IV. Modeling and Model Selection

One of the major themes in this course has been systematic methods for inference under the assumption that the unknowns obeyed some kind of structure. For instance, we have seen many ways to estimate a function from its samples (using least-squares, for example) when that function is assumed to live in a known subspace. We have also seen examples of estimating the probability density function of a random variable/vector when that pdf is assumed to have some parametric form (e.g. the MLE). Finally, we have used pdfs to model the unknowns themselves (Gaussian estimation, MAP estimation, etc).

In this final chapter of the course, we will learn a little bit about how to fit models to data. Problems of this type are typically referred to as "unsupervised learning" problems in the ML literature, but we will shy away from this terminology. We will also encounter, for purposes of illustration, new models for data.

In the very first set of notes in this course, we made the distinction between *geometric* models, and *probabilistic* models. The former is basically the specification of some set (e.g. subspace) that the signal/feature vector/parameters live in or near. The latter is a density function that indicated which values of the vector are expected to be more likely than others.

We will start with the most fundamental geometric model selection problem: how to fit a subspace to a bunch of points.

Principal Components Analysis (PCA)

On many occasions over the last three months, we have made the assumption that some unknown vector that we were interested in belonged to a subspace. PCA is a way in which we can *learn* this subspace from examples.

We are given data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^D$. We would like to find the $K < D$ dimensional subspace that comes closest to containing these points — maybe this is a good fit, maybe it isn't, but the point here is that we can set up an optimization program and solve it (and in the process determine how good the fit is).

PCA gives an easy two-step procedure for solving this problem:

1. Form the sample covariance matrix

$$\hat{\boldsymbol{R}} = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}} \tag{1}$$

2. Factor $\hat{\boldsymbol{R}}$ using an eigenvalue decomposition

$$\hat{\boldsymbol{R}} = \hat{\boldsymbol{V}} \boldsymbol{\Lambda} \hat{\boldsymbol{V}}^{\mathrm{T}}.$$

   Then take the first K eigenvectors (the eigenvectors corresponding to the K largest eigenvalues) — $\mathrm{span}\{\hat{\boldsymbol{v}}_1, \ldots, \hat{\boldsymbol{v}}_K\}$ is the optimal subspace.

Let's make this precise.

Derivation of Principal Components Analysis

We want to find a subspace $\mathcal{T} \subset \mathbb{R}^D$ with $\dim(\mathcal{T}) = K$ that comes closest to containing $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. To make the notion of"closest" precise, we will score a subspace $\mathcal{T}$ by looking at the sum of the squared distances between the $\boldsymbol{x}_n$ and $\mathcal{T}$:

$$\text{Score}(\mathcal{T}) = \sum_{n=1}^{N} \text{dist}(\boldsymbol{x}_n, \mathcal{T})^2.$$

We know that given a orthobasis $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K$ for $\mathcal{T}$, the closest point in $\mathcal{T}$ is

$$\hat{\boldsymbol{x}}_n = \sum_{k=1}^{K} \langle \boldsymbol{x}_n, \boldsymbol{u}_k \rangle \boldsymbol{u}_k = \boldsymbol{U}\boldsymbol{U}^{\mathrm{T}} \boldsymbol{x}_n,$$

with distance

$$\|\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n\|_2^2 = \|(\mathbf{I} - \boldsymbol{U}\boldsymbol{U}^{\mathrm{T}})\boldsymbol{x}_n\|_2^2,$$

where $\boldsymbol{U} = \begin{bmatrix} \boldsymbol{u}_1 & \cdots & \boldsymbol{u}_K \end{bmatrix}$. Thus we can replace our search for a subspace with a search for an *orthobasis*; PCA solves

$$\underset{\boldsymbol{U} \in \mathbb{R}^{D \times K}}{\text{minimize}} \ \sum_{n=1}^{N} \|(\mathbf{I} - \boldsymbol{U}\boldsymbol{U}^{\mathrm{T}})\boldsymbol{x}_n\|_2^2 \quad \text{subject to} \quad \boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = \mathbf{I}. \tag{2}$$

Note that the solution to the program above is certainly not unique — any orthobasis for the same subspace will lead to exactly the same functional value.

Notice that because $\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = \mathbf{I}$, we have

$$(\mathbf{I} - \boldsymbol{U}\boldsymbol{U}^{\mathrm{T}})(\mathbf{I} - \boldsymbol{U}\boldsymbol{U}^{\mathrm{T}}) = (\mathbf{I} - \boldsymbol{U}\boldsymbol{U}^{\mathrm{T}}),$$

and thus

$$\|(\mathbf{I} - \boldsymbol{U}\boldsymbol{U}^{\mathrm{T}})\boldsymbol{x}_n\|_2^2 = \boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{x}_n - \boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{U}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{x}_n = \|\boldsymbol{x}_n\|_2^2 - \|\boldsymbol{U}^{\mathrm{T}}\boldsymbol{x}_n\|_2^2,$$

and since $\|\boldsymbol{x}_n\|_2^2$ does not depend on $\boldsymbol{U}$, we can re-write the PCA program as

$$\underset{\boldsymbol{U}\in\mathbb{R}^{D\times K}}{\text{maximize}} \quad \sum_{n=1}^{N} \|\boldsymbol{U}^{\mathrm{T}}\boldsymbol{x}_n\|_2^2 \quad \text{subject to} \quad \boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = \mathbf{I}.$$

Since[1] $\|\boldsymbol{U}^{\mathrm{T}}\boldsymbol{x}_n\|_2^2 = \mathrm{trace}(\boldsymbol{U}^{\mathrm{T}}\boldsymbol{x}_n\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{U})$, we have

$$\begin{aligned}\sum_{n=1}^{N} \|\boldsymbol{U}^{\mathrm{T}}\boldsymbol{x}_n\|_2^2 &= \sum_{n=1}^{N} \mathrm{trace}(\boldsymbol{U}^{\mathrm{T}}\boldsymbol{x}_n\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{U}) \\ &= \mathrm{trace}\left(\boldsymbol{U}^{\mathrm{T}}\left(\sum_{n=1}^{N}\boldsymbol{x}_n\boldsymbol{x}_n^{\mathrm{T}}\right)\boldsymbol{U}\right) \\ &= N\,\mathrm{trace}(\boldsymbol{U}^{\mathrm{T}}\hat{\boldsymbol{R}}\boldsymbol{U}),\end{aligned}$$

where $\hat{\boldsymbol{R}}$ is the sample covariance defined in (1). We now know that PCA is equivalent to solving

$$\underset{\boldsymbol{U}\in\mathbb{R}^{D\times K}}{\text{maximize}} \quad \mathrm{trace}(\boldsymbol{U}^{\mathrm{T}}\hat{\boldsymbol{R}}\boldsymbol{U}) \quad \text{subject to} \quad \boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = \mathbf{I}.$$

By construction, $\hat{\boldsymbol{R}}$ is symmetric positive semi-definite, so it has an eigenvalue decomposition

$$\boldsymbol{R} = \hat{\boldsymbol{V}}\boldsymbol{\Lambda}\hat{\boldsymbol{V}}^{\mathrm{T}}.$$

[1] If you are uncomfortable with this equality, you should take the time right now to sit down an derive it.

We have

$$\text{trace}(\boldsymbol{U}^{\mathrm{T}}\hat{\boldsymbol{R}}\boldsymbol{U}) = \text{trace}(\boldsymbol{U}^{\mathrm{T}}\hat{\boldsymbol{V}}\boldsymbol{\Lambda}\hat{\boldsymbol{V}}^{\mathrm{T}}\boldsymbol{U}) = \text{trace}(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{W}),$$

where $\boldsymbol{W} = \hat{\boldsymbol{V}}^{\mathrm{T}}\boldsymbol{U}$, and note[2] that $\boldsymbol{W}^{\mathrm{T}}\boldsymbol{W} = \mathbf{I}$. Thus we can solve (2) by solving

$$\underset{\boldsymbol{W}\in\mathbb{R}^{D\times K}}{\text{maximize}} \;\; \text{trace}(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{W}) \quad \text{subject to} \quad \boldsymbol{W}^{\mathrm{T}}\boldsymbol{W} = \mathbf{I},$$

and then taking $\hat{\boldsymbol{U}} = \hat{\boldsymbol{V}}\hat{\boldsymbol{W}}$ to get the minimizer in (2).

We can show that the last maximization program above is equivalent to a simple linear program that we can solve by inspection. Let $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K$ be the columns of $\boldsymbol{W}$. Then

$$\begin{aligned}
\text{trace}(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{W}) &= \sum_{k=1}^{K} \boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{w}_k \\
&= \sum_{k=1}^{K}\sum_{d=1}^{D} (w_k[d])^2\lambda_d \\
&= \sum_{d=1}^{D} h_d\lambda_d, \quad h_d = \sum_{k=1}^{K}(w_k[d])^2 = \sum_{k=1}^{K}(W[d,k])^2.
\end{aligned}$$

The $h_d,\ d = 1, \ldots, D$ above are the sums of the squares of the *rows* of $\boldsymbol{W}$. It is clear that $h_d \geq 0$. It is also true that

$$\sum_{d=1}^{D} h_d = K,$$

[2] $\hat{\boldsymbol{V}}$ is a $D \times D$ orthonormal matrix, so $\hat{\boldsymbol{V}}\hat{\boldsymbol{V}}^{\mathrm{T}} = \mathbf{I}$.

as the fact that the norm of each columns of $\boldsymbol{W}$ is 1 means that

$$\sum_{d=1}^{D}\sum_{k=1}^{K} W[d,k]^2 = \sum_{k=1}^{K}\left(\sum_{d=1}^{D} W[d,k]^2\right) = \sum_{k=1}^{K} 1 = K.$$

Finally, it is also true that $h_d \leq 1$. Here is why: since the columns of $\boldsymbol{W}$ are orthonormal, they can be considered as part of an orthonormal basis for all of $\mathbb{R}^D$. That is, there is a (and actually there are many) $D \times (D-K)$ matrix $\boldsymbol{W}_0$ such that the columns of

$$\boldsymbol{W}' = \begin{bmatrix}\boldsymbol{W} & \boldsymbol{W}_0\end{bmatrix}$$

form an orthonormal basis for $\mathbb{R}^D$. Since $\boldsymbol{W}'$ is square, $\boldsymbol{W}'\boldsymbol{W}'^{\mathrm{T}} = \mathbf{I}$, meaning the sum of the squares of each row are equal to 1. Thus

$$h_d = \sum_{k=1}^{K} W[d,k]^2 = \sum_{k=1}^{K} W'[d,k]^2 \leq \sum_{k=1}^{D} W'[d,k]^2 = 1.$$

With these constraints on the h_d, let's see how large we can make the quantity of interest:

$$\underset{\boldsymbol{h}\in\mathbb{R}^D}{\text{maximize}} \sum_{d=1}^{D} h_j\lambda_d \quad \text{subject to} \quad \sum_{d=1}^{D} h_d = K, \quad 0 \leq h_d \leq 1.$$

This is a linear program, but we can intuit the answer. Since all of the λ_d are positive, we want to have their weights (i.e. the h_d) as large as possible for the largest entries. Since the weights are constrained to be less than 1, and their sum is K, this simply means we assign a weight of 1 to the K largest terms, and 0 to the others:

$$\hat{h}_d = \begin{cases} 1, & j = 1,\ldots,K, \\ 0, & \text{otherwise} \end{cases}$$

This means that the sum of the squares of the entries in the rows of the corresponding $\hat{\boldsymbol{W}}$ are 1 for the first K, and zero below — there are many matrices with orthonormal columns which fit the bill, but a specific one which does is

$$\hat{\boldsymbol{W}} = \begin{bmatrix} \mathbf{I}_{K\times K} \\ \mathbf{0}_{(D-K)\times K} \end{bmatrix}. \tag{3}$$

Taking $\hat{\boldsymbol{U}} = \hat{\boldsymbol{V}}\hat{\boldsymbol{W}}$, this results in

$$\hat{\boldsymbol{U}} = \begin{bmatrix} \hat{\boldsymbol{v}}_1 & \hat{\boldsymbol{v}}_2 & \cdots & \hat{\boldsymbol{v}}_K \end{bmatrix}.$$

Including an affine offset

PCA is easily modified to include an affine offset (so we fit a subspace that has been shifted away from the origin) that we learn as well. Now, we simply solve

$$\underset{\boldsymbol{U}\in\mathbb{R}^{D\times K},\boldsymbol{\mu}\in\mathbb{R}^D}{\text{maximize}} \quad \sum_{n=1}^{N} \|(\mathbf{I}-\boldsymbol{U}\boldsymbol{U}^{\mathrm{T}})(\boldsymbol{x}_n-\boldsymbol{\mu})\|_2^2$$

in place of (2) above. With $\boldsymbol{U}$ fixed, we can take the gradient of the functional above with respect to $\boldsymbol{\mu}$ to get the optimality condition for $\hat{\boldsymbol{\mu}}$:

$$\mathbf{0} = -2\sum_{n=1}^{N}(\mathbf{I}-\boldsymbol{U}\boldsymbol{U}^{\mathrm{T}})(\boldsymbol{x}_n-\hat{\boldsymbol{\mu}}) = -2(\mathbf{I}-\boldsymbol{U}\boldsymbol{U}^{\mathrm{T}})\left(\sum_{n=1}^{N}\boldsymbol{x}_n - N\hat{\boldsymbol{\mu}}\right),$$

which means that $\hat{\boldsymbol{\mu}}$ is the sample mean[3]:

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n.$$

[3]We are using the terms "sample mean" and "sample covariance" just because of the forms of these terms — we do not need to interpret the $\boldsymbol{x}_n$ as outcomes of random variables for this to all make sense.

This answer is independent of $\boldsymbol{U}$, so we can just subtract off $\boldsymbol{\mu}$ and proceed as before.

It is also easy to see that PCA can be interpreted as searching over an orthobasis, an expansion $\boldsymbol{\alpha}_n \in \mathbb{R}^K$ in that orthobasis, and an affine offset $\boldsymbol{\mu}$. We just of course know, because of all the hard work we did in September, that with $\boldsymbol{U}$ fixed, $\hat{\boldsymbol{\alpha}}_n = \boldsymbol{U}^{\mathrm{T}} \boldsymbol{x}_n$.

We now summarize the results in this section:

PCA Theorem

$$\underset{\boldsymbol{\mu}, \boldsymbol{U}, \{\boldsymbol{\alpha}_n\}}{\text{minimize}} \sum_{n=1}^{N} \|\boldsymbol{x}_n - \boldsymbol{\mu} - \boldsymbol{U}\boldsymbol{\alpha}_n\|_2^2, \quad \text{subject to } \boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = \mathbf{I},$$

has solution

$$\hat{\boldsymbol{\mu}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_n, \quad \hat{\boldsymbol{U}} = \begin{bmatrix} \hat{\boldsymbol{v}}_1 & \cdots & \hat{\boldsymbol{v}}_k \end{bmatrix}, \quad \hat{\boldsymbol{\alpha}}_n = \hat{\boldsymbol{U}}^{\mathrm{T}}(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}),$$

where $\hat{\boldsymbol{v}}_1, \ldots, \hat{\boldsymbol{v}}_K$ are the eigenvectors corresponding to the K largest eigenvalues of

$$\hat{\boldsymbol{R}} = \sum_{i=1}^{N} (\boldsymbol{x}_n - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}})^{\mathrm{T}}.$$

Structured Matrix Factorization

PCA is the simplest example of a class of problems that are known as **structured matrix factorization**. The idea is as follows. Given a $D \times N$ matrix $\boldsymbol{X}$, we want to find a $D \times R$ matrix $\boldsymbol{B}$ and a $R \times N$ matrix $\boldsymbol{C}$ such that

$$\boldsymbol{X} \approx \boldsymbol{B}\boldsymbol{C},$$

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

To interpret PCA in this framework, we take $R = K$,

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_N \end{bmatrix},$$

$\boldsymbol{B} = \boldsymbol{U}$ and $\boldsymbol{C} = \begin{bmatrix} \boldsymbol{\alpha}_1 & \boldsymbol{\alpha}_2 & \cdots & \boldsymbol{\alpha}_N \end{bmatrix}$. The structure imposed is that the columns of $\boldsymbol{U}$ are orthonormal.

The following is a more general statement of the PCA theorem (without the affine offset), and is proved in essentially the same way. It one of the fundamental results in modern linear algebra.

Eckart-Young Theorem: Let $\boldsymbol{X}$ be a $D \times N$ matrix with SVD $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$. Then the best rank-$R$ approximation to $\boldsymbol{X}$,

$$\underset{\boldsymbol{Y} \in \mathbb{R}^{D \times N}}{\text{minimize}} \ \|\boldsymbol{X} - \boldsymbol{Y}\|_F^2 \quad \text{subject to} \quad \text{rank}(\boldsymbol{Y}) = R,$$

is

$$\hat{\boldsymbol{Y}} = \boldsymbol{U}_R \boldsymbol{\Sigma}_R \boldsymbol{V}_R^{\mathrm{T}},$$

where $\boldsymbol{\Sigma}_R$ is the $R \times R$ diagonal matrix containing the R largest singular values of $\boldsymbol{X}$, $\boldsymbol{U}_R$ is a $D \times R$ matrix whose columns are the corresponding left singular vectors, and $\boldsymbol{V}_R$ contains the corresponding right singular vectors.

We can also replace the Frobenius norm $\|\cdot\|_F^2$ (sum of the squares of the singular values) in the optimization above with the spectral norm $\|\cdot\|$ (largest singular value). We could also re-write the program as

$$\underset{\substack{\boldsymbol{B}\in\mathbb{R}^{D\times R}\\ \boldsymbol{C}\in\mathbb{R}^{R\times N}}}{\text{minimize}} \ \|\boldsymbol{X}-\boldsymbol{B}\boldsymbol{C}\|_F^2.$$

The rank constraint is now implicit in the size of the matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ (as when had for PCA). A solution to the above is then

$$\hat{\boldsymbol{B}} = \boldsymbol{U}_R\boldsymbol{\Sigma}_R^{1/2}, \quad \hat{\boldsymbol{C}} = \boldsymbol{\Sigma}_R^{1/2}\boldsymbol{V}_R^{\mathrm{T}}.$$

Of course, this solution is not unique, as we may divvy up the $\boldsymbol{\Sigma}_R$ in an arbitrary way, i.e.

$$\hat{\boldsymbol{B}} = \boldsymbol{U}_R\boldsymbol{\Sigma}_R, \quad \hat{\boldsymbol{C}} = \boldsymbol{V}_R^{\mathrm{T}},$$

or even apply a general invertible $R\times R$ matrix $\boldsymbol{A}$ to the left and right:

$$\hat{\boldsymbol{B}} = \boldsymbol{U}_R\boldsymbol{\Sigma}_R^{1/2}\boldsymbol{A}, \quad \hat{\boldsymbol{C}} = \boldsymbol{A}^{-1}\boldsymbol{\Sigma}_R^{1/2}\boldsymbol{V}_R^{\mathrm{T}},$$

etc.

As we saw in the previous section, it is easy (and most natural) to make the factors $\hat{\boldsymbol{B}}$ or $\hat{\boldsymbol{C}}$ have *orthogonal* columns or rows, respectively.

What are we doing when we perform an approximation of this kind? Suppose that the columns of $\boldsymbol{X}$ are data points in $\mathbb{R}^D$:

$$\boldsymbol{X} = \begin{bmatrix}\boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_N\end{bmatrix}$$

and suppose that there exists a small $R \ll \min(D, N)$, such that

$$\begin{aligned} \boldsymbol{X} &\approx \boldsymbol{B}\boldsymbol{C} \\ &= \begin{bmatrix} & & \\ & \boldsymbol{B} & \\ & & \end{bmatrix} \begin{bmatrix} \boldsymbol{c}_1 & \boldsymbol{c}_2 & \cdots & \boldsymbol{c}_N \end{bmatrix}, \end{aligned}$$

i.e. $\boldsymbol{x}_n \approx \boldsymbol{B}\boldsymbol{c}_n$. This means that each $\boldsymbol{x}_n$ is a different (linear) combination (specified by the entries in $\boldsymbol{c}_n$) of a small number of latent features (specified by the columns of $\boldsymbol{B}$). So in solving this optimization algorithm, we are learning a basis for the $\boldsymbol{x}_n$.

Example: Eigenfaces

The data points $\boldsymbol{x}_n$ are D-pixel images[4] of faces (D is on the order of thousands):

There are many such images (N on the order of $1000s$), but $\boldsymbol{X}$ can be approximated very well with rank $R = 200$. Here are the columns of $\hat{\boldsymbol{B}}$:

[4] Example from http://kixor.net/school/2008spring/comp776/assn3/.

Example: Latent Semantic Indexing

LSI is used to automatically group documents together by their semantic content. Here, the data we are given is a *document term matrix*. We parse N documents (encyclopedia/wikipedia entries, newspaper article, etc.) and count the number of times D different words appear; the tally for word d in document n is the entry $X[d, n]$:

$$X[d, n] = \#\text{ times word } d \text{ appeared in document } n.$$

The rows of $\boldsymbol{X}$ tend to be correlated, so we can approximate $\boldsymbol{X} \approx \boldsymbol{BC}$ for $r \ll \min(D, N)$. This means that word frequency counts in a document are basically a linear combination of a small number of "feature counts" given by the columns of $\boldsymbol{B}$.

But in both these cases, it is hard to interpret what the latent factors really tell us about the data, other than being able to assist us in some simple dimensionality reduction. This is because we have not enforced the proper structure on the factors $\boldsymbol{B}, \boldsymbol{C}$.

Non-negative matrix factorization

If we constrain the factors to have positive entries, we get different (and perhaps more interesting) results:

$$\underset{\substack{\boldsymbol{B}\in\mathbb{R}^{D\times R}\\ \boldsymbol{C}\in\mathbb{R}^{R\times N}}}{\text{minimize}} \ \|\boldsymbol{X}-\boldsymbol{B}\boldsymbol{C}\|_F^2, \quad \boldsymbol{B}\geq\boldsymbol{0},\ \boldsymbol{C}\geq\boldsymbol{0}. \tag{4}$$

The idea behind this is that in many cases (including the face and LSI examples above), what we are really looking for is a mixture of *positive* factors. This problem is called *non-negative matrix factorization* (NNMF).

Solving (4) is much more involved than in the unconstrained case. Not only is there no closed-form solution, but the optimization is non-convex, and comes with all the complications there-of.

But notice that if we fix one the factors, say $\boldsymbol{B}=\boldsymbol{B}^{(0)}$, then the program

$$\underset{\boldsymbol{C}\in\mathbb{R}^{R\times N}}{\text{minimize}} \ \|\boldsymbol{X}-\boldsymbol{B}^{(0)}\boldsymbol{C}\|_F^2, \quad \boldsymbol{C}\geq\boldsymbol{0}.$$

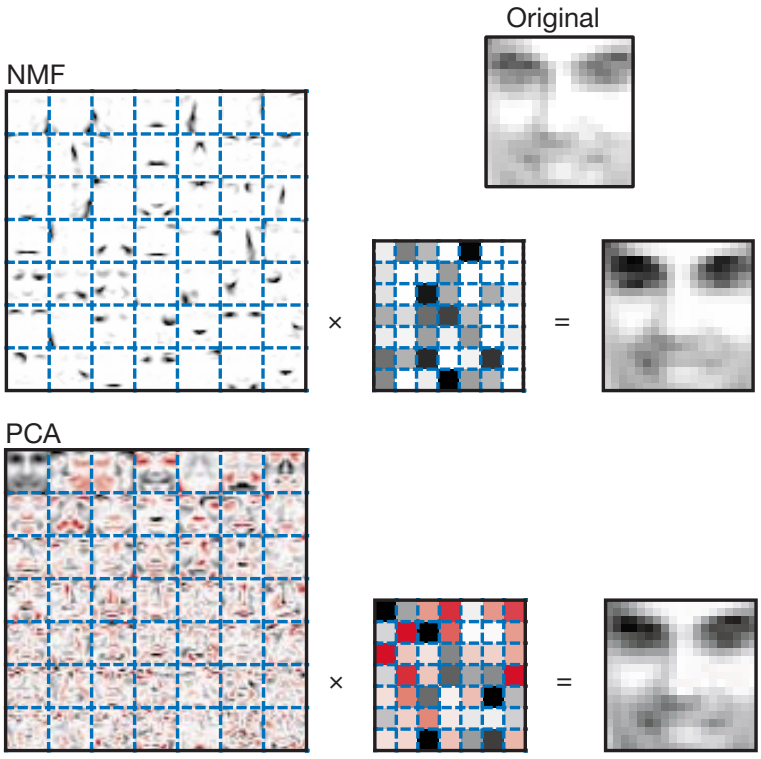
is separable by column; we can solve

$$\underset{\boldsymbol{c}_n\in\mathbb{R}^{R}}{\text{minimize}} \ \|\boldsymbol{x}_n-\boldsymbol{B}^{(0)}\boldsymbol{c}_n\|_2^2 \quad \text{subject to} \quad \boldsymbol{c}_n\geq 0,$$

for $n=1,\ldots,N$ independently. Each of these subproblems is a quadratics program, and so it solvable with off-the-shelf software. The same thing is true if we pin down $\boldsymbol{C}=\boldsymbol{C}^{(0)}$ and optimize over $\boldsymbol{B}$. So NNMF algorithms typically alternate back and forth between solving for $\boldsymbol{B}$ and solving for $\boldsymbol{C}$.

Here are some interesting results from the original paper on NNMF[5]

Faces:



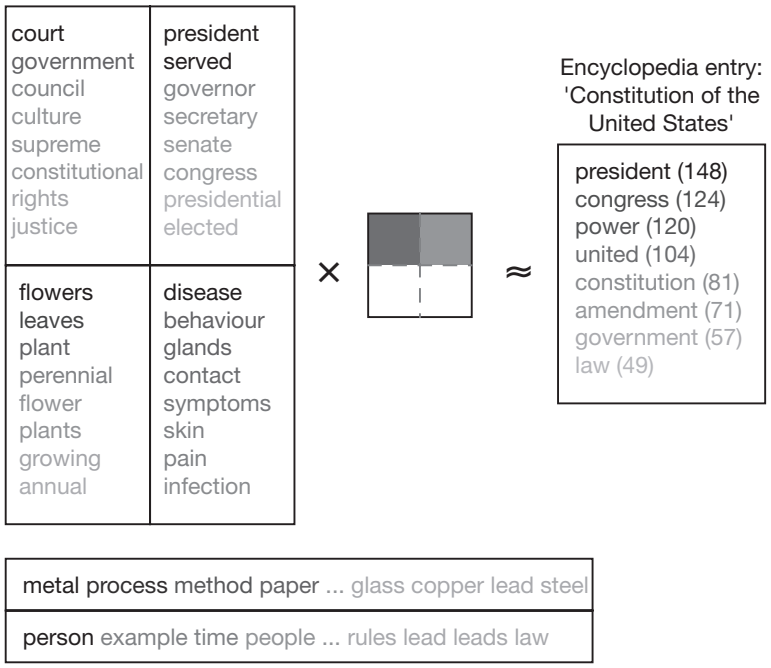
The top is essentially the eigenfaces experiment, but where we restrict the factors ($\boldsymbol{B}$) and mixture coefficients ($\boldsymbol{C}$) to be positive. Notice that, as compared to the standard PCA experiment below it, that the factors actually look like facial features.

[5]Lee and Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, October 1999.

Latent semantic indexing:

Here is an example from the same article for LSI.

$N = 30,991$ articles from an encyclopedia were parsed,
$D = 15,276$ words counted



Now the columns of $\hat{\boldsymbol{B}}$ can be interpreted as *groupings* of words that appear together frequently.

On the top right, we see example weights for four different factors that decomposed the (word count of) the US Constitution ... the two factors (columns of $\boldsymbol{B}$) on top, learned while training on a separate data set, were given significant weights (entries in $\boldsymbol{c}$), while the two factors on bottom had very small weights.

The example on bottom shows the words associated with the two factors whose entry in the row corresponding to the word 'lead' was the greatest.