ECE 417, Lecture 6: Discrete Cosine Transform

Mark Hasegawa-Johnson

9/6/2019

Outline

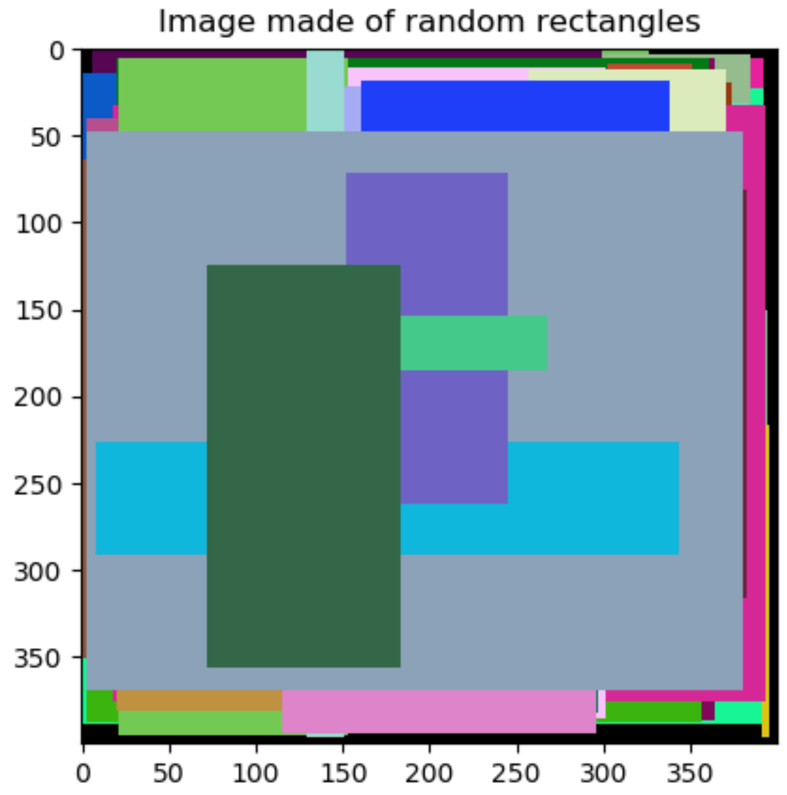
- DCT
- KNN
- How to draw the contour plots of a multivariate Gaussian pdf

Discrete Cosine Transform

- Last time: PCA
- Why it’s useful: PCs are uncorrelated with one another, so you can keep just the top-N (for N<<D), and still get a pretty good nearest-neighbor classifier.
- Why it’s difficult: PCA can only be calculated when you’ve already collected the whole dataset.
- Question: can we estimate what the PCA will be in advance, before we have the whole dataset? For example, what are the PC axes for the set of “all natural images”?

A model of natural images

1. Choose an object of a random color,
2. Make it a random size,
3. Position it at a random location in the image,
4. Repeat.



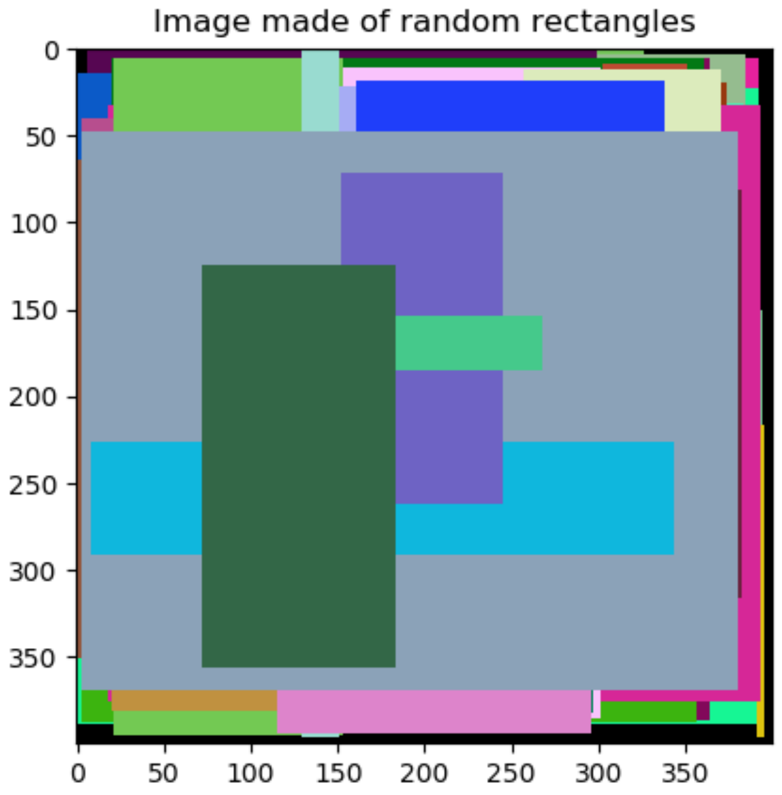
Result: PCA = DFT!

Define the 2D DFT, $X[k_1,k_1]$, as

$$\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x[n_1,n_1]e^{-j\frac{2\pi k_1 n_1}{N_1}}e^{-j\frac{2\pi k_2 n_2}{N_2}}$$

It turns out that the pixels, $x[n_1,n_1]$, are highly correlated with one another (often exactly the same!)

But on average, as # images $\rightarrow\infty$, the DFT coefficients $X[k_1,k_1]$ become uncorrelated with one another (because object sizes are drawn at random).



2D DFT as a vector transform

- Suppose we vectorize the image, for example, in raster-scan order, so that

$$\vec{x} = \begin{bmatrix} x[0,0] \\ x[0,1] \\ \vdots \\ x[N_1-1, N_2-1] \end{bmatrix}$$

- … and suppose we invent some mapping from k to (k_1, k_2), for example, it could be in diagonal order: $0{:}\,(0,0), 1{:}\,(1,0), 2{:}\,(0,1), 3{:}\,(2,0), 4{:}\,(1,1), 5{:}\,(0,2), 6{:}\,(3,0), \cdots$. Then the features are $y_k = \vec{x}^T \vec{v}_k$, with basis vectors

$$\vec{v}_k = \begin{bmatrix} v_{0k} \\ \vdots \\ v_{N_1 N_2 - 1, k} \end{bmatrix}, v_{nk} = e^{-j\frac{2\pi k_1 n_1}{N_1}} e^{-j\frac{2\pi k_2 n_2}{N_2}}$$

The problem with DFT…

… is that it's complex-valued! That makes it hard to do some types of statistical analysis and machine learning (some types of derivatives, for example, do not have a definition if the variable is complex-valued).

$$\vec{v}_k = \begin{bmatrix} v_{0k} \\ \vdots \\ v_{N_1 N_2 - 1, k} \end{bmatrix}, v_{nk} = e^{-j\frac{2\pi k_1 n_1}{N_1}} e^{-j\frac{2\pi k_2 n_2}{N_2}}$$

How to make the DFT real

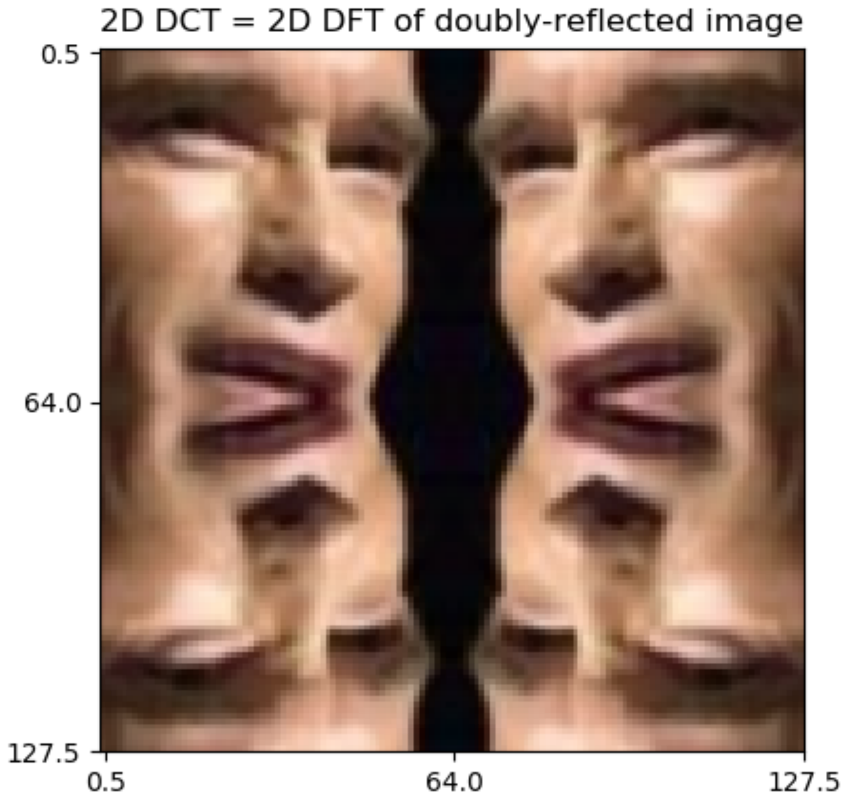
The DFT of a real symmetric sequence is real & symmetric.

$$x[n] = x^*[N-n] \leftrightarrow \ \mathrm{Im}\{X[k]\} = 0$$

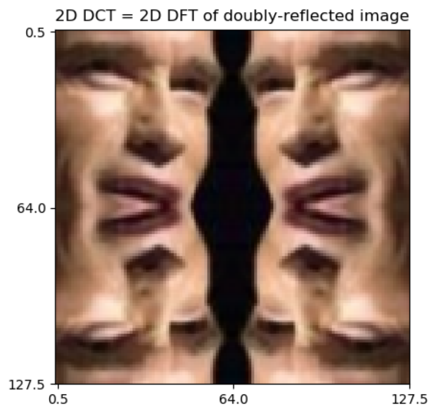
$$\mathrm{Im}\{x[n]\} = 0 \leftrightarrow X[k] = X^*[N-k]$$

How to make the DFT real

- Most natural images are real-valued.
- Let’s also make it symmetric: pretend that the observed image is just ¼ of a larger, mirrored image.



Discrete Cosine Transform



Define $s[m] = \begin{cases} \frac{1}{2} x\left[m - \frac{1}{2}\right], & m = \frac{1}{2}, \frac{3}{2}, \cdots, N - \frac{1}{2} \\ \frac{1}{2} x\left[2N - m - \frac{1}{2}\right], & m = N + \frac{1}{2}, N + \frac{3}{2}, \cdots, 2N - \frac{1}{2} \end{cases}$

$$e^{j\frac{\pi k n}{N}} + e^{-j\frac{\pi k n}{N}} = 2\cos\left(\frac{\pi k m}{N}\right)$$

$$m = n + \frac{1}{2}$$

Then:

$$S[k] = \sum_{m=\frac{1}{2}}^{2N-\frac{1}{2}} s[m] e^{-j\frac{2\pi k m}{2N}} = \sum_{m=\frac{1}{2}}^{N-\frac{1}{2}} s[m] 2\cos\left(\frac{\pi k m}{N}\right) = \sum_{n=0}^{N-1} x[n] \cos\left(\frac{\pi k \left(n + \frac{1}{2}\right)}{N}\right)$$

2D DCT as a vector transform

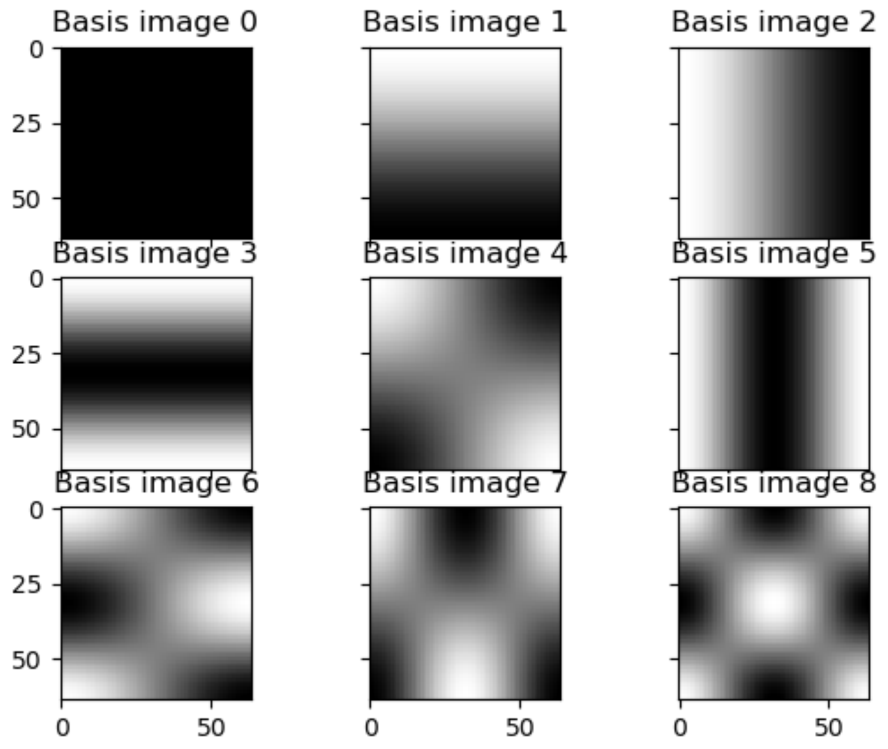
Assume that you have some reasonable mapping from n to (n_1, n_2), and from k to (k_1, k_2). Then $\vec{y} = V^T\vec{x}$, where $V = \left[\vec{v}_0, \cdots, \vec{v}_{N_1N_2-1}\right]$, and

$$\vec{v}_k = \begin{bmatrix} v_{0k} \\ \vdots \\ v_{N_1N_2-1,k} \end{bmatrix}, v_{nk} = \cos\left(\frac{\pi k_1\left(n_1 + \frac{1}{2}\right)}{N_1}\right)\cos\left(\frac{\pi k_2\left(n_2 + \frac{1}{2}\right)}{N_2}\right)$$

Basis Images: 9th-order 2D DCT

$$\cos\left(\frac{\pi k_1\left(n_1+\frac{1}{2}\right)}{N_1}\right)\cos\left(\frac{\pi k_2\left(n_2+\frac{1}{2}\right)}{N_2}\right)$$

- The k1=0, k2=0 case represents the average intensity of all pixels in the image.
- The k1=1 or k2=1 basis vectors capture the brightness gradient from top to bottom, or from left to right, respectively.
- The k1=2 or k2=2 basis vectors capture the difference in pixel intensity between the center vs. the edges of the image.



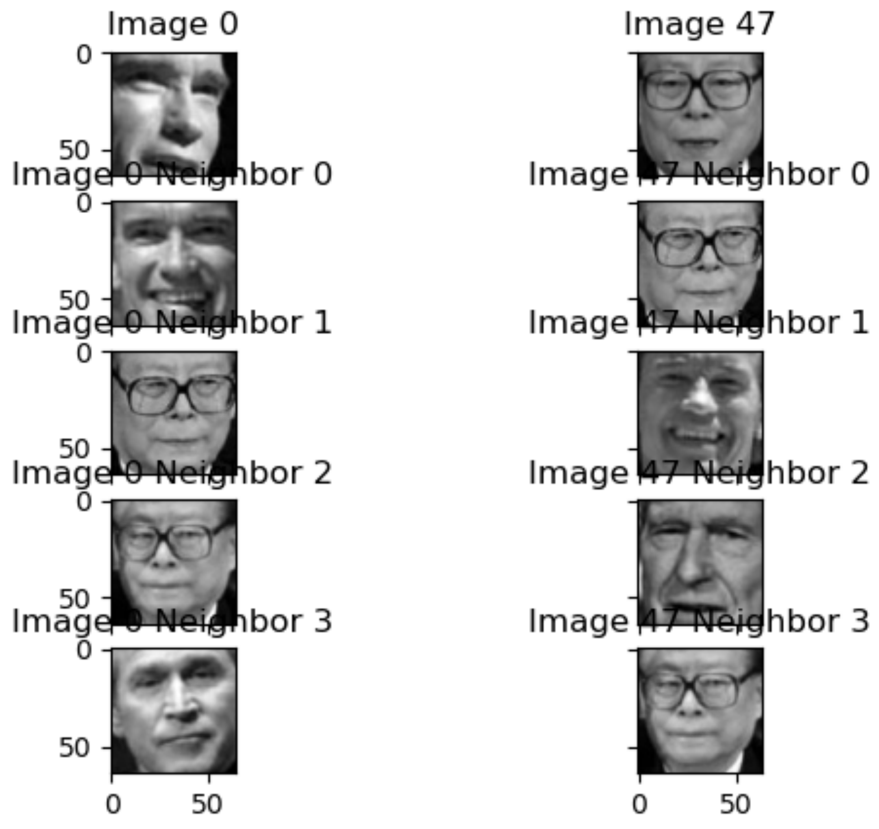
Nearest neighbors: 9th-order 2D DCT

This image shows the four nearest neighbors of “Image 0” (Arnold Schwarzenegger) and “Image 47” (Jiang Zemin), calculated using a 9th-order 2D DCT.

Neighbors of “Image 0” are dark on the right-hand-side, and in the lower-left corner.

Neighbors of “Image 47” are darker on the bottom than the top.

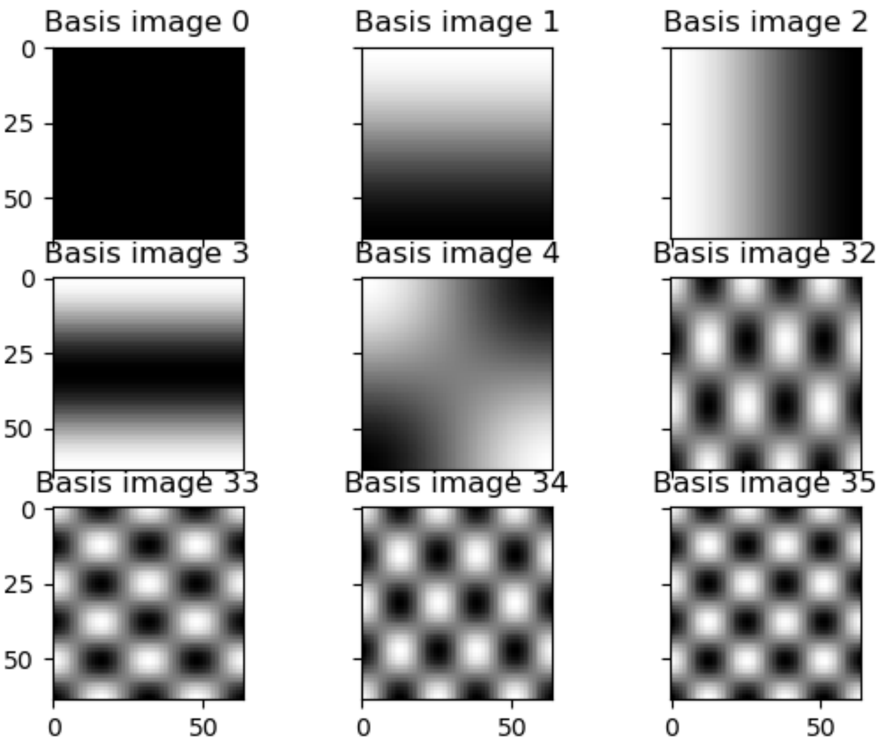
Neither of these features captures person identity very well…



Basis Images: 36th-order 2D DCT

$$\cos\left(\frac{\pi k_1\left(n_1+\frac{1}{2}\right)}{N_1}\right)\cos\left(\frac{\pi k_2\left(n_2+\frac{1}{2}\right)}{N_2}\right)$$

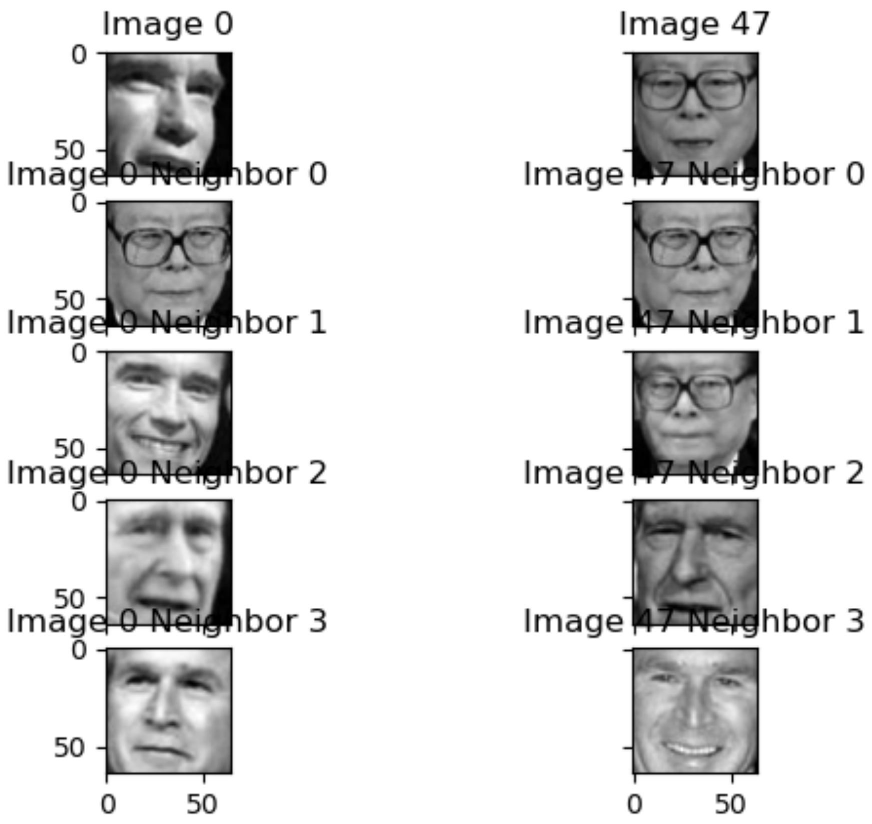
With a 36th order DCT (up to k1=5,k2=5), we can get a bit more detail about the image.



Nearest neighbors: 36th-order 2D DCT

The 36 order DCT is, at least, capturing the face orientation: most of the images considered “similar” are at least looking in the same way.

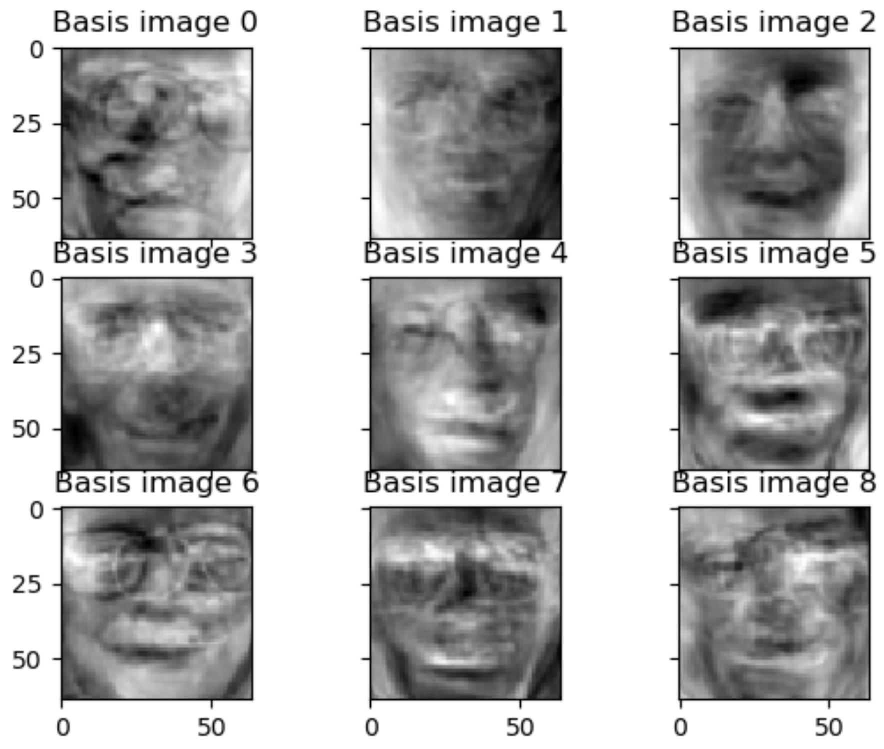
Jiang Zemin seems to be correctly identified (2 of the 4 neighbors are the same person), but Arnold Schwarzenegger isn’t (each of the 4 “similar” images shows a different person!)



PCA vs. DCT

PCA is like DCT in some ways. In this example, $\vec{v}_0$ might be measuring average brightness; $\vec{v}_1$ is left-to-right gradient; $\vec{v}_2$ is measuring center-vs-edges.

But PCA can also learn what's important to represent sample covariance of the given data. For example, eyeglasses ($\vec{v}_5$, $\vec{v}_6$), short vs. long nose ($\vec{v}_5$), narrow vs. wide chin ($\vec{v}_7$).

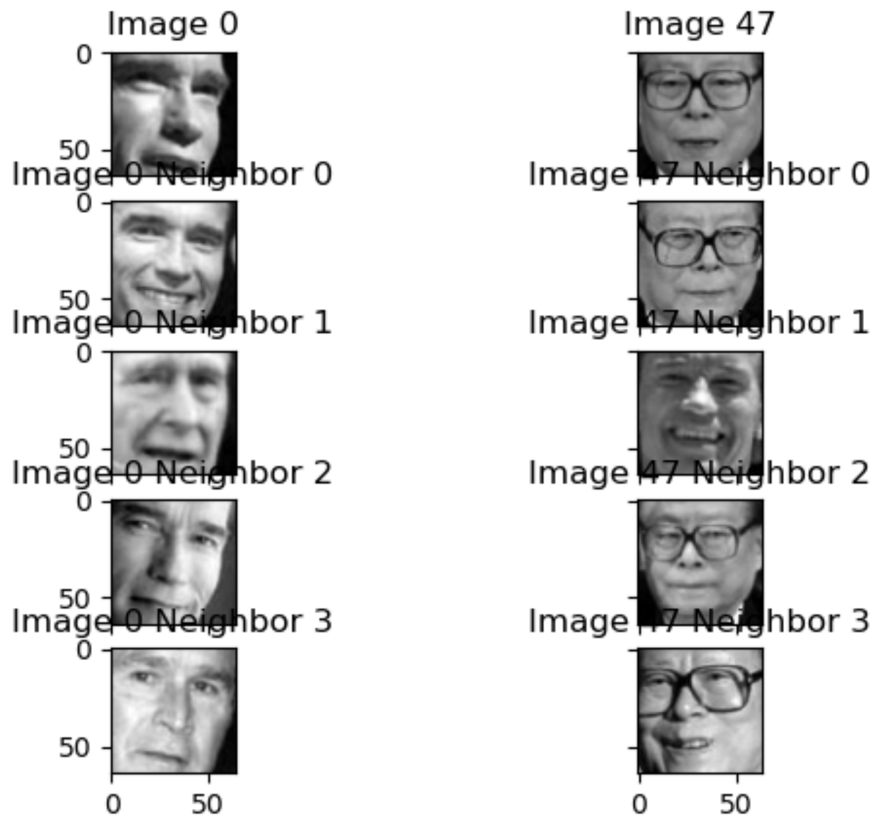


Nearest neighbors: 9th-order PCA

For these two test images, 9th-order PCA has managed to identify both people.

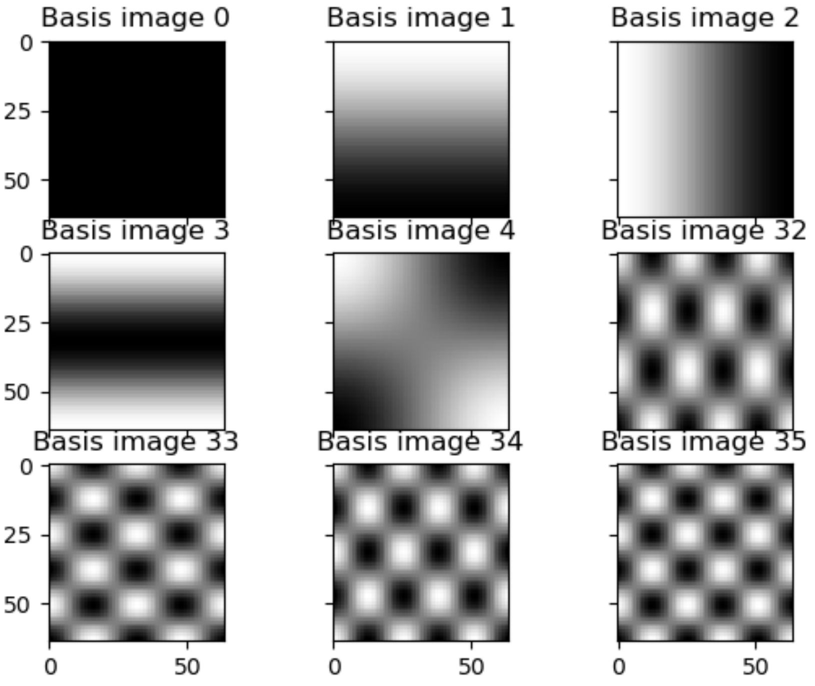
Two of the four neighbors of ”Image 0” are Arnold Schwarzenegger.

Three of the four neighbors of “Image 47” are Jiang Zemin.

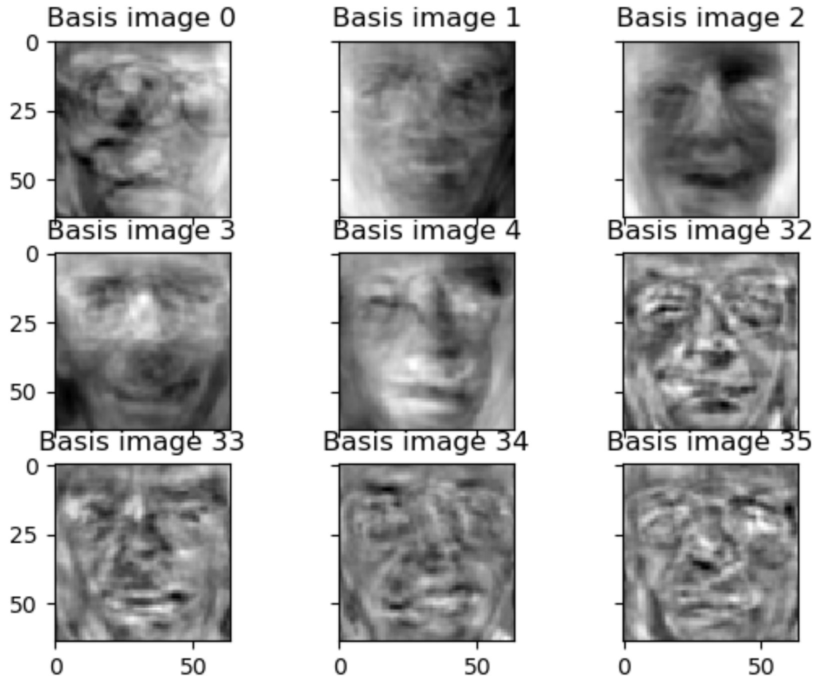


High-order PCA might be just noise!

It is not always true that PCA outperforms DCT. Especially for higher-dimension feature vectors, PCA might just learn random variation in the training dataset, which might not be useful for identifying person identity.



...vs...



Summary

- As $\mathrm{M} \rightarrow \infty$, PCA of randomly generated images $\rightarrow$ DFT
- DCT = half of the real symmetric DFT of a real mirrored image.
- As order of the DCT grows, details of the image start to affect its nearest neighbor calculations, allowing it capture more about person identity.
- PCA can pick out some details with smaller feature vectors than DCT, because it models the particular problem under study (human faces) rather than a theoretical model of all natural images.
- With larger feature vectors, PCA tends to learn quirks of the given dataset, which are usually not useful for person identification. DCT is a bit more robust (maybe because it's like using $\mathrm{M} \rightarrow \infty$).

Outline

- DCT
- KNN
- How to draw the contour plots of a multivariate Gaussian pdf

K-Nearest Neighbors (KNN) Classifier

1. To classify each test token, find the K training tokens that are closest.
2. Look up the **reference** labels (known true person IDs) of those K neighbors. Let them vote. If there is a winner, then use that person ID as the **hypothesis** for the test token.
   - If there is no winner, then fall back to 1NN.

Confusion Matrix

Hypothesis

Reference

	0	1	2	3
0	# times that person 0 was classified correctly (sometimes abbreviated C(0|0))	# times that person 0 was classified as person 1 (sometimes abbreviated C(1|0))	…	…
1	…	…	…	
2				
3				

Accuracy, Recall, and Precision

Accuracy:

$$A = \frac{\sum_{r=0}^{3} C(r|r)}{\sum_{r=0}^{3}\sum_{h=0}^{3} C(h|r)} = \frac{\#\, correct}{\#\, data}$$

Recall:

$$R = \frac{1}{4}\sum_{r=0}^{3} \frac{C(r|r)}{\sum_{h=0}^{3} C(h|r)} = \frac{1}{4}\sum_{r=0}^{3} \frac{\#\, times\ r\ correctly\ recognized}{\#\, times\ r\ presented}$$

Precision:

$$P = \frac{1}{4}\sum_{h=0}^{3} \frac{C(h|h)}{\sum_{r=0}^{3} C(h|r)} = \frac{1}{4}\sum_{h=0}^{3} \frac{\#\, times\ h\ correctly\ recognized}{\#\, times\ h\ guessed}$$

Outline

- DCT
- KNN
- How to draw the contour plots of a multivariate Gaussian pdf

The Multivariate Gaussian probability density function

If the dimensions of $\vec{x}$ are jointly Gaussian, then we can write their joint probability density function (pdf) as

$$f_{\vec{X}}(\vec{x}) = \mathcal{N}(\vec{x};\vec{\mu},R) = \frac{1}{|2\pi R|^{1/2}}\, e^{-\frac{1}{2}\,(\vec{x}-\vec{\mu})^T R^{-1}(\vec{x}-\vec{\mu})}$$

The exponent is sometimes called the Mahalanobis distance (with weight matrix R) between $\vec{x}$ and $\vec{\mu}$ (named after Prasanta Chandra Mahalanobis, 1893-1972):

$$d_R^2(\vec{x},\vec{\mu}) = (\vec{x}-\vec{\mu})^T R^{-1}(\vec{x}-\vec{\mu})$$

Contour lines of a Gaussian pdf

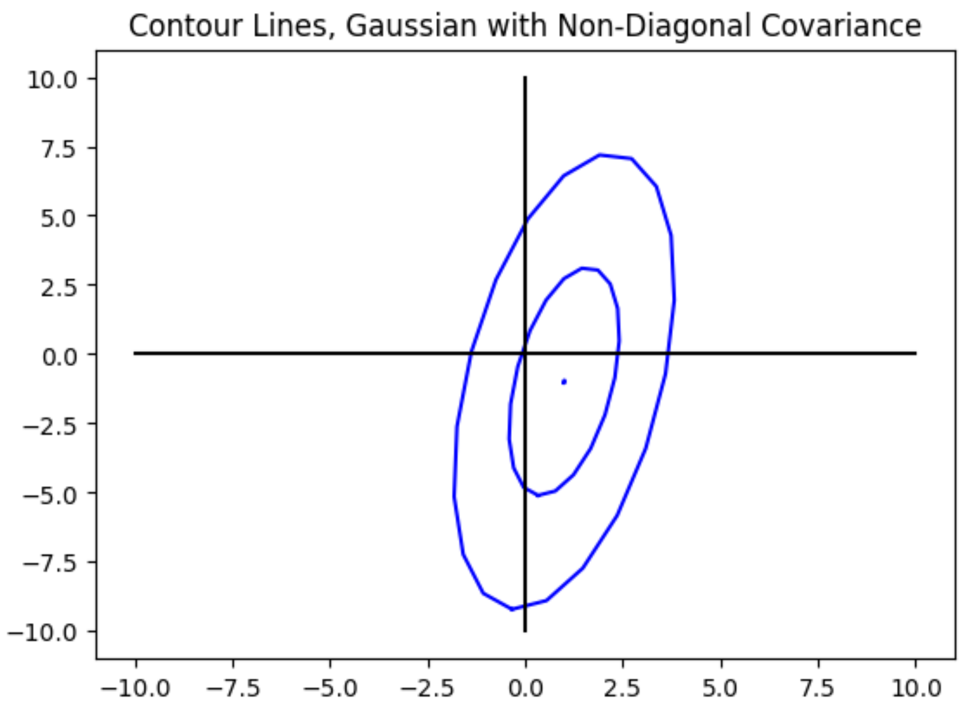
The contour lines of a Gaussian pdf are the lines of constant Mahalanobis distance between $\vec{x}$ and $\vec{\mu}$. For example:

$\frac{f_{\vec{X}}(\vec{x})}{f_{\vec{X}}(\vec{\mu})} = e^{-\frac{1}{2}}$ when $1 = d_R^2(\vec{x},\vec{\mu})$ which happens when

$$1 = d_R^2(\vec{x},\vec{\mu}) = (\vec{x}-\vec{\mu})^T R^{-1}(\vec{x}-\vec{\mu})$$

$\frac{f_{\vec{X}}(\vec{x})}{f_{\vec{X}}(\vec{\mu})} = e^{-2}$ when $4 = d_R^2(\vec{x},\vec{\mu})$ which happens when

$$4 = d_R^2(\vec{x},\vec{\mu}) = (\vec{x}-\vec{\mu})^T R^{-1}(\vec{x}-\vec{\mu})$$



Inverse of a positive definite matrix

The inverse of a positive definite matrix is:

$$R^{-1} = V\Lambda^{-1}V^T$$

Proof:

$$R\ R^{-1} = V\Lambda V^T V\Lambda^{-1}V^T = V\Lambda\Lambda^{-1}V^T = VV^T = I$$

So

$$d_R^2(\vec{x},\vec{\mu}) = (\vec{x}-\vec{\mu})^T R^{-1}(\vec{x}-\vec{\mu}) = (\vec{x}-\vec{\mu})^T\ V\Lambda^{-1}V^T(\vec{x}-\vec{\mu})$$

$$= \vec{y}^T\ \Lambda^{-1}\vec{y}$$

Facts about ellipses

The formula

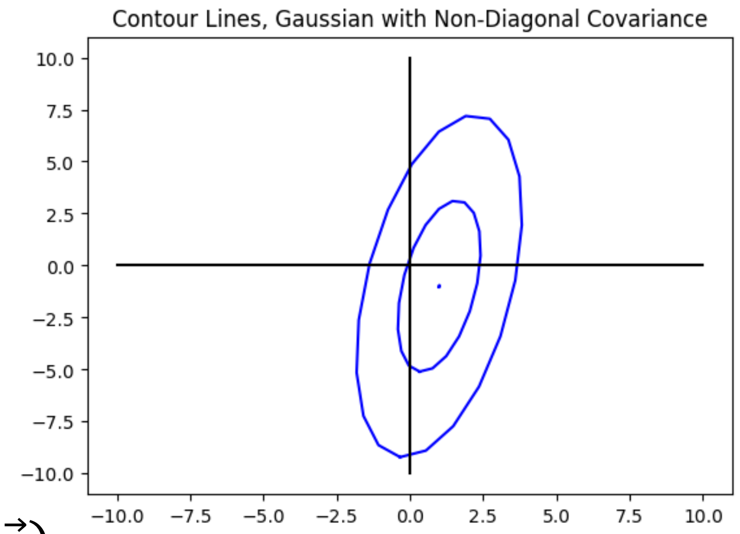
$$1 = \vec{y}^T \Lambda^{-1} \vec{y}, \quad \text{where} \quad \vec{y} = V^T(\vec{x} - \vec{\mu})$$

… or equivalently

$$1 = \frac{{y_0}^2}{\lambda_0} + \cdots + \frac{{y_{D-1}}^2}{\lambda_{D-1}} \quad \text{where} \ \ y_d = {\vec{v}_d}^T(\vec{x} - \vec{\mu})$$

… is the formula for an ellipsoid (in 2D, an ellipse).

- The principal axes are the vectors $\vec{v}_0, \vec{v}_1, \ldots$
- The radius of the ellipse in the $\vec{v}_d$ direction is $\sqrt{\lambda_d}$
- If $\lambda_0 = \lambda_1 = \cdots$, then it’s a circle.



Example

Suppose that x_1 and x_2 are linearly correlated Gaussians with means 1 and -1, respectively, and with variances 1 and 4, and covariance 1.

$$\vec{\mu} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Remember the definitions of variance and covariance:

$$\sigma_1{}^2 = E[(x_1 - \mu_1)^2] = 1$$
$$\sigma_2{}^2 = E[(x_2 - \mu_2)^2] = 4$$
$$\rho_{12} = \rho_{21} = E[(x_1 - \mu_1)(x_2 - \mu_2)] = 1$$

$$R = \begin{bmatrix} 1 & 1 \\ 1 & 4 \end{bmatrix}$$

Example

We have that

$$R = \begin{bmatrix} 1 & 1 \\ 1 & 4 \end{bmatrix}$$

We get the eigenvalues from the determinant equation: $|R - \lambda I| = (1-\lambda)(4-\lambda) - 1 = \lambda^2 - 5\lambda + 3$ which equals zero for $\lambda = \frac{5 \pm \sqrt{13}}{2}$.

We get the eigenvectors by solving $\lambda \vec{v} = R\vec{v}$, which gives

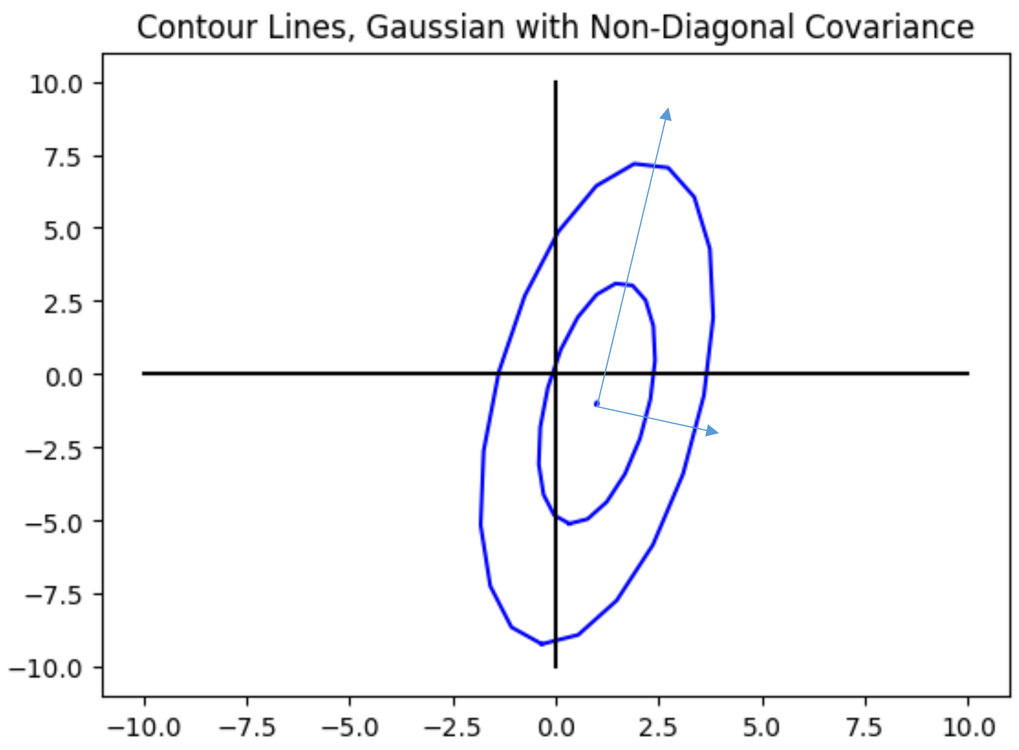
$$\vec{v}_1 \propto \begin{bmatrix} 1 \\ \frac{3+\sqrt{13}}{2} \end{bmatrix}, \vec{v}_2 \propto \begin{bmatrix} 1 \\ \frac{3-\sqrt{13}}{2} \end{bmatrix}$$

Example

So the principal axes of the ellipse are in the directions

$$\vec{v}_1 \propto \begin{bmatrix} 1 \\ \frac{3+\sqrt{13}}{2} \end{bmatrix},$$

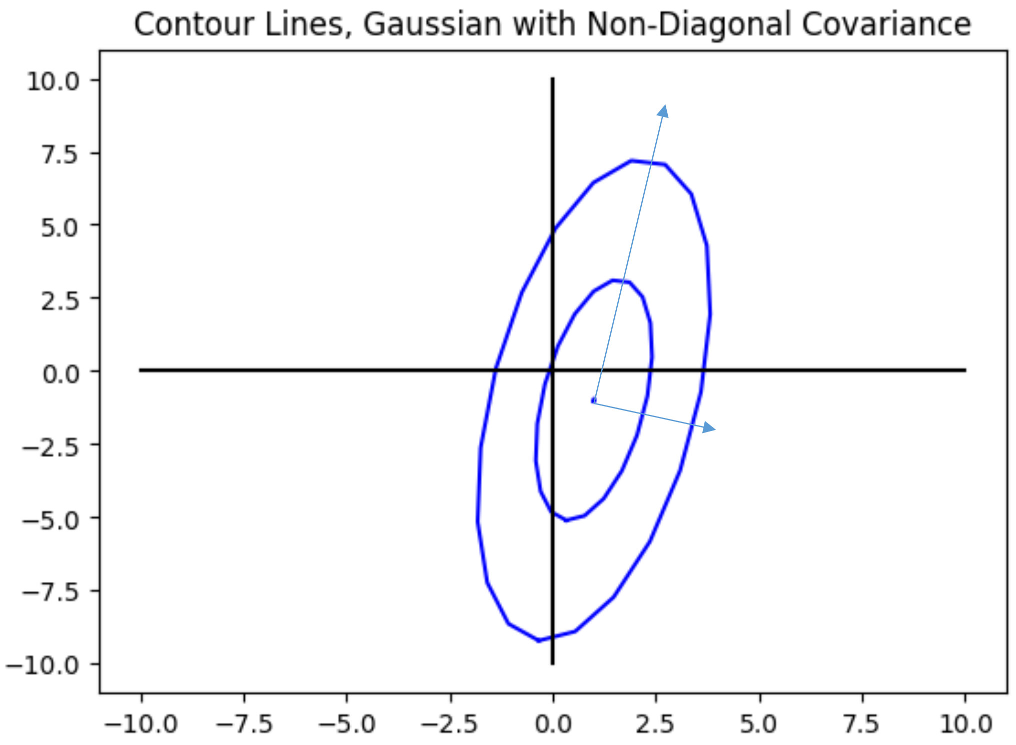
$$\vec{v}_2 \propto \begin{bmatrix} 1 \\ \frac{3-\sqrt{13}}{2} \end{bmatrix}$$



Summary

In fact, it's useful to talk about R in this way:

- The first principal component, y_1, is the part of $(\vec{x}-\vec{\mu})$ that's in the $\vec{v}_1$ direction. It has a variance of λ_1.
- The second principal component, y_2, is the part of $(\vec{x}-\vec{\mu})$ that's in the $\vec{v}_2$ direction. It has a variance of λ_2.
- The principal components are uncorrelated with each other.
- If $\vec{x}$ is Gaussian, then y_1 and y_2 are independent Gaussian random variables.


Contour Lines, Gaussian with Non-Diagonal Covariance