Math 415 - Lecture 38

Applications of SVD

Monday December 7th 2015

Textbook reading: Chapter 6.3

Textbook reading: Chapter 6.3

Strang lecture: Lecture 29: Singular Value Decomposition

- Final exam is on Thursday 12/17/2015 8am - 11am. We will announce the room assignment this week.

- Final exam is on Thursday 12/17/2015 8am - 11am. We will announce the room assignment this week.
- Conflict final exam is on Tuesday 12/15/2015 8am-11am. If you send your TA an email, you will receive an email with the location this week.

- Final exam is on Thursday 12/17/2015 8am - 11am. We will announce the room assignment this week.
- Conflict final exam is on Tuesday 12/15/2015 8am-11am. If you send your TA an email, you will receive an email with the location this week.
- Lecture on Wednesday 12/9 will be used for review.

- Final exam is on Thursday 12/17/2015 8am - 11am. We will announce the room assignment this week.
- Conflict final exam is on Tuesday 12/15/2015 8am-11am. If you send your TA an email, you will receive an email with the location this week.
- Lecture on Wednesday 12/9 will be used for review.
- Please check your scores online. If incorrect, contact TA.

Review

Singular Value Decomposition:

$$A = \underbrace{\begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_m \\ | & & | \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} \sigma_1 & 0 & \\ 0 & \sigma_2 & \\ & & \ddots \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} - & \mathbf{v}_1^T & - \\ & \vdots & \\ - & \mathbf{v}_n^T & - \end{bmatrix}}_{V^T}$$

Singular Value Decomposition:

$$A = \underbrace{\begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_m \\ | & & | \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} \sigma_1 & 0 & \\ 0 & \sigma_2 & \\ & & \ddots \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} - & \mathbf{v}_1^T & - \\ & \vdots & \\ - & \mathbf{v}_n^T & - \end{bmatrix}}_{V^T}$$

- $A = U\Sigma V^T$, where U $(m \times m)$, V $(n \times n)$ are orthogonal.

Singular Value Decomposition:

$$A = \underbrace{\begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_m \\ | & & | \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} \sigma_1 & 0 & \\ 0 & \sigma_2 & \\ & & \ddots \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} - & \mathbf{v}_1^T & - \\ & \vdots & \\ - & \mathbf{v}_n^T & - \end{bmatrix}}_{V^T}$$

- $A = U\Sigma V^T$, where U $(m \times m)$, V $(n \times n)$ are orthogonal.
- Σ is rectangular $m \times n$ and diagonal, the r non zero diagonal entries are called **singular values**, they are positive.

Singular Value Decomposition:

$$A = \underbrace{\begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_m \\ | & & | \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} \sigma_1 & 0 & \\ 0 & \sigma_2 & \\ & & \ddots \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} - & \mathbf{v}_1^T & - \\ & \vdots & \\ - & \mathbf{v}_n^T & - \end{bmatrix}}_{V^T}$$

- $A = U\Sigma V^T$, where U $(m \times m)$, V $(n \times n)$ are orthogonal.
- Σ is rectangular $m \times n$ and diagonal, the r non zero diagonal entries are called **singular values**, they are positive.
- Can be rewritten in *column times row form*

$$A = \mathbf{u}_1\sigma_1\mathbf{v}_1^T + \mathbf{u}_2\sigma_2\mathbf{v}_2^T + \ldots + \mathbf{u}_r\sigma_r\mathbf{v}_r^T$$

Singular Value Decomposition:

$$A = \underbrace{\begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_m \\ | & & | \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} \sigma_1 & 0 & \\ 0 & \sigma_2 & \\ & & \ddots \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} - & \mathbf{v}_1^T & - \\ & \vdots & \\ - & \mathbf{v}_n^T & - \end{bmatrix}}_{V^T}$$

- $A = U\Sigma V^T$, where U $(m \times m)$, V $(n \times n)$ are orthogonal.
- Σ is rectangular $m \times n$ and diagonal, the r non zero diagonal entries are called **singular values**, they are positive.
- Can be rewritten in *column times row form*

$$A = \mathbf{u}_1\sigma_1\mathbf{v}_1^T + \mathbf{u}_2\sigma_2\mathbf{v}_2^T + \ldots + \mathbf{u}_r\sigma_r\mathbf{v}_r^T$$

- $\mathbf{u}_1, \ldots, \mathbf{u}_m$ orthonormal eigenbasis of AA^T.

Singular Value Decomposition:

$$A = \underbrace{\begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_m \\ | & & | \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} \sigma_1 & 0 & \\ 0 & \sigma_2 & \\ & & \ddots \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} - & \mathbf{v}_1^T & - \\ & \vdots & \\ - & \mathbf{v}_n^T & - \end{bmatrix}}_{V^T}$$

- $A = U\Sigma V^T$, where U $(m \times m)$, V $(n \times n)$ are orthogonal.
- Σ is rectangular $m \times n$ and diagonal, the r non zero diagonal entries are called **singular values**, they are positive.
- Can be rewritten in *column times row form*

$$A = \mathbf{u}_1\sigma_1\mathbf{v}_1^T + \mathbf{u}_2\sigma_2\mathbf{v}_2^T + \ldots + \mathbf{u}_r\sigma_r\mathbf{v}_r^T$$

- $\mathbf{u}_1, \ldots, \mathbf{u}_m$ orthonormal eigenbasis of AA^T.

Singular Value Decomposition:

$$A = \underbrace{\begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_m \\ | & & | \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} \sigma_1 & 0 & \\ 0 & \sigma_2 & \\ & & \ddots \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} - & \mathbf{v}_1^T & - \\ & \vdots & \\ - & \mathbf{v}_n^T & - \end{bmatrix}}_{V^T}$$

- $A = U\Sigma V^T$, where U $(m \times m)$, V $(n \times n)$ are orthogonal.
- Σ is rectangular $m \times n$ and diagonal, the r non zero diagonal entries are called **singular values**, they are positive.
- Can be rewritten in *column times row form*

$$A = \mathbf{u}_1\sigma_1\mathbf{v}_1^T + \mathbf{u}_2\sigma_2\mathbf{v}_2^T + \ldots + \mathbf{u}_r\sigma_r\mathbf{v}_r^T$$

- $\mathbf{u}_1, \ldots, \mathbf{u}_m$ orthonormal eigenbasis of AA^T.$\mathbf{v_1}, \ldots, \mathbf{v_n}$ orthonormal eigenbasis for A^TA.

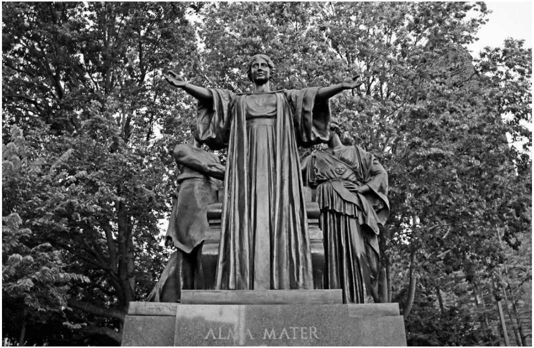
Image Compression

Idea

Idea

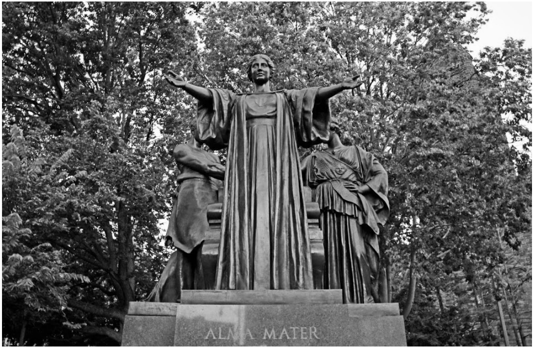
We have a grayscale picture that is $m \times n$ pixels in size:

We have a grayscale picture that is $m \times n$ pixels in size:



Each pixel is a shade of gray from 0 (black) to 255 (white).

We have a grayscale picture that is $m \times n$ pixels in size:



Each pixel is a shade of gray from 0 (black) to 255 (white). This gives an $m \times n$ matrix A. Each entry of A is one pixel of the image; that entry is some integer from 0 to 255, giving the brightness of that pixel.

Encoding a large picture takes up a lot of space:

- Say our picture is 625×960 pixels.

Encoding a large picture takes up a lot of space:

- Say our picture is 625×960 pixels.
- Each pixel has 256 possible values. (This takes up exactly 8 bits, or 1 byte, of memory on a computer).

Encoding a large picture takes up a lot of space:

- Say our picture is 625×960 pixels.
- Each pixel has 256 possible values. (This takes up exactly 8 bits, or 1 byte, of memory on a computer).
- The whole picture requires $625 \times 960 = 600000$ bytes, so 600 kB.

Encoding a large picture takes up a lot of space:

- Say our picture is 625×960 pixels.
- Each pixel has 256 possible values. (This takes up exactly 8 bits, or 1 byte, of memory on a computer).
- The whole picture requires $625 \times 960 = 600000$ bytes, so 600 kB.
- That is not so bad, but color pictures from your personal camera would be about 30MB each without any compression... and that quickly adds up.

Encoding a large picture takes up a lot of space:

- Say our picture is 625×960 pixels.
- Each pixel has 256 possible values. (This takes up exactly 8 bits, or 1 byte, of memory on a computer).
- The whole picture requires $625 \times 960 = 600000$ bytes, so 600 kB.
- That is not so bad, but color pictures from your personal camera would be about 30MB each without any compression... and that quickly adds up.

Question

Can we do better?

Use singular value decomposition of A:

Use singular value decomposition of A:

$$A = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_{625} \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \\ 0 & \sigma_2 & \\ & & \ddots \end{bmatrix} \begin{bmatrix} - & \mathbf{v}_1^T & - \\ & \vdots & \\ - & \mathbf{v}_{960}^T & - \end{bmatrix}$$

Use singular value decomposition of A:

$$A = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \ldots & \mathbf{u}_{625} \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \\ 0 & \sigma_2 & \\ & & \ddots \end{bmatrix} \begin{bmatrix} - & \mathbf{v}_1^T & - \\ & \vdots & \\ - & \mathbf{v}_{960}^T & - \end{bmatrix}$$

Recall we can rewrite this as

$$A = \mathbf{u}_1\sigma_1\mathbf{v}_1^T + \mathbf{u}_2\sigma_2\mathbf{v}_2^T + \ldots + \mathbf{u}_r\sigma_r\mathbf{v}_r^T$$

Use singular value decomposition of A:

$$A = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_{625} \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \\ 0 & \sigma_2 & \\ & & \ddots \end{bmatrix} \begin{bmatrix} - & \mathbf{v}_1^T & - \\ & \vdots & \\ - & \mathbf{v}_{960}^T & - \end{bmatrix}$$

Recall we can rewrite this as

$$A = \mathbf{u}_1\sigma_1\mathbf{v}_1^T + \mathbf{u}_2\sigma_2\mathbf{v}_2^T + \ldots + \mathbf{u}_r\sigma_r\mathbf{v}_r^T$$

Idea

Throw away the term $\mathbf{u}_i\sigma_i\mathbf{v}_i^T$ when σ_i is small.

Use singular value decomposition of A:

$$A = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_{625} \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \\ 0 & \sigma_2 & \\ & & \ddots \end{bmatrix} \begin{bmatrix} - & \mathbf{v}_1^T & - \\ & \vdots & \\ - & \mathbf{v}_{960}^T & - \end{bmatrix}$$

Recall we can rewrite this as

$$A = \mathbf{u}_1\sigma_1\mathbf{v}_1^T + \mathbf{u}_2\sigma_2\mathbf{v}_2^T + \ldots + \mathbf{u}_r\sigma_r\mathbf{v}_r^T$$

Idea

Throw away the term $\mathbf{u}_i\sigma_i\mathbf{v}_i^T$ when σ_i is small.

Use singular value decomposition of A:

$$A = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_{625} \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \\ 0 & \sigma_2 & \\ & & \ddots \end{bmatrix} \begin{bmatrix} - & \mathbf{v}_1^T & - \\ & \vdots & \\ - & \mathbf{v}_{960}^T & - \end{bmatrix}$$

Recall we can rewrite this as

$$A = \mathbf{u}_1\sigma_1\mathbf{v}_1^T + \mathbf{u}_2\sigma_2\mathbf{v}_2^T + \ldots + \mathbf{u}_r\sigma_r\mathbf{v}_r^T$$

Idea

Throw away the term $\mathbf{u}_i\sigma_i\mathbf{v}_i^T$ when σ_i is small. If $k \leq r$, define

$$A_k = \mathbf{u}_1\sigma_1\mathbf{v}_1^T + \mathbf{u}_2\sigma_2\mathbf{v}_2^T + \ldots + \mathbf{u}_k\sigma_k\mathbf{v}_k^T$$

Use singular value decomposition of A:

$$A = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_{625} \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \\ 0 & \sigma_2 & \\ & & \ddots \end{bmatrix} \begin{bmatrix} - & \mathbf{v}_1^T & - \\ & \vdots & \\ - & \mathbf{v}_{960}^T & - \end{bmatrix}$$

Recall we can rewrite this as

$$A = \mathbf{u}_1\sigma_1\mathbf{v}_1^T + \mathbf{u}_2\sigma_2\mathbf{v}_2^T + \dots + \mathbf{u}_r\sigma_r\mathbf{v}_r^T$$

Idea

Throw away the term $\mathbf{u}_i\sigma_i\mathbf{v}_i^T$ when σ_i is small. If $k \leq r$, define

$$A_k = \mathbf{u}_1\sigma_1\mathbf{v}_1^T + \mathbf{u}_2\sigma_2\mathbf{v}_2^T + \dots + \mathbf{u}_k\sigma_k\mathbf{v}_k^T$$

The matrix A_k is very close to the matrix A, if $\sigma_{k+1}, \dots, \sigma_r$ are small.

For example, take A_{100}:

A_k is also easier to store:

A_k is also easier to store:

- If $k = 100$, then to store the matrix A_{100} we need the numbers $\sigma_1, \ldots, \sigma_{100}$, the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_{100}$ and $\mathbf{v}_1, \ldots, \mathbf{v}_{100}$.

A_k is also easier to store:

- If $k = 100$, then to store the matrix A_{100} we need the numbers $\sigma_1, \ldots, \sigma_{100}$, the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_{100}$ and $\mathbf{v}_1, \ldots, \mathbf{v}_{100}$.
- That's

$$100 + 100(625) + 100(960) = 158600$$

numbers

A_k is also easier to store:

- If $k = 100$, then to store the matrix A_{100} we need the numbers $\sigma_1, \ldots, \sigma_{100}$, the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_{100}$ and $\mathbf{v}_1, \ldots, \mathbf{v}_{100}$.
- That's
$$100 + 100(625) + 100(960) = 158600$$
numbers
- Compare to the original matrix which had
$$625 \cdot 960 = 600000$$
numbers.

A_k is also easier to store:

- If $k = 100$, then to store the matrix A_{100} we need the numbers $\sigma_1, \ldots, \sigma_{100}$, the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_{100}$ and $\mathbf{v}_1, \ldots, \mathbf{v}_{100}$.
- That's
$$100 + 100(625) + 100(960) = 158600$$
numbers
- Compare to the original matrix which had
$$625 \cdot 960 = 600000$$
numbers.

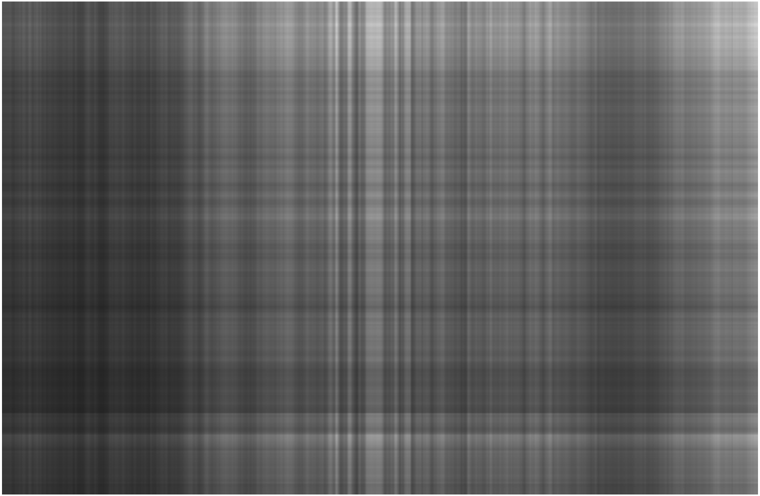
A_k is also easier to store:

- If $k = 100$, then to store the matrix A_{100} we need the numbers $\sigma_1, \dots, \sigma_{100}$, the vectors $\mathbf{u}_1, \dots, \mathbf{u}_{100}$ and $\mathbf{v}_1, \dots, \mathbf{v}_{100}$.
- That's
$$100 + 100(625) + 100(960) = 158600$$
numbers
- Compare to the original matrix which had
$$625 \cdot 960 = 600000$$
numbers.

We reduced the file size by a factor of four!

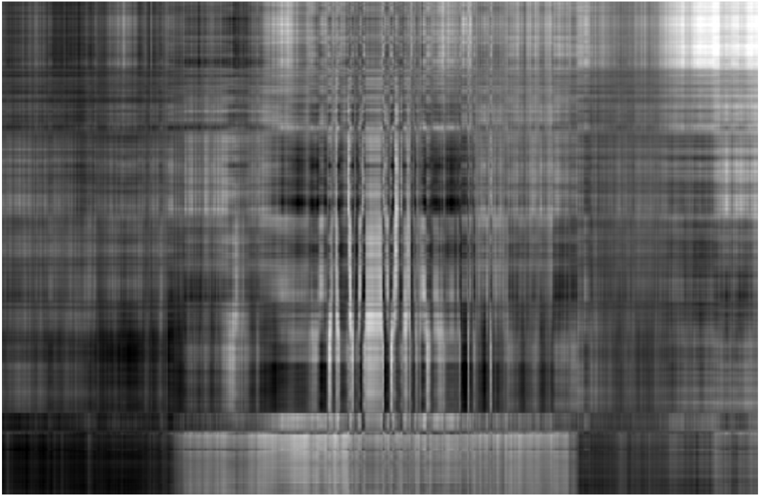
Examples

Examples



A_1

Examples



A_5

Examples

A_{25}

Examples

A_{50}

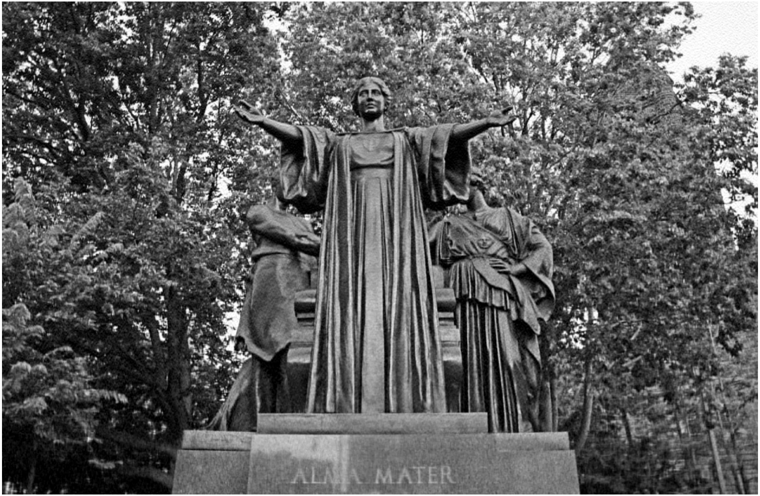
Examples

A_{100}

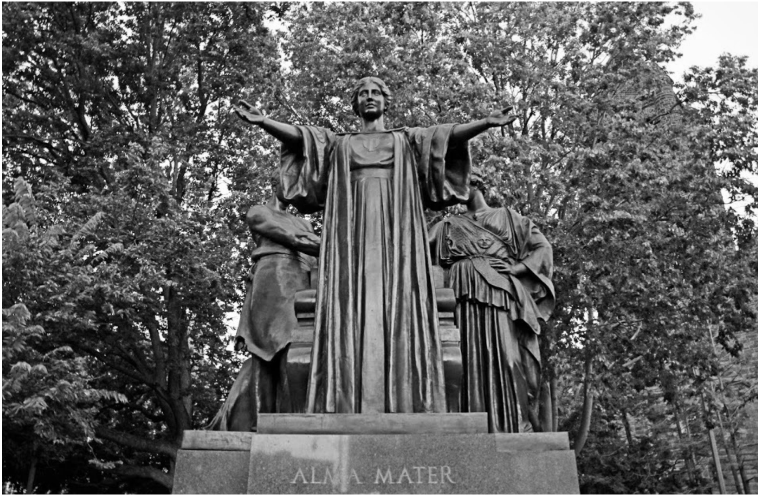
Examples

A_{150}

Examples



A_{200}



A. Probably rank A is 625, so $A = A_{625}$.

Face recognition

Idea

A set of eigenfaces can be generated by performing a mathematical process called principal component analysis (PCA) on a large set of images depicting different human faces. Informally, eigenfaces can be considered a set of "standardized face ingredients", derived from statistical analysis of many pictures of faces. Any human face can be considered to be a combination of these standard faces. For example, one's face might be composed of the average face plus 10% from eigenface 1, 55% from eigenface 2, and even -3% from eigenface 3.

Remarkably, it does not take many eigenfaces combined together to achieve a fair approximation of most faces. Also, because a person's face is not recorded by a digital photograph, but instead as just a list of values (one value for each eigenface in the database used), much less space is taken for each person's face.

Algorithm

Part I - Set up

Step 1. Start with a set of d face images - same resolution, say $r \times c$. Each image is treated as one vector by concatenating the columns of pixels in the original image, resulting in vector in $\mathbb{R}^{rc}$. Set $n = rc$. This gives us d vectors $\mathbf{x}_1, \ldots, \mathbf{x}_d$ in $\mathbb{R}^n$.

Part I - Set up

Step 1. Start with a set of d face images - same resolution, say $r \times c$. Each image is treated as one vector by concatenating the columns of pixels in the original image, resulting in vector in $\mathbb{R}^{rc}$. Set $n = rc$. This gives us d vectors $\mathbf{x}_1, \ldots, \mathbf{x}_d$ in $\mathbb{R}^n$.

Step 2. Calculate average face $\mathbf{a} = \frac{\mathbf{x}_1 + \cdots + \mathbf{x}_d}{d}$. Set $\mathbf{y}_i := \mathbf{x}_i - \mathbf{a}$.

Part I - Set up

Step 1. Start with a set of d face images - same resolution, say $r \times c$. Each image is treated as one vector by concatenating the columns of pixels in the original image, resulting in vector in $\mathbb{R}^{rc}$. Set $n = rc$. This gives us d vectors $\mathbf{x}_1, \ldots, \mathbf{x}_d$ in $\mathbb{R}^n$.

Step 2. Calculate average face $\mathbf{a} = \frac{\mathbf{x}_1 + \cdots + \mathbf{x}_d}{d}$. Set $\mathbf{y}_i := \mathbf{x}_i - \mathbf{a}$.Let X be the $n \times d$ matrix whose i-th column is $\mathbf{y}_i$.

Part I - Set up

Step 1. Start with a set of d face images - same resolution, say $r \times c$. Each image is treated as one vector by concatenating the columns of pixels in the original image, resulting in vector in $\mathbb{R}^{rc}$. Set $n = rc$. This gives us d vectors $\mathbf{x}_1, \ldots, \mathbf{x}_d$ in $\mathbb{R}^n$.

Step 2. Calculate average face $\mathbf{a} = \frac{\mathbf{x}_1 + \cdots + \mathbf{x}_d}{d}$. Set $\mathbf{y}_i := \mathbf{x}_i - \mathbf{a}$.Let X be the $n \times d$ matrix whose i-th column is $\mathbf{y}_i$.

Step 3. Calculate the singular value decomposition of X. Say $X = U\Sigma V^T$. Note that $U = \begin{bmatrix} \mathbf{u}_1 & \ldots & \mathbf{u}_n \end{bmatrix}$ is a $n \times n$-matrix, V is $d \times d$-matrix.

Part I - Set up

Step 1. Start with a set of d face images - same resolution, say $r \times c$. Each image is treated as one vector by concatenating the columns of pixels in the original image, resulting in vector in $\mathbb{R}^{rc}$. Set $n = rc$. This gives us d vectors $\mathbf{x}_1, \ldots, \mathbf{x}_d$ in $\mathbb{R}^n$.

Step 2. Calculate average face $\mathbf{a} = \frac{\mathbf{x}_1 + \cdots + \mathbf{x}_d}{d}$. Set $\mathbf{y}_i := \mathbf{x}_i - \mathbf{a}$.Let X be the $n \times d$ matrix whose i-th column is $\mathbf{y}_i$.

Step 3. Calculate the singular value decomposition of X. Say $X = U\Sigma V^T$. Note that $U = \begin{bmatrix} \mathbf{u}_1 & \ldots & \mathbf{u}_n \end{bmatrix}$ is a $n \times n$-matrix, V is $d \times d$-matrix.

Step 4. The vectors $\mathbf{u}_1, \ldots, \mathbf{u_n}$ are called eigenfaces. Forget all eigenfaces, where the corresponding singular value is small. Let's say, we keep $\mathbf{u}_1, \ldots \mathbf{u}_s$ for some $s < n$.

So what are eigenfaces?

So what are eigenfaces?

We said: ‘Eigenfaces can be considered a set of ”standardized face ingredients”, derived from statistical analysis of many pictures of faces.’

So what are eigenfaces?

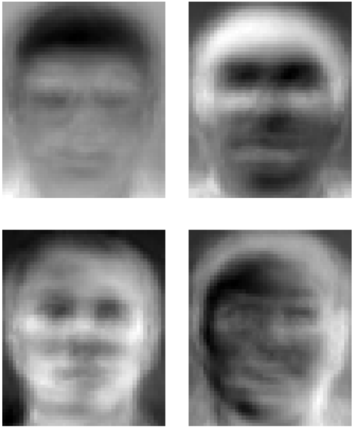
We said: 'Eigenfaces can be considered a set of "standardized face ingredients", derived from statistical analysis of many pictures of faces.'

Note that $\mathbf{u}_1, \ldots, \mathbf{u_n}$ are the eigenvalues (principal components/axes) of XX^T.

So what are eigenfaces?

We said: 'Eigenfaces can be considered a set of "standardized face ingredients", derived from statistical analysis of many pictures of faces.'

Note that $\mathbf{u}_1, \ldots, \mathbf{u}_\mathbf{n}$ are the eigenvalues (principal components/axes) of XX^T. We can think of XX^T as the matrix of all possible combination of the faces. Therefore $\mathbf{u}_1, \ldots, \mathbf{u}_\mathbf{n}$ are the principal components of all the possible faces.



Some eigenfaces from AT&T Laboratories Cambridge

Part II - Learning new faces

Suppose we want to add a face $\mathbf{f} \in \mathbb{R}^n$ to the database. Then instead of saving f, we save the following vector in $\mathbb{R}^s$

$$\begin{bmatrix} (\mathbf{f}-\mathbf{a}) \cdot \mathbf{u}_1 \\ \vdots \\ (\mathbf{f}-\mathbf{a}) \cdot \mathbf{u}_s \end{bmatrix}.$$

We call this vector $\mathbf{w}_f$.

Part II - Learning new faces

Suppose we want to add a face $\mathbf{f} \in \mathbb{R}^n$ to the database. Then instead of saving f, we save the following vector in $\mathbb{R}^s$

$$\begin{bmatrix} (\mathbf{f}-\mathbf{a}) \cdot \mathbf{u}_1 \\ \vdots \\ (\mathbf{f}-\mathbf{a}) \cdot \mathbf{u}_s \end{bmatrix}.$$

We call this vector $\mathbf{w}_f$.

Do you know what the entries represent?

Part II - Learning new faces

Suppose we want to add a face $\mathbf{f} \in \mathbb{R}^n$ to the database. Then instead of saving f, we save the following vector in $\mathbb{R}^s$

$$\begin{bmatrix} (\mathbf{f}-\mathbf{a}) \cdot \mathbf{u}_1 \\ \vdots \\ (\mathbf{f}-\mathbf{a}) \cdot \mathbf{u}_s \end{bmatrix}.$$

We call this vector $\mathbf{w}_f$.

Do you know what the entries represent? The weights used in the projection of $\mathbf{f}-\mathbf{a}$ onto $\text{span}(\mathbf{u}_1, \ldots, \mathbf{u}_s)$!

Part II - Learning new faces

Suppose we want to add a face $\mathbf{f} \in \mathbb{R}^n$ to the database. Then instead of saving f, we save the following vector in $\mathbb{R}^s$

$$\begin{bmatrix} (\mathbf{f} - \mathbf{a}) \cdot \mathbf{u}_1 \\ \vdots \\ (\mathbf{f} - \mathbf{a}) \cdot \mathbf{u}_s \end{bmatrix}.$$

We call this vector $\mathbf{w}_f$.

Do you know what the entries represent? The weights used in the projection of $\mathbf{f} - \mathbf{a}$ onto $\text{span}(\mathbf{u}_1, \ldots, \mathbf{u}_s)$! So $\mathbf{f}$ is composed of the average face $\mathbf{a}$ plus scalar $(\mathbf{f} - \mathbf{a}) \cdot \mathbf{u}_1$ times eigenface 1 plus $(\mathbf{f} - \mathbf{a}) \cdot \mathbf{u}_2$ times eigenface 2, etc.

Part III - Recognizing a face

Suppose we are now given a photo of a person $\mathbf{p} \in \mathbb{R}^n$.

Part III - Recognizing a face

Suppose we are now given a photo of a person $\mathbf{p} \in \mathbb{R}^n$. First calculate $w_{\mathbf{p}}$.

Part III - Recognizing a face

Suppose we are now given a photo of a person $\mathbf{p} \in \mathbb{R}^n$. First calculate $w_{\mathbf{p}}$. How do we know which person in the database is most likely this person?

Part III - Recognizing a face

Suppose we are now given a photo of a person $\mathbf{p} \in \mathbb{R}^n$. First calculate $w_{\mathbf{p}}$. How do we know which person in the database is most likely this person? Easy, simply take the face f in the database such that $\|w_p - w_f\|$ is minimal!