Non-negative Matrix Factorization

UBCO MDS — DATA 573

Non-negative Matrix Factorization

- We have just finished discussions surrounding factor analysis, and previously discussed principal component analysis.
- Rather than jumping headfirst into a new, somewhat similar technique...let's motivate it through an image processing example

Image Analysis

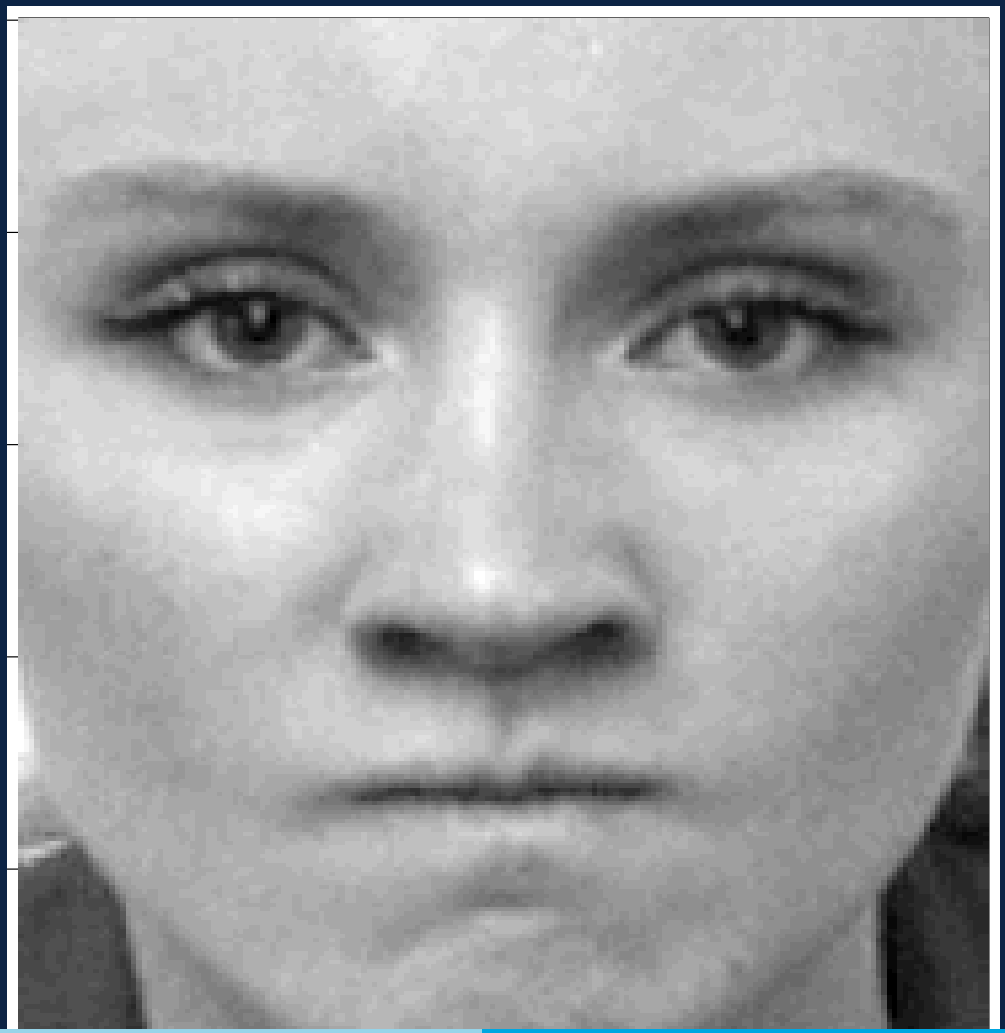
- As we haven't yet dealt with any image data, you may feel as though you do not have any tools at your disposal in this realm...but you would be wrong!
- PCA is often one of the first analytic techniques learned in image analysis.
- Can be used for compression, along with various other imaging goals.
- Before we get to deep, let's discuss some preliminaries.

Data Prep

- The core information from image files is numeric.
- Each pixel of an image can have a single value (grayscale, say 0-1 for black to white) or several values (three for RGB, four for CMYK, etc)
- I will only discuss grayscale to simplify things.
- R has many built in packages for image reading, manipulation, etc... Python perhaps more.
- But if the images are already set up in a relatively tidy data form, you probably won't even need additional packages beyond base R. I have not used any additional packages for generating the material in this lecture.

Pain Data

- I will use the "pain" data set I tracked down online. 84 grayscale images of cropped faces, all 241 x 181 pixels.
  `http://pics.psych.stir.ac.uk/2D_face_sets.htm`



Face 1 (first 5 x 24 pixels)


```
> head(pain[,,1])
          y1        y2        y3        y4        y5        y6        y7        y8
x1 0.3098039 0.3215686 0.3529412 0.3568627 0.3254902 0.3137255 0.3215686 0.3215686
x2 0.2588235 0.2823529 0.3058824 0.3058824 0.2862745 0.2705882 0.2705882 0.2784314
x3 0.2784314 0.2862745 0.2901961 0.2823529 0.2705882 0.2666667 0.2745098 0.2901961
x4 0.2862745 0.2862745 0.2823529 0.2745098 0.2666667 0.2666667 0.2745098 0.2862745
x5 0.2901961 0.2862745 0.2862745 0.2823529 0.2823529 0.2784314 0.2745098 0.2745098
          y9       y10       y11       y12       y13       y14       y15       y16
x1 0.3333333 0.3411765 0.3490196 0.3215686 0.3372549 0.3490196 0.3254902 0.3411765
x2 0.2901961 0.3058824 0.2941176 0.2784314 0.2901961 0.2980392 0.2941176 0.2980392
x3 0.2862745 0.3058824 0.2941176 0.2784314 0.2941176 0.2980392 0.2941176 0.2980392
x4 0.2823529 0.3019608 0.2941176 0.2784314 0.2941176 0.3019608 0.2980392 0.3019608
x5 0.2823529 0.3019608 0.2941176 0.2823529 0.2980392 0.3058824 0.2980392 0.3019608
         y17       y18       y19       y20       y21       y22       y23       y24
x1 0.3294118 0.3254902 0.3058824 0.2862745 0.2470588 0.2431373 0.2274510 0.1294118
x2 0.2862745 0.2901961 0.2470588 0.2392157 0.2156863 0.2313725 0.1686275 0.1176471
x3 0.2745098 0.2862745 0.2509804 0.2392157 0.2039216 0.2117647 0.1568627 0.1137255
x4 0.2745098 0.2901961 0.2627451 0.2509804 0.2078431 0.2078431 0.1490196 0.1137255
x5 0.2941176 0.2980392 0.2666667 0.2627451 0.2235294 0.2235294 0.1607843 0.1215686
```

Data Prep

- First issue: images have a 2D structure
- Easy enough, we can reshape each image into a 1D vector, thereby forming a matrix where each row is an individual image of a face and each column corresponds to the pixels.

5 faces (first 24 pixels vertical)


```
> flim[1:5, 1:24]
           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [
f01_a 0.3098039 0.2588235 0.2784314 0.2862745 0.2901961 0.2980392 0.3137255 0.3019
f01_d 0.2901961 0.2745098 0.2823529 0.2862745 0.2862745 0.2823529 0.2784314 0.2784
f01_f 0.3794466 0.3241107 0.3438735 0.3557312 0.3557312 0.3596838 0.3675889 0.3596
f01_h 0.2980392 0.2823529 0.2745098 0.2627451 0.2549020 0.2627451 0.2784314 0.2941
f01_n 0.2117647 0.2470588 0.2470588 0.2470588 0.2470588 0.2431373 0.2509804 0.2588
           [,9]     [,10]     [,11]     [,12]     [,13]     [,14]     [,15]     [,1
f01_a 0.2823529 0.2705882 0.2666667 0.2823529 0.3098039 0.3137255 0.2941176 0.2901
f01_d 0.2823529 0.2588235 0.2549020 0.2509804 0.2588235 0.2705882 0.2823529 0.2901
f01_f 0.3438735 0.3438735 0.3359684 0.3241107 0.3241107 0.3359684 0.3517787 0.3596
f01_h 0.3058824 0.3137255 0.3058824 0.3019608 0.3058824 0.3215686 0.3294118 0.3294
f01_n 0.2705882 0.2705882 0.2862745 0.3098039 0.3254902 0.3215686 0.3098039 0.3254
          [,17]     [,18]     [,19]     [,20]     [,21]     [,22]     [,23]     [,2
f01_a 0.3019608 0.3137255 0.3215686 0.3058824 0.2823529 0.2862745 0.3137255 0.3176
f01_d 0.2941176 0.2862745 0.2862745 0.2862745 0.2862745 0.2823529 0.2745098 0.2705
f01_f 0.3596838 0.3517787 0.3557312 0.3596838 0.3636364 0.3715415 0.3754941 0.3794
f01_h 0.3215686 0.3372549 0.3411765 0.3529412 0.3568627 0.3450980 0.3254902 0.3372
f01_n 0.3490196 0.3803922 0.4039216 0.4392157 0.6352941 0.7843137 0.8078431 0.8313

> dim(flim)
[1]    84 43621
```

Data Prep

- Now the data is formatted in a way that we're used to...
- So let's run it through PCA, doing the usual scaling...

PCA

```
>prface <- prcomp(flim, scale.=TRUE)
> dim(prface$rotation)
[1] 43621    84
> dim(prface$x)
[1] 84 84
```

PCA

```
> summary(prface)
Importance of components:
                            PC1     PC2      PC3      PC4      PC5      PC6     PC7
Standard deviation     110.4661 71.0666 50.12856 48.63727 42.62239 38.94299 35.9961
Proportion of Variance   0.2797  0.1158  0.05761  0.05423  0.04165  0.03477  0.0297
Cumulative Proportion    0.2797  0.3955  0.45313  0.50736  0.54901  0.58378  0.6135
                           PC10     PC11     PC12     PC13     PC14     PC15     PC
Standard deviation     31.74923 29.59248 28.01701 27.02542 24.81858 24.76744 23.845
Proportion of Variance  0.02311  0.02008  0.01799  0.01674  0.01412  0.01406  0.013
Cumulative Proportion   0.68889  0.70897  0.72696  0.74371  0.75783  0.77189  0.784
                           PC19     PC20     PC21     PC22     PC23    PC24     PC2
Standard deviation     21.67246 20.02499 18.70156 17.86329 17.58793 17.0994 17.0683
Proportion of Variance  0.01077  0.00919  0.00802  0.00732  0.00709  0.0067  0.0066
Cumulative Proportion   0.81852  0.82771  0.83573  0.84304  0.85013  0.8568  0.8635
                           PC28     PC29     PC30     PC31     PC32    PC33     PC3
Standard deviation     15.17276 14.96189 14.72391 14.39746 14.00067 13.8513 13.6200
Proportion of Variance  0.00528  0.00513  0.00497  0.00475  0.00449  0.0044  0.0042
Cumulative Proportion   0.88026  0.88539  0.89036  0.89511  0.89960  0.9040  0.9082
                           PC37     PC38    PC39     PC40     PC41     PC42     PC4
Standard deviation     12.74030 12.38721 12.1822 11.93798 11.78207 11.72539 11.2820
Proportion of Variance  0.00372  0.00352  0.0034  0.00327  0.00318  0.00315  0.0029
Cumulative Proportion   0.91963  0.92314  0.9265  0.92981  0.93300  0.93615  0.9390
```

PCA

	PC46	PC47	PC48	PC49	PC50	PC51	PC52
Standard deviation	10.49716	10.40377	10.26327	10.04208	9.95375	9.60014	9.43158
Proportion of Variance	0.00253	0.00248	0.00241	0.00231	0.00227	0.00211	0.00204
Cumulative Proportion	0.94724	0.94972	0.95214	0.95445	0.95672	0.95883	0.96087

	PC56	PC57	PC58	PC59	PC60	PC61	PC62	PC
Standard deviation	8.94607	8.84429	8.58608	8.52505	8.3430	8.25341	8.23033	7.95
Proportion of Variance	0.00183	0.00179	0.00169	0.00167	0.0016	0.00156	0.00155	0.00
Cumulative Proportion	0.96842	0.97021	0.97190	0.97357	0.9752	0.97673	0.97828	0.97

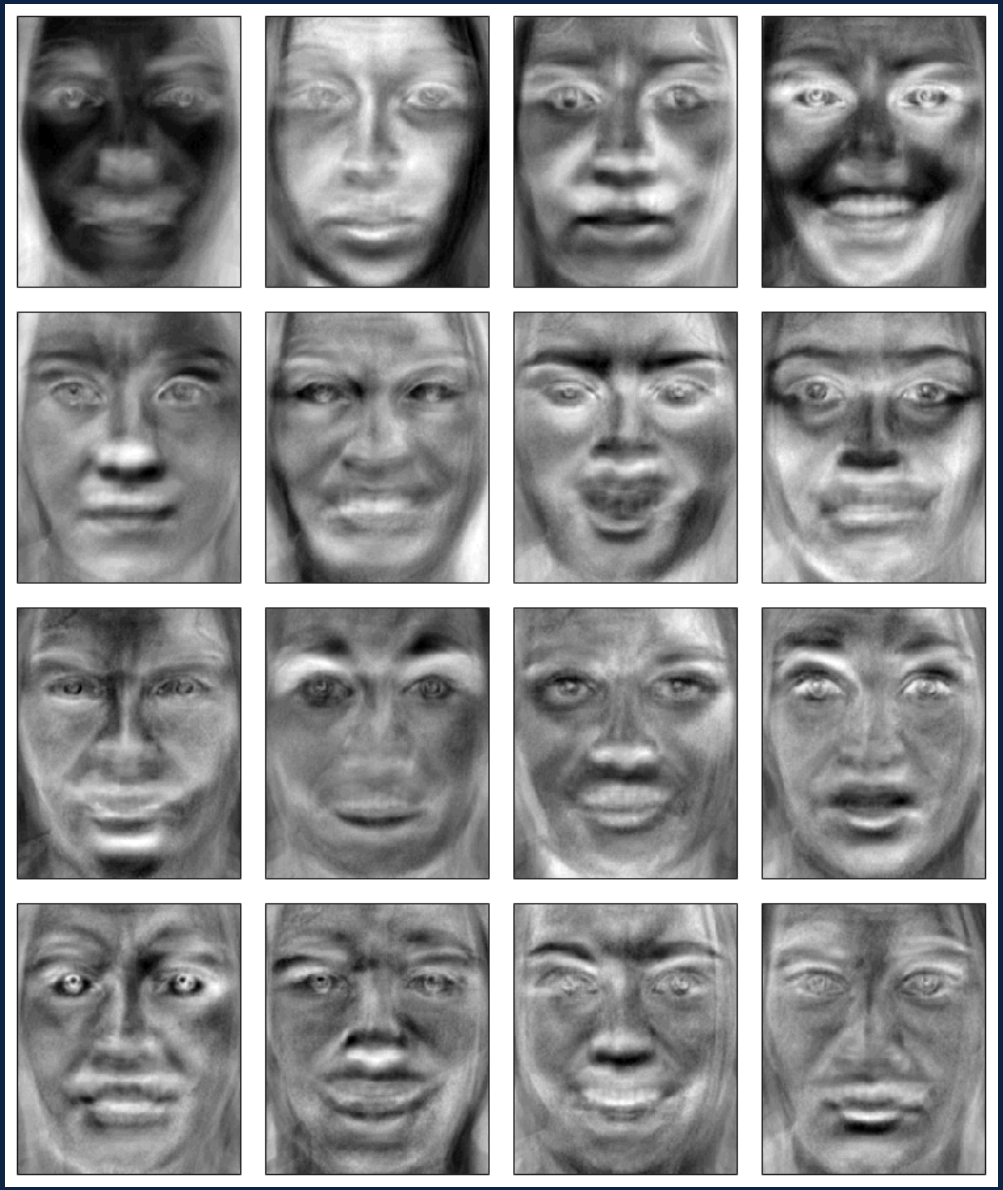
	PC66	PC67	PC68	PC69	PC70	PC71	PC72	PC
Standard deviation	7.59716	7.5437	7.43347	7.36412	7.19047	7.02242	6.85074	6.70
Proportion of Variance	0.00132	0.0013	0.00127	0.00124	0.00119	0.00113	0.00108	0.00
Cumulative Proportion	0.98379	0.9851	0.98636	0.98760	0.98879	0.98992	0.99099	0.99

	PC76	PC77	PC78	PC79	PC80	PC81	PC82	P
Standard deviation	6.39875	6.22907	5.86576	5.82745	5.55216	5.50508	5.35070	4.7
Proportion of Variance	0.00094	0.00089	0.00079	0.00078	0.00071	0.00069	0.00066	0.00
Cumulative Proportion	0.99498	0.99587	0.99666	0.99744	0.99814	0.99884	0.99949	1.00

Average Face

- Wait...we scaled, so we subtracted a mean for each pixel? An average face? Yes...

First 16 Eigenvectors



Reconstructing with First PC

```
      f03_s        f04_h
 94.689917 -103.280485
      f07_d        f10_d
187.933583    8.411657
```

Reconstructing from First PC to Last PC

Results

- Suppose we decide the reconstruction with 25 PCs is suitable

Results

- What have we effectively done?
- The full data is stored in a matrix with total number of entries

$$84 * 43621 = 3,664,164$$

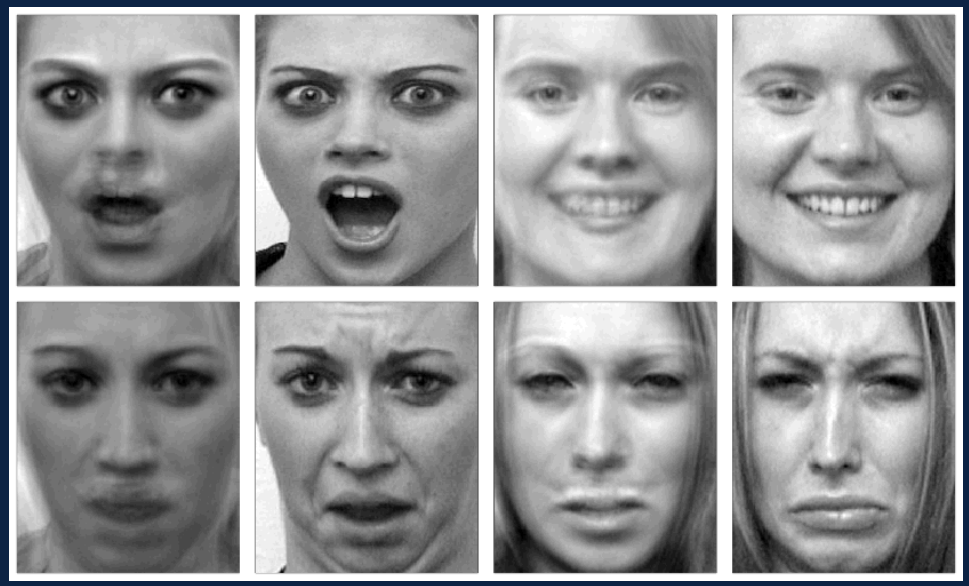
- Our reconstructed from PCs version uses

$$25 * 43621 + 84 * 25 + 2 * 43621 = 1,179,867$$

- AKA, roughly 1/3 of the storage.

Results

- Furthermore, suppose the goal was to predict the emotion in the picture.
- Perhaps 12 PCs (or maybe even less) could be suitable for a supervised algorithm (or human) to classify the expressed emotion



- Then we only need

$$12 * 43621 + 84 * 12 + 2 * 43621 = 611,702$$

Results

- On top of this, it's noteworthy how this storage grows if we include additional images.
- Let's say we add 600 more images to our data. We go from 84 images to 684, then

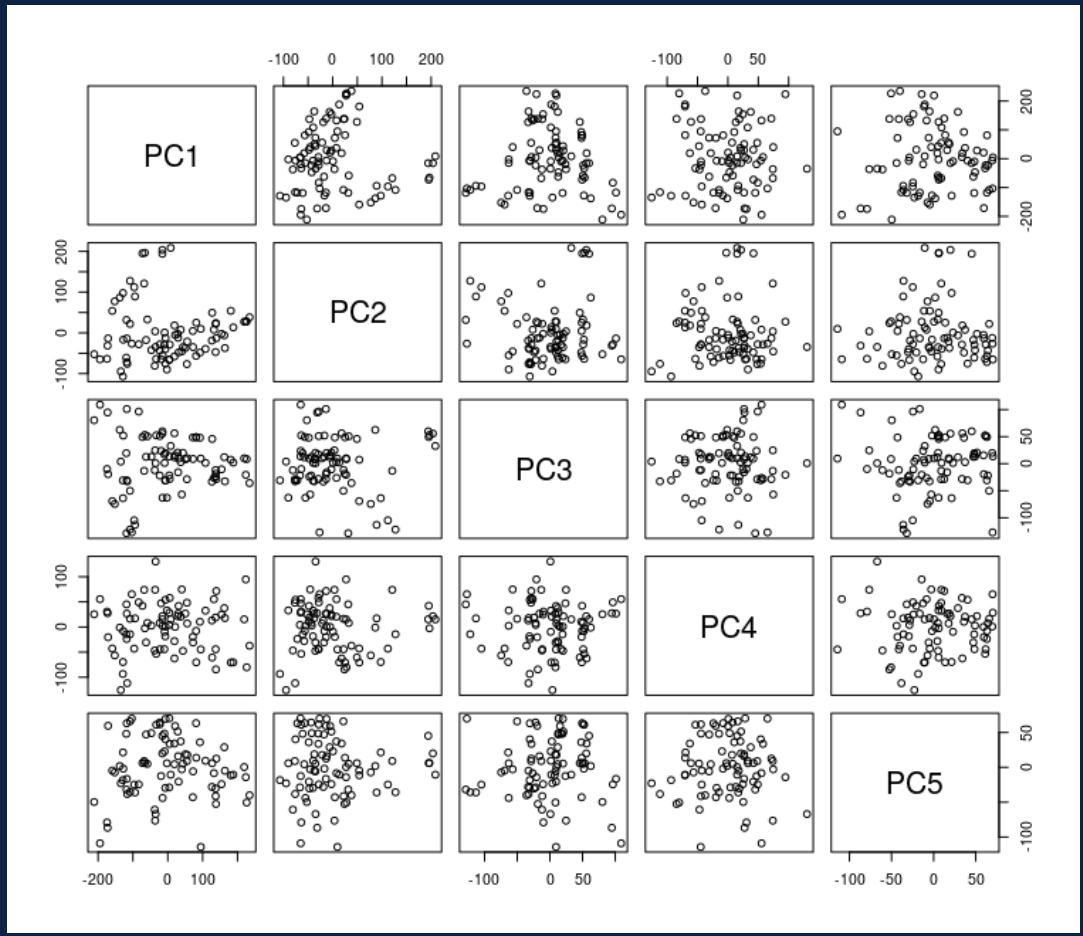
$$12 * 43621 + 84 * 12 + 2 * 43621 = 611,702$$

to

$$12 * 43621 + 684 * 12 + 2 * 43621 = 618,902$$

Results

- Finally, while we focussed on reconstruction as a useful visual tool, our PCA scores can of course be viewed in the usual manner and/or used for further analysis



Comments

- Images (and frankly several other sources of data) have an interesting characteristic that PCA does not and cannot take advantage of
- Our grayscale images are, in fact, bounded on 0-1. They are therefore non-negative.
- PCA estimates both eigenvectors and scores in a non-bounded way.
- The result is that the linear combination that makes up the reconstruction includes both additive and subtractive components — aka, you can add a big nose and then subtract a smaller nose to get a medium size.
- It's actually more complicated than that. In fact, we are basically adding and subtracting those entire eigenvector faces to approximate the original faces

Comments

- There are several analyses where adding and subtracting should not be permitted
- One such example is in spectral data, where the observed (non-negative) signal is assumed to be made up of several other non-negative signals.
- In this domain, there is no physical explanation for 'subtracting' a signal. Either an element is there in some amount, or its not...it cannot exist negatively.
- For images though, not allowing subtractable parts may have interesting ramifications for us...let's finally look at a new model...

Non-negative matrix factorization

- Non-negative matrix factorization (or NMF) is another decomposition technique. Unlike PCA or FA, it is not performed on covariance matrices.
- We seek to decompose our non-negative data matrix X into two other non-negative matrices W and H via

$$X \approx WH$$

  - where X is $n \times p$
  - W is $n \times q$
  - H is $q \times p$

- Note if you're googling: often X is defined transposed, and therefore what some sources call W would be H in my notation.

Non-negative matrix factorization

- Because of the note on the previous slide, matching terminology to other sources might get messy.
- Instead, I will give terminology that partially matches our other related analyses
- We will call W the 'scores' as they are the equivalent of PCA or FA scores in this realm. They are how the individual observations map onto H.
- We will call H the 'basis' as they can be viewed as the essential components (in terms of the variables) of the original data.

Non-negative matrix factorization

- Assuming a model is all well and good, but how is it estimated?
- There is no one answer to this question. It depends on the type of error you wish to minimize, and the type of iterative process you would like to undertake.
- At present, standard softwares will likely provide tens of options for the optimization procedure
- I will introduce the original, most basic form from Lee and Seung's (1999) seminal paper.

Multiplicative Updates

1. Initialize both W and H randomly.
2. Update H:

$$H \times \frac{W'X}{W'WH}$$

3. Update W:

$$W \times \frac{XH'}{WHH'}$$

4. Repeat 2 and 3 until convergence (or perhaps a maximum number of iterations).

Note that in the above, both the $\times$ and the divisor are element-wise operations — not matrix operators.

Application

- Apply homemade function for this on our faces with 600 iterations (for time) and 16 bases (for easy comparison)
- We can compare the estimated bases to our eigenvectors
- It's worth noting that there should be no inherent order of importance to our NMF bases, unlike in PCA

Comparison of Eigenvectors/Bases

PCA

NMF

Reconstruction: 16 PCs vs 16 Bases vs Original

Comments

- Similar good news on compression, PCA with 16 PCs would be

$$16 * 43621 + 84 * 16 + 2 * 43621 = 786,522$$

whereas NMF with 16 bases would be

$$16 * 43621 + 84 * 16 = 699,280$$

- So some slight savings over PCA — but note this is only because we scaled the data in PCA (requiring usage of the means and stdevs of each variable to properly reconstruct approximate images).
- Predominantly, we get a more interesting/useful interpretation of the underlying bases vs components in PCA
- Again, this is due to the additivity requirement when the decomposition enforces non-negativity for all aspects of the decomposition.


THE UNIVERSITY OF BRITISH COLUMBIA