Structured matrix factorizations

An extremely large variety of interesting and important problems in machine learning can be formulated as:

Given a $d \times n$ matrix $\mathbf{X}$, find a $d \times r$ matrix $\mathbf{B}$ and a $r \times n$ matrix $\mathbf{C}$ such that

$$\mathbf{X} \approx \mathbf{BC}$$

under the constraints that $\mathbf{B}$ and $\mathbf{C}$ have some kind of ***structure***

Examples we have seen so far have included PCA, MDS, and extensions like kernel PCA, Isomap, LLE, etc.

In these examples, the only structure we have enforced has been ***orthonormality*** – this is not always quite enough…

Example: Eigenfaces

Fill up the columns of $\mathbf{X}$ with vectorized images of faces

Given thousands of such faces, we can still approximate $\mathbf{X}$ very well as a low-rank matrix (e.g., rank 200)

Given a low-rank factorization $\mathbf{X} \approx \mathbf{BC}$, the columns of $\mathbf{B}$ represent a small number of "eigenfaces" which we can use to build up any face

Example: Eigenfaces

Example: Topic modeling

Topic modeling refers to a broad class of algorithms that to automatically sort documents into groups which share a common theme/topic based on their semantic content

The input to these algorithms is the familiar "bag of words" representation, i.e., the matrix $\mathbf{X}$ with entries

$$X[i,j] = \begin{array}{l}\text{\# of times word } i \text{ appeared} \\ \text{in document } j\end{array}$$

The rows of $\mathbf{X}$ tend to be correlated, so we can typically approximate $\mathbf{X} \approx \mathbf{BC}$ as a low-rank matrix

This expresses each document as a combination of the columns of $\mathbf{B}$, each of which ideally each represent a coherent "topic"

Non-negative matrix factorization

In both of the two previous cases, it is actually very hard to interpret what the latent factors tell us about the data, because we are not enforcing the proper structure on $\mathbf{B}, \mathbf{C}$

If we instead constrain $\mathbf{B}, \mathbf{C}$ to have non-negative entries, we get different (and perhaps more interesting) results:

$$\underset{\mathbf{B},\mathbf{C}}{\text{minimize}} \ \|\mathbf{X} - \mathbf{B}\mathbf{C}\|_F^2 \quad \text{s.t.} \quad \mathbf{B} \geq \mathbf{0}, \mathbf{C} \geq \mathbf{0}$$

The idea behind this is that by forcing the factors to be positive, we must build up $\mathbf{X}$ only by adding components (instead of relying on potentially complicated cancellations)

This will hopefully make it easier to interpret the results

Solving the NNMF problem

$$\underset{\mathbf{B},\mathbf{C}}{\text{minimize}} \ \|\mathbf{X} - \mathbf{B}\mathbf{C}\|_F^2 \quad \text{s.t.} \quad \mathbf{B} \geq \mathbf{0}, \mathbf{C} \geq \mathbf{0}$$

Solving this problem is much more involved than the unconstrained case

- no closed form solution (the SVD does not help us)
- non-convex optimization problem

However, notice that if we fix $\mathbf{B} = \mathbf{B}^{(0)}$, then the program

$$\underset{\mathbf{C}}{\text{minimize}} \ \|\mathbf{X} - \mathbf{B}^{(0)}\mathbf{C}\|_F^2 \quad \text{s.t.} \quad \mathbf{C} \geq \mathbf{0}$$

is column separable, i.e., for each i we can separately solve

$$\underset{\mathbf{c}_i}{\text{minimize}} \ \|\mathbf{x}_i - \mathbf{B}^{(0)}\mathbf{c}_i\|_2^2 \quad \text{s.t.} \quad \mathbf{c}_i \geq \mathbf{0}$$

NNMF and alternating minimization

$$\underset{\mathbf{c}_i}{\text{minimize}} \ \|\mathbf{x}_i - \mathbf{B}^{(0)}\mathbf{c}_i\|_2^2 \quad \text{s.t.} \quad \mathbf{c}_i \geq \mathbf{0}$$

This is a quadratic program that is easily solvable using off-the-shelf software

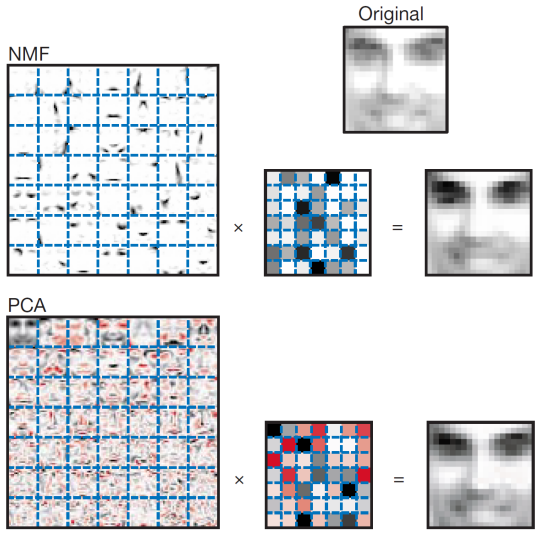
The same is true if we fix $\mathbf{C} = \mathbf{C}^{(0)}$ and optimize over $\mathbf{B}$

Alternating minimization algorithm

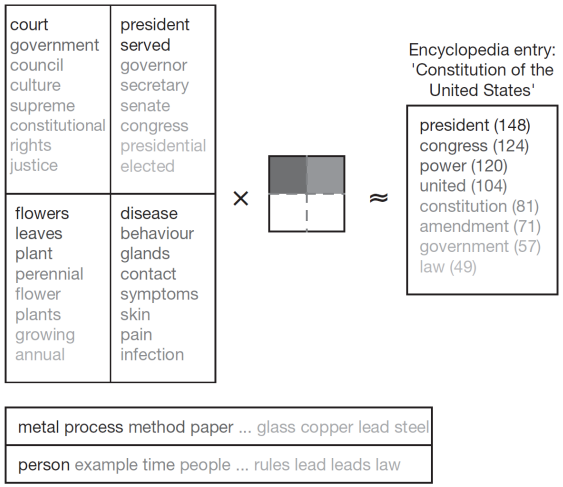
Input: $\mathbf{X}$ together with initial guess for $\mathbf{B} = \mathbf{B}^{(0)}$

1. Compute $\mathbf{C}^{(j)}$ treating $\mathbf{B} = \mathbf{B}^{(j)}$ as fixed
2. Compute $\mathbf{B}^{(j+1)}$ treating $\mathbf{C} = \mathbf{C}^{(j)}$
3. Iterate until convergence

Example: NNMF for faces

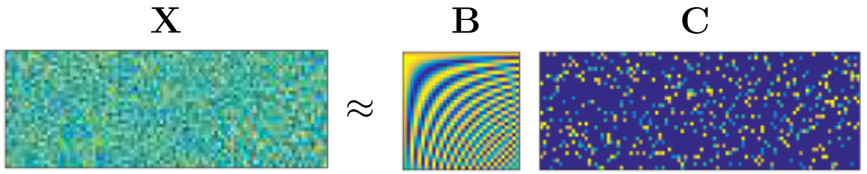


Example: NNMF for topic modeling



Dictionary learning

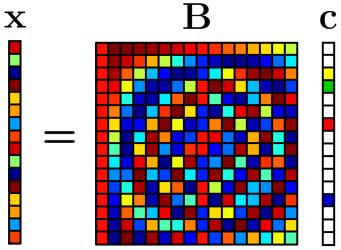
In ***dictionary learning***, our goal is again to form a factorization of the form $\mathbf{X} \approx \mathbf{BC}$, but now with the structure that $\mathbf{C}$ is ***sparse***



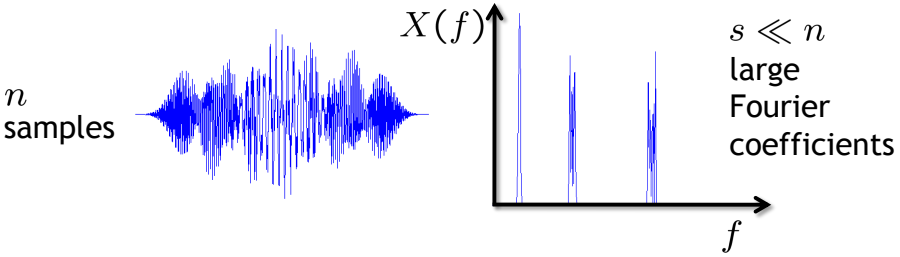
The idea is that we are searching for a basis or dictionary $\mathbf{B}$ such that every $\mathbf{x}_i$ can be expressed as a weighted combination of just a few columns of $\mathbf{B}$

Sparsity

$$\mathbf{x} = \mathbf{Bc} = \sum_j c_j \mathbf{b}_j$$



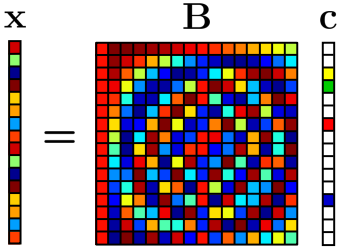
s nonzero entries



$s \ll n$ large Fourier coefficients

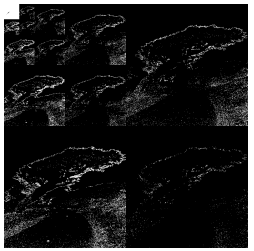
Sparsity

$$\mathbf{x} = \mathbf{Bc} = \sum_j c_j \mathbf{b}_j$$



s nonzero entries

n pixels



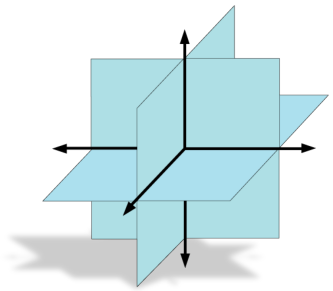
$s \ll n$ large wavelet coefficients

Sparsity as a nonlinear model

Sparse models are fundamentally different than subspace models like PCA

We may use only $s \ll n$ coefficients for each $\mathbf{x}$, but ***which*** coefficients changes depending on $\mathbf{x}$

Sparsity is really a ***union of subspaces*** model



Sparse approximation

Before we discuss dictionary learning, let us first consider the problem of finding a sparse representation for a given $\mathbf{x}$

One way to pose this problem is via the optimization problem

$$\underset{\mathbf{c}}{\text{minimize}} \ \|\mathbf{x} - \mathbf{B}\mathbf{c}\|_2^2 \quad \text{s.t.} \quad \|\mathbf{c}\|_0 \leq s$$

If $\mathbf{B}$ is orthonormal, then

$$\|\mathbf{x} - \mathbf{B}\mathbf{c}\|_2^2 = \|\mathbf{B}^T(\mathbf{x} - \mathbf{B}\mathbf{c})\|_2^2 = \|\mathbf{B}^T\mathbf{x} - \mathbf{c}\|_2^2$$

In this case, solving the problem is easy

Simply compute $\mathcal{I} = \{s \text{ largest elements of } \mathbf{y} = \mathbf{B}^T\mathbf{x}\}$ and set

$$\widehat{c}(j) = \begin{cases} y(j) & j \in \mathcal{I} \\ 0 & j \notin \mathcal{I} \end{cases}$$

Sparse approximation

In the more general setting where $\mathbf{B}$ is not orthonormal, this problem is not so easy to solve

NP-hard in general

This is important in practice, since typically one can obtain $\mathbf{B}$ that do a ***much*** better job of "sparsifying" our data if we allow them to be ***overcomplete*** (more columns than rows)

However, there are some good heuristics which also have strong theoretical justification when $\mathbf{B}$ satisfies some nice regularity conditions

LASSO: $\underset{\mathbf{c}}{\text{minimize}} \ \|\mathbf{x} - \mathbf{B}\mathbf{c}\|_2^2 \quad \text{s.t.} \quad \|\mathbf{c}\|_1 \leq \tau$

Greedy algorithms: ***orthogonal matching pursuit***

Dictionary learning formulation

Getting back to dictionary learning, the problem we would like to solve can be stated as the optimization problem

$$\underset{\mathbf{B},\mathbf{C}}{\text{minimize}} \ \|\mathbf{X} - \mathbf{B}\mathbf{C}\|_F^2 \ \text{ s.t. } \ \|\mathbf{c}_i\|_0 \leq s, \ i = 1, \ldots, n$$

Just as with NNMF, this is nonconvex and extremely difficult to solve exactly

However, we can take the same approach as before and consider an alternating minimization strategy to tackling this:

- fix $\mathbf{B}$ and optimize over $\mathbf{C}$
- then, fix $\mathbf{C}$ and optimize over $\mathbf{B}$
- repeat until convergence

K-SVD algorithm

There are many variants of this approach, but perhaps the most popular is the ***K-SVD*** algorithm

Input: $\mathbf{X}$, $\mathbf{B}^{(0)}$, desired sparsity level s

1. With $\mathbf{B}^{(j)}$ fixed, solve a series of n sparse approximation problems using orthogonal matching pursuit to obtain $\mathbf{C}^{(j)}$
2. With $\mathbf{C}^{(j)}$ fixed, solve the optimization problem

$$\mathbf{B}^{(j+1)} = \arg\min_{\mathbf{B}} \|\mathbf{X} - \mathbf{B}\mathbf{C}^{(j)}\|_F^2$$

3. Iterate until convergence

The optimization problem in step 2 can be broken into independent problems for each column of $\mathbf{B}$, and can be solved by taking an SVD

Example: Natural images

The first people to look at dictionary learning in this way were Olshausen and Field in a seminal 1996 paper

- Olshausen gave a talk recently here at Georgia Tech

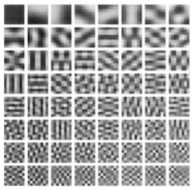
They were studying human vision, and wondered if it might be the case that the human visual system is actually trained to produce sparse representations of natural images

They formed a large collection of patches of images from natural scenes

Compared PCA to dictionary learning

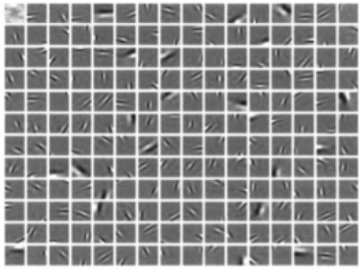
Example: Natural images

PCA



very similar to DCT

Learned dictionary

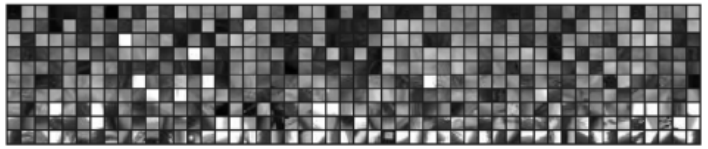


actually matches experimental data for neural responses in human/animal visual systems

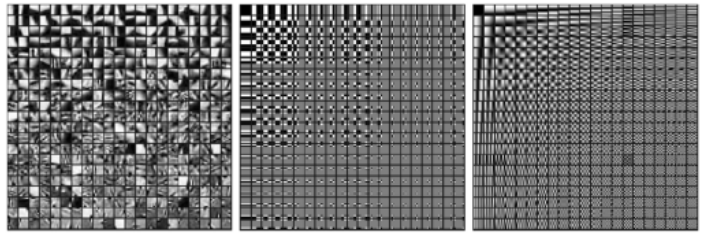
Example: Natural images

The K-SVD algorithm provides similar results:

Example image patches



K-SVD dictionary Haar wavelets DCT



Missing and corrupted data

Let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ denote a $d \times n$ "data matrix"

Recall that we can compute the principal components simply by computing the singular value decomposition

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

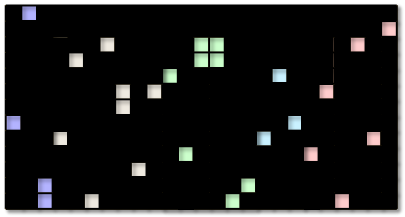
The principal eigenvectors are the first k columns of $\mathbf{U}$

The inherent assumption in PCA is that $\mathbf{X}$ is (at least approximately) low rank

- i.e., most of the singular values are close to zero

What happens if some of the entries in $\mathbf{X}$ are missing?

PCA with missing data



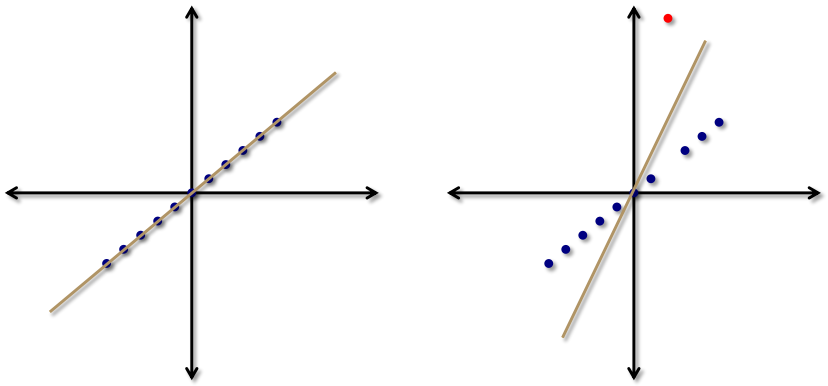
Given a set Ω of indices, we get to observe $\mathbf{Y} = \mathbf{X}_\Omega$, and we would like to recover $\mathbf{X}$

- also known as ***matrix completion***
- can also be extended to the case where some of the entries have been corrupted (i.e., the set Ω is unknown) under the name of ***robust PCA***

Outliers

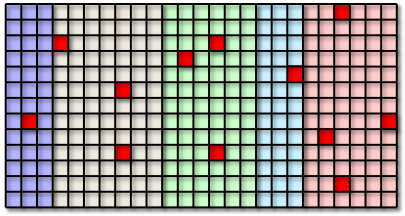
What happens if there are outliers in our data set?

PCA can be ***extremely sensitive*** to outliers/corruptions



Modeling outliers

In the presence of outliers, our data matrix $\mathbf{X}$ is no longer low-rank because some of the entries have been corrupted



$$\mathbf{X} = \underbrace{\mathbf{L}}_{\text{low-rank}} + \underbrace{\mathbf{S}}_{\text{corruptions}}$$

Robust PCA

The problem of robust PCA boils down to a separation problem

Given $\mathbf{X} = \mathbf{L} + \mathbf{S}$, determine $\mathbf{L}$ (and hence $\mathbf{S}$)

Is this possible?

Example

If I tell you $3 = x + y$, what are x and y?

We need to make some ***assumptions*** on $\mathbf{L}$ and $\mathbf{S}$ to make this problem tractable

- $\mathbf{L}$ is ***low-rank*** (i.e., PCA even makes sense)
- $\mathbf{S}$ is ***sparse*** (only a few entries are corrupted)

How to perform separation/recovery?

Matrix completion

$$\min_{\mathbf{X}} \ \text{rank}(\mathbf{X})$$
$$\text{s.t. } \mathbf{X}_\Omega = \mathbf{Y}$$

Robust PCA

$$\min_{\mathbf{L},\mathbf{S}} \ \text{rank}(\mathbf{L}) + \lambda \|\mathbf{S}\|_0$$
$$\text{s.t. } \mathbf{L} + \mathbf{S} = \mathbf{X}$$

What's the problem with these approaches?

- nonconvex, intractable, NP hard, etc.

There are alternating minimization approaches to both, or…

Convex relaxation

We have already seen that an effective proxy for $\|\mathbf{S}\|_0$ that encourages sparsity is the "convex relaxation"

$$\|\mathbf{S}\|_1 := \sum_{i,j} |S_{i,j}|$$

What about a convex relaxation of $\text{rank}(\mathbf{L})$?

Observe that if we let $\boldsymbol{\sigma}$ denote the vector containing the singular values of $\mathbf{L}$, then $\text{rank}(\mathbf{L}) = \|\boldsymbol{\sigma}\|_0$

What if we replace $\text{rank}(\mathbf{L})$ with $\|\mathbf{L}\|_* = \|\boldsymbol{\sigma}\|_1$?

It turns out that $\|\mathbf{L}\|_*$ is indeed a convex function, and hence something we can potentially optimize efficiently

- typically called the ***nuclear norm*** or ***Shatten 1-norm***
- we can write $\|\mathbf{L}\|_* = \text{trace}(\sqrt{\mathbf{L}^T\mathbf{L}})$

Convex programs for separation/recovery

Matrix completion

$$\min_{\mathbf{X}} \ \|\mathbf{X}\|_*$$
$$\text{s.t. } \mathbf{X}_\Omega = \mathbf{Y}$$

Robust PCA

$$\min_{\mathbf{L},\mathbf{S}} \ \|\mathbf{L}\|_* + \lambda \|\mathbf{S}\|_1$$
$$\text{s.t. } \mathbf{L} + \mathbf{S} = \mathbf{X}$$

Application: Collaborative filtering

The "Netflix Problem"

$X(i,j) =$ how much user i likes movie j

Rank 1 model: $u_i =$ how much user i likes romantic movies

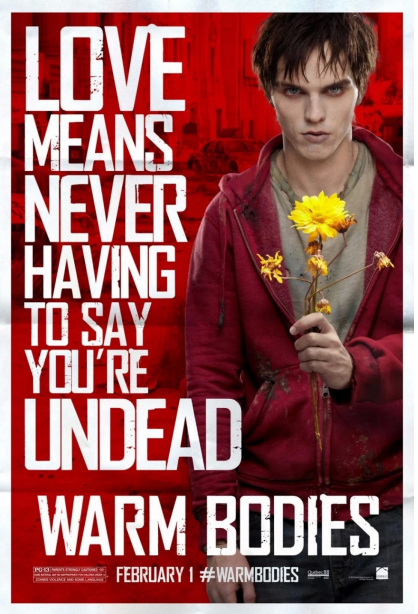
$v_j =$ amount of romance in movie j

$X(i,j) = u_i v_j$

Rank 2 model: $w_i =$ how much user i likes zombie movies

$x_j =$ amount of zombies in movie j

$X[i,j] = u_i v_j + w_i x_j$



Application: Response data

More generally, we might have

$X[i,j] =$ response from person i to question j

Low-rank models are commonly applied to all kinds of "response" data

- surveys (census)
- standardized tests
- market research

If we have lots of questions and lots of participants

- we might not be able to ask all participants all possible questions
- we might need to deal with malicious responses

Application: Multidimensional scaling

Suppose that $\mathbf{X}$ represents the pairwise distances between a set of objects, i.e.,

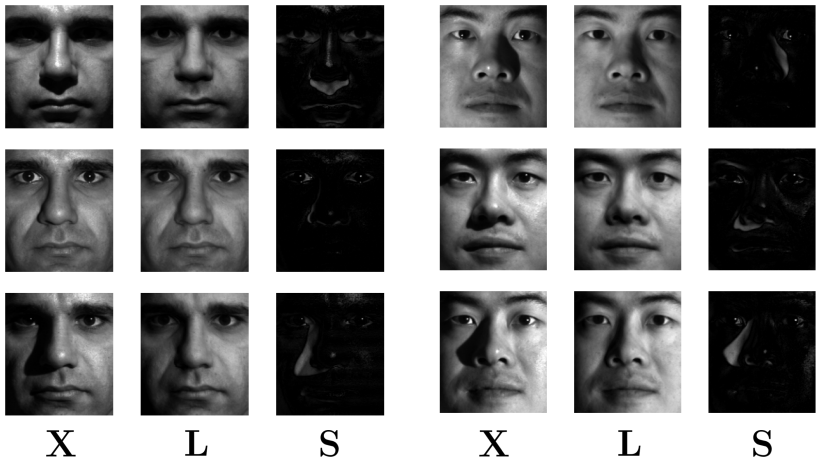
$$X[i,j] = \|\mathbf{x}_i - \mathbf{x}_j\|^2$$

Recall from before that in this case $\mathrm{rank}(\mathbf{X}) = d + 2$ where d is the dimension of the $\mathbf{x}_i$

If we have a large number of objects, we may not be able to observe/measure all possible pairwise distances

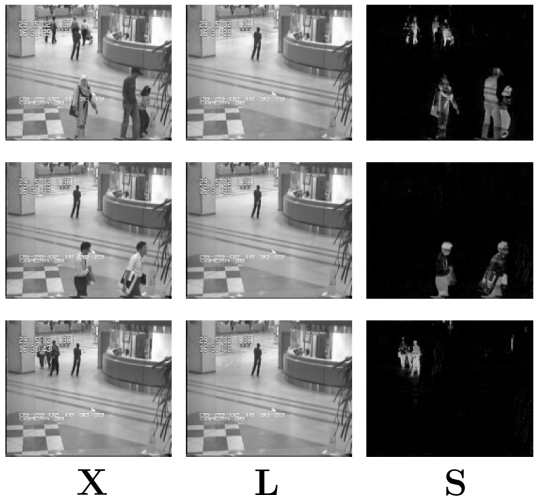
If these distances are calculated from similarity judgments by people, we may have outliers/corruptions

Application: Removing face illumination



[Candès et al., 2009]

Application: Background subtraction



[Candès et al., 2009]

When does it work?

In general, we cannot always guarantee that we can solve the separation/recovery problems

Examples where separation/recovery might be impossible:

- too many of the entries are corrupted
  - in this case our corruptions are not really sparse
- we do not observe enough of the entries
  - e.g., we do not even observe some rows/columns
- the low-rank portion of the matrix is also sparse
  - we will not be able to separate it from sparse corruptions
  - we might not observe any of the relevant entries

How much data do we need?

Consider the matrix completion problem

To recover a matrix with rank k, how many observations do we expect to need?

If $\mathbf{X}$ is $d \times n$, then there are

$$dk - \frac{k(k+1)}{2} + nk - \frac{k(k+1)}{2} + k = k(d + n - k)$$

degrees of freedom

We can reasonably expect that we will need to have

$$|\Omega| \gtrsim Ck(d+n)$$

In fact, we need a little bit more:

$$|\Omega| \gtrsim Ck(d+n)\log(d+n)$$

Incoherence

We also need to avoid the case where $\mathbf{X}$ is sparse

We will assume that $\mathbf{X}$ satisfies the incoherence condition:

If $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, the incoherence condition with parameter μ states that

$$\max_i \|\mathbf{U}^T\mathbf{e}_i\|^2 \leq \frac{\mu k}{d} \qquad \max_i \|\mathbf{V}^T\mathbf{e}_i\|^2 \leq \frac{\mu k}{n}$$

$$\|\mathbf{U}\mathbf{V}^T\|_\infty \leq \sqrt{\frac{\mu k}{dn}}$$

Robust PCA

Theorem

Suppose that $\mathbf{X} = \mathbf{L} + \mathbf{S}$ where $\mathbf{L}$ satisfies the incoherence property,

$$\text{rank}(\mathbf{L}) \leq C'_\mu \min(d, n)/\log^2(\max(d, n)),$$

and the support set of $\mathbf{S}$ is chosen uniformly at random from all sets of size $m \leq C' n_1 n_2$.

Then if we set $\lambda = 1/\sqrt{n_{\min}}$ we achieve exact separation of $\mathbf{L}$ and $\mathbf{S}$ with high probability.

See papers by Candès, Li, Ma, and Wright, and others

Matrix completion

Theorem

Suppose that $\mathbf{X}$ has rank k and satisfies the incoherence condition. If we observe m entries of $\mathbf{X}$ sampled uniformly at random, then as long as

$$m \gtrsim C_\mu k(d + n)\log^2(d + n)$$

we will recover $\mathbf{X}$ ***exactly*** with high probability

See papers by Recht, Candès, Tao, and many others