THE UNIVERSITY OF TEXAS AT AUSTIN
Department of Statistics and Data Sciences
College of Natural Sciences

Advanced Predictive Models for Complex Data

Lecture 4: Matrix completion - more sophisticated methods

Purnamrita Sarkar
Department of Statistics and Data Science
The University of Texas at Austin

https://psarkar.github.io/teaching

Matrix factorization : non-negative matrix factorization angle

- So, SVD returns directions or principal components
- But these are not interpretable.
- But what if we optimized the following?

$$\min_{\substack{U \in \mathbb{R}^{+}_{m\times k} \\ V \in \mathbb{R}^{+}_{n\times k}}} \|A - UV^T\|_F^2$$

Matrix factorization : non-negative matrix factorization angle

- So, SVD returns directions or principal components
- But these are not interpretable.
- But what if we optimized the following?

$$\min_{\substack{U \in \mathbb{R}^{+}_{m \times k} \\ V \in \mathbb{R}^{+}_{n \times k}}} \|A - UV^T\|_F^2$$

- Is this factorization unique?

Matrix factorization : non-negative matrix factorization angle

- So, SVD returns directions or principal components
- But these are not interpretable.
- But what if we optimized the following?

$$\min_{\substack{U \in \mathbb{R}^+_{m \times k} \\ V \in \mathbb{R}^+_{n \times k}}} \|A - UV^T\|_F^2$$

- Is this factorization unique?
- No – I could multiply U by a positive constant, and divide V by the same and that will give me the same UV^T

The non-negative matrix factorization angle

- Typically, the issues with uniqueness can be resolved by putting constraints on norm or sparsity.
- Despite that, we now have a non-convex loss. There a variety of algorithms, most of them based on alternating minimization type methods.

The non-negative matrix factorization angle

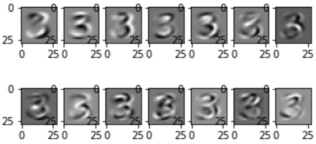
- Typically, the issues with uniqueness can be resolved by putting constraints on norm or sparsity.
- Despite that, we now have a non-convex loss. There a variety of algorithms, most of them based on alternating minimization type methods.
- Here is the loss function minimized by the buit-in NMF code in scikit-learn

$$\begin{aligned}\min_{\substack{U\in\mathbb{R}^+_{m\times k}\\ V\in\mathbb{R}^+_{n\times k}}} \; & \|A - UV^T\|_F^2 + \alpha\beta\left(\|\mathrm{vec}(W)\|_1 + \|\mathrm{vec}(H)\|_1\right) \\ & + \frac{1}{2}\alpha(1-\beta)\left(\|W\|_F^2 + \|H\|_F^2\right)\end{aligned}$$

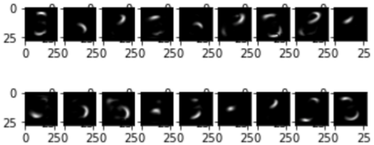
- α, β are regularization parameters

Why Non-negative matrix factorization

- Let us compare the basis vectors obtained using NMF and matrix factorization.
- Look at the right singular vectors or the V in the aforementioned optimization problem with $k = 20$.



PCA basis



NMF basis with 20 components

- Take five minutes to think how the two are different.
- Drumrolls——

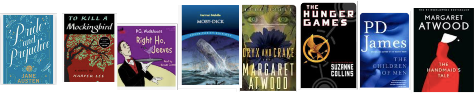
Why Non-negative matrix factorization

- The basis vectors from SVD are global, they are picking up a linear combination of the individual pixel values (which are the features)
- On the other hand, NMF is actually picking up the different parts of the threes, which can be thought of as pieces which are combined together in different ways to give many different handwritten 3's.

Why Non-negative matrix factorization

- The basis vectors from SVD are global, they are picking up a linear combination of the individual pixel values (which are the features)
- On the other hand, NMF is actually picking up the different parts of the threes, which can be thought of as pieces which are combined together in different ways to give many different handwritten 3's.
- So NMF is interpretable, and columns of U and V are not orthogonal.
- But we need conditions to make sure that algorithms return the global optima, and one needs to also think about uniqueness.

Matrix completion - NMF angle



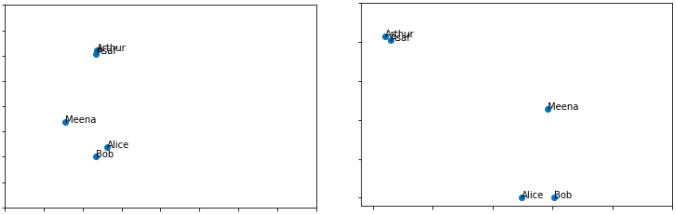
Alice	4	3	5	4	1	1	1	2
Bob	4	5	4	5	1	2	2	1
Meena	4	5	4	4	4	5	5	3
Asaf	1	1	1	1	4	4	4	5
Arthur	2	1	1	1	5	4	4	4

- Remember our user-book rating matrix?
- We random pick 5 elements and set them to zero (think missing).

4	0	5	4	1	1	0	2
4	4	4	5	1	0	2	1
4	5	4	4	0	5	5	3
1	1	1	1	4	4	4	5
2	1	0	1	5	4	4	4

Matrix completion - NMF angle

- We will do SVD to get $Y = U_1 V_1^T$
- We will do NMF to get $Y = U_2 V_2^T$ Now we will use U_1 and U_2 to embed the users as we had before.



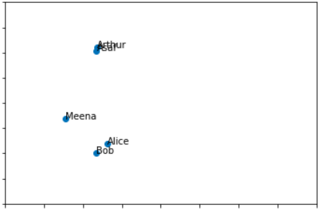
(A) (B)

Table 1: (A) embedding with SVD, (B) embedding with NMF

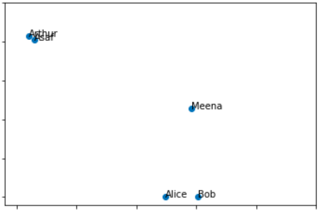
- Take a few minutes to ponder over why these two are different and which one is more interpretable and why.

Matrix completion - NMF angle

- We will do SVD to get $Y = U_1 V_1^T$
- We will do NMF to get $Y = U_2 V_2^T$ Now we will use U_1 and U_2 to embed the users as we had before.



(A)



(B)

Table 2: (A) embedding with SVD, (B) embedding with NMF

- NMF is more interpretable, because Alice/Bob are placed on the X axis (approximately) and Arthur/Asaf on the Y axis, so its almost like the different directions are for the different genres of books, classics and dystopian fiction.

Matrix completion, for real, this time

- So far, we have kind of avoided talking about what to do with unobserved entries, there are many design choices about how to fill them up.
- Now we will provide optimization objectives which deals directly with observed and unobserved entries.
- Notation – Let Ω denote the set of pairs (i, j) such that X_{ij} is observed.

$$\min_{Y} \underbrace{\sum_{ij \in \Omega} (X_{ij} - Y_{ij})^2}_{\text{loss over } \Omega} + \underbrace{\lambda \mathcal{R}(Y)}_{\text{regularization}}$$

Matrix completion, for real, this time

$$\min_{Y \in \mathbb{R}^{m \times n}} \underbrace{\sum_{ij \in \Omega} (X_{ij} - Y_{ij})^2}_{\text{loss over } \Omega} + \underbrace{\lambda \mathcal{R}(Y)}_{\text{regularization that involves all entries}}$$

- So what kind of regularization can we use?
- How about a rank constraint?

Matrix completion, for real, this time

$$\min_{Y \in \mathbb{R}^{m \times n}} \underbrace{\sum_{ij \in \Omega} (X_{ij} - Y_{ij})^2}_{\text{loss over } \Omega} + \underbrace{\lambda \mathcal{R}(Y)}_{\text{regularization that involves all entries}}$$

- So what kind of regularization can we use?
- How about a rank constraint?

$$\min_{Y} \sum_{ij \in \Omega} (X_{ij} - Y_{ij})^2$$

$$\text{s.t.} \operatorname{rank}(Y) = k$$

Matrix completion, for real, this time

- Rank constraints in the above setting can be combinatorially very hard.

Matrix completion, for real, this time

- Rank constraints in the above setting can be combinatorially very hard.
- Funny right? because some rank minimization problems are easy – if I ask you to return the best rank k approximation of a matrix. But the moment you add more structure, i.e. minimize frobenius norm over a set of pairs, things get hairy.

> There are several special cases of the RMP that have well known solutions. For example, approximating a given matrix with a low-rank matrix in spectral or Frobenius norm is an RMP that can be solved via singular value decomposition (SVD) [15]. However, in general, the RMP is known to be computationally intractable (NP-hard) [26]..

Rank Minimization and Applications in System Theory.Fazel, Hindi and Boyd, 2004

Matrix completion, for real, this time

- Instead, what is often used is the nuclear norm penalty.

$$\min_{Y \in \mathbb{R}^{m \times n}} \sum_{ij \in \Omega} (X_{ij} - Y_{ij})^2 + \lambda \|Y\|_*$$

- Recall that the nuclear norm is basically the sum of the singular values of a matrix.
- The rank can be thought of as a ℓ_0 "norm" of the vector of singular values, constraints based on which are not convex
- The nuclear norm is like a ℓ_1 norm.

Matrix completion, for real, this time

- Instead, what is often used is the nuclear norm penalty.

$$\min_{Y \in \mathbb{R}^{m \times n}} \sum_{ij \in \Omega} (X_{ij} - Y_{ij})^2 + \lambda \|Y\|_*$$

- Recall that the nuclear norm is basically the sum of the singular values of a matrix.
- The rank can be thought of as a ℓ_0 "norm" of the vector of singular values, constraints based on which are not convex
- The nuclear norm is like a ℓ_1 norm.
- As it turns out the nuclear norm is the tightest convex relaxation of of rank of a matrix. (See Fazel, Hindi and Boyd)

Matrix completion, for real, this time

- Instead, what is often used is the nuclear norm penalty.

$$\min_{Y \in \mathbb{R}^{m \times n}} \sum_{ij \in \Omega} (X_{ij} - Y_{ij})^2 + \lambda \|Y\|_*$$

- Recall that the nuclear norm is basically the sum of the singular values of a matrix.
- The rank can be thought of as a ℓ_0 "norm" of the vector of singular values, constraints based on which are not convex
- The nuclear norm is like a ℓ_1 norm.
- As it turns out the nuclear norm is the tightest convex relaxation of of rank of a matrix. (See Fazel, Hindi and Boyd)
  - In plain words, over a bounded set, the nuclear norm function is the largest convex function smaller than the rank function, otherwise also known as the convex envelop.

Matrix completion, for real, this time

- Instead, what is often used is the nuclear norm penalty.

$$\min_{Y \in \mathbb{R}^{m \times n}} \sum_{ij \in \Omega} (X_{ij} - Y_{ij})^2 + \lambda \|Y\|_*$$

- Set $Z^{(0)} = 0$
- Set $Y_{ij}^{(t+1)} = \begin{cases} X_{ij} & (i,j) \in \Omega \\ Z_{ij}^{(t)} & (i,j) \notin \Omega \end{cases}$
- Compute $Y^{(t)} = U\text{diag}[\sigma_1, \ldots, \sigma_r]V^T$
- Compute $Z^{(t+1)} = U\text{diag}[(\sigma_1 - \lambda)_+, \ldots, (\sigma_r - \lambda)_+]V^T$

Lets do a real example

- We will load an image and convert its grayscale version into a matrix.

```
from keras.preprocessing.image import load_img
# load the image
img = load_img('bondi_beach.jpeg',grayscale=True)
# report details about the image
print(type(img))
print(img.format)
print(img.mode)
print(img.size)
# show the image
img.show()
```

```
<class 'PIL.Image.Image'>
None
L
(640, 427)
```

Lets do a real example

- We will load an image and convert its grayscale version into a matrix.

```
from keras.preprocessing.image import img_to_array
img_array = np.squeeze(img_to_array(img))
print(img_array.dtype)
print(img_array.shape)
```

```
float32
(427, 640)
```

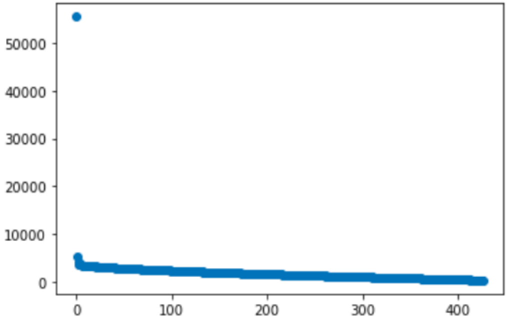
Lets do a real example

- Now we will sample 100,000 entries at random and withhold them.

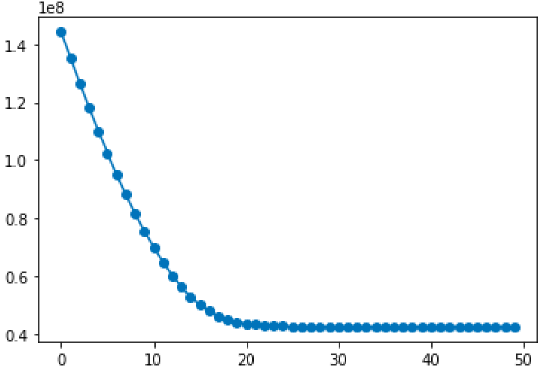
```
from keras.preprocessing.image import array_to_img

img2=copy.copy(img_array)
n=range(100000)
nrow=427
ncol=640
for i in n:
    row=random.choice(range(nrow))
    col=random.choice(range(ncol))
    row
    img2[row,col]=0;
plt.matshow(img2)
```

Applying the Soft Impute algorithm

- But what kind of a λ do we use?
- Here is a plot of the singular values of the matrix `img2`



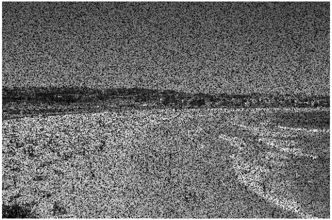
Applying the Soft Impute algorithm



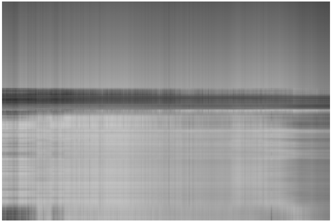
- Good sanity check to see if the loss is going down with the number of iterations.

Applying the Soft Impute algorithm with $\lambda = 2000$ and $\lambda = 50$

Original



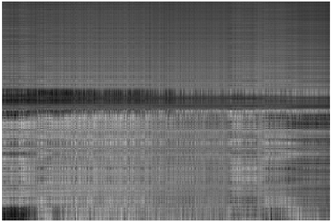
Noisy



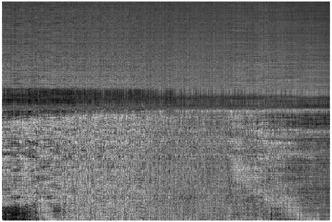
$\lambda = 2000$

$\lambda = 50$

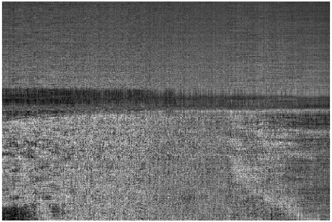
Applying SVD



$K = 5$



$K = 20$



$K = 30$

$K = 50$