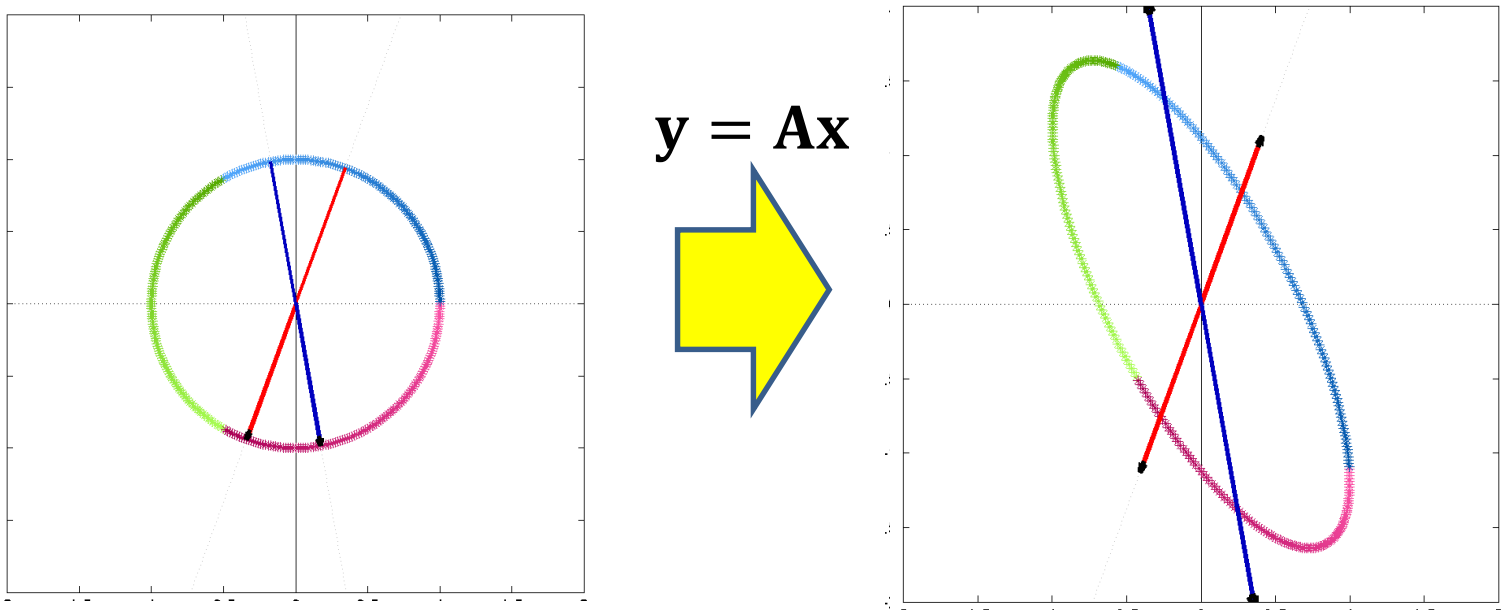
Machine Learning for Signal Processing

Data driven representations: 1. Eigenrepresentations

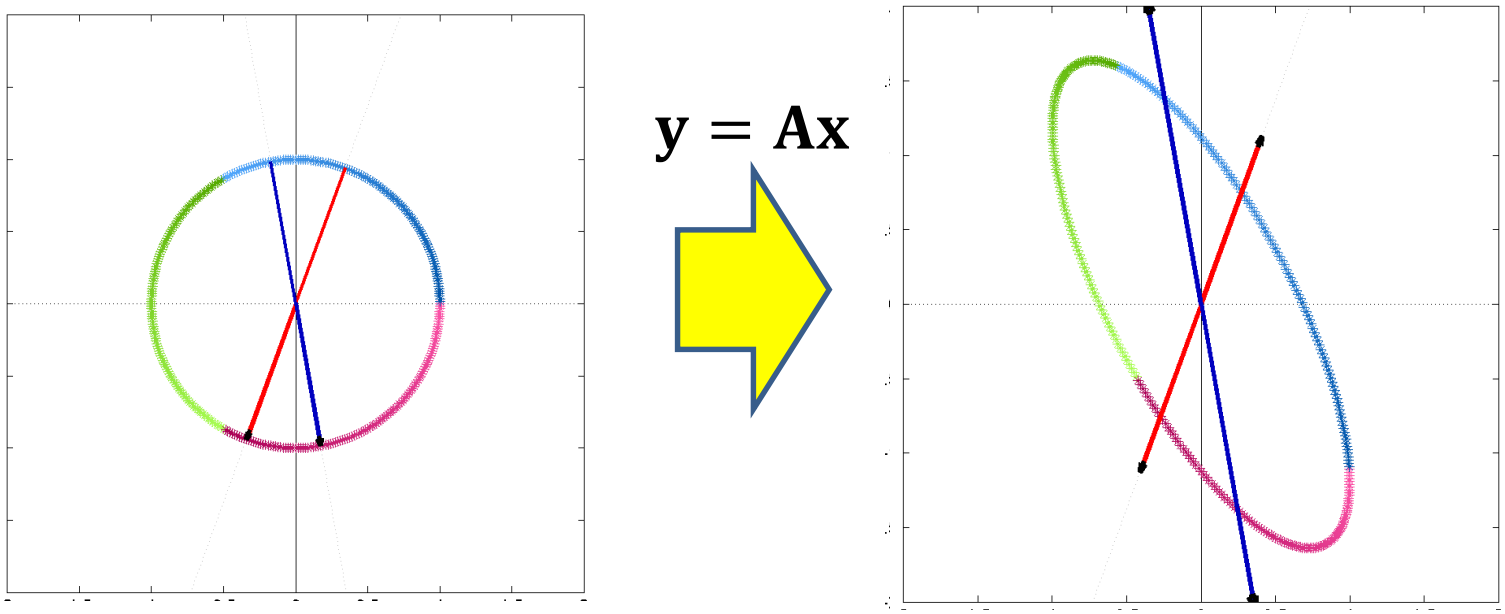
Instructor: Bhiksha Raj

Linear Algebra Reminders: 1



- A matrix transforms a sphereoid to an ellipsoid
- The Eigenvectors of the matrix are the vectors who do not change direction during this transformation

Linear Algebra Reminders: 1.5



- Any square matrix $\mathbf{A}$ can be "Eigen decomposed" as

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$$

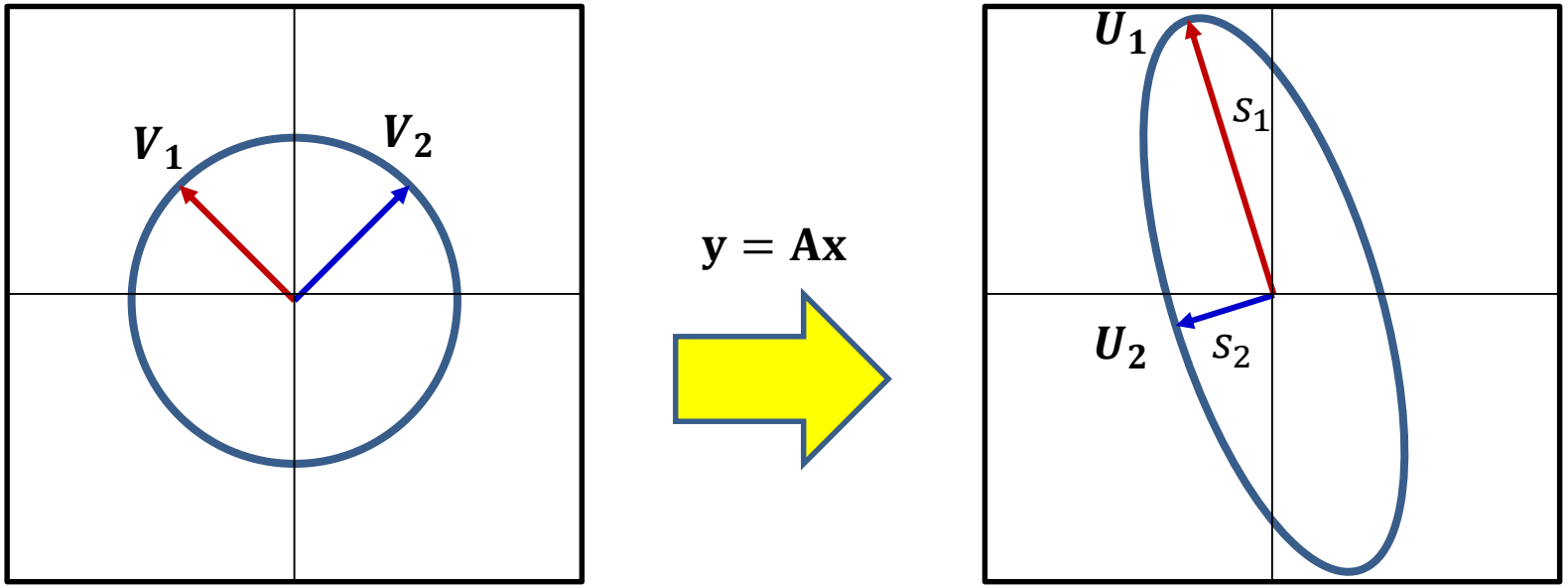
  - $\mathbf{V}$ is the set of Eigen vectors.$\mathbf{\Lambda}$is a diagonal matrix of scaling terms
- If $\mathbf{A}$ is symmetric, we will get

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{T}$$

  - The vectors in $\mathbf{V}$ are orthogonal to one another. $\mathbf{V}$ is an *orthogonal matrix*
  - $\mathbf{V}\mathbf{V}^{T} = \mathbf{V}^{T}\mathbf{V} = \mathbf{I}$

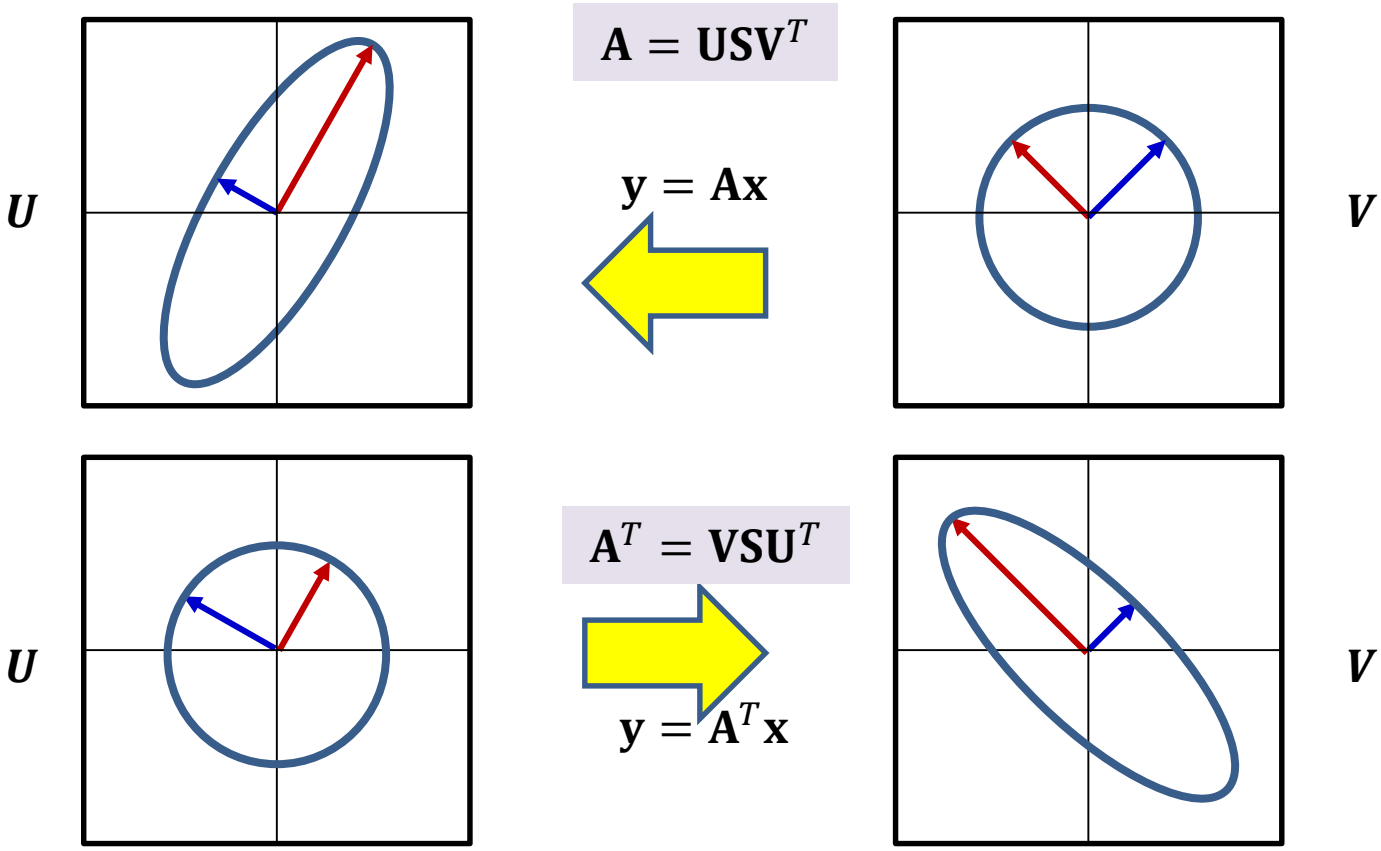
Linear Algebra Reminders: 2

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$



- A matrix transforms the orthogonal set of right singular vectors to the orthogonal set of left singular vectors
  - These are the major axes of the ellipsoid obtained from the sphereoid
  - The scaling factors are the singular values

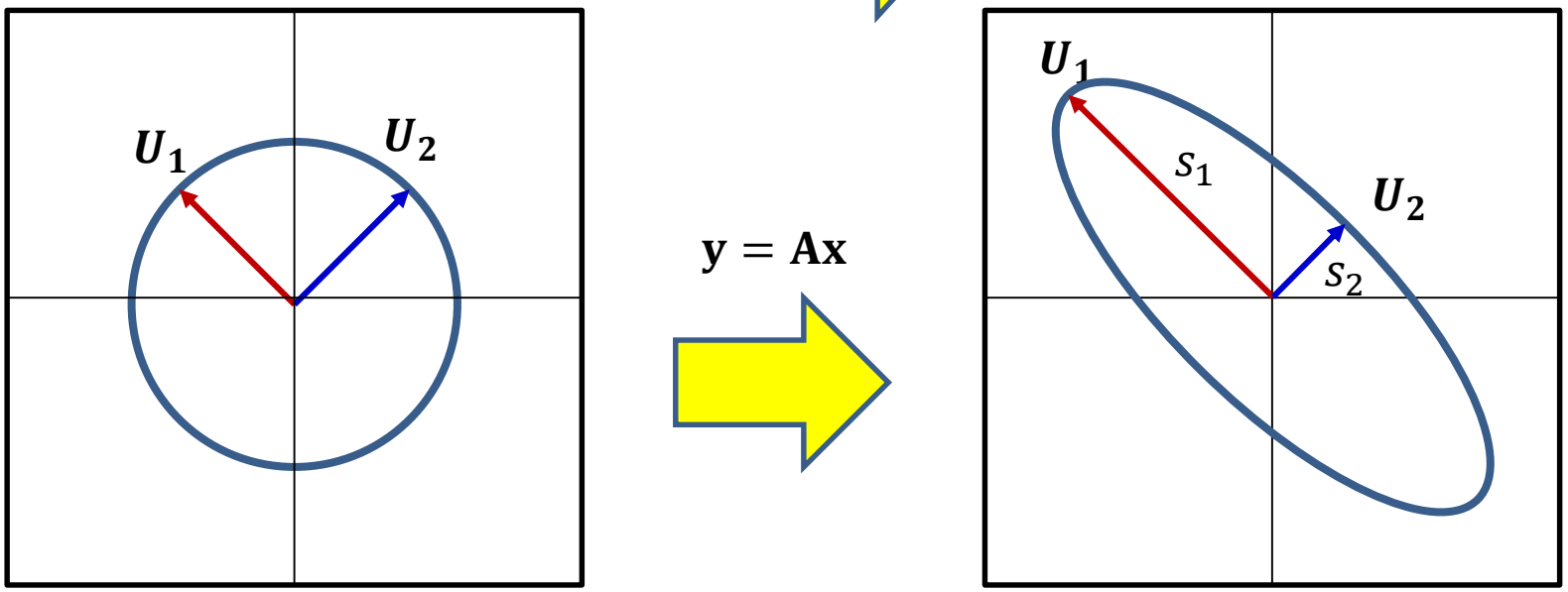
Linear Algebra Reminders: 2



- A matrix transforms the orthogonal set of right singular vectors to the orthogonal set of left singular vectors
  - These are the major axes of the ellipsoid obtained from the sphereoid
  - The scaling factors are the singluar values
- The *transpose* of a matrix transforms the left singular vectors to the right singular vectors

Linear Algebra Reminders: 3

$\mathbf{A} = \mathbf{A}^T$ $\mathbf{USV}^T = \mathbf{VSU}^T$ $\mathbf{U} = \mathbf{V}$ $\boldsymbol{A} = \mathbf{USU}^T$



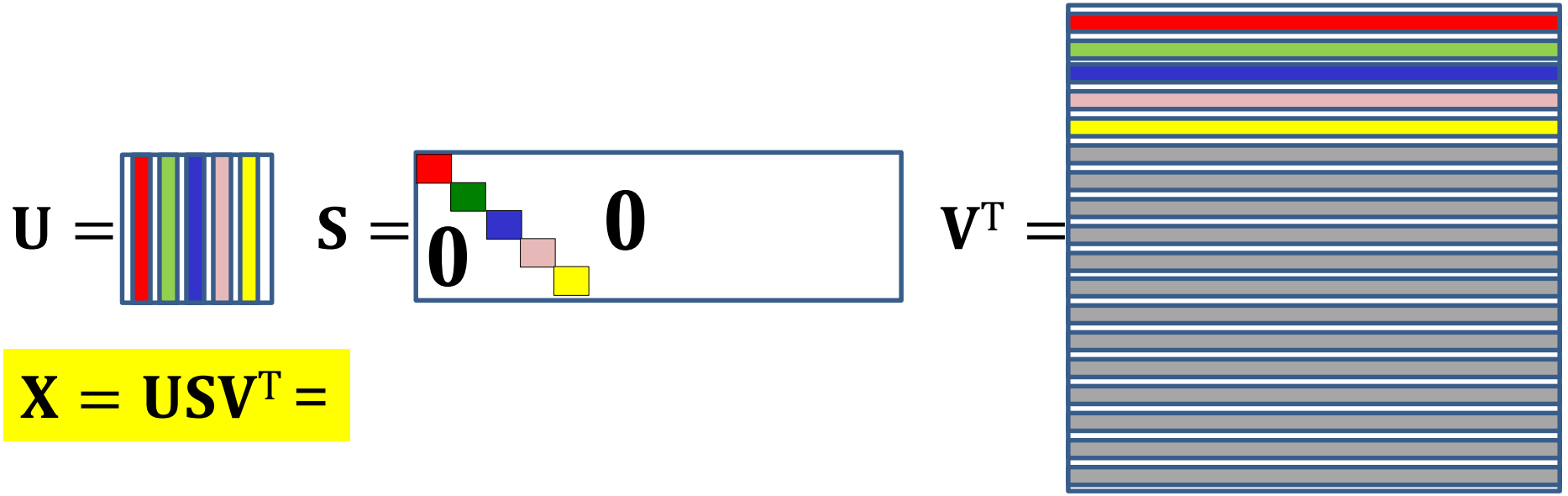
- For a symmetric matrix left and right singular vectors are identical
  - Orthogonal vectors which do not change direction from the transform
  - These are the major axes of the ellipsoid obtained from a sphereoid
- These are also the *eigenvectors* of the matrix
  - Since they do not change direction
  - SVD gives you Eigen decomposition, with $\boldsymbol{\Lambda} = \mathbf{S}^2$

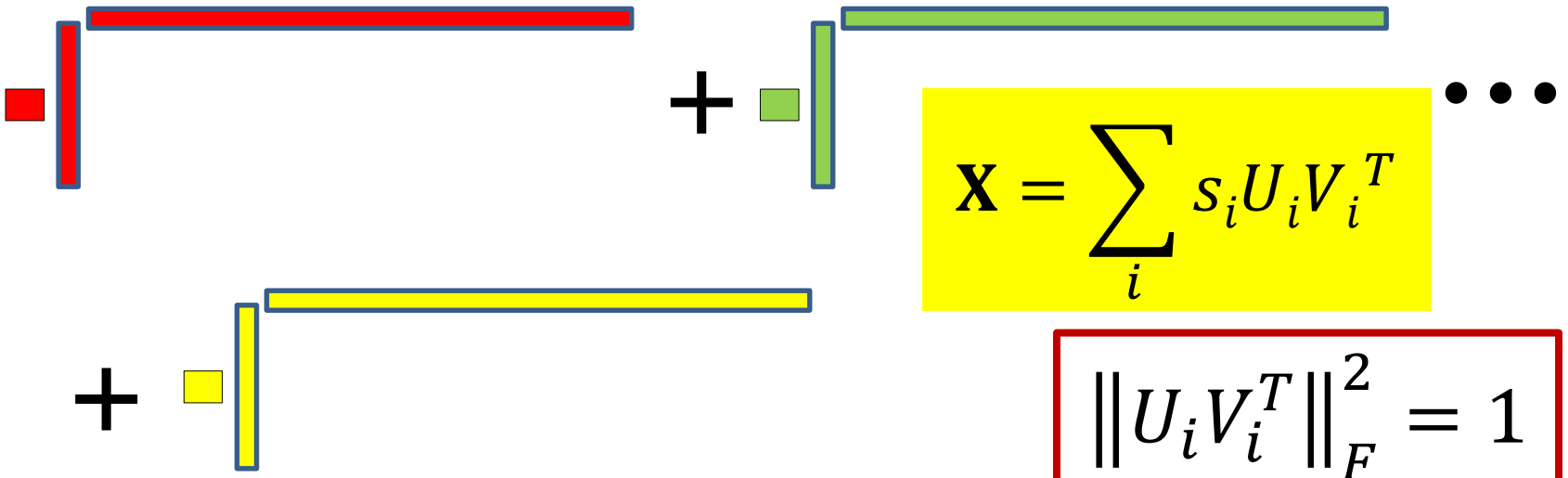
Linear Algebra Reminders: 4 –> SVD

- SVD decomposes a matrix into a the sum of a sequence of “unit-energy” matrices weighted by the corresponding singular values

- Retaining only the “high-singular-value” components retains most of the energy in the matrix

SVD on data-container matrices

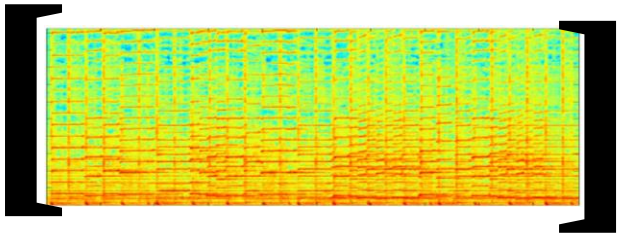




$$\mathbf{X} = \sum_i s_i U_i V_i^T$$

$$\|U_i V_i^T\|_F^2 = 1$$

SVD decomposes the data



$$\mathbf{X} = [X_1 \; X_2 \quad \cdots X_N]$$

$$\mathbf{X} = \mathbf{USV}^{\mathrm{T}}$$

$$\mathbf{X} = s_1 U_1 V_1^T + s_2 U_2 V_2^T + s_3 U_3 V_3^T + s_4 U_4 V_4^T + \ldots$$

- Each left singular vector and the corresponding right singular vector contribute one “basic” component to the data
- The “magnitude” of its contribution is the corresponding singular value

Expanding the SVD

$$\mathbf{X} = s_1 U_1 V_1^T + s_2 U_2 V_2^T + s_3 U_3 V_3^T + s_4 U_4 V_4^T + \ldots$$

- Each left singular vector and the corresponding right singular vector contribute on "basic" component to the data
- The "magnitude" of its contribution is the corresponding singular value
- Low singular-value components contribute little, if anything
  - Carry little information
  - Are often just "noise" in the data

Expanding the SVD

$$\mathbf{X} = s_1 U_1 V_1^T + s_2 U_2 V_2^T + s_3 U_3 V_3^T + s_4 U_4 V_4^T + \ldots$$

$$\mathbf{X} \approx s_1 U_1 V_1^T + s_2 U_2 V_2^T$$

- Low singular-value components contribute little, if anything
  - Carry little information
  - Are often just “noise” in the data
- Data can be recomposed using only the “major” components with minimal change of value
  - Minimum squared error between original data and recomposed data
  - Sometimes eliminating the low-singular-value components will, in fact “clean” the data

Linear Algebra recall

- What is $\mathbf{x}^T\mathbf{y}$
  - When $\mathbf{y}$ is unit length

Linear Algebra recall

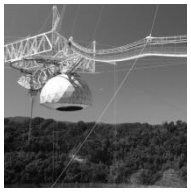
- What is $\mathbf{x}^T\mathbf{y}$
  - When $\mathbf{y}$ is unit length
- What is the projection of $\mathbf{x}$ onto $\mathbf{y}$
  - When $\mathbf{y}$ is unit length

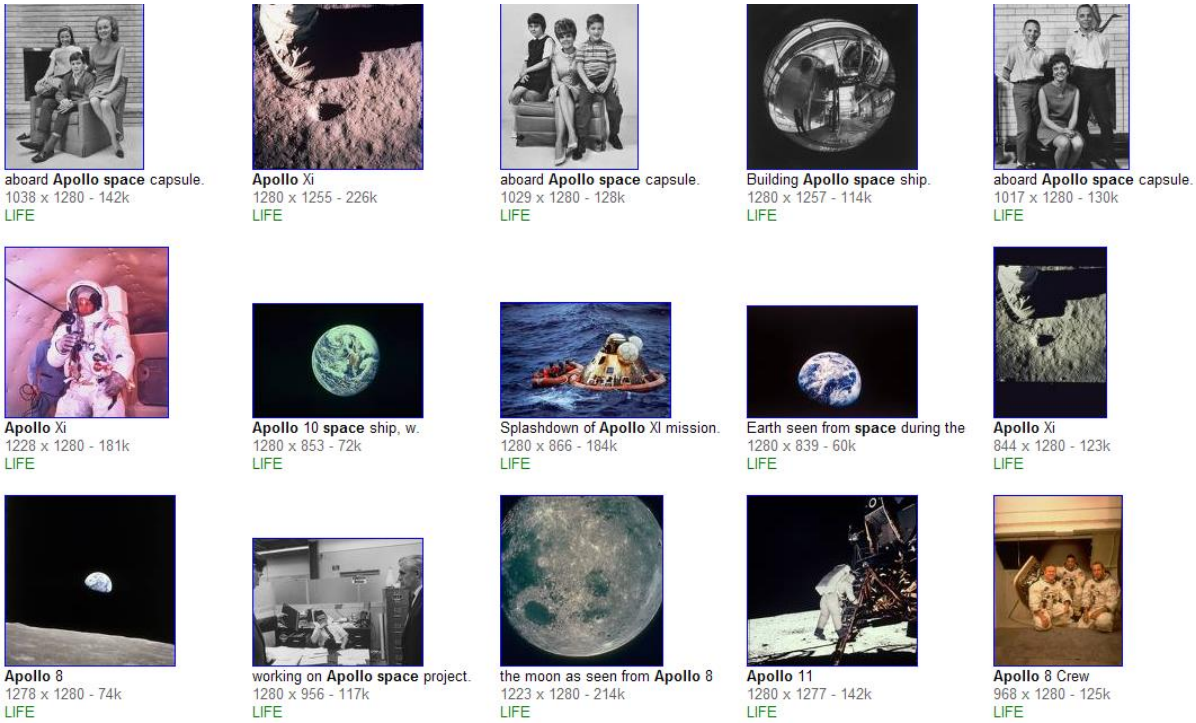
Linear Algebra recall

- What is $\mathbf{x}^T\mathbf{y}$
  - When $\mathbf{y}$ is unit length
- What is the projection of $\mathbf{x}$ onto $\mathbf{y}$
  - When $\mathbf{y}$ is unit length
- What is the projection of $\mathbf{x}$ onto $\mathbf{Y} = [\mathbf{y}_1\mathbf{y}_1 \ldots \mathbf{y}_K]$
  - When$\mathbf{Y}$ is an orthogonal matrix

- **On with the topic for today…**

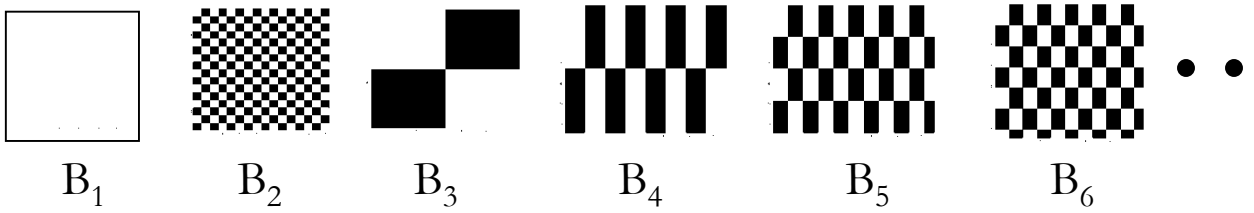
Recall: Representing images





- The most common element in the image: background
  - Or rather large regions of relatively featureless shading
  - Uniform sequences of numbers

Adding more bases



- Checkerboards with different variations

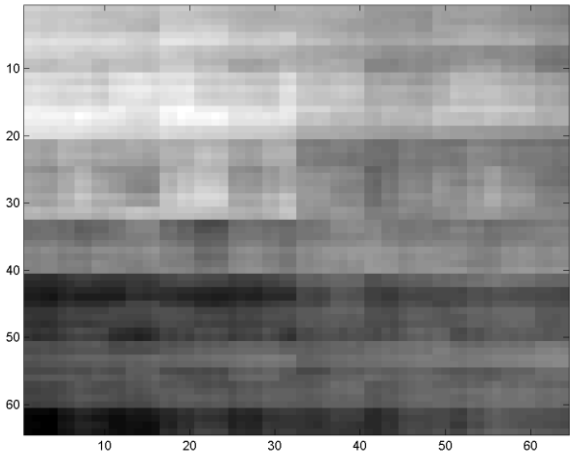
$$Image \approx w_1 B_1 + w_2 B_2 + w_3 B_3 + \ldots$$

$$W = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ . \\ . \end{bmatrix} \qquad B = [B_1 \;\; B_2 \;\; B_3]$$

$$BW \approx Image$$

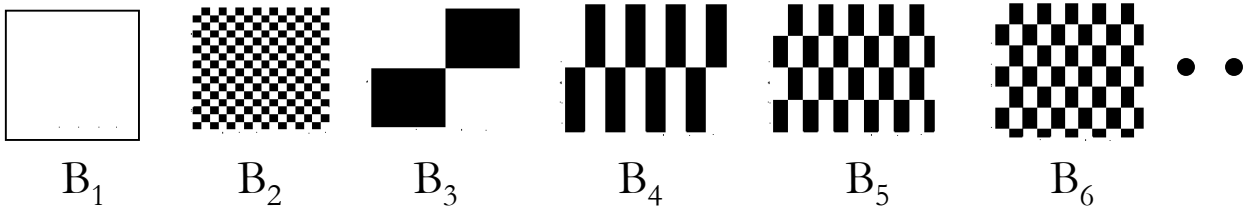
$$W = pinv(B)Image$$

$$PROJECTION = BW$$



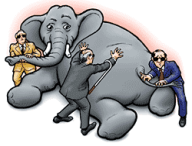
Getting closer at 625 bases!

“Bases”



$$image \approx w_1 B_1 + w_2 B_2 + w_3 B_3 + \ldots$$

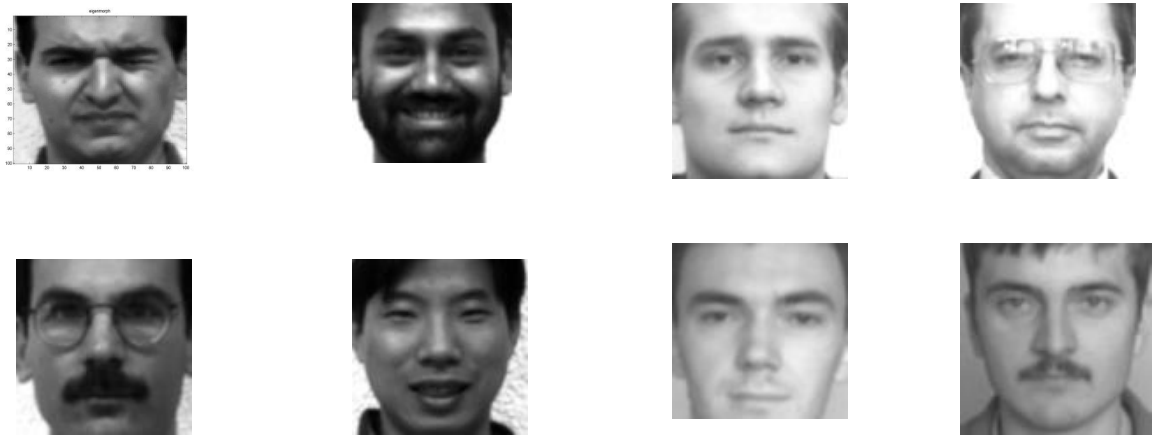
- “Bases” are the “standard” units such that all instances can be expressed a weighted combinations of these units
- Ideal requirements: Bases must be orthogonal
- Checkerboards are one choice of bases
  - Orthogonal
  - But not “smooth”
- Other choices of bases: Complex exponentials, Wavelets, etc..



Data specific bases?

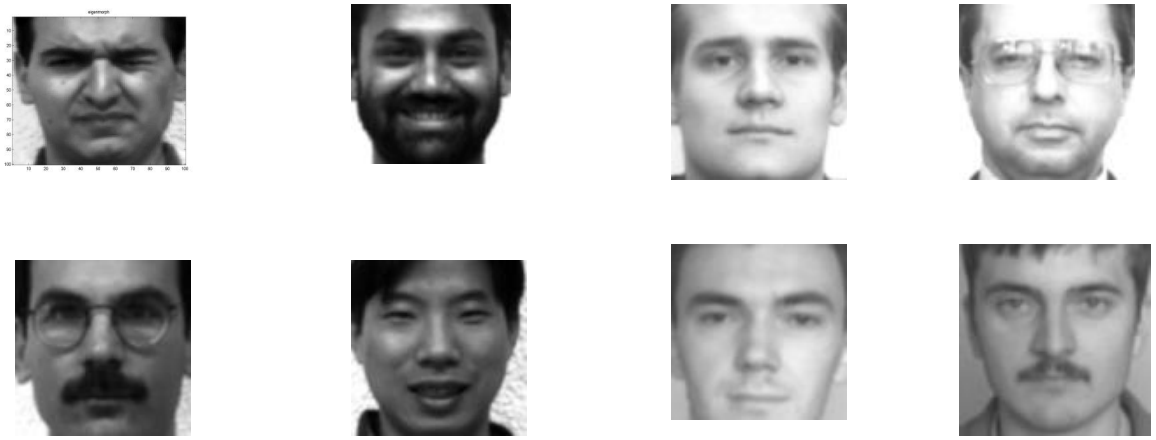
- **Issue**: **The bases we have considered so far are *data agnostic***
  - Checkerboards, Complex exponentials, Wavelets..
  - We use the same bases regardless of the data we analyze
    - Image of face vs. Image of a forest
    - Segment of speech vs. Seismic rumble
- How about data specific bases
  - Bases that consider the underlying data
    - E.g. is there something better than checkerboards to describe faces
    - Something better than complex exponentials to describe music?

Data-specific description of faces



- A collection of images
  - All normalized to 100x100 pixels
- What is common among all of them?
  - Do we have a common descriptor?

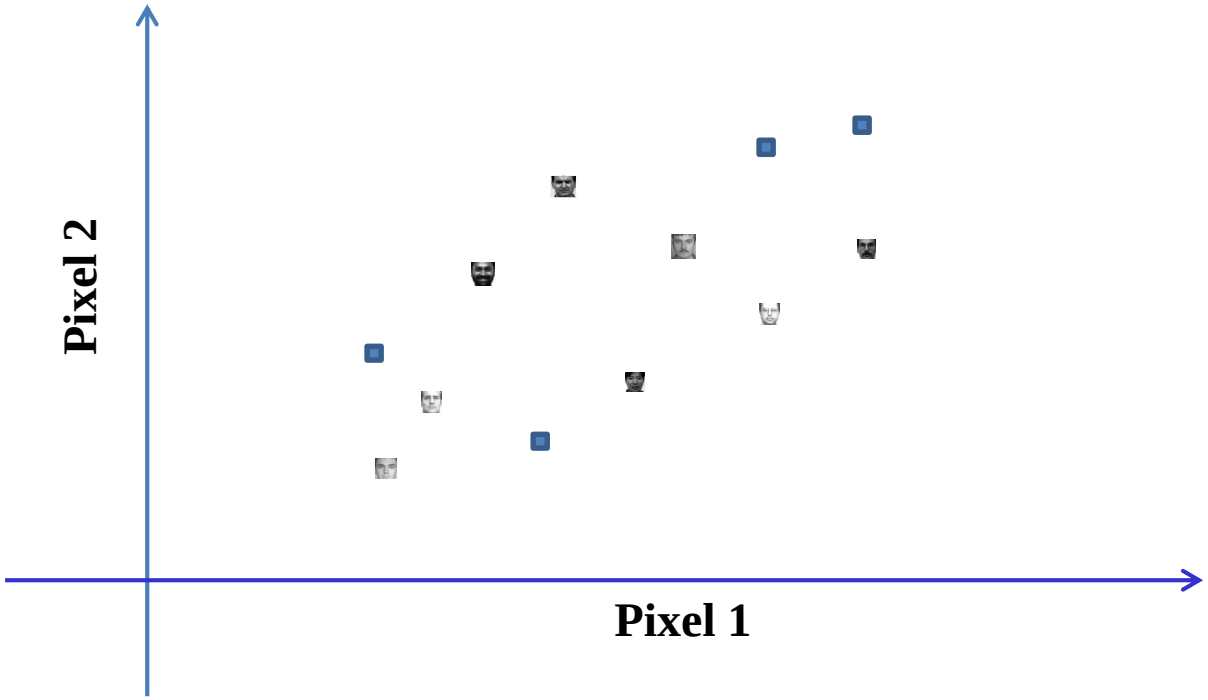
A typical face



The typical face

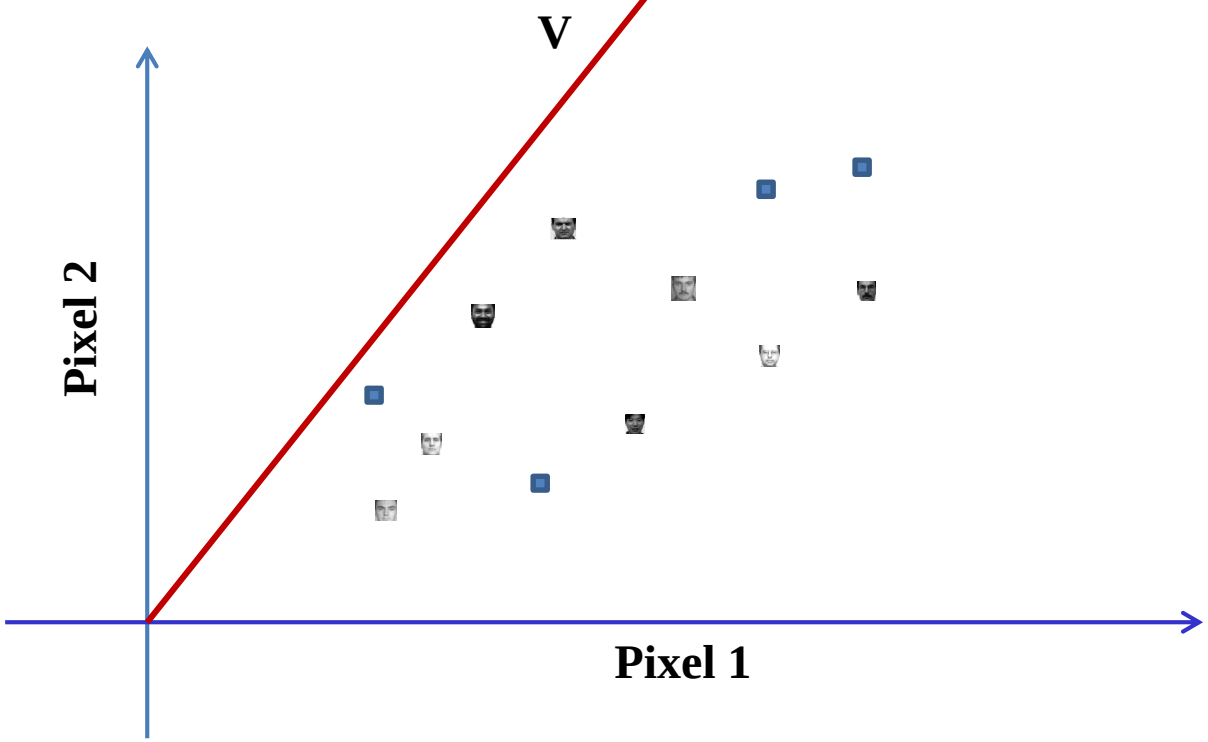
- **Assumption: There is a "typical" face that captures most of what is common to all faces**
  - Every face can be represented by a scaled version of a typical face
  - We will denote this face as V
- Approximate **every** face f as $\mathrm{f} = \mathrm{w}_\mathrm{f}\ \mathrm{V}$
- Estimate V to minimize the squared error
  - How? What is V?

Abstracting the problem:
Finding the typical face



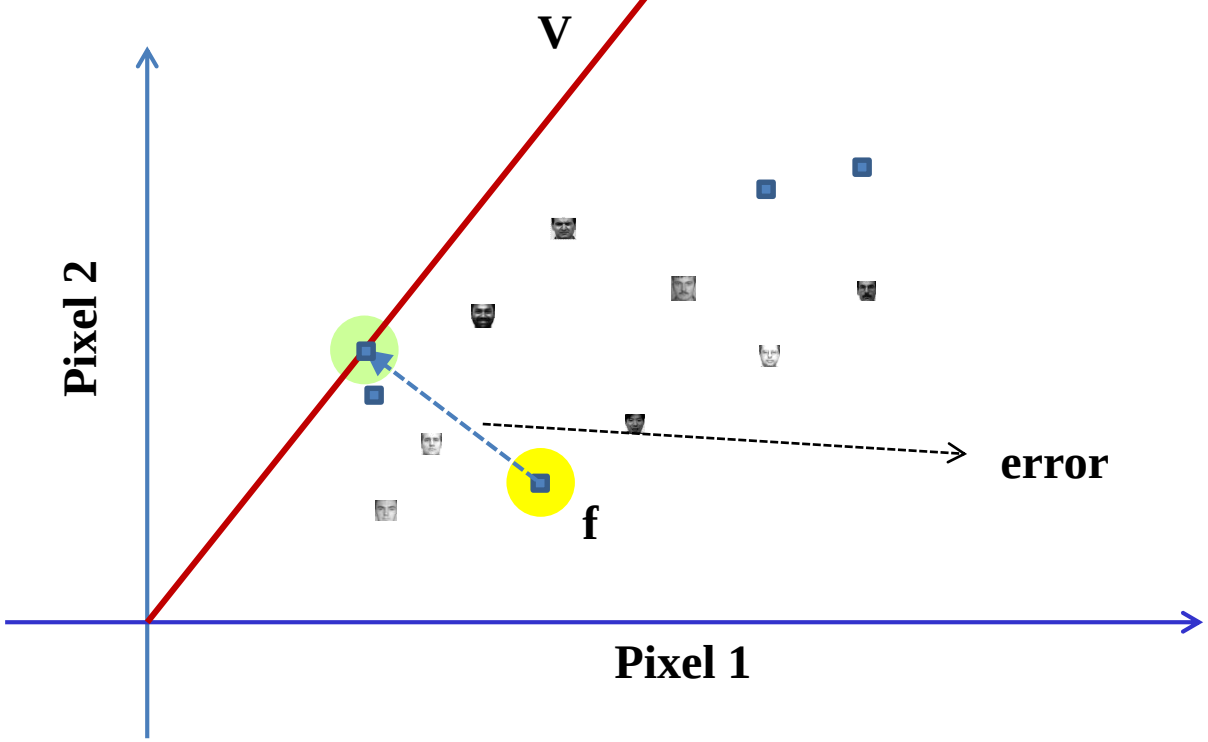
- Each “point” represents a face in “pixel space”

Abstracting the problem: Finding the typical face



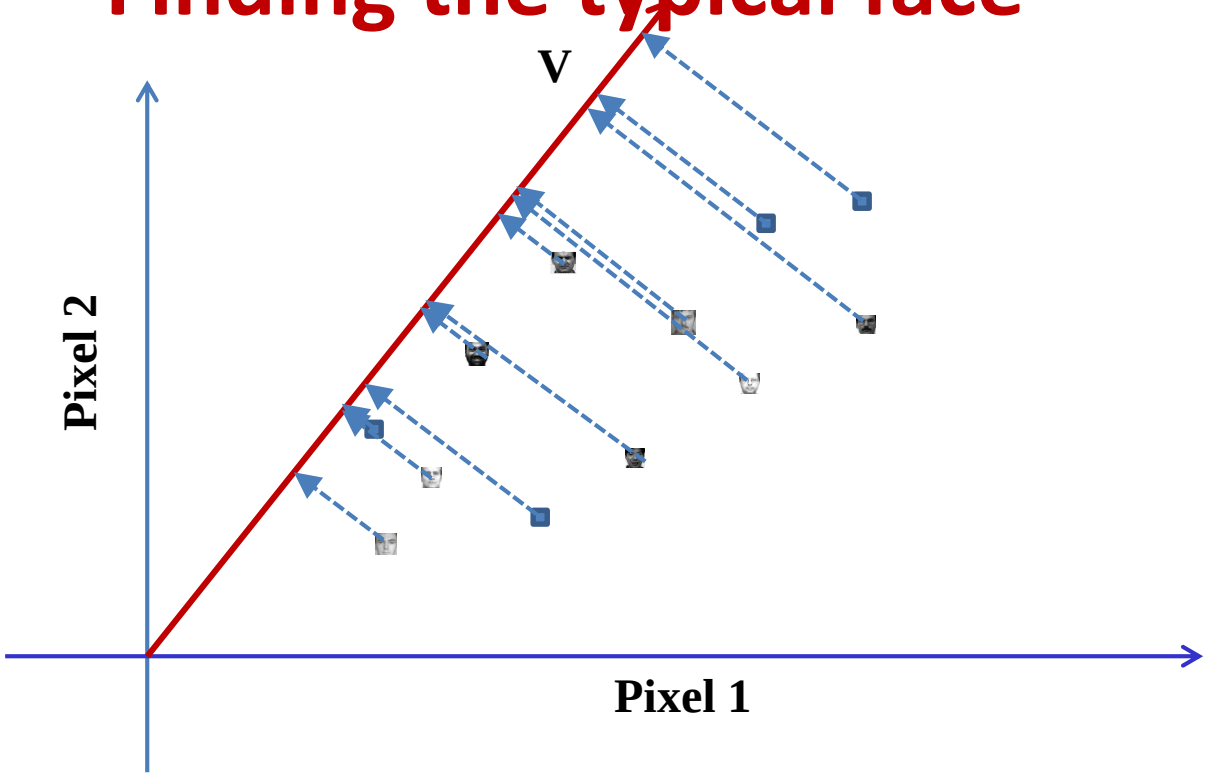
- Each “point” represents a face in “pixel space”
- Any “typical face” V is a vector in this space

Abstracting the problem: Finding the typical face



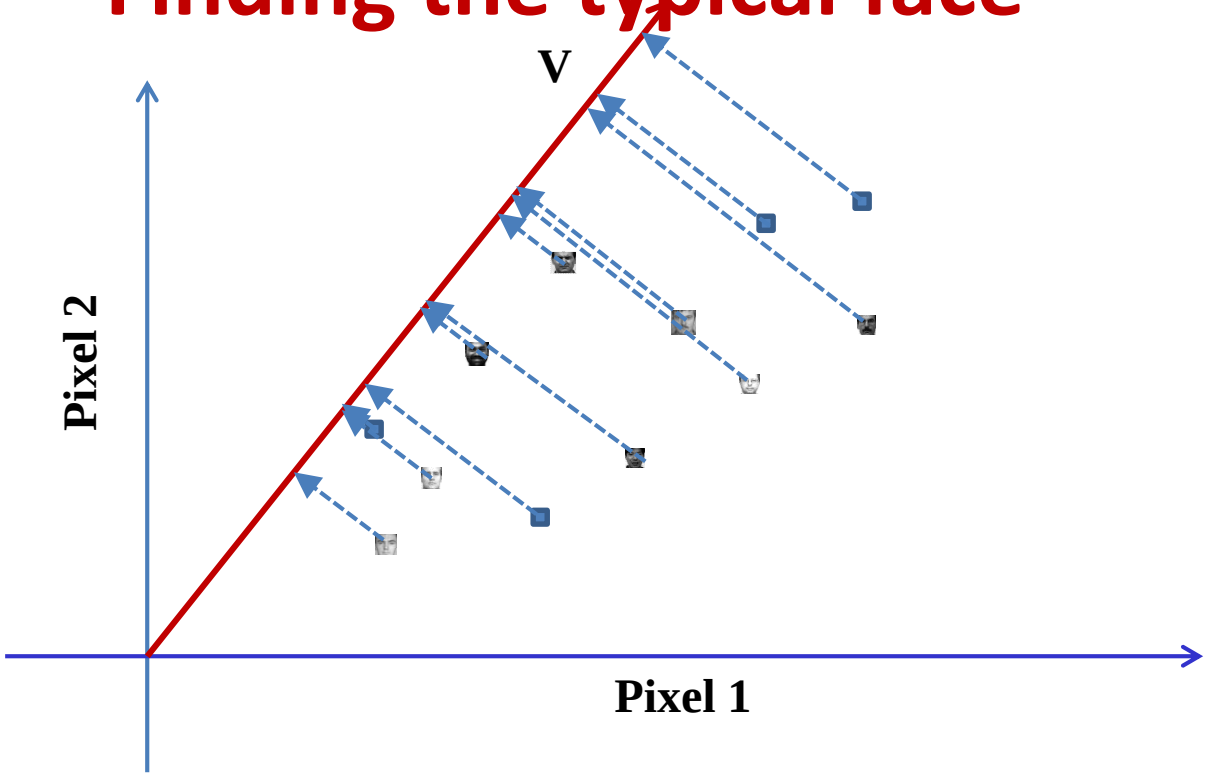
- Each "point" represents a face in "pixel space"
- The "typical face" V is a vector in this space
- The ***approximation*** $\mathrm{w_{f,}}\,\mathrm{V}$ for any face f is the *projection* of f onto V
- The distance between f and its projection $\mathrm{w_f V}$ is the *projection error* for f

Abstracting the problem: Finding the typical face



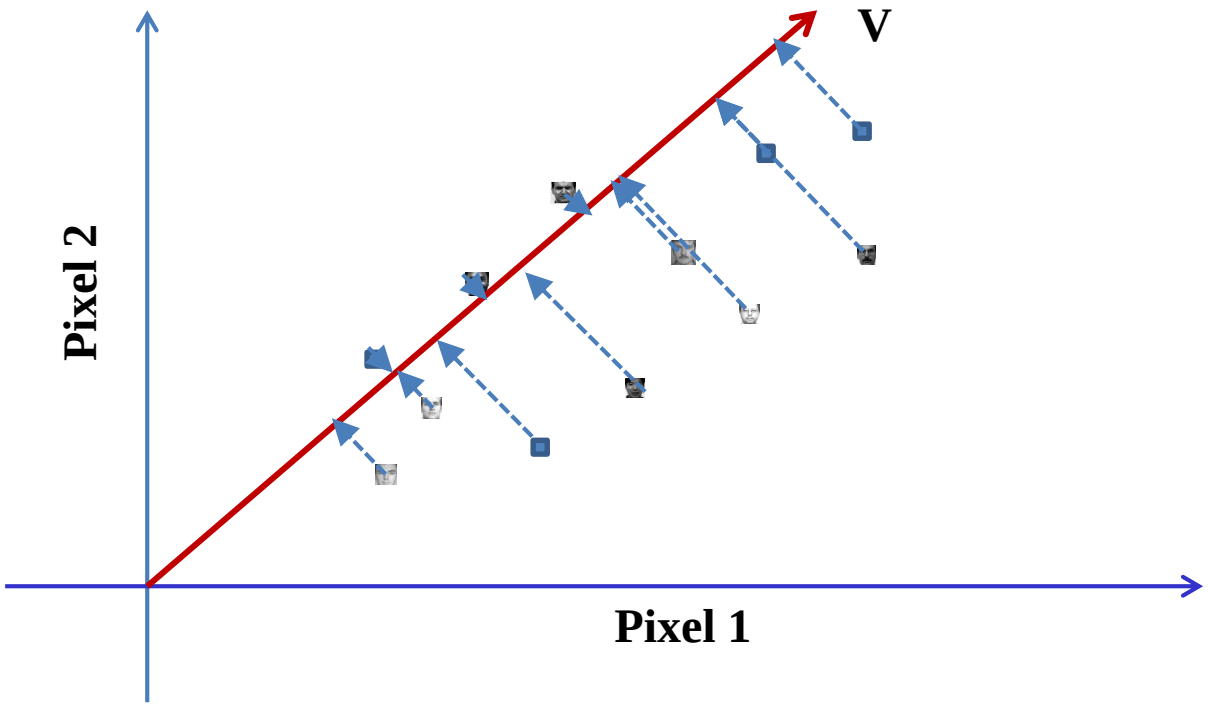
- *Every* face in our data will suffer error when approximated by its projection on V
- The total squared length of all error lines is the *total squared projection error*

Abstracting the problem: Finding the typical face



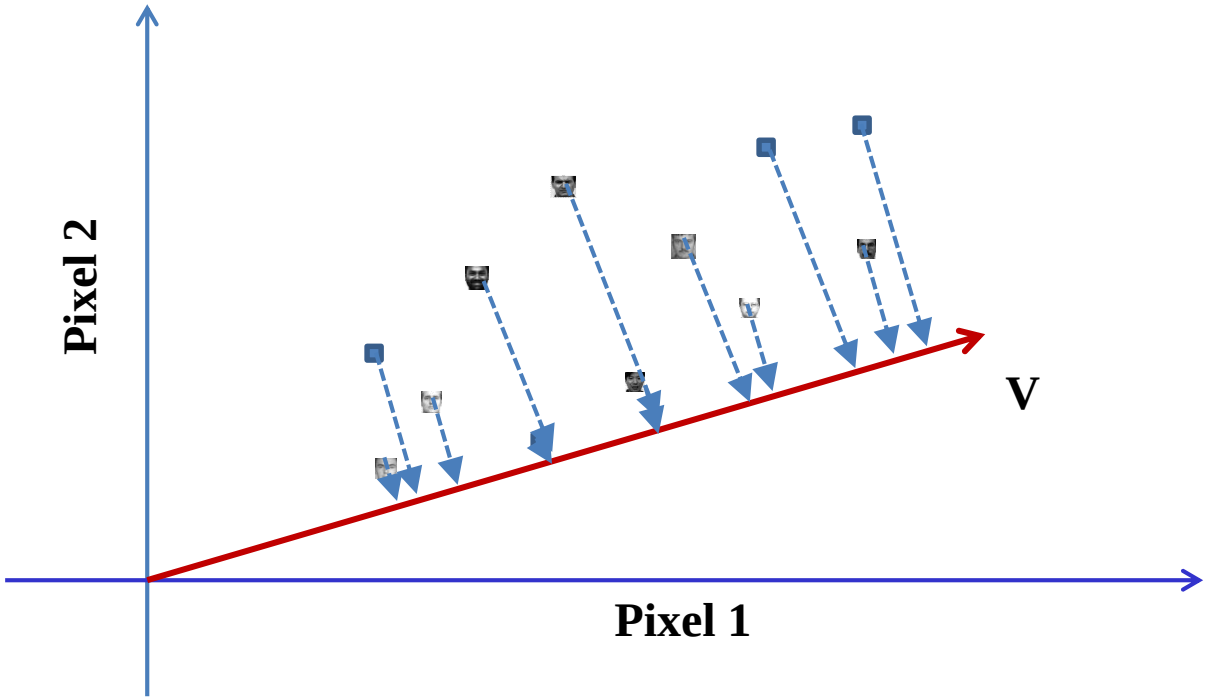
- The problem of finding the first typical face V_1:
  Find the V for which the total projection error is minimum!

Abstracting the problem: Finding the typical face



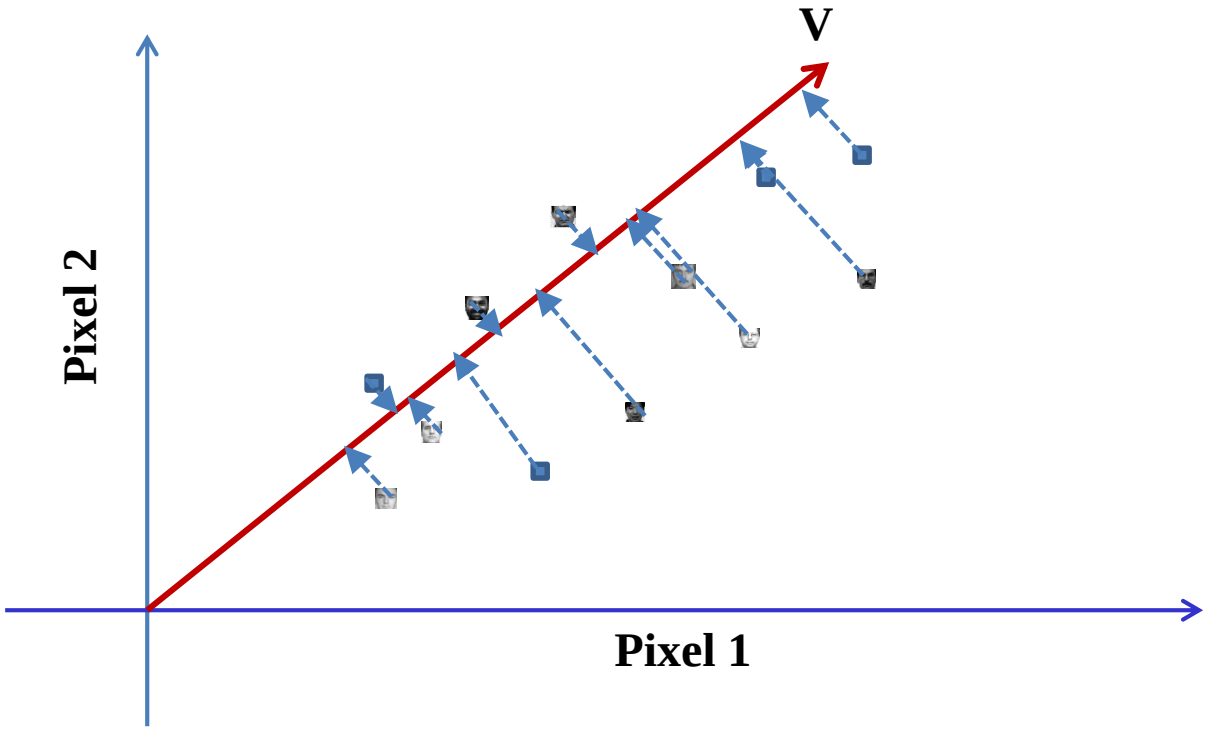
- The problem of finding the first typical face V_1:
  Find the V for which the total projection error is minimum!

Abstracting the problem: Finding the typical face



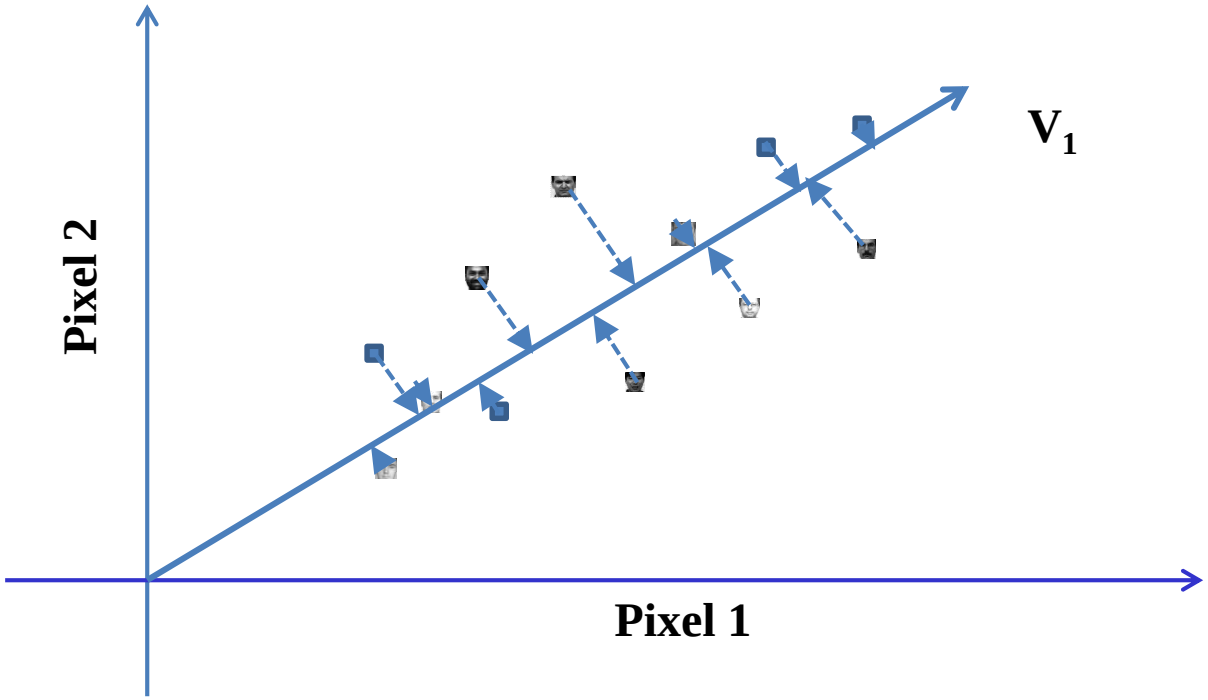
- The problem of finding the first typical face V_1:
  Find the V for which the total projection error is minimum!

Abstracting the problem: Finding the typical face



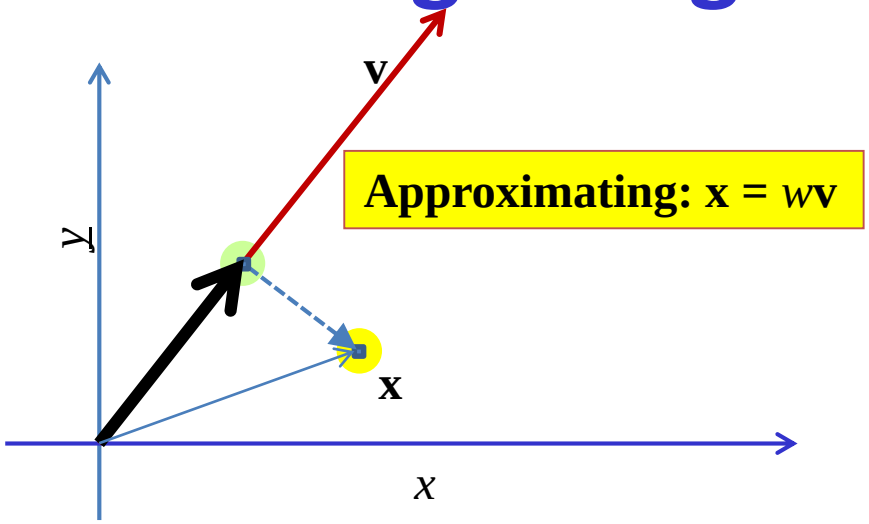
- The problem of finding the first typical face V_1:
  Find the V for which the total projection error is minimum!

Abstracting the problem: Finding the typical face



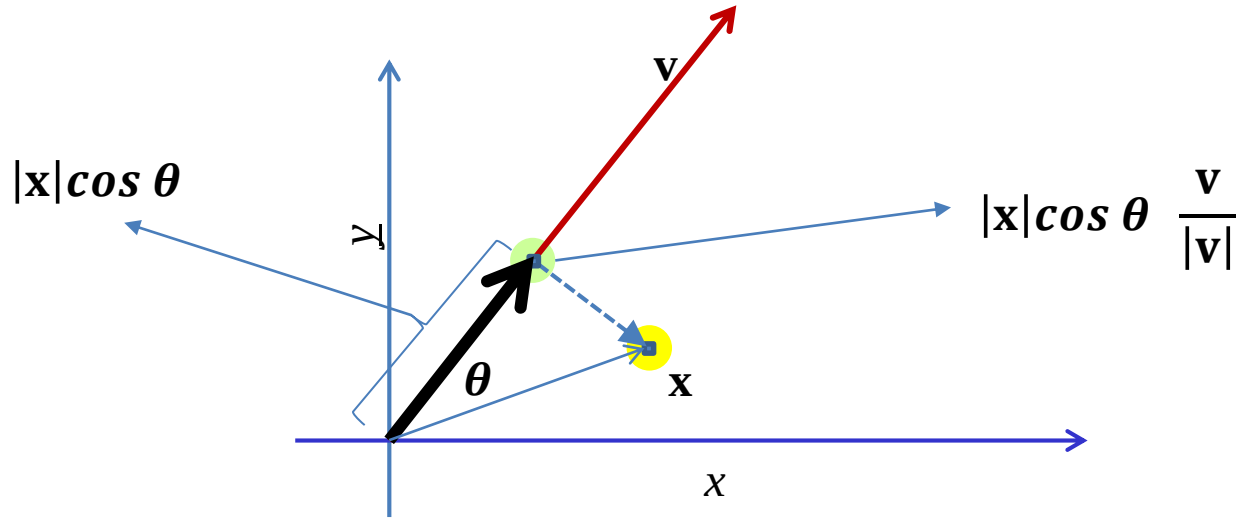
- The problem of finding the first typical face V_1:
  Find the V for which the total projection error is minimum!
- This "minimum squared error" V is our "best" first typical face
- **It is also the first *Eigen face***

Formalizing the Problem: Error from approximating a single vector



- Consider: approximating $\mathbf{x} = w\mathbf{v}$
  - E.g **x** is a face, and "**v**" is the "typical face"
- Finding an approximation $w\mathbf{v}$ which is closest to **x**
  - In a Euclidean sense
  - Basically projecting **x** onto **v**

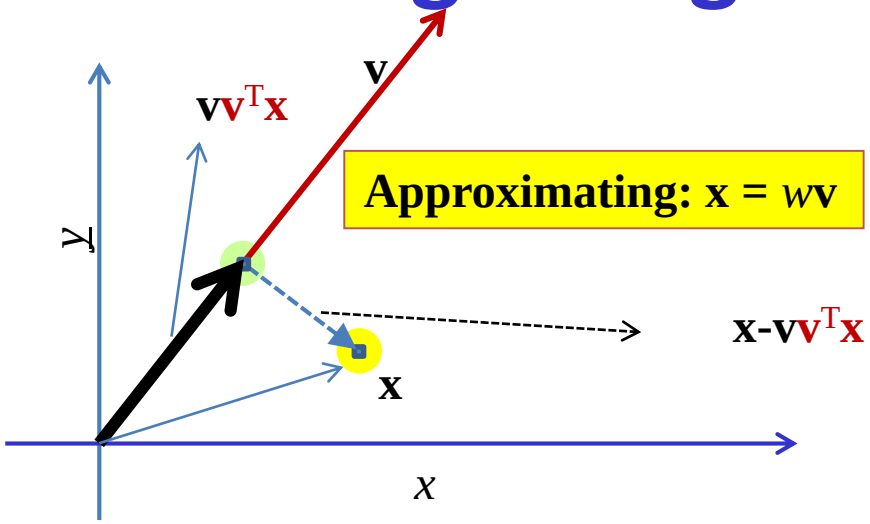
Projection of a vector on another



- The black arrow is the *projection* of $\mathbf{x}$ on $\mathbf{v}$
- $\frac{\mathbf{v}}{|\mathbf{v}|}$ is a *unit* vector in the direction of v
- $\mathbf{x}_{proj} = |\mathbf{x}|\cos\theta \frac{\mathbf{v}}{|\mathbf{v}|} = |\mathbf{x}||\mathbf{v}|\cos\theta \frac{\mathbf{v}}{|\mathbf{v}|^2}$

$$= \mathbf{x}^T\mathbf{v}\frac{\mathbf{v}}{|\mathbf{v}|^2}$$

Formalizing the Problem: Error from approximating a single vector



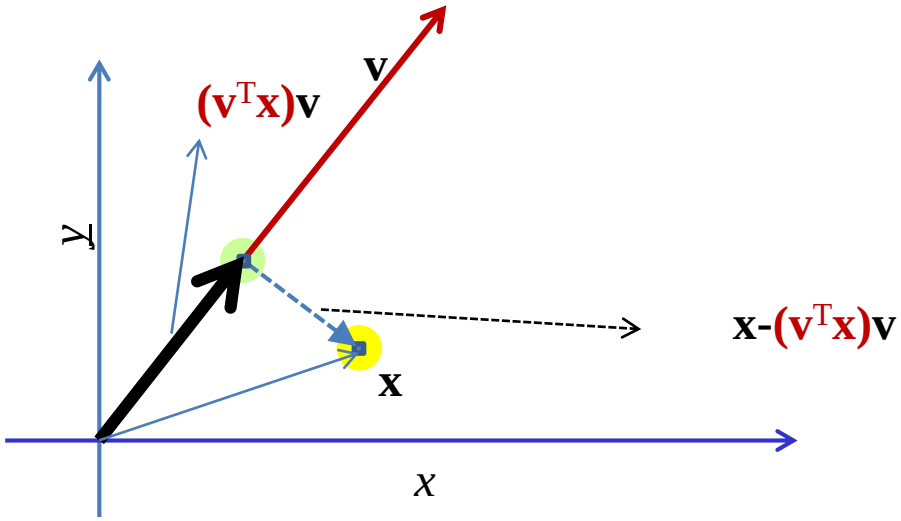
- Projection of a vector $\mathbf{x}$ on to a vector $\mathbf{v}$

$$\hat{\mathbf{x}} = \frac{\mathbf{x}^T\mathbf{v}}{|\mathbf{v}|^2}\mathbf{v}$$

- Assuming $\mathbf{v}$ is of unit length: $\hat{\mathbf{x}} = (\mathbf{x}^T\mathbf{v})\mathbf{v}$

$$error = \mathbf{x} - \hat{\mathbf{x}} = \mathbf{x} - (\mathbf{x}^T\mathbf{v})\mathbf{v} \qquad \text{squared error} = \left\|\mathbf{x} - (\mathbf{x}^T\mathbf{v})\mathbf{v}\right\|^2$$

Error from approximating a single vector



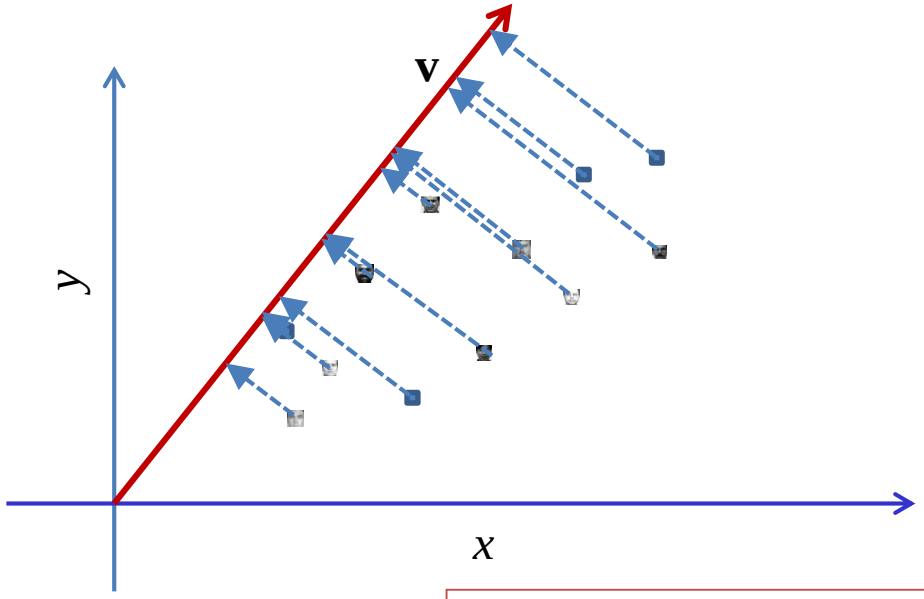
- Projection $\hat{\mathbf{x}} = (\mathbf{x}^T\mathbf{v})\mathbf{v}$
- Squared length of projection

$$\|\hat{\mathbf{x}}\|^2 = (\mathbf{x}^T\mathbf{v})^2 = (\mathbf{x}^T\mathbf{v})^T(\mathbf{x}^T\mathbf{v}) = \mathbf{v}^T\mathbf{x}\mathbf{x}^T\mathbf{v}$$

- Pythogoras theorem: Squared length of error $e(\mathbf{x}) = \|\mathbf{x}\|^2 - \|\hat{\mathbf{x}}\|^2$

$$e(\mathbf{x}) = \mathbf{x}^T\mathbf{x} - \mathbf{v}^T\mathbf{x}\mathbf{x}^T\mathbf{v}$$

Error for *many* vectors



- Error for one vector: $e(\mathbf{x}) = \mathbf{x}^T\mathbf{x} - \mathbf{v}^T\mathbf{x}\mathbf{x}^T\mathbf{v}$
- Error for many vectors

$$E = \sum_i e(\mathbf{x}_i) = \sum_i \mathbf{x}_i^T\mathbf{x}_i - \sum_i \mathbf{v}^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{v} = \sum_i \mathbf{x}_i^T\mathbf{x}_i - \mathbf{v}^T\left(\sum_i \mathbf{x}_i\mathbf{x}_i^T\right)\mathbf{v}$$

- **Goal: Estimate v to minimize this error!**

Definition: The correlation matrix

$$\left(\sum_i \mathbf{x}_i \mathbf{x}_i^T \right)$$

- The encircled term is the ***correlation matrix***

$$\mathbf{X} = [\mathbf{x}_1 \ \mathbf{x}_2 \ \ldots \mathbf{x}_N]$$

$$\sum_i \mathbf{x}_i \mathbf{x}_i^T = \mathbf{X}\mathbf{X}^T = \mathbf{R}$$

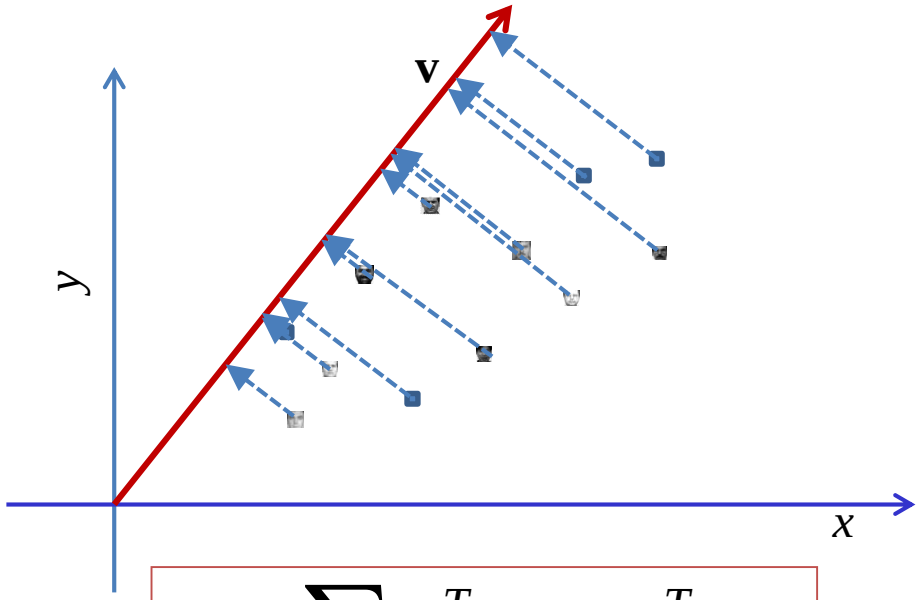
$\mathbf{X}$ = Data Matrix

$\mathbf{X}^T$ = Transposed Data Matrix

=

Correlation

Error for *many* vectors



- Total error: $E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \mathbf{v}^T \mathbf{R} \mathbf{v}$
- Add constraint: $\mathbf{v}^T\mathbf{v} = 1$
- Constrained objective to minimize:

$$L = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \mathbf{v}^T \mathbf{R} \mathbf{v} + \lambda\left(\mathbf{v}^T \mathbf{v} - 1\right)$$

Two Matrix Identities

- Derivative w.r.t $\mathbf{v}$

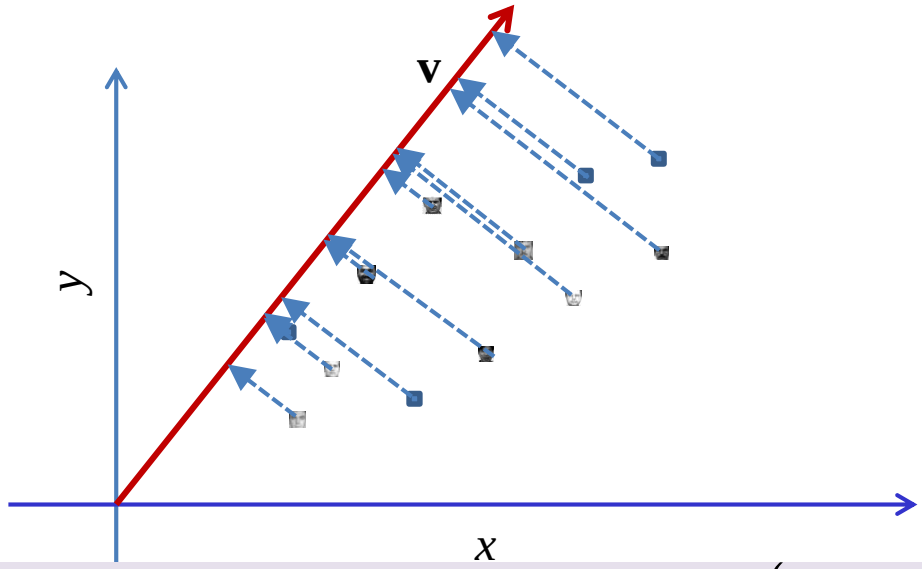
$$L = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \mathbf{v}^T \mathbf{R} \mathbf{v} + \lambda \left(\mathbf{v}^T \mathbf{v} - 1 \right)$$

$$\nabla_{\mathbf{v}} \left(\mathbf{v}^T \mathbf{v} \right) = 2\mathbf{v}$$

$$\nabla_{\mathbf{v}} \mathbf{v}^T \mathbf{R} \mathbf{v} = 2\mathbf{R}\mathbf{v}$$

These are actually *transposed* derivatives, but that does not affect the overall result that follows

Minimizing error



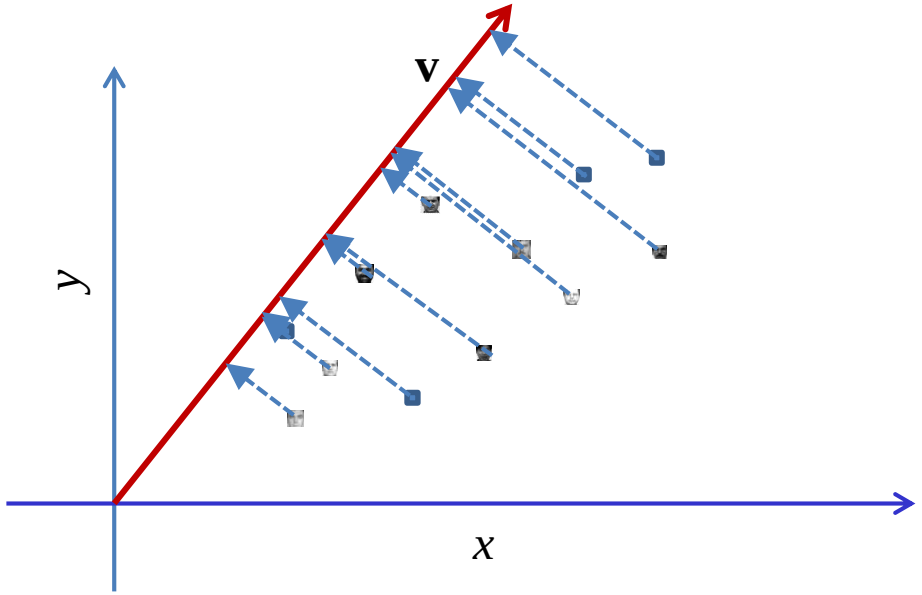
$$L = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \mathbf{v}^T \mathbf{R}\mathbf{v} + \lambda\left(\mathbf{v}^T \mathbf{v} - 1\right)$$

- Differentiating w.r.t $\mathbf{v}$ and equating to 0

$$-2\mathbf{R}\mathbf{v} + 2\lambda\mathbf{v} = 0$$

$$\mathbf{R}\mathbf{v} = \lambda\mathbf{v}$$

The best "basis"



- The minimum-error basis is found by solving

$$\mathbf{R}\mathbf{v} = \lambda\mathbf{v}$$

- $\mathbf{v}$ is an Eigen vector of the correlation matrix $\mathbf{R}$
  - λ is the corresponding Eigen value

What about the total error?

$$E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \mathbf{v}^T \mathbf{R} \mathbf{v}$$

$$= \sum_i \mathbf{x}_i^T \mathbf{x}_i - \mathbf{v}^T \lambda \mathbf{v} = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \lambda \mathbf{v}^T \mathbf{v}$$

$$E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \lambda$$

Minimizing the error

- The total error is $E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \lambda$

- We already know that the optimal basis is an Eigen vector
- The total error depends on the *negative* of the corresponding Eigen value
- To *minimize* error, we must *maximize* λ
- i.e. Select the Eigen vector with the largest Eigen value

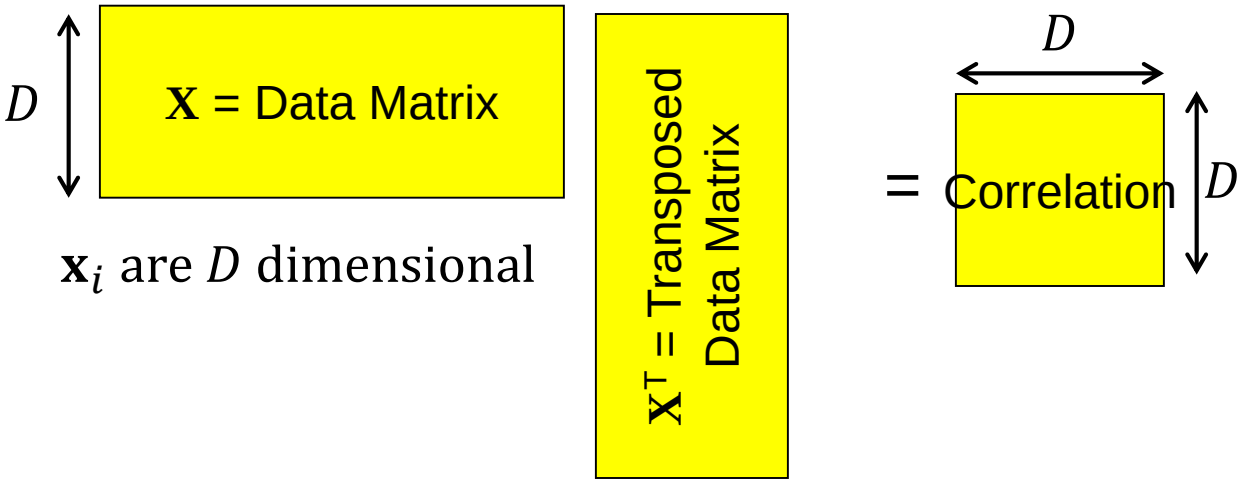
Poll 1

A detour: The correlation matrix

A new definition:
The *correlation* matrix

$$\mathbf{X} = \left[\mathbf{x}_1\ \mathbf{x}_2\ \ldots\ \mathbf{x}_N\right]$$

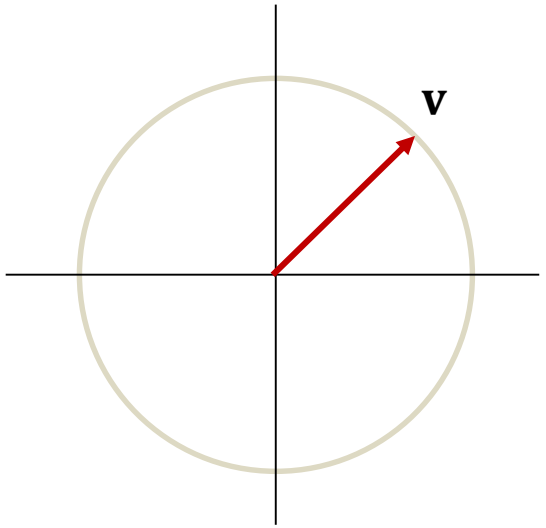
$$\sum_i \mathbf{x}_i \mathbf{x}_i^T = \mathbf{X}\mathbf{X}^T = \mathbf{R}$$



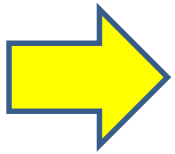
$$\mathbf{R} = \mathbf{R}^T$$

- For data-holder matrices: the product of a matrix and its transpose
  - Also equal to the sum of the outer products of the columns of the matrix
  - **The correlation matrix is symmetric**
  - **It quantifies the average dependence of individual *components* of the data on other components**

Interpreting the correlation matrix



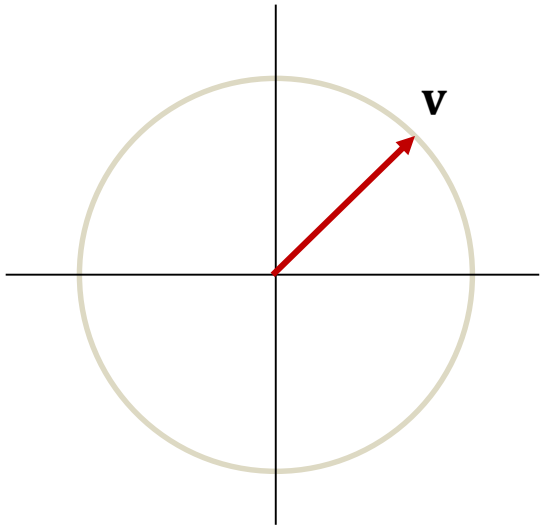
$\mathbf{y} = \mathbf{R}\mathbf{v}$



?

- Consider the effect of multiplying a unit vector by $\mathbf{R}$

Interpreting the correlation matrix



$$\mathbf{y} = \mathbf{R}\mathbf{v}$$

$$\mathbf{R}\mathbf{v} = \mathbf{X}\mathbf{X}^T\mathbf{v}$$

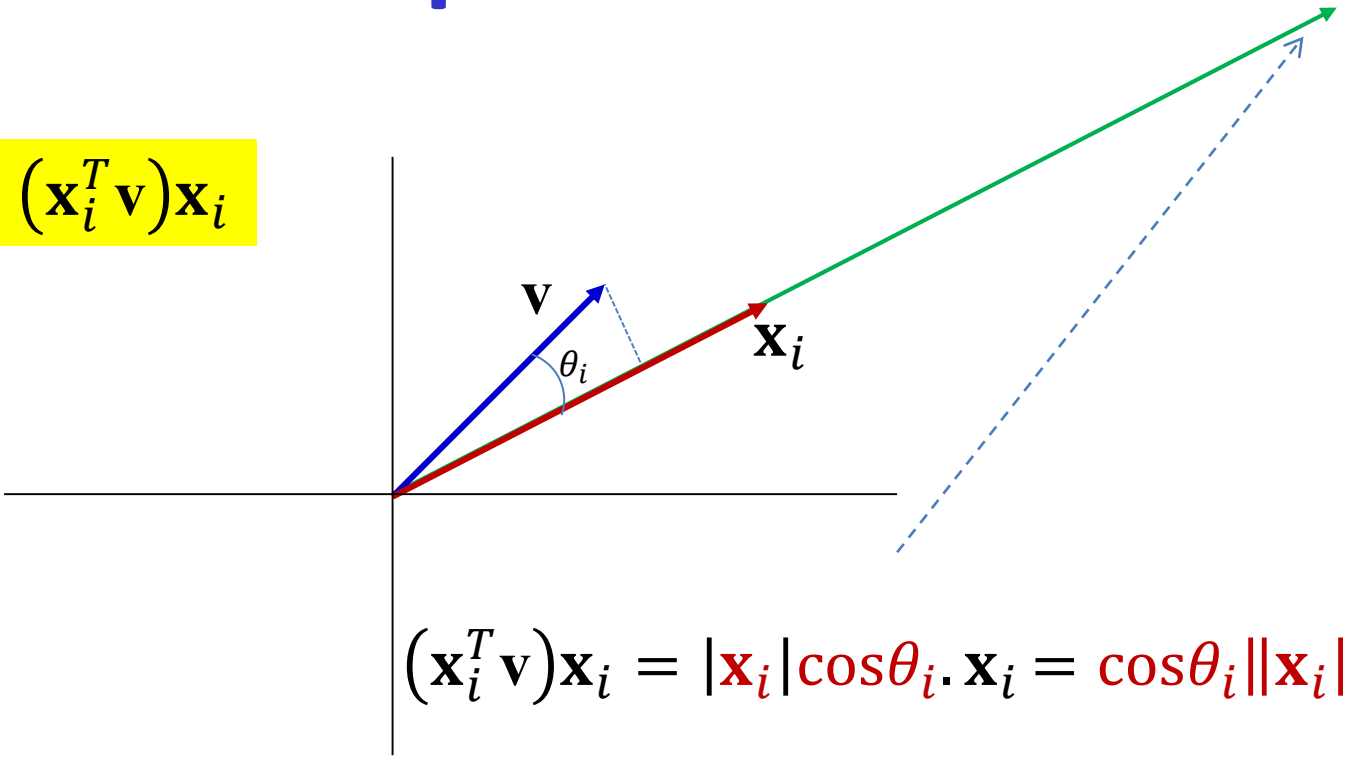
$$\mathbf{R}\mathbf{v} = \sum_i \mathbf{x}_i \mathbf{x}_i^T \mathbf{v}$$

$$\mathbf{R}\mathbf{v} = \sum_i (\mathbf{x}_i^T \mathbf{v})\mathbf{x}_i$$

- Consider the effect of multiplying a unit vector by $\mathbf{R}$

The inner product term

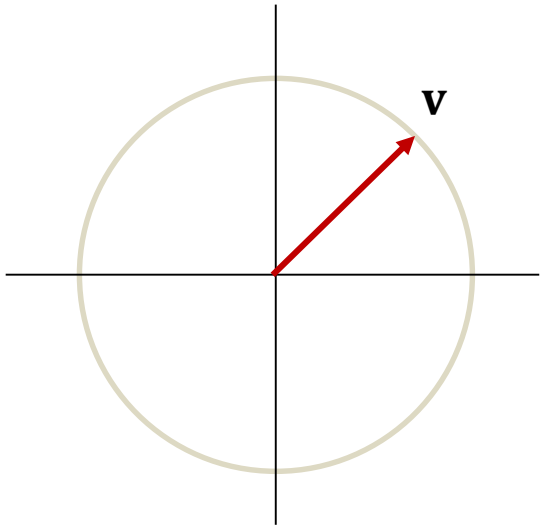
Understanding $\left(\mathbf{x}_i^T\mathbf{v}\right)\mathbf{x}_i$



$$\left(\mathbf{x}_i^T\mathbf{v}\right)\mathbf{x}_i = |\mathbf{x}_i|\cos\theta_i.\mathbf{x}_i = \cos\theta_i\|\mathbf{x}_i\|^2\bar{\mathbf{x}}_i$$

- Consider $\left(\mathbf{x}_i^T\mathbf{v}\right)\mathbf{x}_i$
- This is the projection of unit vector $\mathbf{v}$ on $\mathbf{x}_i$, scaled by the squared length of $\mathbf{x}_i$

Interpreting the correlation matrix



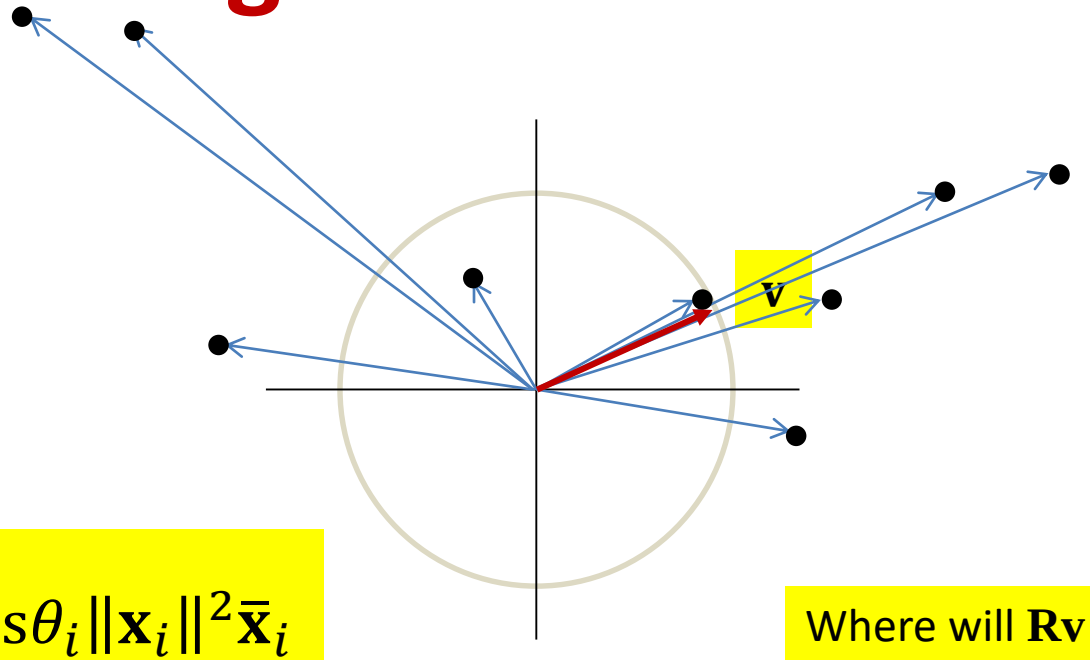
$$\mathbf{y} = \mathbf{R}\mathbf{v}$$

$$\mathbf{R}\mathbf{v} = \sum_i (\mathbf{x}_i^T \mathbf{v})\mathbf{x}_i$$

$$\mathbf{R}\mathbf{v} = \sum_i \cos\theta_i \|\mathbf{x}_i\|^2 \bar{\mathbf{x}}_i$$

- Consider the effect of multiplying a unit vector by $\mathbf{R}$

Interpreting the correlation matrix

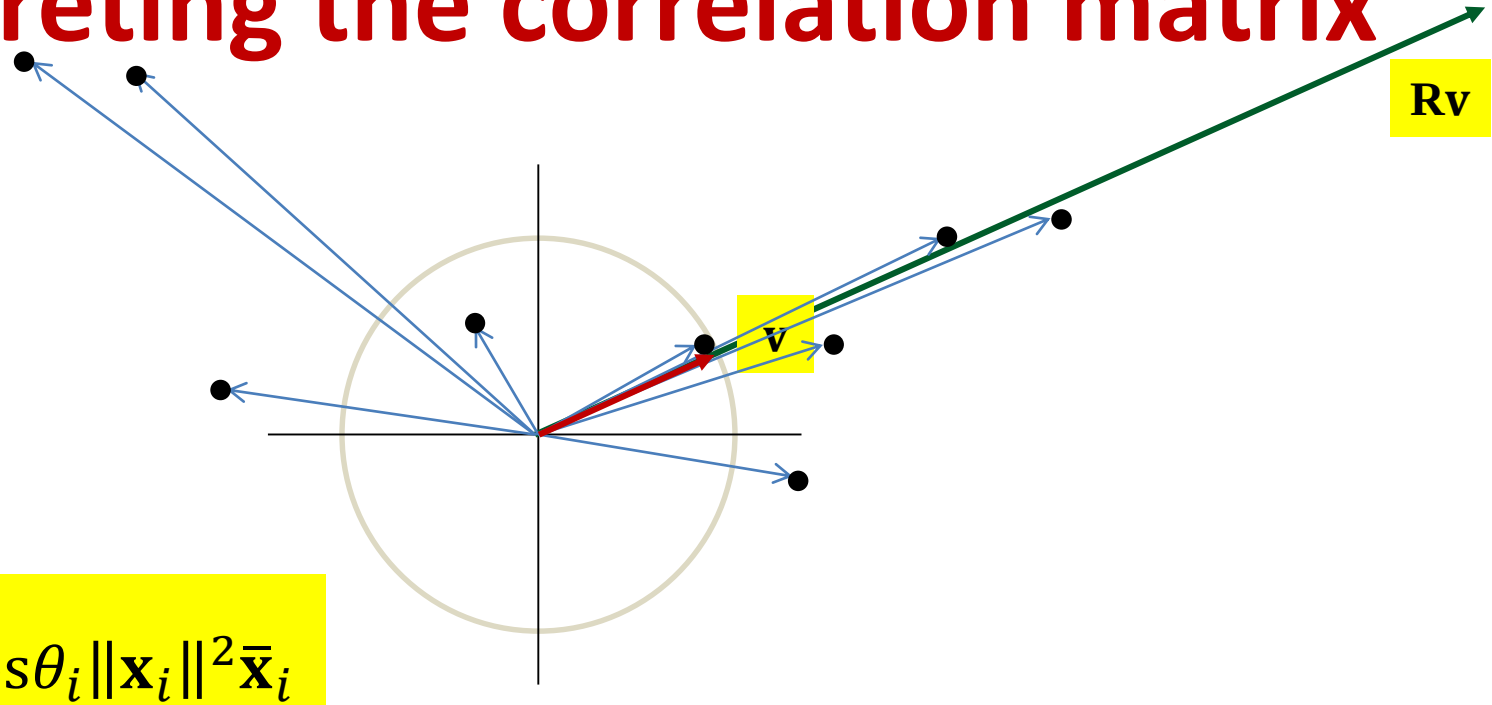


$$\mathbf{Rv} = \sum_{i} \cos\theta_i \|\mathbf{x}_i\|^2 \bar{\mathbf{x}}_i$$

Where will $\mathbf{Rv}$ be?

- Each unit vector is transformed to the sum of cosine-weighted squared-length versions of the individual vectors
  - Approximately the sum of the squared-length version of vectors that are close to it in angle

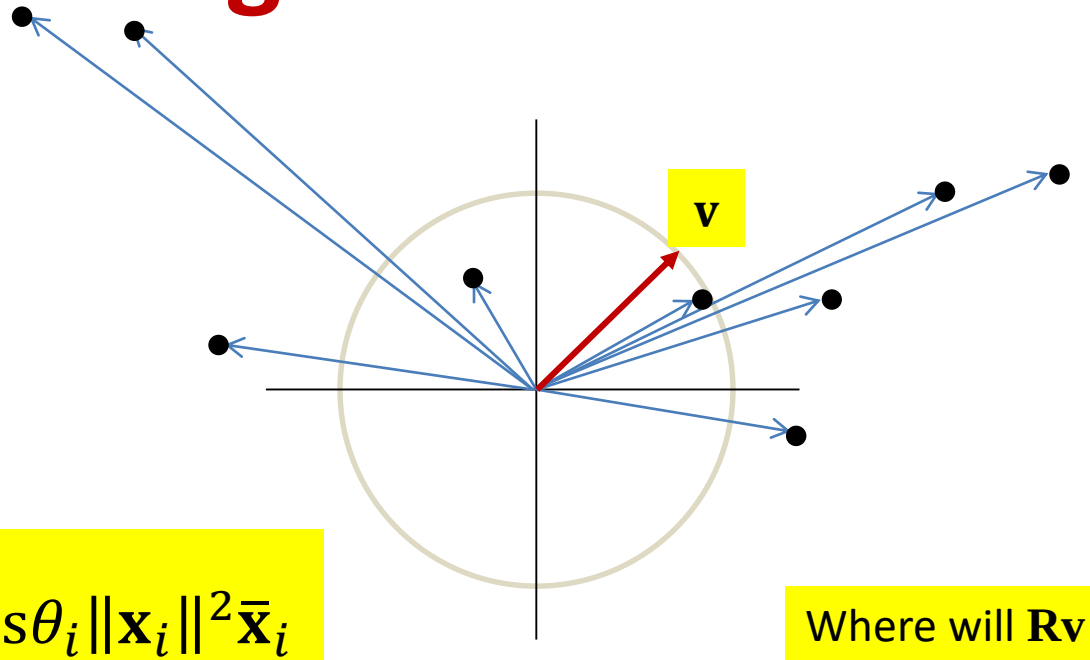
Interpreting the correlation matrix



$$\mathbf{Rv} = \sum_{i} \cos\theta_i \|\mathbf{x}_i\|^2 \bar{\mathbf{x}}_i$$

- Each unit vector is transformed to the sum of cosine-weighted squared-length versions of the individual vectors
  - Approximately the sum of the squared-length version of vectors that are close to it in angle

Interpreting the correlation matrix

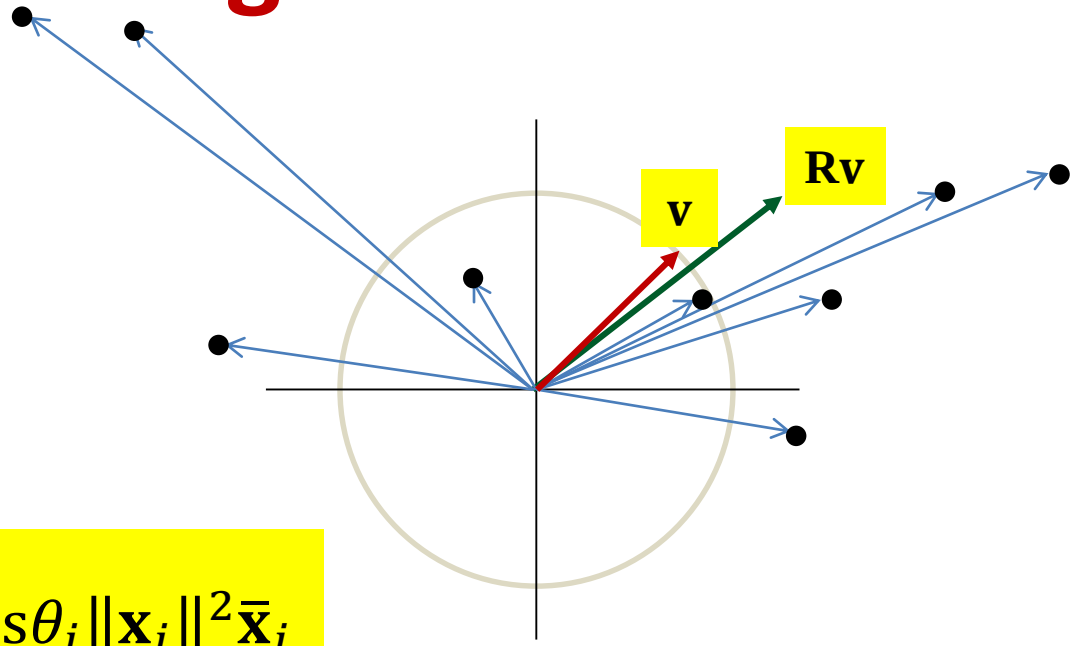


$$\mathbf{Rv} = \sum_{i} \cos\theta_i \|\mathbf{x}_i\|^2 \bar{\mathbf{x}}_i$$

Where will $\mathbf{Rv}$ be?

- Each unit vector is transformed to the sum of cosine-weighted squared-length versions of the individual vectors
  - Approximately the sum of the squared-length version of vectors that are close to it in angle

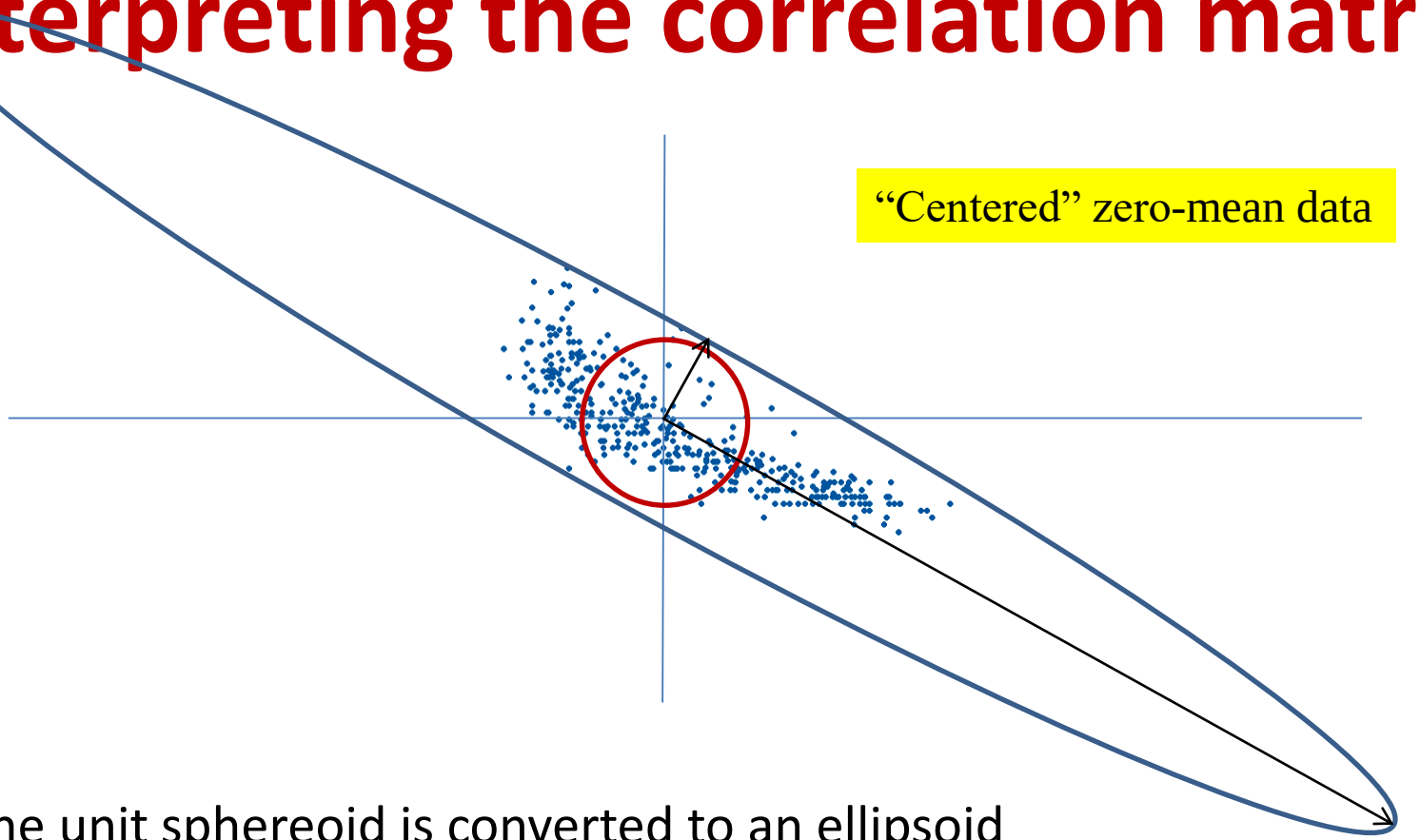
Interpreting the correlation matrix



$$\mathbf{Rv} = \sum_{i} \cos\theta_i \|\mathbf{x}_i\|^2 \bar{\mathbf{x}}_i$$

- Each unit vector is transformed to the sum of cosine-weighted squared-length versions of the individual vectors
  - Approximately the sum of the squared-length version of vectors that are close to it in angle

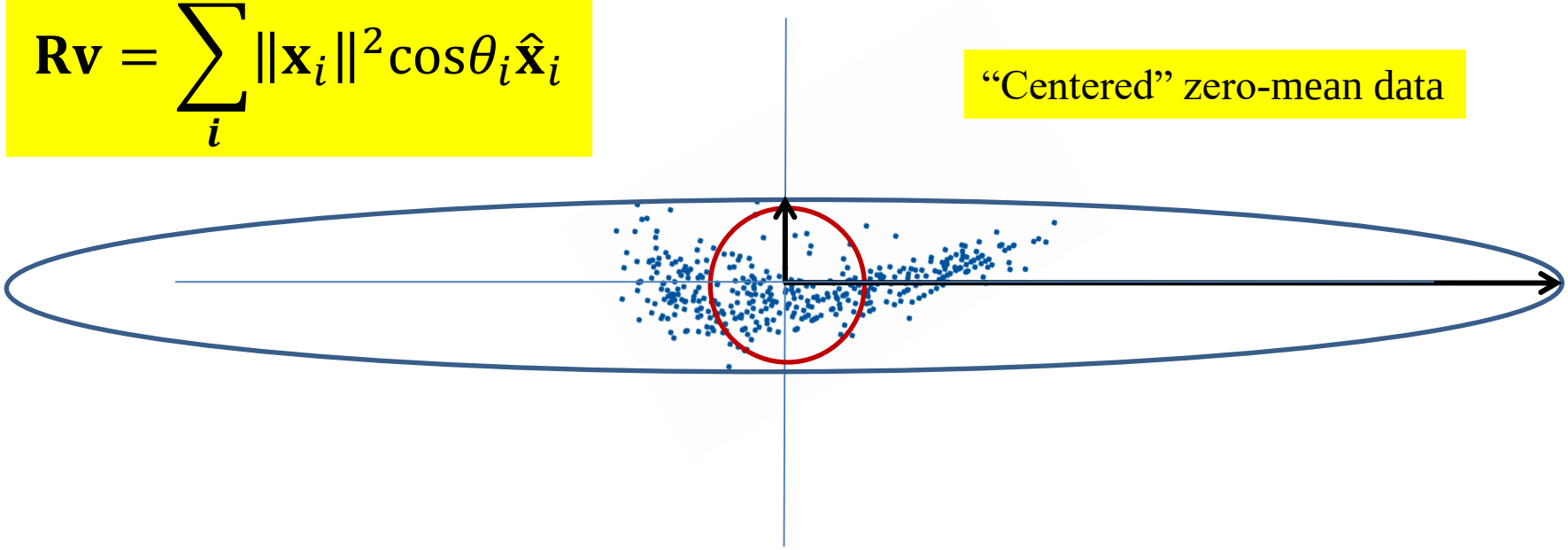
Interpreting the correlation matrix



- The unit sphereoid is converted to an ellipsoid
  - The major axes point to the directions of greatest energy
  - These are the *eigenvectors*
  - Their length is proportional to the *square* of the lengths of the data vectors
    - *Why?*

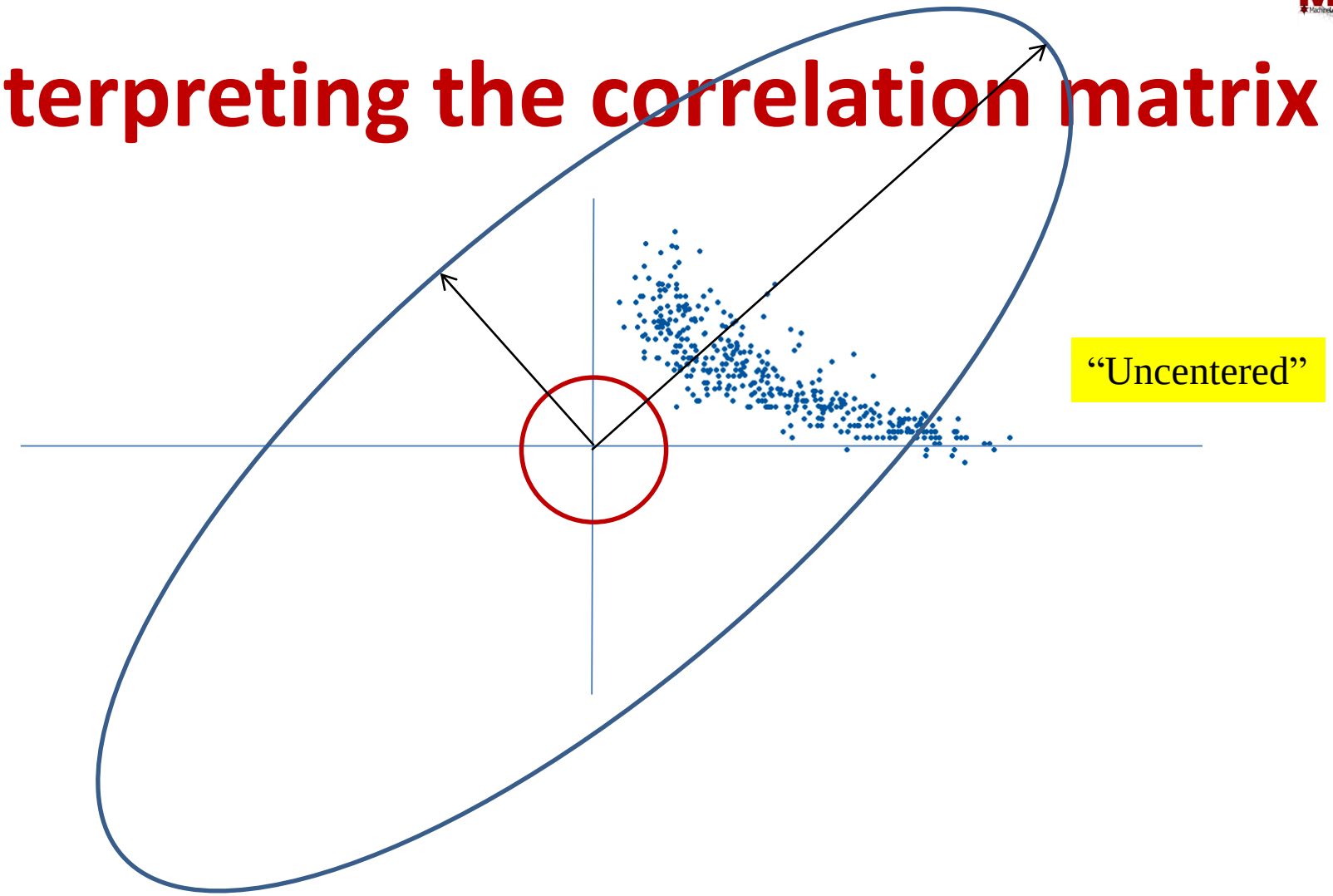
"Uncorrelated" data

$$\mathbf{Rv} = \sum_i \|\mathbf{x}_i\|^2 \cos\theta_i \hat{\mathbf{x}}_i$$



- When the scatter of the data is aligned to the axes, the transformed ellipse is also aligned to the axes
  - The data are "**uncorrelated**"

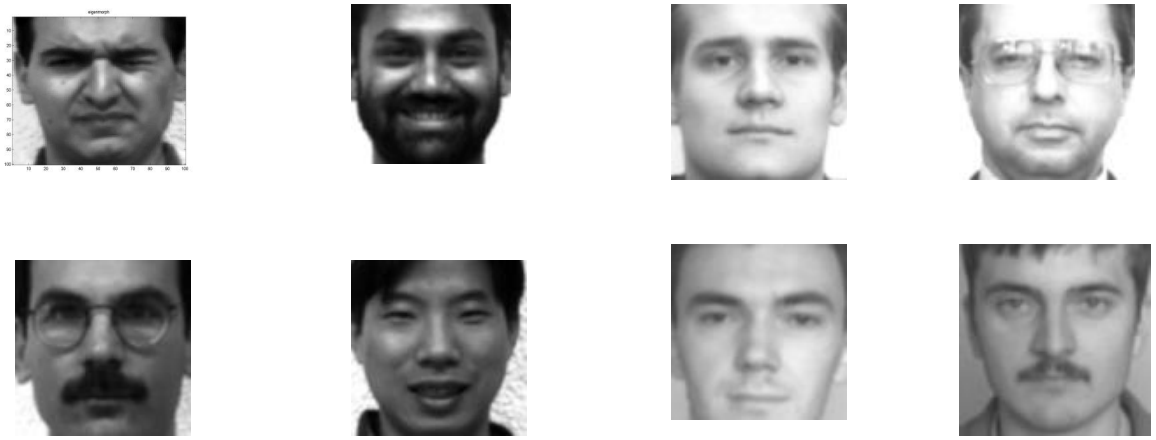
Interpreting the correlation matrix



- For "uncentered" data..
  - Note although the vectors near the major axis are shorter, there are more of them, so the ellipse is wider in that direction

Returning to our problem..

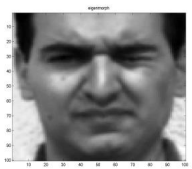
The typical face



The typical face

- Compute the correlation matrix for your data
  - Arrange them in matrix **X** and compute $\mathbf{R} = \mathbf{X}\mathbf{X}^T$
- Compute the *principal* Eigen vector of R
  - The Eigen vector with the largest Eigen value
  - Explains most of the "energy" in the faces
- This is the typical face

The *approximation* with the first typical face

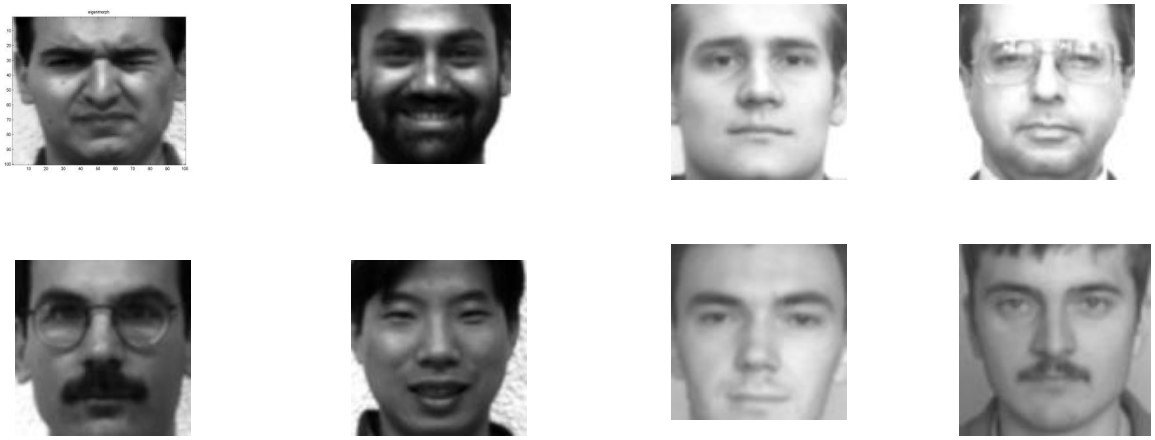







- The first typical face models some of the characteristics of the faces
  - Simply by scaling its grey level
- But the approximation has error
- Can we do better?

The *second* typical face



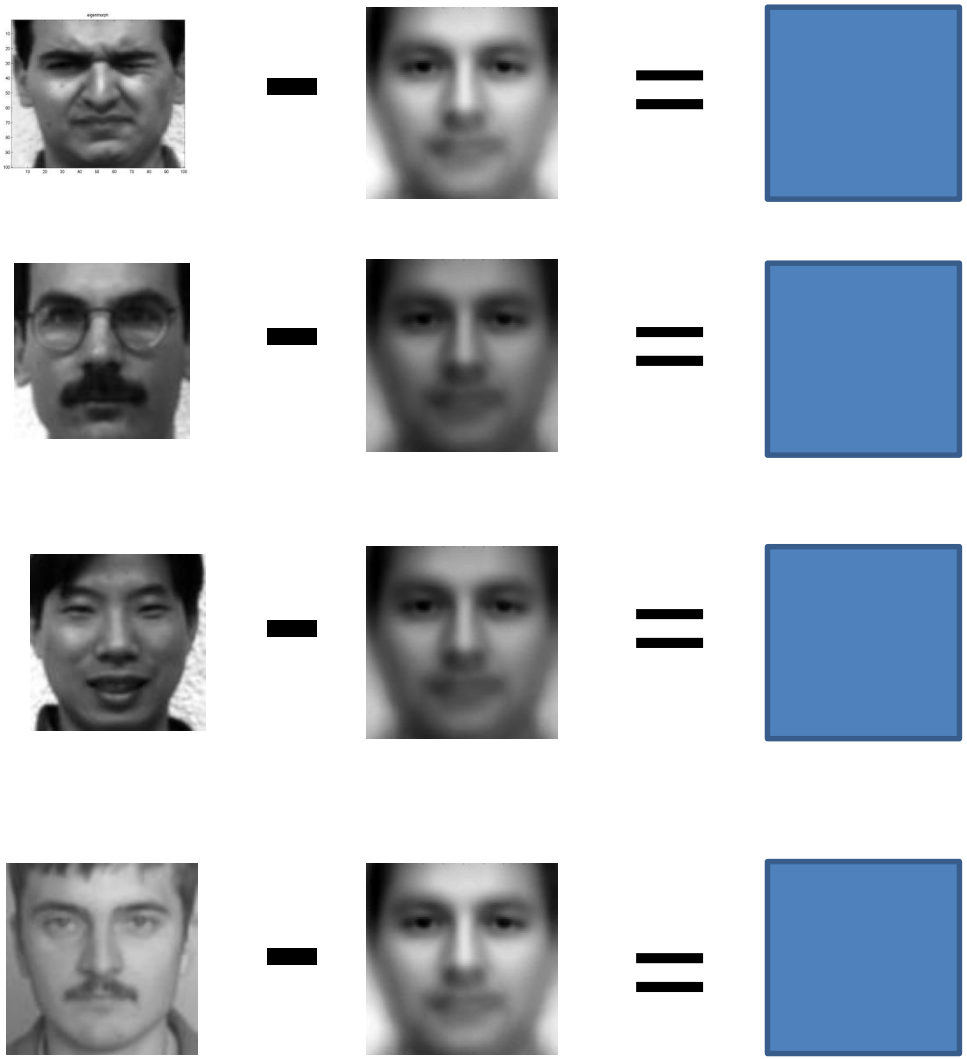
The *first* typical face

The *second* typical face?

?

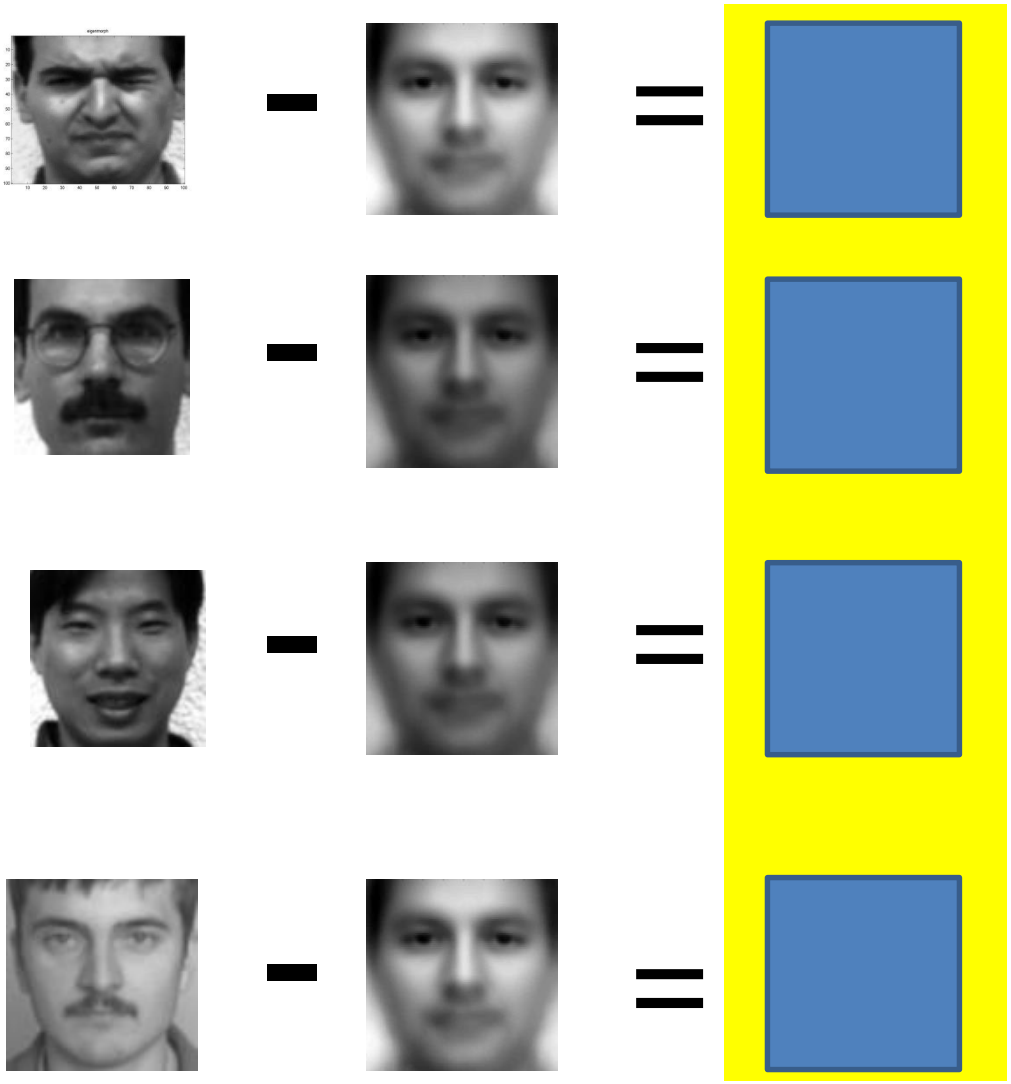
- Approximation with only one typical face V_1 has error
  - Approximating every face as $f = w_{f1} V_1$ is incomplete
- Lets add *second* face to explain this error
  - Add a *second* typical face V_1. Explain each face now as
  - $f = w_{f1} V_1 + w_{f2} V_2$
- How do we find this second face?

Solution: Iterate



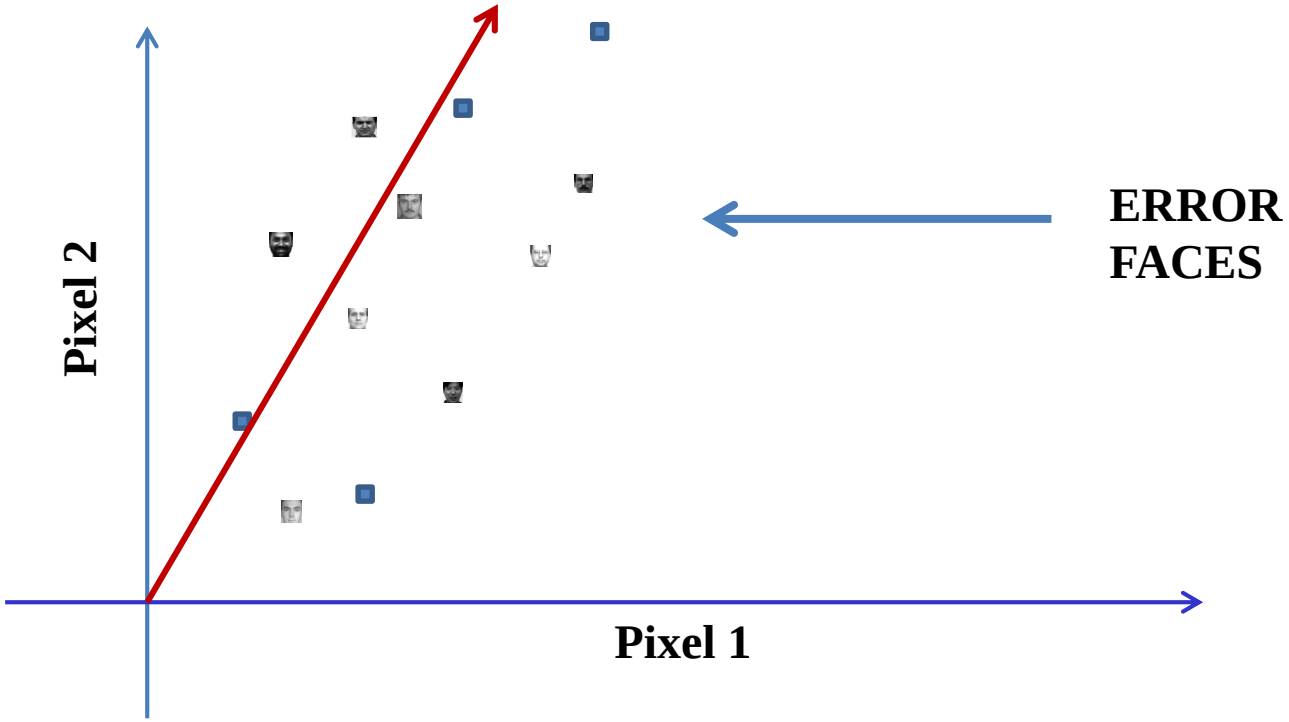
- Get the “error” faces by subtracting the first-level approximation from the original image

Solution: Iterate



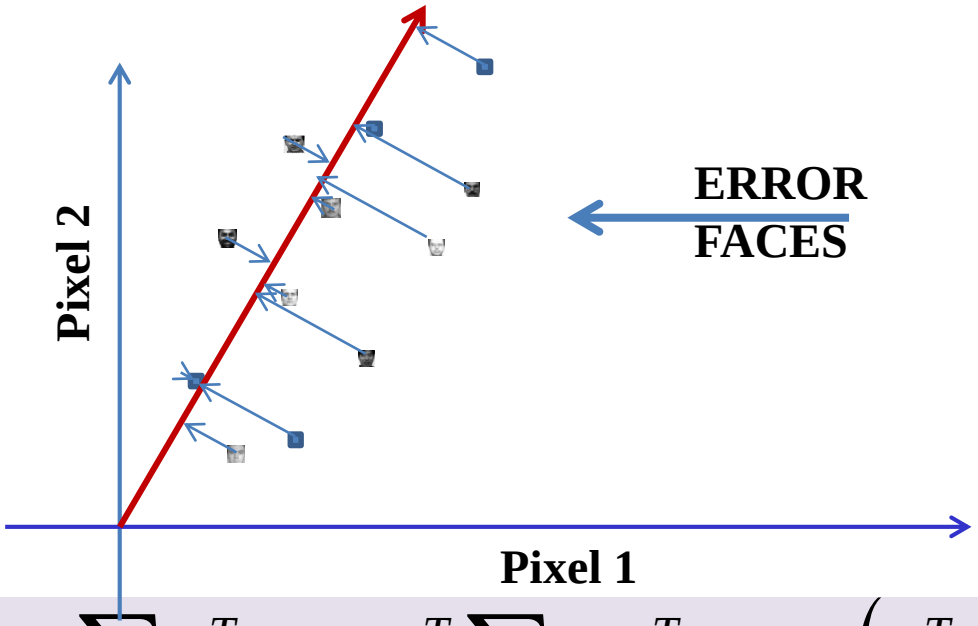
- Get the “error” faces by subtracting the first-level approximation from the original image
- Repeat the estimation on the “error” images

Abstracting the problem:
Finding the *second* typical face



- Each "point" represents an *error* face in "pixel space"
- Find the vector V_2 such that the projection of these error faces on V_2 results in the least error

Minimizing error



The same math applies but now to the set of *error data points*

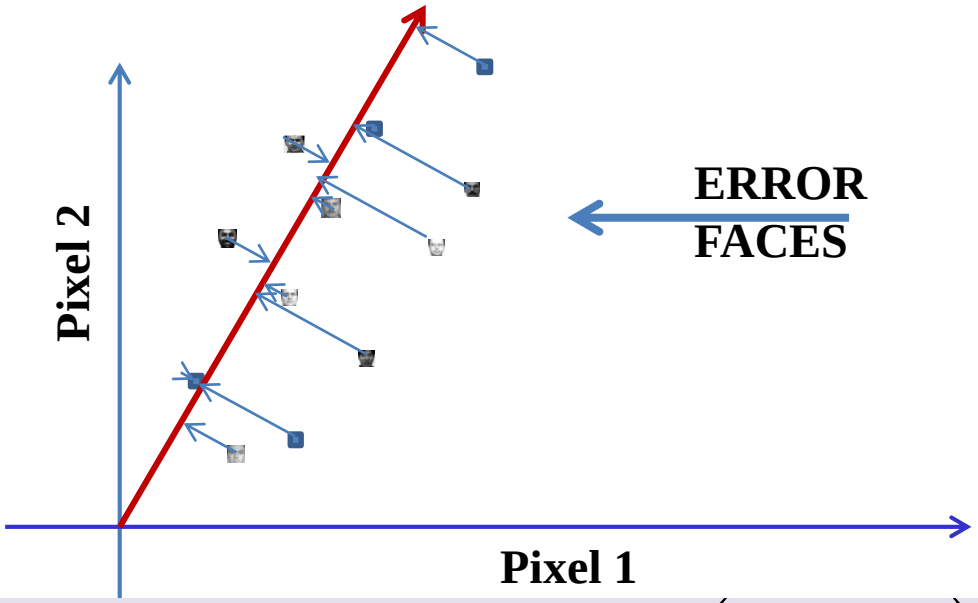
$$L = \sum_i \mathbf{e}_i^T \mathbf{e}_i - \mathbf{v}^T \sum_i \mathbf{e}_i \mathbf{e}_i^T \mathbf{v} + \lambda\left(\mathbf{v}^T \mathbf{v} - 1\right)$$

- Defining the autocorrelation of the error

$$\mathbf{R}_e = \sum \mathbf{e}\mathbf{e}^T$$

$$L = \sum_i \mathbf{e}_i^T \mathbf{e}_i - \mathbf{v}^T \mathbf{R}_e \mathbf{v} + \lambda\left(\mathbf{v}^T \mathbf{v} - 1\right)$$

Minimizing error



The same math applies but now to the set of *error data points*

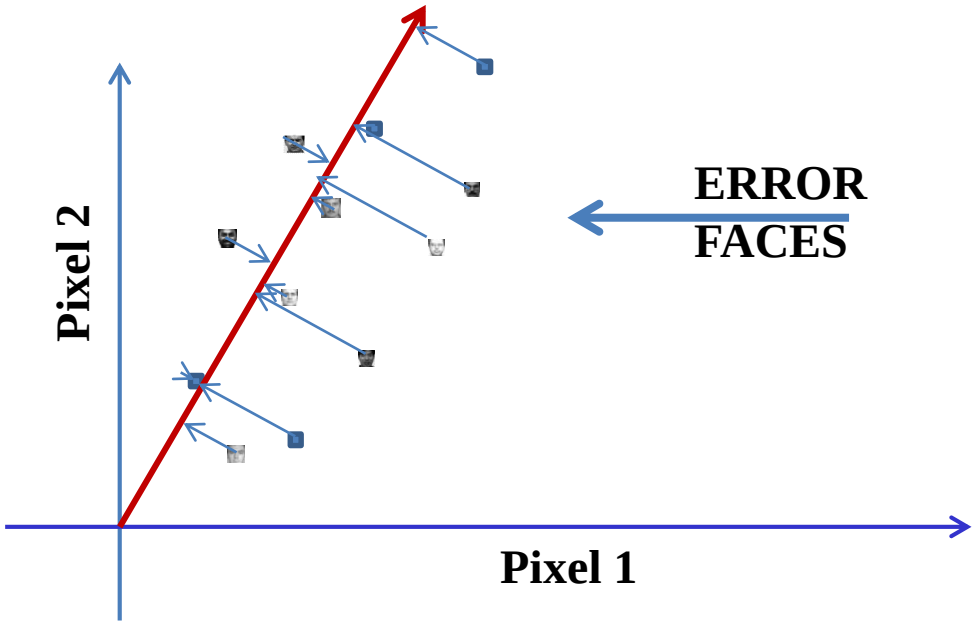
$$L = \sum_i \mathbf{e}_i^T \mathbf{e}_i - \mathbf{v}^T \mathbf{R}_e \mathbf{v} + \lambda\left(\mathbf{v}^T \mathbf{v} - 1\right)$$

- Differentiating w.r.t $\mathbf{v}$ and equating to 0

$$-2\mathbf{R}_e \mathbf{v} + 2\lambda \mathbf{v} = 0$$

$$\mathbf{R}_e \mathbf{v} = \lambda \mathbf{v}$$

Minimizing error



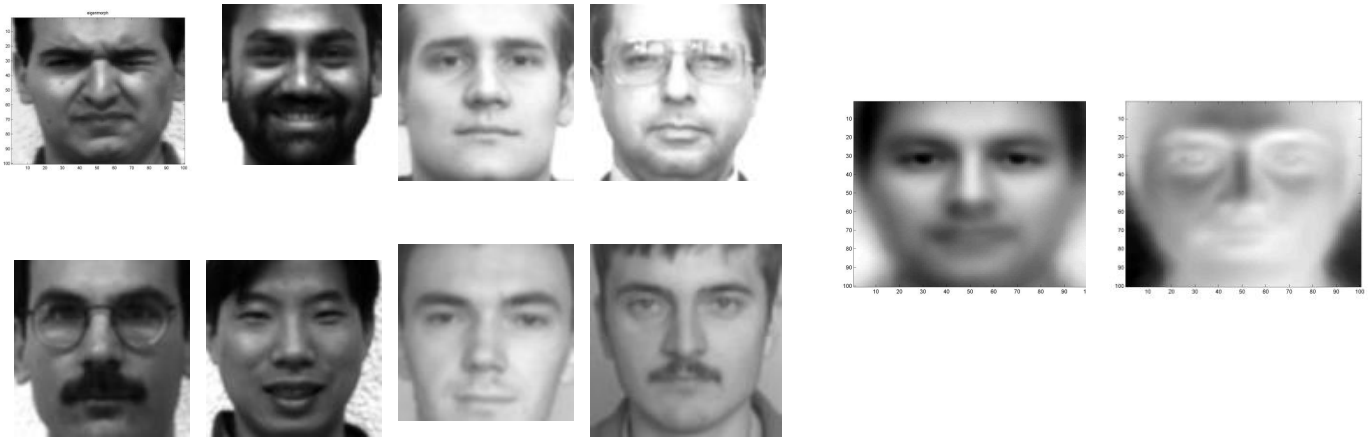
The same math applies but now to the set of *error data points*

- The minimum-error basis is found by solving

$$\mathbf{R}_e \mathbf{v}_2 = \lambda \mathbf{v}_2$$

- $\mathbf{v}_2$ is an Eigen vector of the correlation matrix $\mathbf{R}_e$ corresponding to the largest eigen value λ of $\mathbf{R}_e$

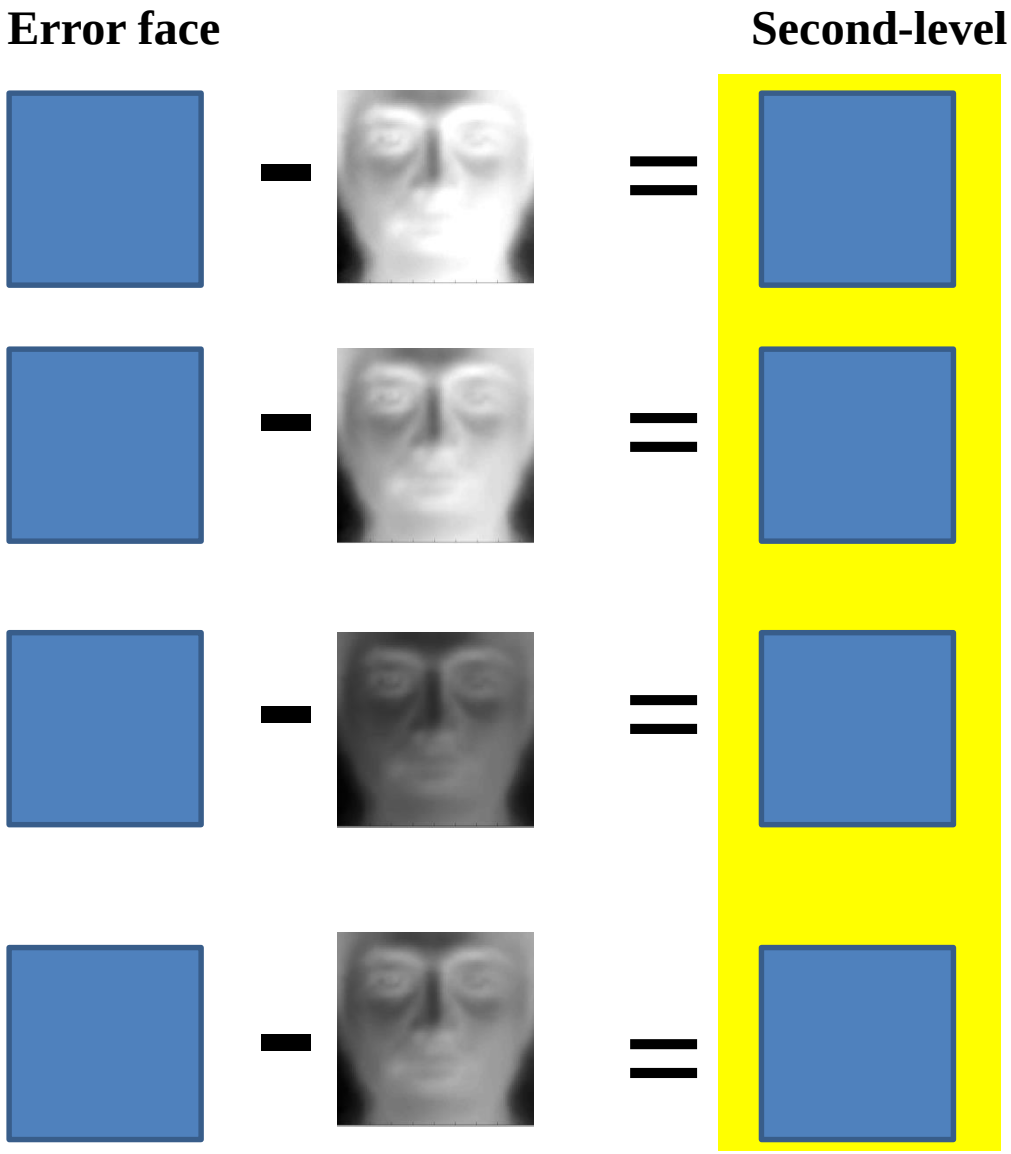
Which gives us our second typical face



- But approximation with the two faces will *still* result in error
- So we need more typical faces to explain *this* error

- We can do this by subtracting the appropriately scaled version of the second “typical” face from the error images and repeating the process

Solution: Iterate



- Get the second-level "error" faces by subtracting the scaled second typical face from the first-level error
- Repeat the estimation on the second-level "error" images

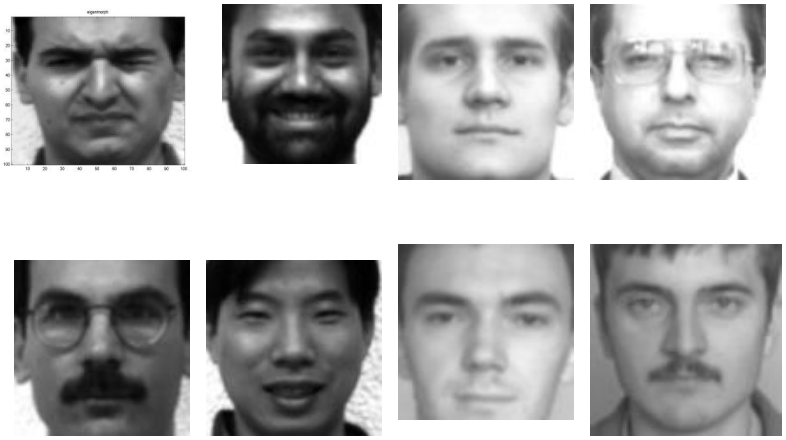
An interesting property

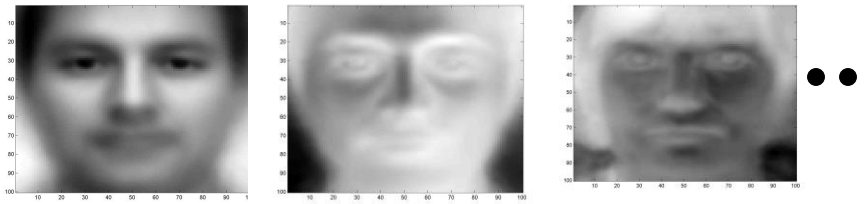
- Each “typical face” will be orthogonal to all other typical faces
  - Because each of them is learned to explain what the rest could not
  - None of these faces can explain one another!

To add more faces

- We can continue the process, refining the error each time
  - An instance of a procedure is called "Gram-Schmidt" orthogonalization
- So what are we really doing?

A collection of least squares typical faces





- Assumption: There are a set of *K* "typical" faces that captures most of all faces
- Approximate **every** face f as $f = w_{f,1} V_1 + w_{f,2} V_2 + w_{f,3} V_3 + .. + w_{f,k} V_k$
  - V_2 is used to "correct" errors resulting from using only V_1. So on average

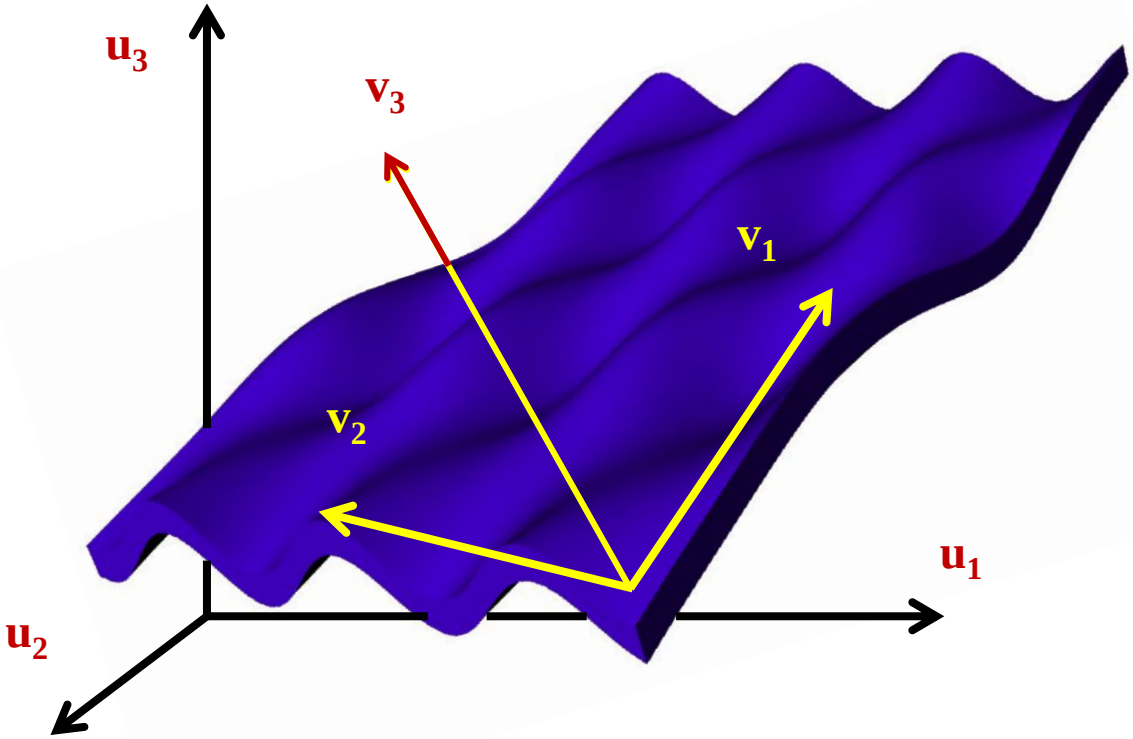
$$\left\| f - (w_{f,1}V_{f,1} + w_{f,2}V_{f,2}) \right\|^2 < \left\| f - w_{f,1}V_{f,1} \right\|^2$$

  - V_3 corrects errors remaining after correction with V_2

$$\left\| f - (w_{f,1}V_{f,1} + w_{f,2}V_{f,2} + w_{f,3}V_{f,3}) \right\|^2 < \left\| f - (w_{f,1}V_{f,1} + w_{f,2}V_{f,2}) \right\|^2$$

  - And so on..
  - $\mathbf{V} = [V_1\ V_2\ V_3]$
- Estimate V to minimize the squared error
  - ***What is V?***

Recall: Basis based representation



- *The most important challenge* in ML: Find the best set of bases for a given data set

The Energy Compaction Property

- **Define "best"?**
- The description

$$X = w_1 B_1 + w_2 B_2 + w_3 B_3 + \ldots + w_N B_N$$

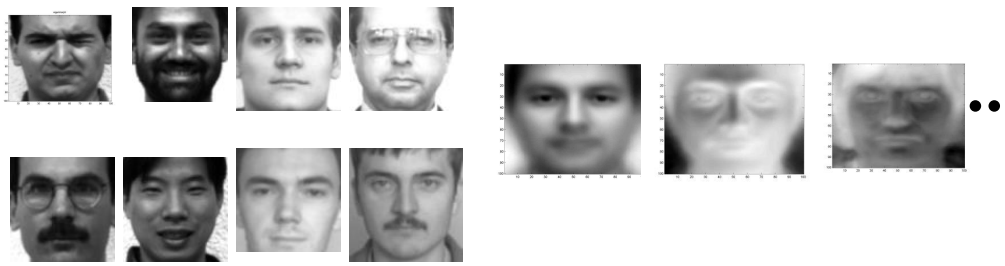
- The ideal:

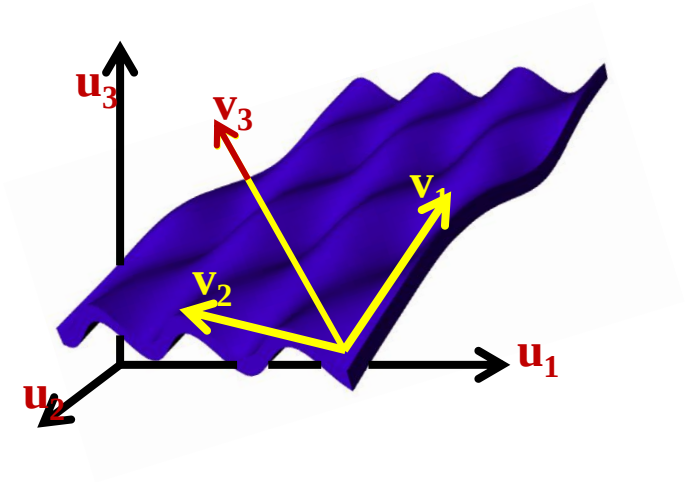
$$\hat{X}_i \approx w_1 B_1 + w_2 B_2 + \ldots + w_i B_i \qquad Error_i = \left\| X - \hat{X}_i \right\|^2$$

$$Error_i < Error_{i-1}$$

  - If the description is terminated at any point, we should still get most of the information about the data
    - No other set of bases (for any leading subset of bases) should result in lower Error for the same number of bases

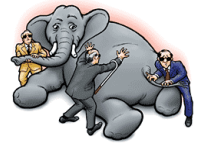
Finding the bases



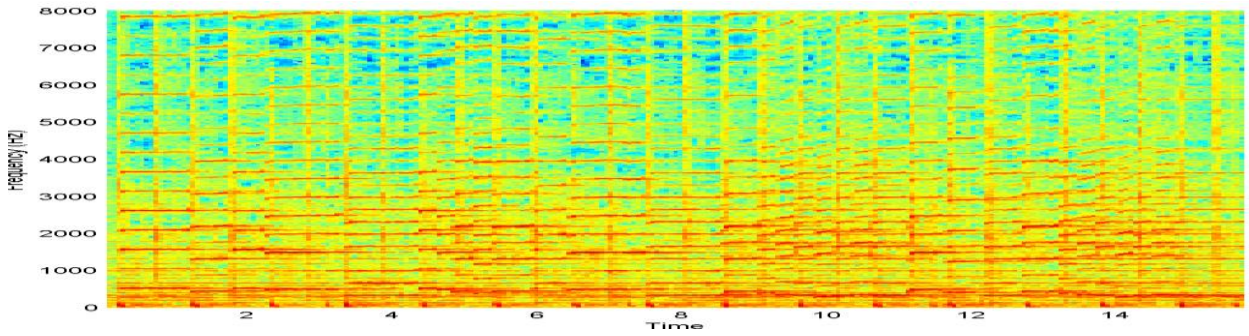


- Finding the optimal set of “typical faces” in this example is the problem of finding the optimal basis set for the data

A recollection

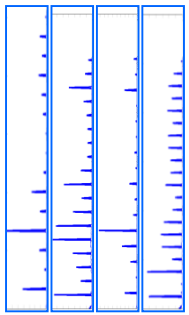


M =



S=Pinv(N) M

N =



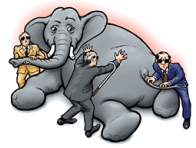
U = ?

$U = NS \approx M$

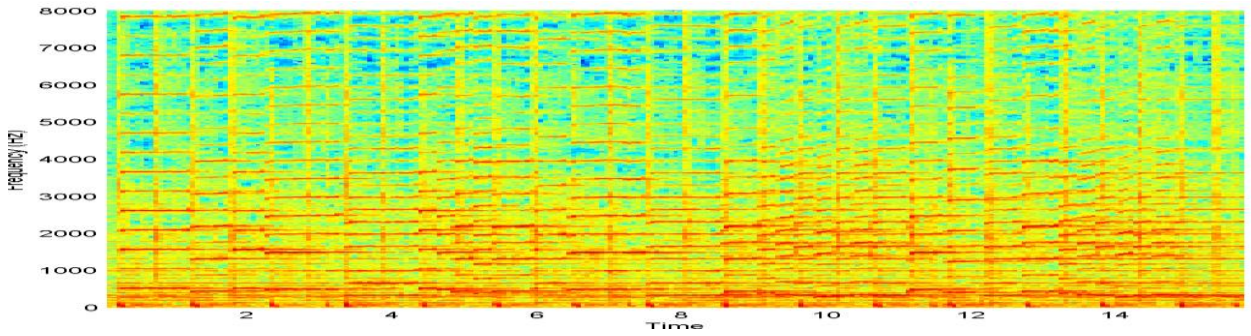
$S = pinv(N)M$

- Finding the best explanation of music M in terms of notes N
- Also finds the *score* S of M in terms of N

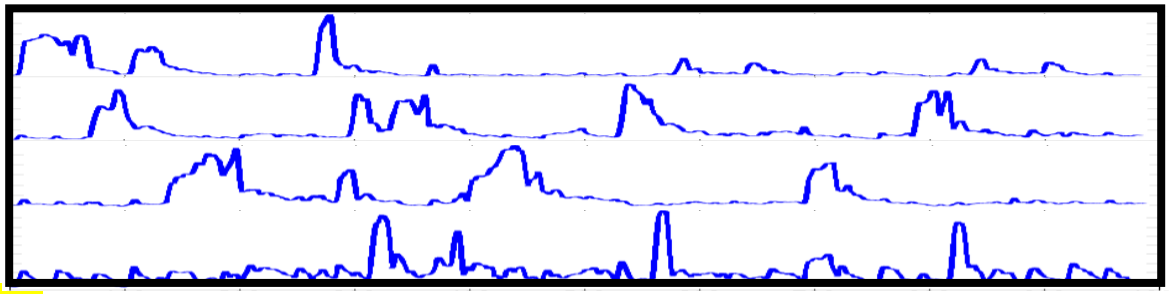
How about the other way?



M =



S =



N = M Pinv(S)

N = ?

U = ?

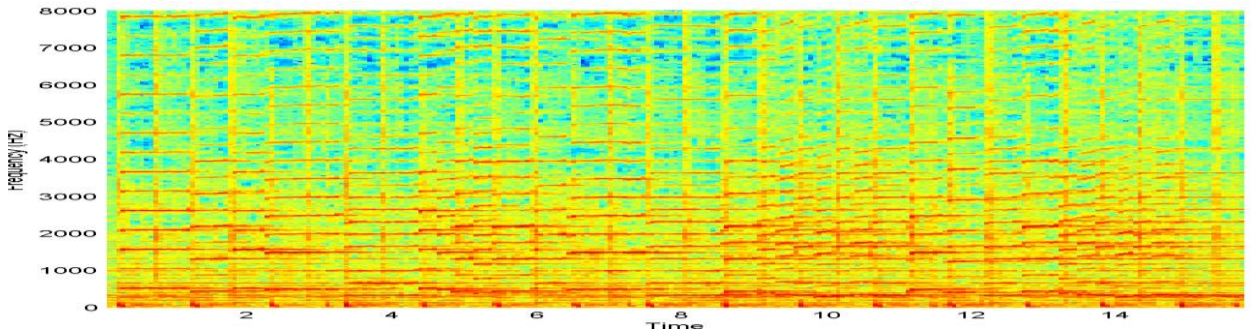
$$U = NS \approx M$$

$$N = M\ pinv(S)$$

- Finding the *notes* N given music M and score S
- Also finds best explanation of M in terms of S

Finding Everything

M =



S = ?

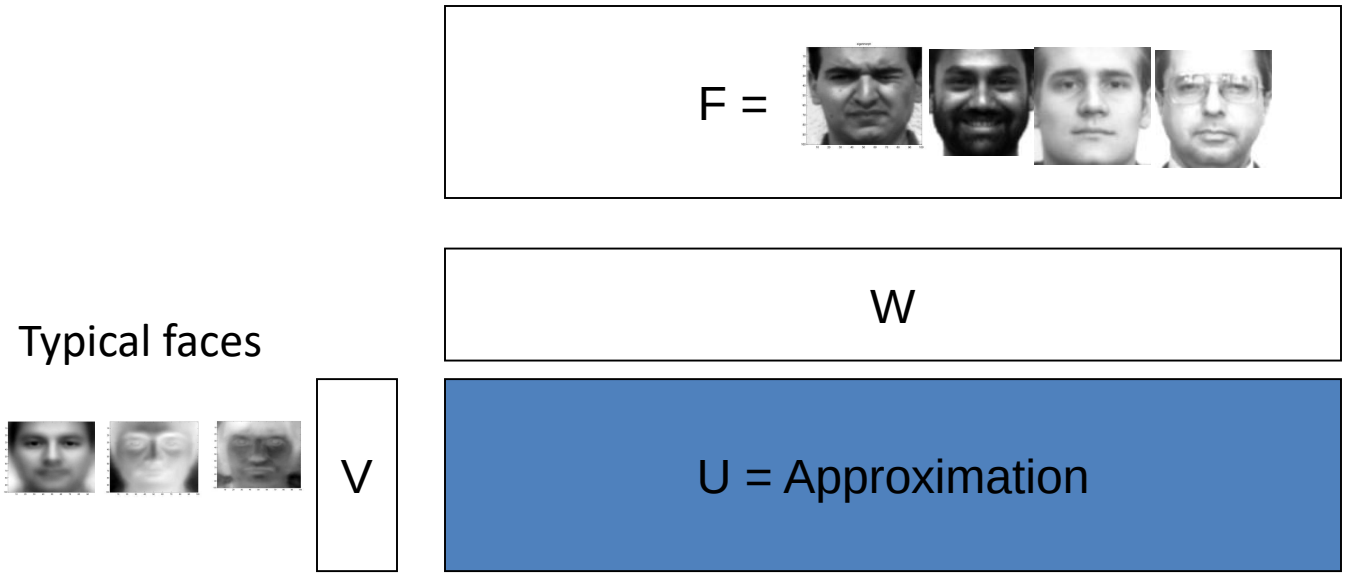
N = ?

U = ?

$$U = NS \approx M$$

- Find the four notes and their score that generate the closest approximation to M

The Same Problem



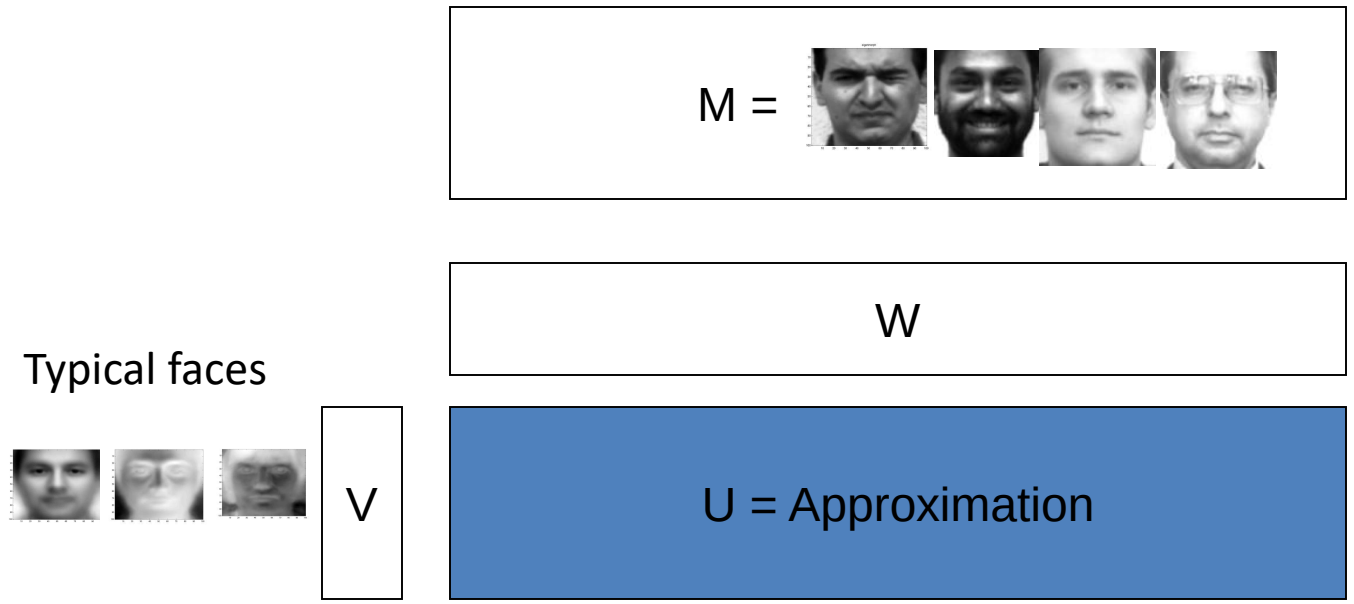
- Here U, V and W are all unknown and must be estimated
  - Such that the total squared error between F and U is minimized
- For each face f
  - $f = w_{f,1}V_1 + w_{f,2}V_2 + \cdots + w_{f,K}V_K$
- For the collection of faces $F \approx VW$
  - V is $D \times K$, W is $K \times N$
    - D is the number of pixels, N is the number of faces in the set

Poll 2

Finding the bases

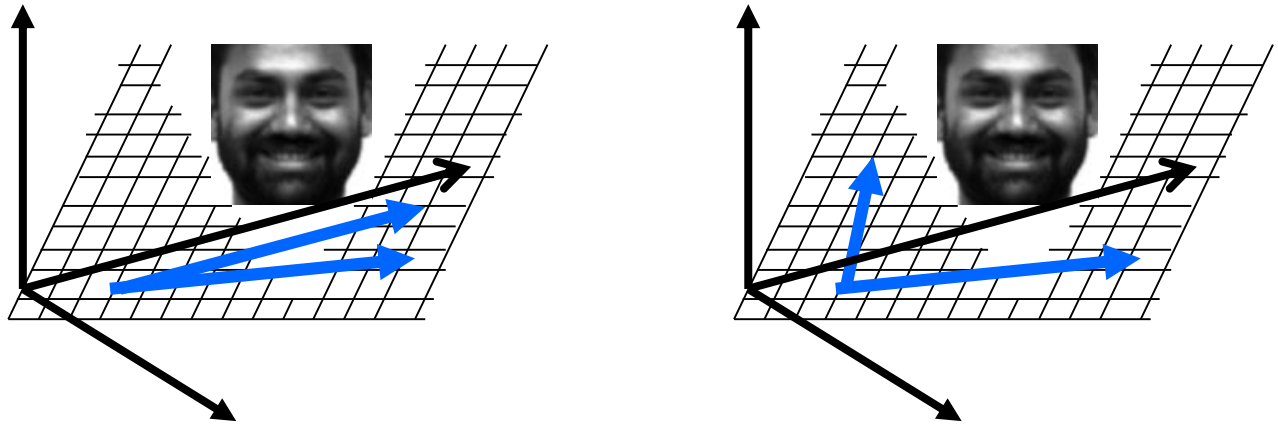
- We just saw an incremental procedure for finding the bases
  - Finding one new basis at a time that explains residual error not explained by previous bases
  - An instance of a procedure is called "Gram-Schmidt" orthogonalization
- We can also do it all at once

With many typical faces



- Approximate **every** face f as $f = w_{f,1} V_1 + w_{f,2} V_2 + \ldots + w_{f,k} V_k$
- Here W, V and U are ALL unknown and must be determined
  - Such that the squared error between U and M is minimum

With multiple bases

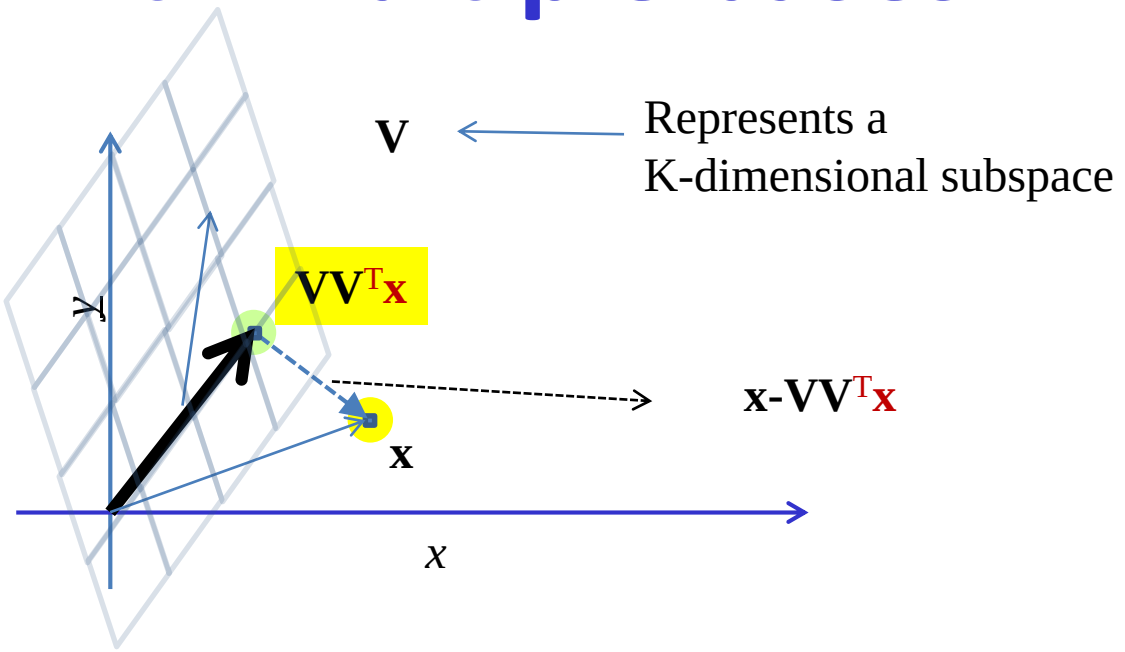


- **Assumption: all bases v_1 v_2 v_3.. are unit length**
- **Assumption: all bases are orthogonal to one another: $v_i^T v_j = 0$ if i != j**
  - We are trying to find the optimal K-dimensional subspace to project the data
  - Any set of basis vectors in this subspace will define the subspace
  - Constraining them to be orthogonal does not change this

- I.e. if $\mathbf{V} = [\mathbf{v}_1\ \mathbf{v}_2\ \mathbf{v}_3\ \ldots]$, $\mathbf{V}^T\mathbf{V} = \mathbf{I}$
  - Pinv($\mathbf{V}$) = $\mathbf{V}^T$

- Projection matrix for $\mathbf{V}$ = $\mathbf{V}$Pinv($\mathbf{V}$) = $\mathbf{V}\mathbf{V}^T$

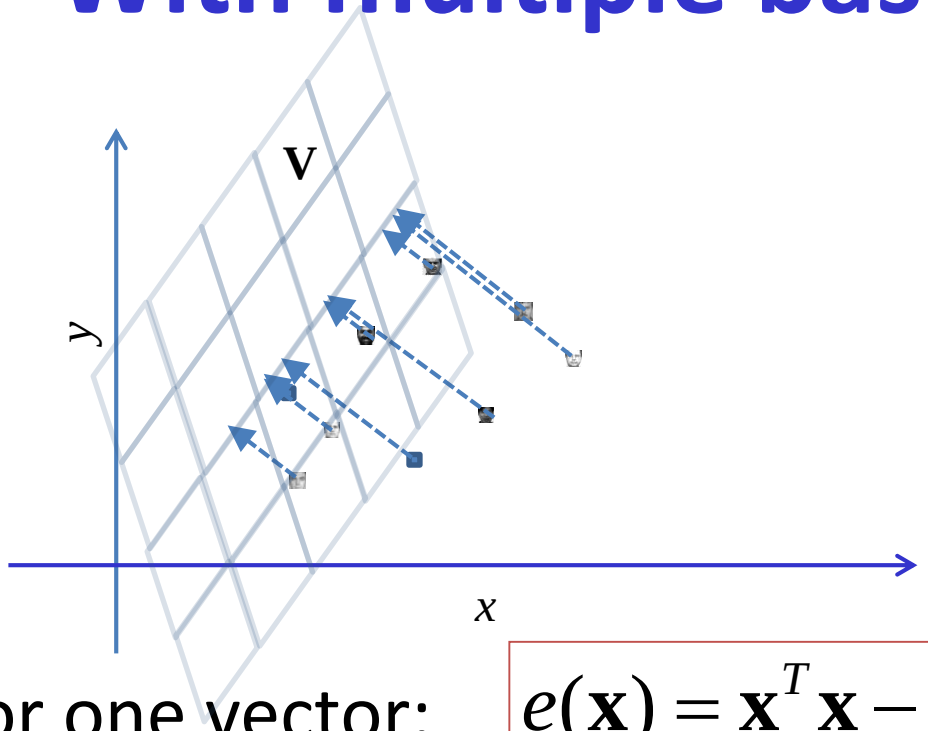
With multiple bases



- Projection for a vector $\hat{\mathbf{x}} = \mathbf{V}\mathbf{V}^T\mathbf{x}$
- Error vector = $\mathbf{x} - \hat{\mathbf{x}} = \mathbf{x} - \mathbf{V}\mathbf{V}^T\mathbf{x}$

- Error length = $e(\mathbf{x}) = \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x}$

With multiple bases



- Error for one vector: $e(\mathbf{x}) = \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x}$
- Error for many vectors

$$E = \sum_i \mathbf{x}_i^T\mathbf{x}_i - \sum_i \mathbf{x}_i^T\mathbf{V}\mathbf{V}^T\mathbf{x}_i$$

- **Goal: Estimate $\mathbf{V}$ to minimize this error!**

Minimizing Error

- With constraint $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, we get the modified objective

$$L = \sum_i \mathbf{x}_i^T\mathbf{x}_i - \sum_i \mathbf{x}_i^T\mathbf{V}\mathbf{V}^T\mathbf{x}_i + trace\big(\Lambda(\mathbf{V}^T\mathbf{V} - \mathbf{I})\big)$$

  - Λ is a diagonal Lagrangian matrix
  - Constraints are $\mathbf{v}_i^T\mathbf{v}_i = 1$ and $\mathbf{v}_i^T\mathbf{v}_j = 0$ for $i \neq j$
- Differentiating w.r.t $\mathbf{V}$ and equating to 0

$$-2\left(\sum_i \mathbf{x}_i^T\mathbf{x}_i\right)\mathbf{V} + 2\mathbf{V}\Lambda = 0 \quad \Rightarrow \quad \mathbf{R}\mathbf{V} = \mathbf{V}\Lambda$$

Finding the optimal K bases

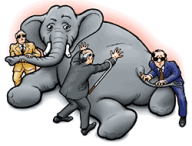
$$\mathbf{R}\mathbf{V} = \mathbf{V}\Lambda$$

- Compute the Eigendecompsition of the correlation matrix
- Select *K* Eigen vectors
- But which *K*?
- Total error = $E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \sum_{j=1}^{K} \lambda_j$
- Select *K* eigen vectors corresponding to the *K* largest Eigen values

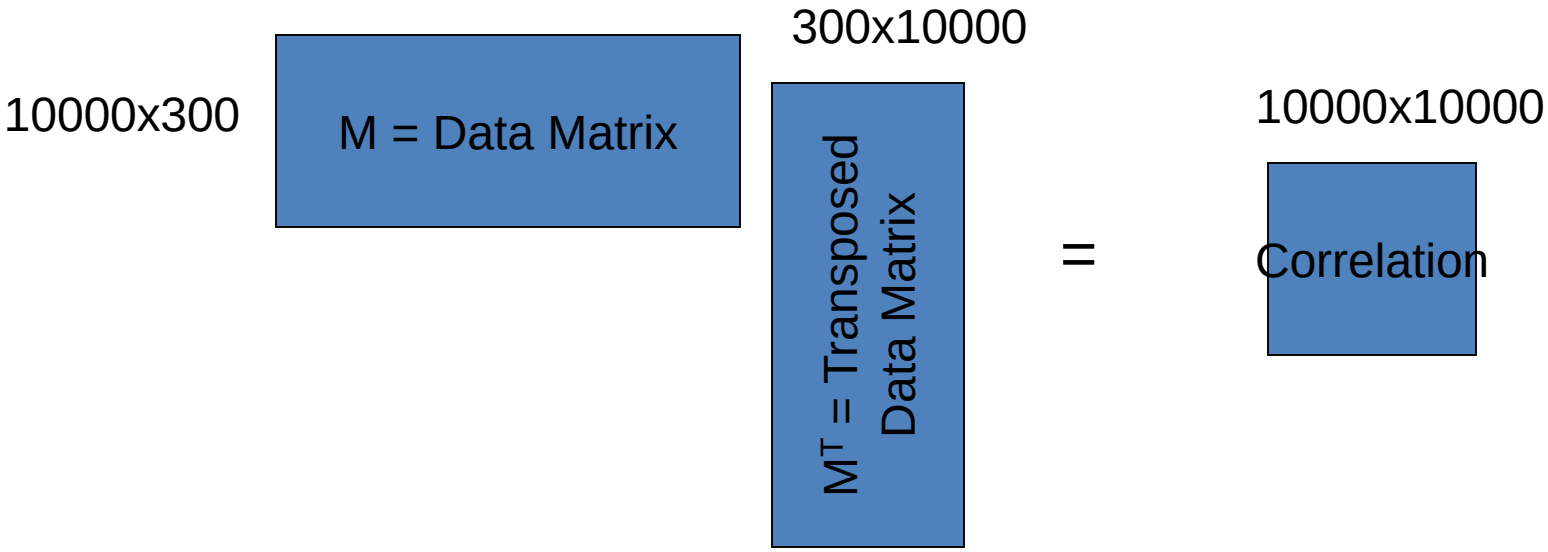
Eigen Faces!

- Arrange your input data into a matrix $\mathbf{X}$
- Compute the correlation $\mathbf{R} = \mathbf{X}\mathbf{X}^{\mathrm{T}}$
- Solve the Eigen decomposition: $\mathbf{R}\mathbf{V} = \Lambda\mathbf{V}$
- The Eigen vectors corresponding to the *K* largest eigen values are our optimal bases
- We will refer to these as *eigen faces.*

Poll 3



How many Eigen faces

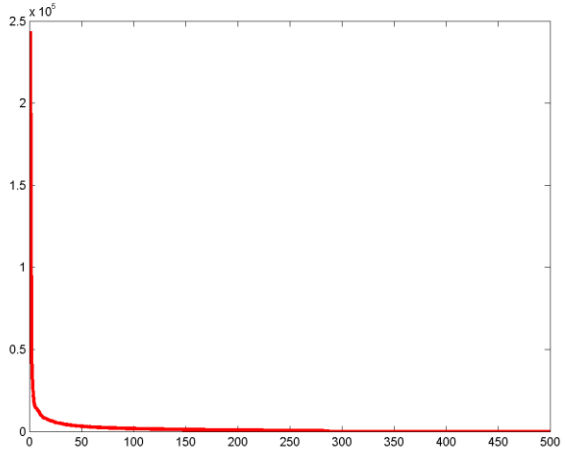


- How to choose “K” (number of Eigen faces)
- Lay all faces side by side in vector form to form a matrix
  - In my example: 300 faces. So the matrix is 10000 x 300
- Multiply the matrix by its transpose
  - The correlation matrix is 10000x10000

Eigen faces

[U,S] = eig(correlation)

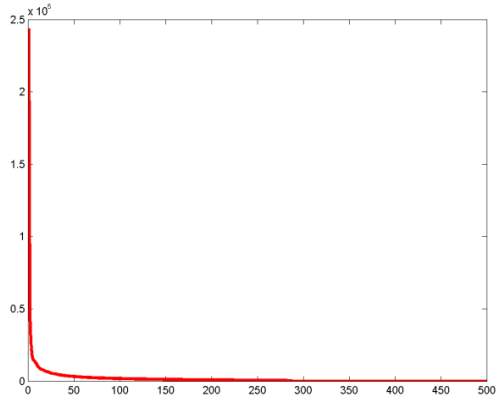
$$S = \begin{bmatrix} \lambda_1 & . & 0 & . & 0 \\ 0 & \lambda_2 & 0 & . & 0 \\ . & . & . & . & . \\ . & . & . & . & . \\ 0 & . & 0 & . & \lambda_{10000} \end{bmatrix} \quad U = \begin{bmatrix} \text{eigenface1} & \text{eigenface2} & \bullet\bullet\bullet \end{bmatrix}$$



- Compute the eigen vectors
  - Only 300 of the 10000 eigen values are non-zero
    - Why?
- Retain eigen vectors with high eigen values (>0)
  - Could use a higher threshold

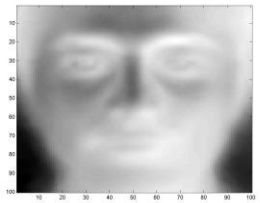
Eigen Faces

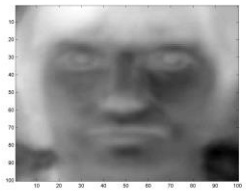
$$U = \left[\begin{array}{ccc} \text{eigenface1} & \text{eigenface2} & \bullet\bullet\bullet \end{array}\right]$$



eigenface1

eigenface2

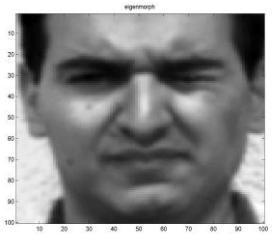


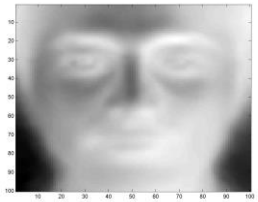
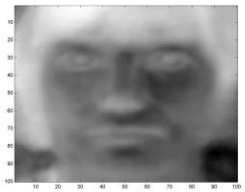


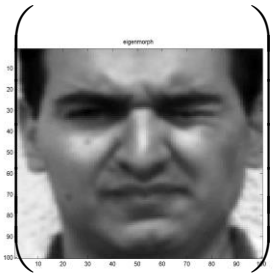
eigenface3

- The eigen vector with the highest eigen value is the first typical face
- The vector with the second highest eigen value is the second typical face.
- Etc.

Representing a face



$= w_1$  $+ w_2$  $+ w_3$  $\bullet\bullet\bullet$

Representation  $= [w_1\ w_2\ w_3\ \ldots.]^T$

- The weights with which the eigen faces must be combined to compose the face are used to represent the face!

Energy Compaction Example

- One outcome of the "energy compaction principle": the approximations are recognizable

- Approximating a face with one basis:

$$f = w_1 \mathbf{v}_1$$

Energy Compaction Example

- One outcome of the "energy compaction principle": the approximations are recognizable


- Approximating a face with one Eigenface:

$$f = w_1 \mathbf{v}_1$$

Energy Compaction Example

- One outcome of the "energy compaction principle": the approximations are recognizable

- Approximating a face with 10 eigenfaces:

$$f = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + \ldots w_{10}\mathbf{v}_{10}$$

Energy Compaction Example

- One outcome of the "energy compaction principle": the approximations are recognizable

- Approximating a face with 30 eigenfaces:

$$f = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + \ldots + w_{10}\mathbf{v}_{10} + \ldots + w_{30}\mathbf{v}_{30}$$

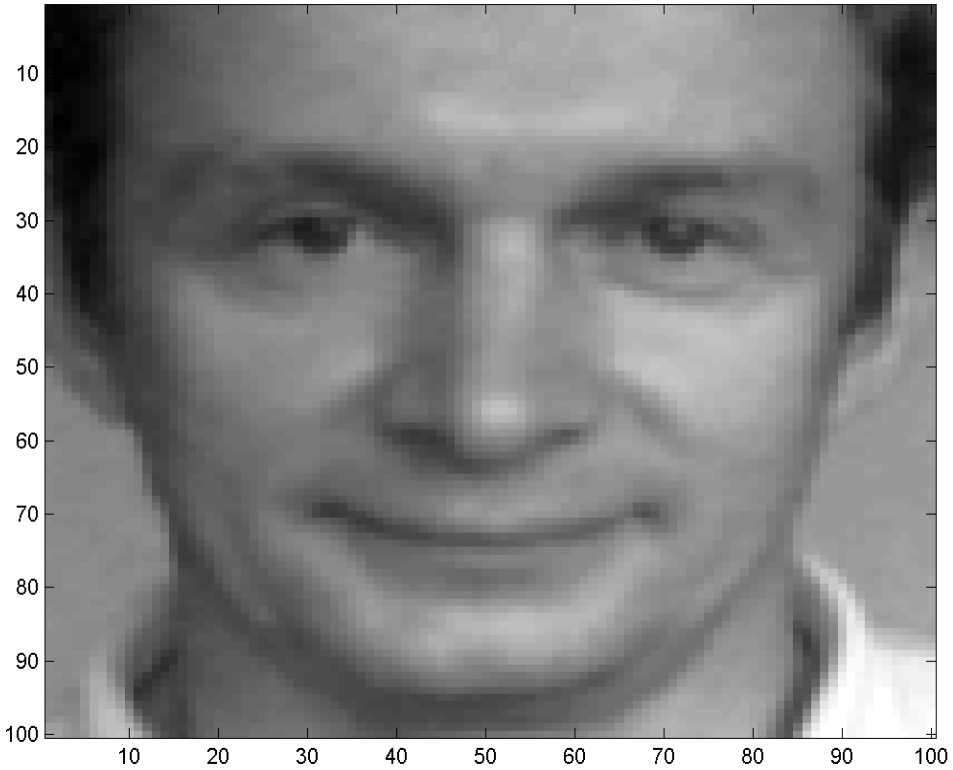
Energy Compaction Example

- One outcome of the "energy compaction principle": the approximations are recognizable

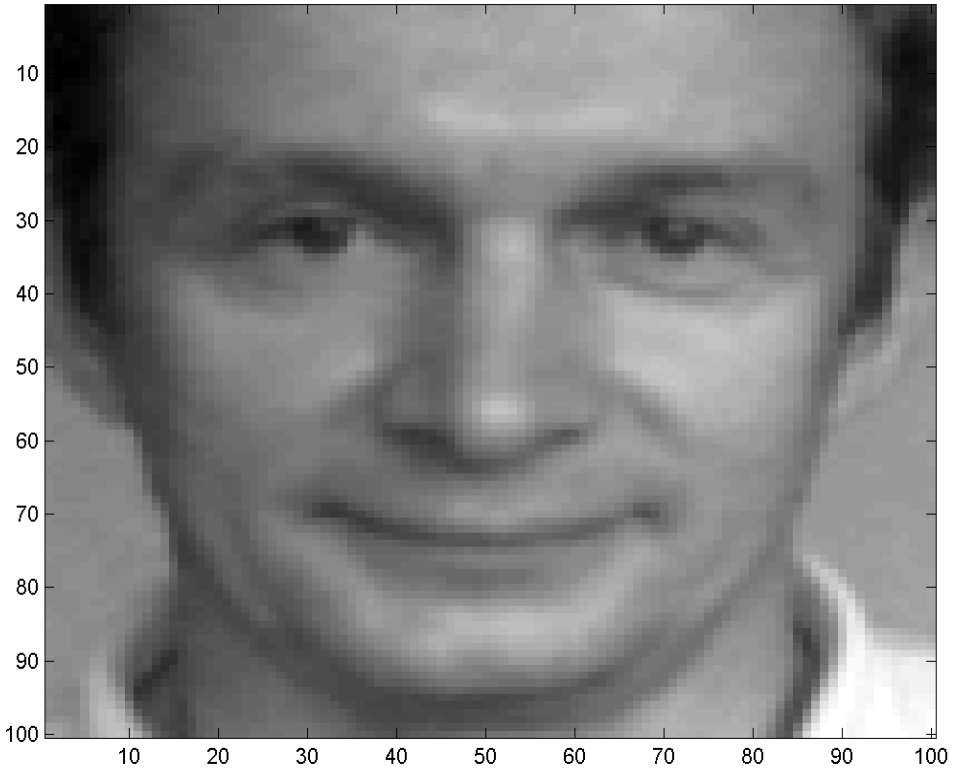
- Approximating a face with 60 eigenfaces:

$$f = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + \ldots + w_{10}\mathbf{v}_{10} + \ldots + w_{30}\mathbf{v}_{30} + \ldots + w_{60}\mathbf{v}_{60}$$

How did I do this?



How did I do this?

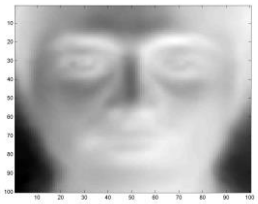


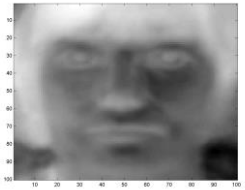
- Hint: only changing weights assigned to Eigen faces..

Class specificity

eigenface1

eigenface2





eigenface3

- The Eigenimages (bases) are very specific to the class of data they are trained on
  - Faces here
- They will not be useful for other classes

Class specificity

- Eigen bases are class specific

- Composing a fishbowl from Eigenfaces

Class specificity

- Eigen bases are class specific

- Composing a fishbowl from Eigenfaces
- With 1 basis

$$f = w_1 \mathbf{v}_1$$

Class specificity

- Eigen bases are class specific



- Composing a fishbowl from Eigenfaces
- With 10 bases

$$f = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + \ldots + w_{10}\mathbf{v}_{10}$$

Class specificity

- Eigen bases are class specific

- Composing a fishbowl from Eigenfaces
- With 30 bases

$$f = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + \ldots + w_{10}\mathbf{v}_{10} + \ldots + w_{30}\mathbf{v}_{30}$$

Class specificity

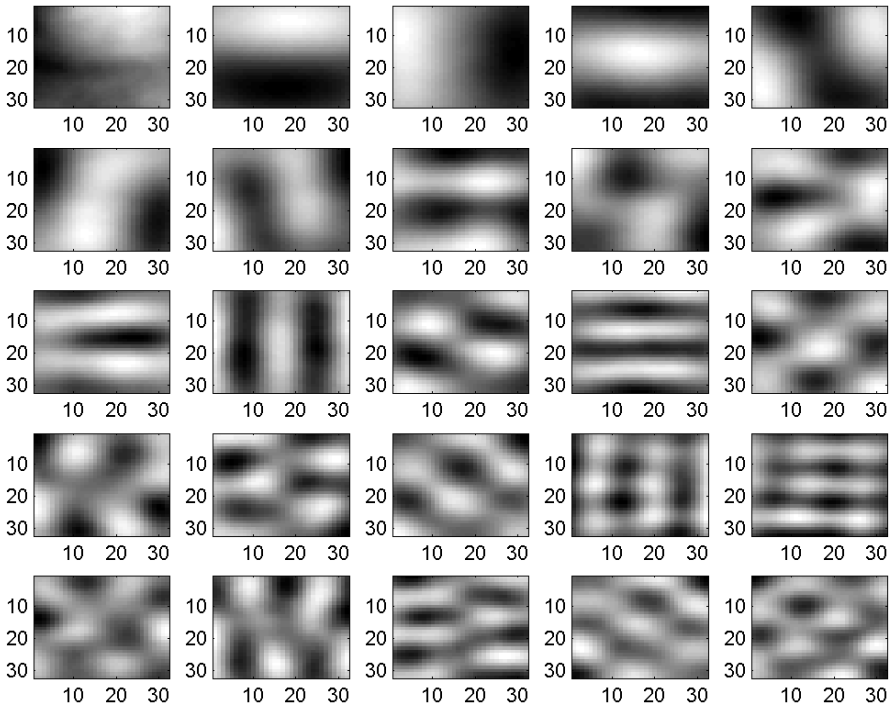
- Eigen bases are class specific

- Composing a fishbowl from Eigenfaces
- With 100 bases

$$f = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + \ldots + w_{10}\mathbf{v}_{10} + \ldots + w_{30}\mathbf{v}_{30} + \ldots + w_{100}\mathbf{v}_{100}$$

Universal bases

- Universal bases..



- End up looking a lot like *discrete cosine transforms!!!!*
- *DCTs are the best "universal" bases*
  - *If you don't know what your data are, use the DCT*

Poll 4

Relation of Eigen decomposition to SVD

Eigen Decomposition of the Correlation Matrix

$$\mathbf{X}\mathbf{X}^T = \mathbf{R} = \mathbf{E}\mathbf{D}\mathbf{E}^T$$

SVD of the Data Matrix

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

$$\mathbf{X}\mathbf{X}^T = \mathbf{U}\mathbf{S}\mathbf{V}^T\ \mathbf{V}\mathbf{S}\mathbf{U}^T = \mathbf{U}\mathbf{S}^2\mathbf{U}^T$$

Comparing $\mathbf{E} = \mathbf{U}$ $\mathbf{D} = \mathbf{S}^2$

- Eigen decomposition of the correlation matrix gives you left singular vectors of data matrix

Dimensionality Reduction

- $\mathbf{R} = \mathbf{E}\mathbf{D}\mathbf{E}^T$
  - The columns of $\mathbf{E}$ are our "Eigen" bases

- We can express any vector X as a combination of these bases

$$X = w_D^X E_1 + w_D^X E_D + \cdots + w_D^X E_D$$

- Using only the "top" K bases
  - Corresponding to the top K Eigen values

$$X \approx w_D^X E_1 + w_D^X E_D + \cdots + w_K^X E_K$$

Dimensionality Reduction

- Using only the "top" K bases
  - Corresponding to the top K Eigen values

$$X \approx w_D^X E_1 + w_D^X E_D + \cdots + w_K^X E_K$$

- In vector form:

$$X \approx \boldsymbol{E}_{1:K} \mathbf{w}_K^X$$

$$\mathbf{w}_K^X = Pinv(\boldsymbol{E}_{1:K}) X = \boldsymbol{E}_{1:K}^T X$$

$$\mathbf{W}_K^X = \boldsymbol{E}_{1:K}^T \mathbf{X}$$

- If "$\boldsymbol{E}$" is agreed upon, knowing $\mathbf{W}_K^X$ is sufficient to reconstruct $\mathbf{X}$
  - Store only K numbers per vector instead of D without losing too much information
  - **Dimensionality Reduction**

Lets give it a name

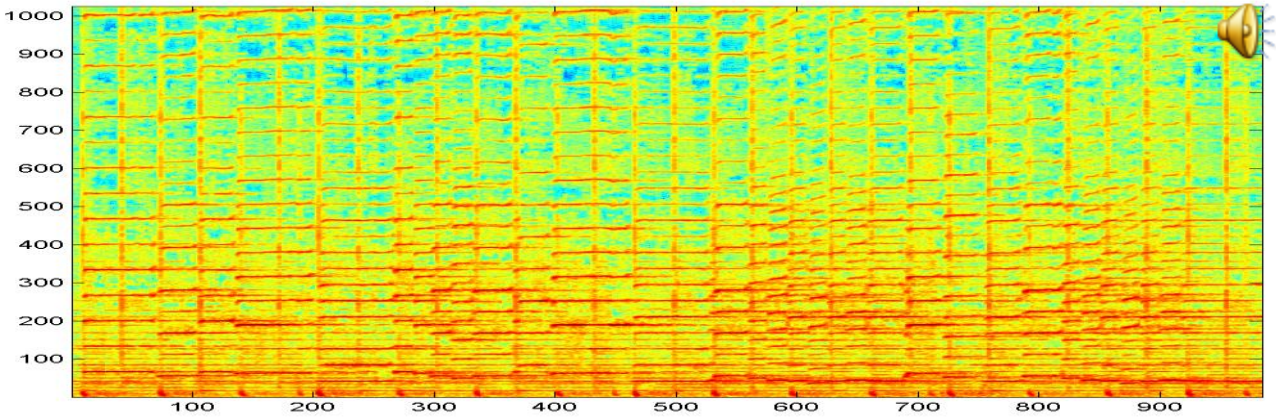
$$\mathbf{R} = \mathbf{E}\mathbf{D}\mathbf{E}^T$$

$\mathbf{E}$ are the "Eigen Bases"

$$\mathbf{W}_K^X = \boldsymbol{E}_{1:K}^T \mathbf{X}$$

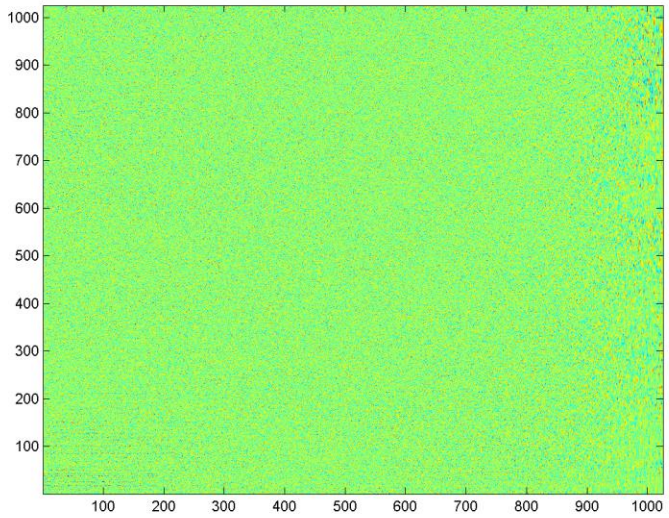
- Retaining only the top *K* weights for every data vector
  - Computed by multiplying the data matrix by the transpose of the top *K* Eigen vectors of ***R***
- This is called the *Karhunen Loeve Transform*
  - ***Not PCA!***

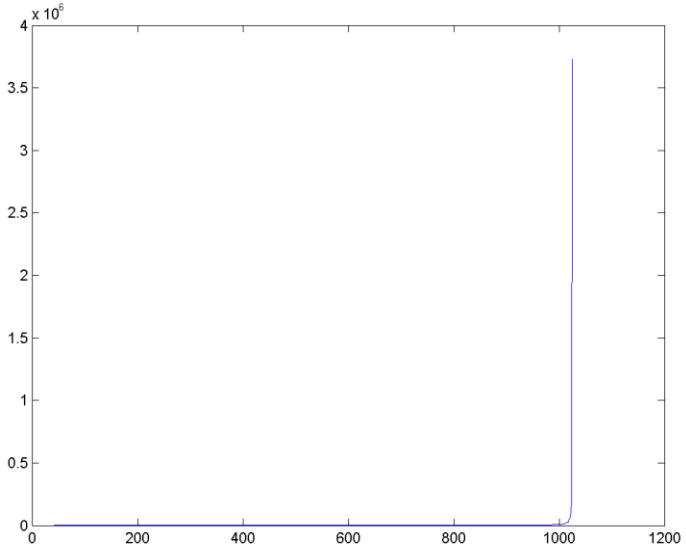
An audio example



- The spectrogram has 974 vectors of dimension 1025
- The covariance matrix is size 1025 x 1025
- There are 1025 eigenvectors

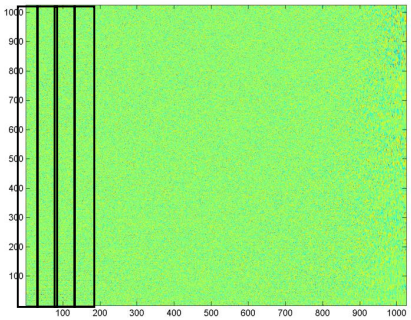
Eigenvalues and Eigenvectors

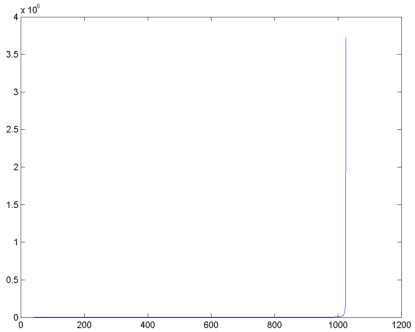


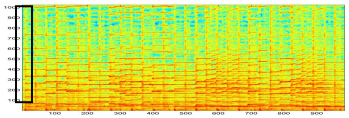


- Left panel: Matrix with 1025 eigen vectors
- Right panel: Corresponding eigen values
  - Most Eigen values are close to zero
    - The corresponding eigenvectors are “unimportant”

Eigenvalues and Eigenvectors





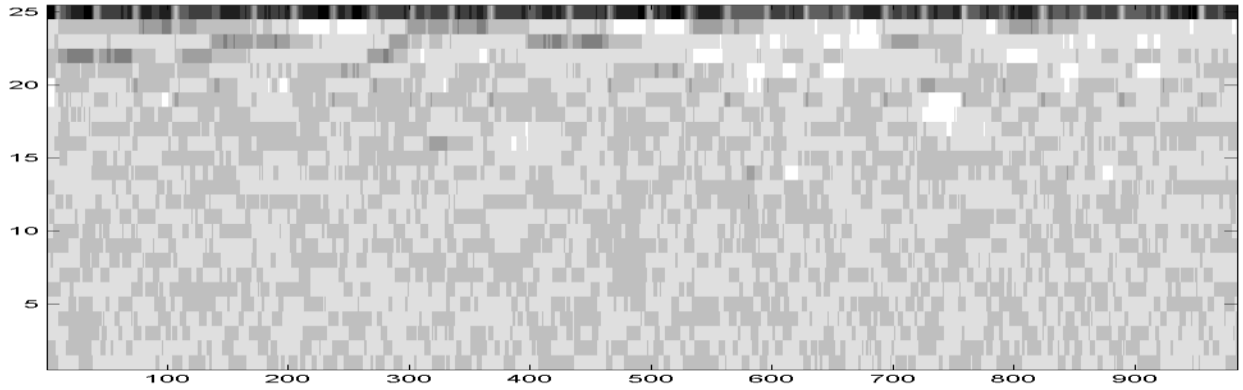


Vec = a1 *eigenvec1 + a2 * eigenvec2 + a3 * eigenvec3 …

- The vectors in the spectrogram are linear combinations of all 1025 Eigen vectors
- The Eigen vectors with low Eigen values contribute very little
  - The average value of a_i is proportional to the square root of the Eigenvalue
  - Ignoring these will not affect the composition of the spectrogram

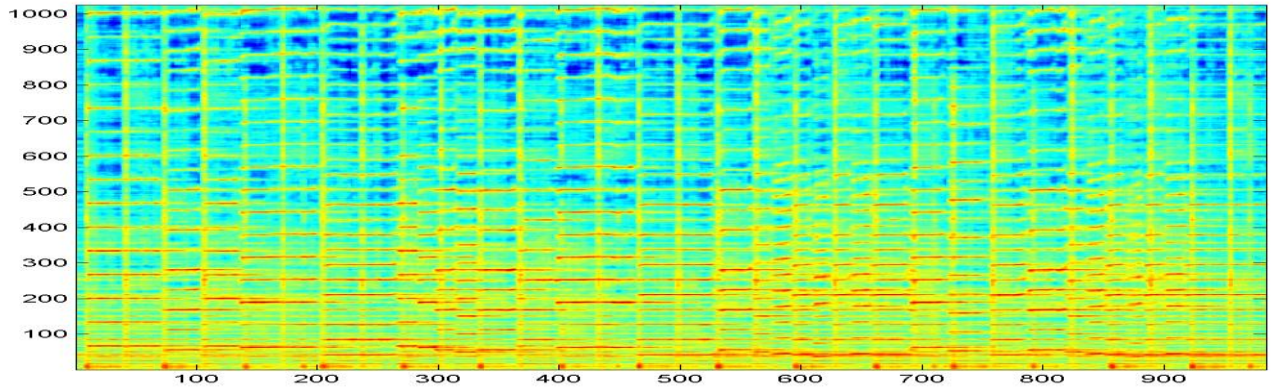
An audio example

$$V_{reduced} = [V_1 \quad . \quad . \quad V_{25}]$$
$$M_{lowdim} = Pinv(V_{reduced})M$$



- The same spectrogram projected down to the 25 eigen vectors with the highest eigen values
  - Only the 25-dimensional weights are shown
    - The weights with which the 25 eigen vectors must be added to compose a least squares approximation to the spectrogram

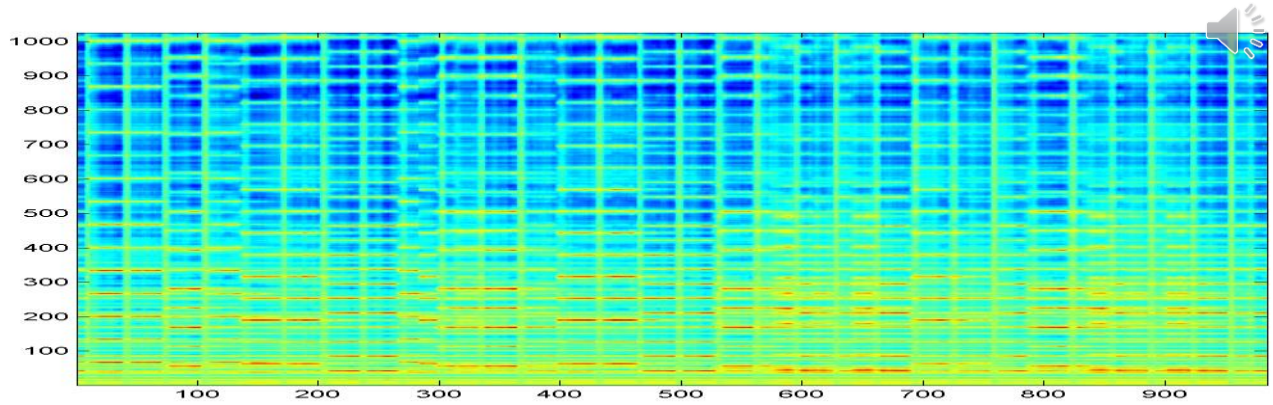
An audio example



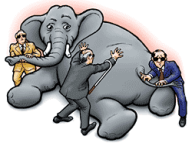
$$M_{reconstructed} = V_{reduced} M_{low\dim}$$

- The same spectrogram constructed from only the 25 Eigen vectors with the highest Eigen values
  - Looks similar
    - With 100 Eigenvectors, it would be indistinguishable from the original
  - Sounds pretty close
  - But now sufficient to store 25 numbers per vector (instead of 1024)

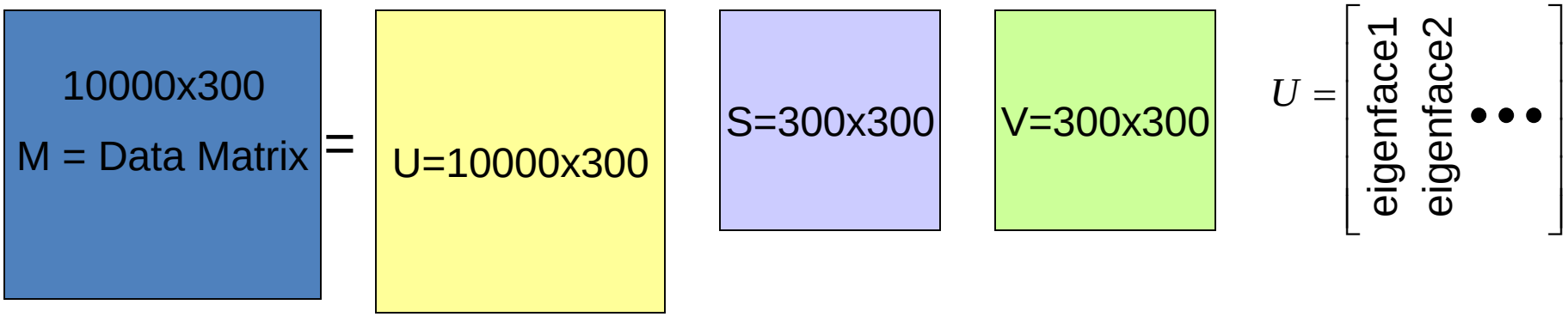
With only 5 eigenvectors



- The same spectrogram constructed from only the 5 Eigen vectors with the highest Eigen values
  - Highly recognizable



SVD instead of Eigen



- Do we need to compute a 10000 x 10000 correlation matrix and then perform Eigen analysis?
  - Will take a very long time on your laptop
- SVD
  - Only need to perform "Thin" SVD. Very fast
    - U = 10000 x 300
      - The columns of U are the eigen faces!
      - The Us corresponding to the "zero" eigen values are not computed
    - S = 300 x 300
    - V = 300 x 300

Using SVD to compute Eigenbases

[U, S, V] = SVD(X)

- U will have the Eigenvectors
- Thin SVD for 100 bases:

  [U,S,V] = svds(X, 100)
- Much more efficient

Eigen Decomposition of data

- Nothing magical about faces or sound – can be applied to any data.
  - Eigen analysis is one of the key components of data compression and representation
  - Represent N-dimensional data by the weights of the K leading Eigen vectors
    - Reduces effective dimension of the data from N to K
    - But requires knowledge of Eigen vectors

What kind of representation?

- What we just saw: *Karhunen Loeve Expansion*
- What you may be familiar with: *Principal Component Analysis*

- The two are similar, but not the same!!

Linear vs. Affine

- The model we saw (KLE)
  - Approximate **every** face f as

$$f = w_{f,1} V_1 + w_{f,2} V_2 + \ldots + w_{f,k} V_k$$

  - Linear combination of bases

- If you add a constant (PCA)

$$f = w_{f,1} V_1 + w_{f,2} V_2 + \ldots + w_{f,k} V_k + m$$

  - *Affine* combination of bases

Affine expansion

- Estimate

$$\mathbf{f} = \mathbf{m} + w_{f,1}\,\mathbf{V}_1 + w_{f,2}\,\mathbf{V}_2 + \ldots + w_{f,k}\,\mathbf{V}_k$$

- Using the *energy compaction* principle leads to the usual incremental estimation rule
  - $\mathbf{m}$ must explain most of the energy
  - Each new basis must explain most of the residual energy

Estimation with the constant

- Estimate

$$f = w_{f,1} V_1 + w_{f,2} V_2 + \ldots + w_{f,k} V_k + m$$

- Lets do this incrementally first:
- $f \approx m$
  - For every face
  - Find *m* to optimize the approximation

Estimation with the constant

- Estimate

  $\mathrm{f} \approx \mathrm{m}$

  – for *every f!*

- Error over all faces $E = \sum_f ||f - m||^2$
- Minimizing the error with respect to *m*, we simply get

  – $m = \frac{1}{N}\sum_f f$

- The *mean* of the data

Estimation the remaining

- Same procedure as before:
  - Remaining "typical faces" must model what the constant *m* could not
- Subtract the constant from every data point
  - $\hat{f} = f - m$
- Now apply the model:
  - $\hat{f} = w_{f,1} V_1 + w_{f,2} V_2 + \ldots + w_{f,k} V_k$
- This is just Eigen analysis of the "mean-normalized" data
  - Also called the "centered" data

Estimating the Affine model

$$f = w_{f,1} V_1 + w_{f,2} V_2 + \ldots + w_{f,k} V_k + m$$

- First estimate the mean *m*

$$m = \frac{1}{N} \sum_f f$$

- Compute the correlation matrix of the "centered" data $\hat{f} = f - m$
  - $\boldsymbol{C} = \sum_f \hat{f} \hat{f}^T = \sum_f (f - m)(f - m)^T$
  - This is the *covariance* matrix of the set of f

Estimating the Affine model

$$f = w_{f,1} V_1 + w_{f,2} V_2 + \ldots + w_{f,k} V_k + m$$

- First estimate the mean m

$$m = \frac{1}{N} \sum_f f$$

- Compute the covariance matrix
  - $\boldsymbol{C} = \sum_f (f - m)(f - m)^T$

- Eigen decompose!

$$\mathbf{CV} = \mathbf{V}\Lambda$$

- The Eigen vectors corresponding to the top *k* Eigen values give us the bases V_k

Linear vs. Affine

- The model we saw
  - Approximate **every** face f as

$$f = w_{f,1} V_1 + w_{f,2} V_2 + \ldots + w_{f,k} V_k$$

  - The ***Karhunen Loeve Expansion***
  - Retains maximum ***Energy*** for any order k

- If you add a constant

$$f = w_{f,1} V_1 + w_{f,2} V_2 + \ldots + w_{f,k} V_k + m$$

  - ***Principal Component Analysis***
  - Retains maximum ***Variance*** for any order k

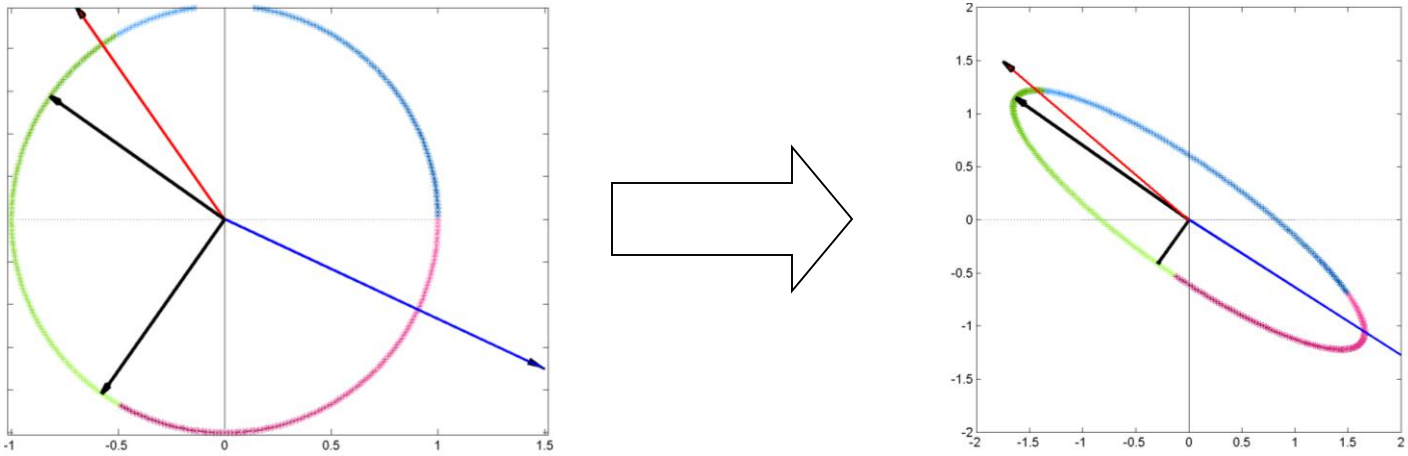
How do they relate

- Relationship between correlation matrix and covariance matrix

$$\mathbf{R} = \mathbf{C} + mm^{\mathrm{T}}$$

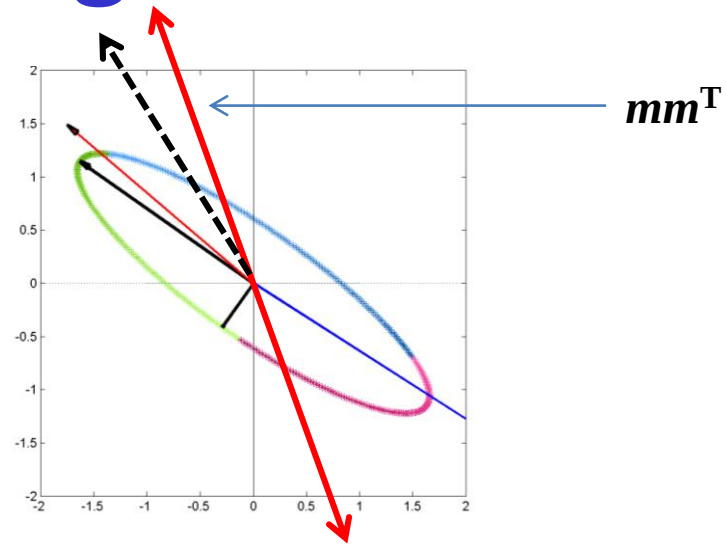
- *Karhunen Loeve* bases are Eigen vectors of **R**
- *PCA* bases are Eigen vectors of **C**
- How do they relate
  - Not easy to say..

The Eigen vectors



- The Eigen vectors of ***C*** are the major axes of the ellipsoid ***Cv***, where ***v*** are the vectors on the unit sphere

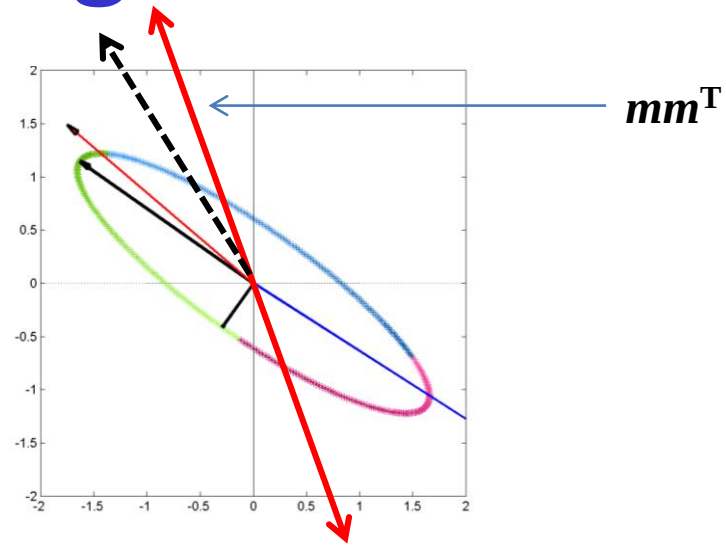
The Eigen vectors



- The Eigen vectors of $\boldsymbol{R}$ are the major axes of the ellipsoid $\boldsymbol{Cv} + \boldsymbol{mm}^T\boldsymbol{v}$

- Note that $\boldsymbol{mm}^T$ has rank 1 and $\boldsymbol{mm}^T\boldsymbol{v}$ is a line

The Eigen vectors



- The principal Eigenvector of ***R*** lies between the principal Eigen vector of ***C*** and ***m***

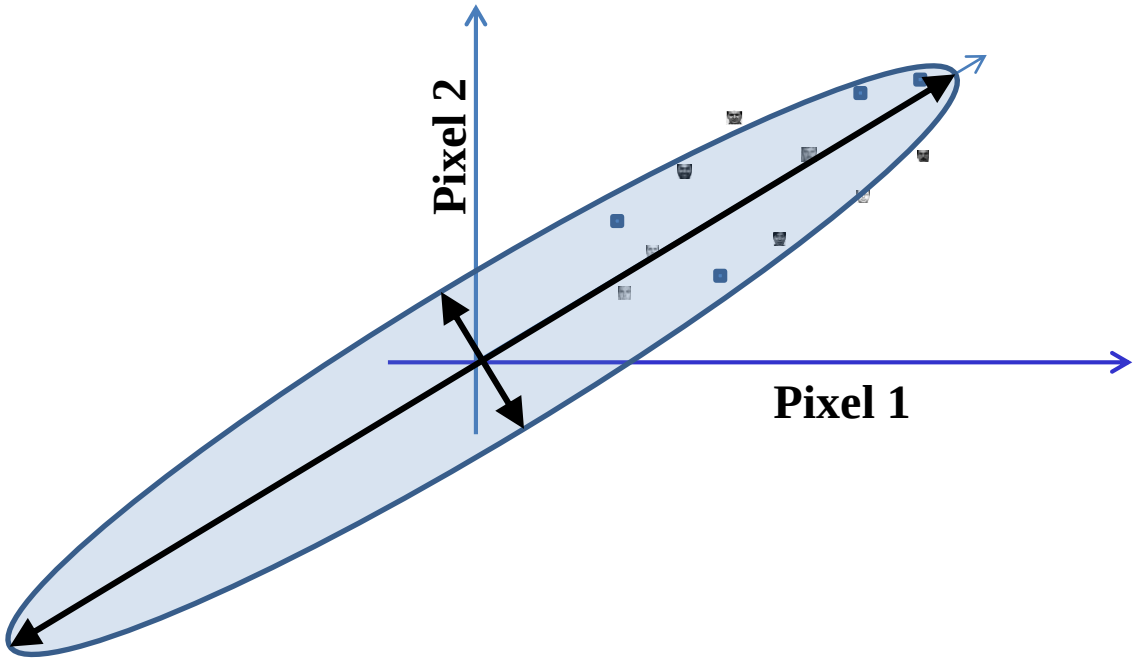
$$\mathbf{e}_R = \alpha \mathbf{e}_C + (1-\alpha)\frac{\mathbf{m}}{\|\mathbf{m}\|} \qquad 0 \le \alpha \le 1$$

- Similarly the principal Eigen *value*

$$\lambda_R = \alpha \lambda_C + (1-\alpha)\|\mathbf{m}\|^2$$

- Similar logic is not easily extendable to the other Eigenvectors, however

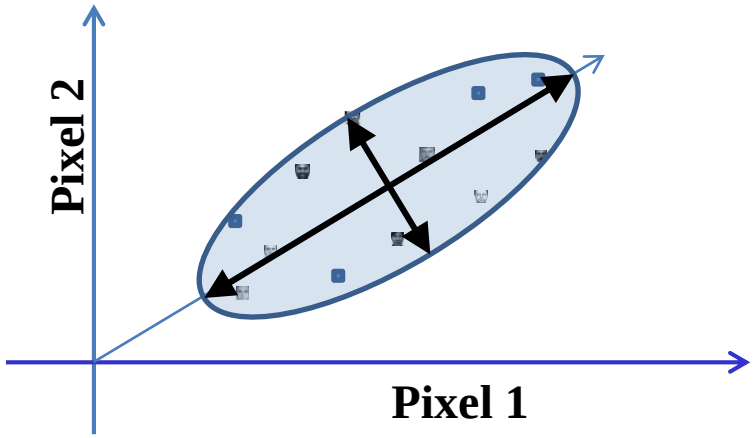
Eigenvectors



- Turns out: Eigenvectors of the *correlation* matrix represent the major and minor axes of an ellipse centered at the origin which encloses the data most compactly

- The SVD of data matrix X uncovers these vectors
  - **KLT**

Eigenvectors



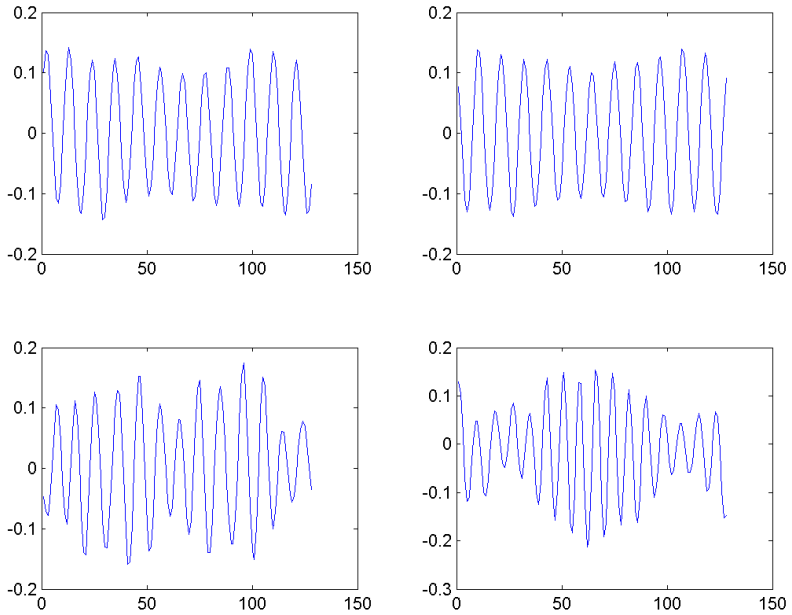
- Turns out: Eigenvectors of the *covariance* represent the major and minor axes of an ellipse centered at the *mean* which encloses the data most compactly

- PCA uncovers these vectors
- In practice, "Eigen faces" refers to *PCA* faces, and not KLT faces

What about sound?

- Finding Eigen bases for speech signals:

- Look like DFT/DCT
- Or wavelets



- DFTs are pretty good most of the time

Eigen Analysis

- Can often find surprising features in your data
- Trends, relationships, more
- Commonly used in recommender systems

- An interesting example..

Eigen Analysis

Figure1. Experiment setup @Wean Hall mechanical space. Pipe with arrow indicates a 10" diameter hot water pipe carrying pressurized hot water flow, on which piezoelectric sensors are installed every 10 ft. A National instruments data acquisition system is used to acquire and store the data for later processing.

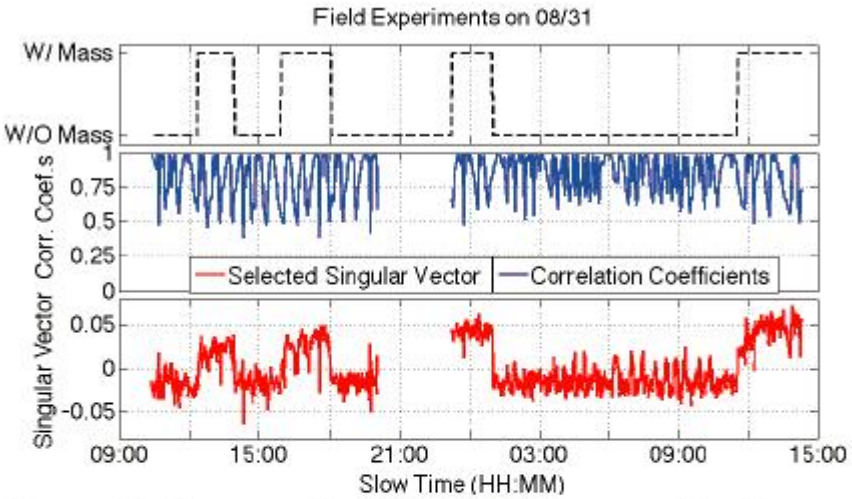


Figure 2. Damage detection results compared with conventional methods. **Top**: Ground truth of whether the pipe is damaged or not. **Middle**: Conventional method only captures temperature variations, and shows no indication of the presence of damage. **Bottom**: The SVD method clearly picks up the steps where damage are introduced and removed.

- Cheng Liu's research on pipes..
- SVD automatically separates useful and uninformative features