Dimensionality Reduction
Principal Component Analysis (PCA)

CS229: Machine Learning

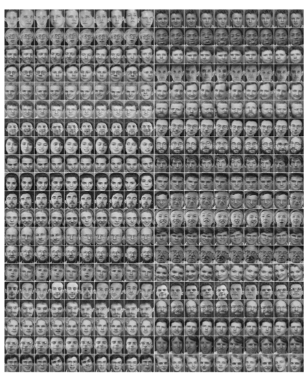
Carlos Guestrin
Stanford University

Slides include content developed by and co-developed with Emily Fox

Embedding

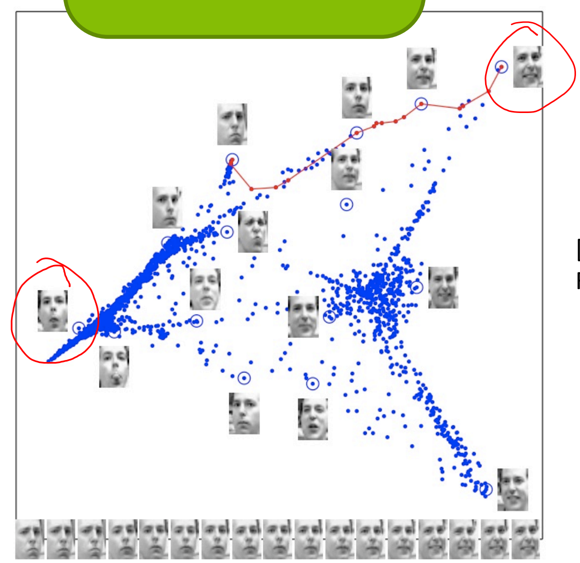
Example: Embedding images to visualize data

Data → PCA → Intelligence



Images with thousands or millions of pixels

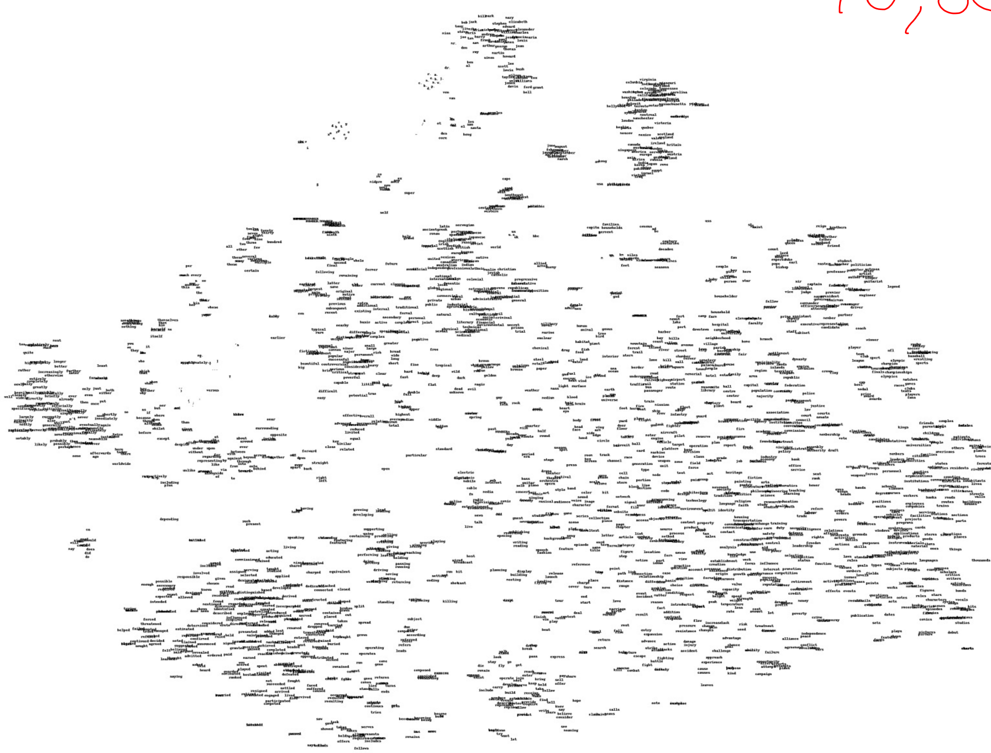
Can we give each image a coordinate, such that similar images are near each other?



[Saul & Roweis '03]

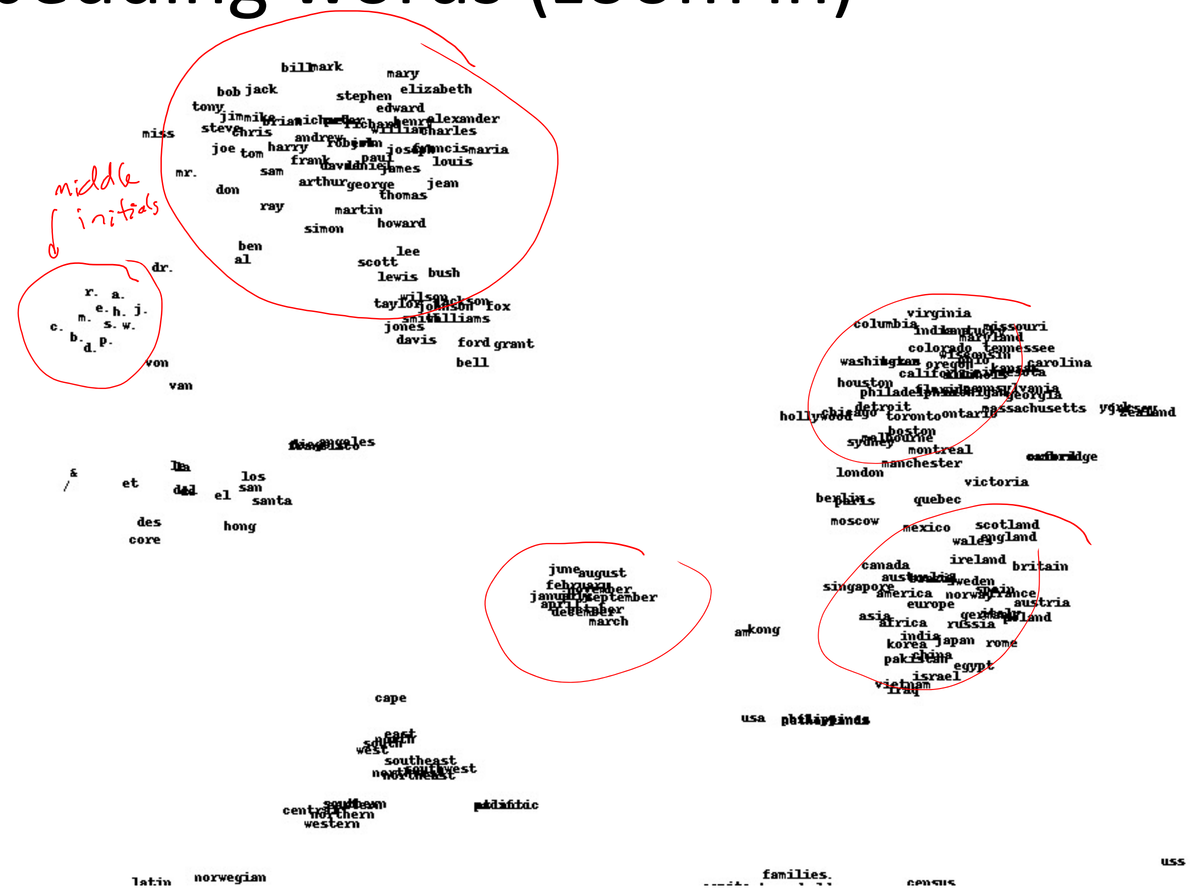
Embedding words

10,000 words



[Joseph Turian]

Embedding words (zoom in)



[Joseph Turian]

Dimensionality reduction

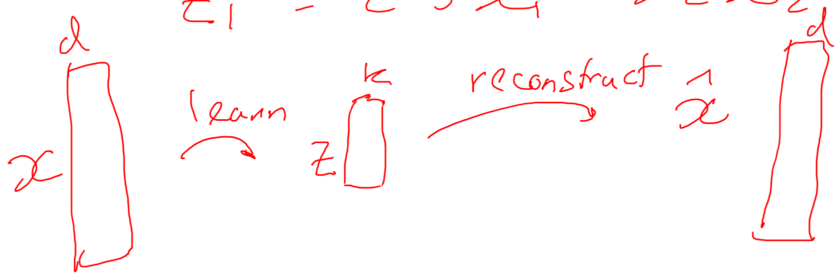
- Input data may have thousands or millions of dimensions!
  - e.g., text data
- **Dimensionality reduction**: represent data with fewer dimensions
  - easier learning – fewer parameters
  - visualization – hard to visualize more than 3D or 4D
  - discover "intrinsic dimensionality" of data
    - high dimensional data that is truly lower dimensional

Lower dimensional projections

$d \gg k$

- Rather than picking a subset of the features, we can create new features that are combinations of existing features

$z_1 = 2.5x_1 - 3.2x_2 - 3.1x_3 + \cdots$

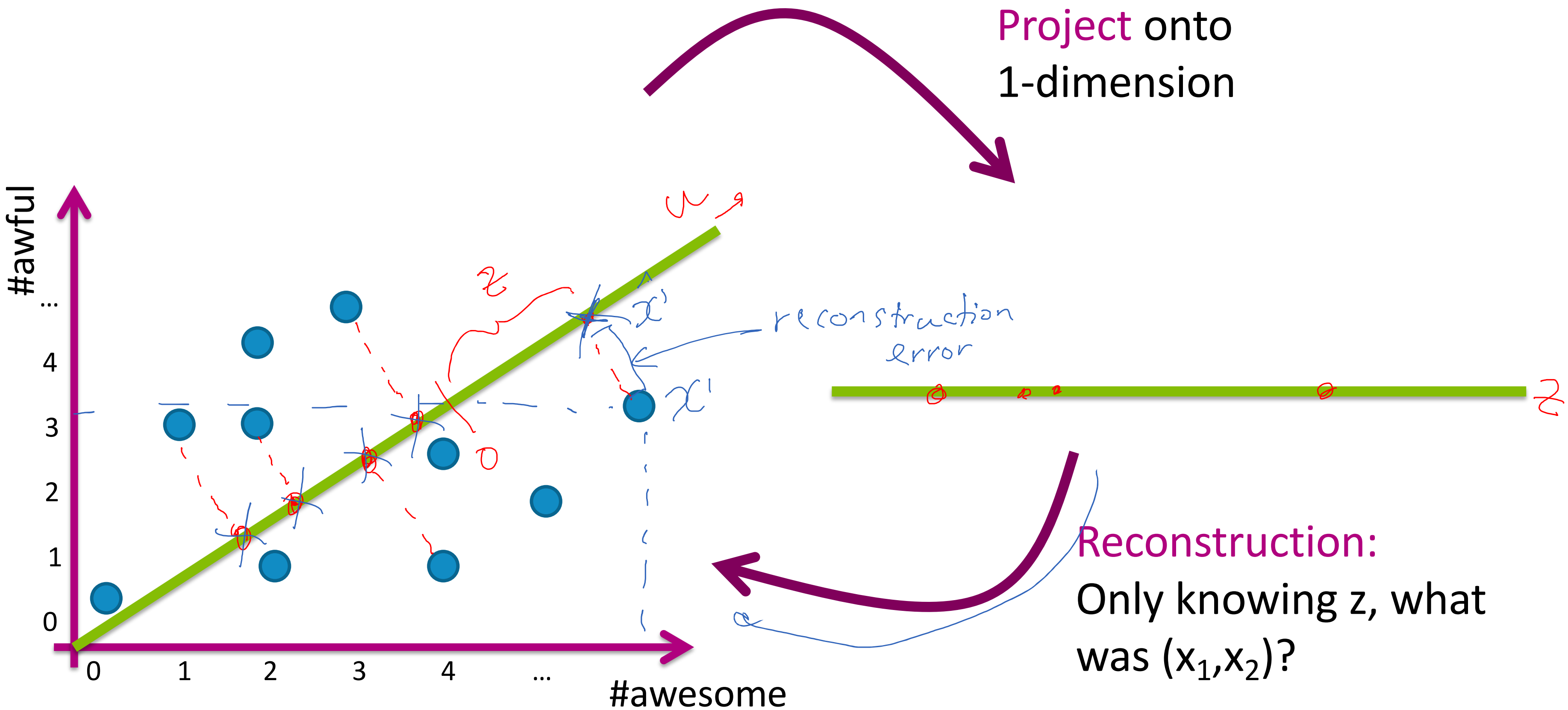


$\hat{x}$ close to x

$z = {}_k A_d \, x$

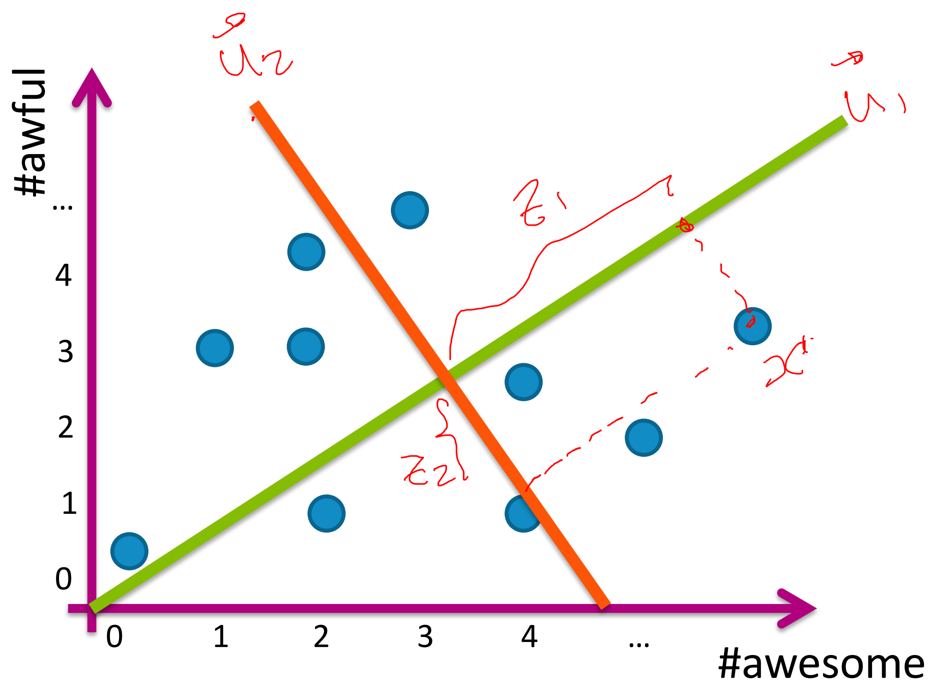
- Let's see this in the unsupervised setting
  - just **x**, but no y

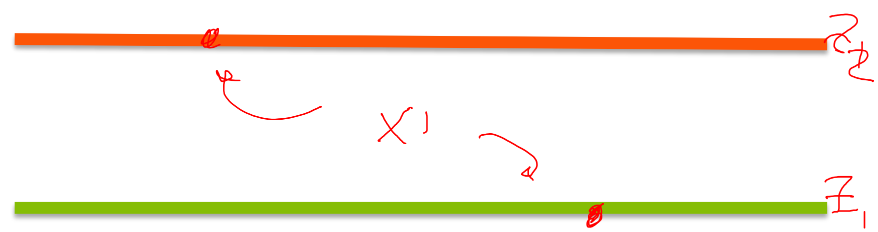
Linear projection and reconstruction



What if we project onto d vectors?

$x' = z_1 \vec{u}_1 + z_2 \vec{u}_2$ (ignoring offset)

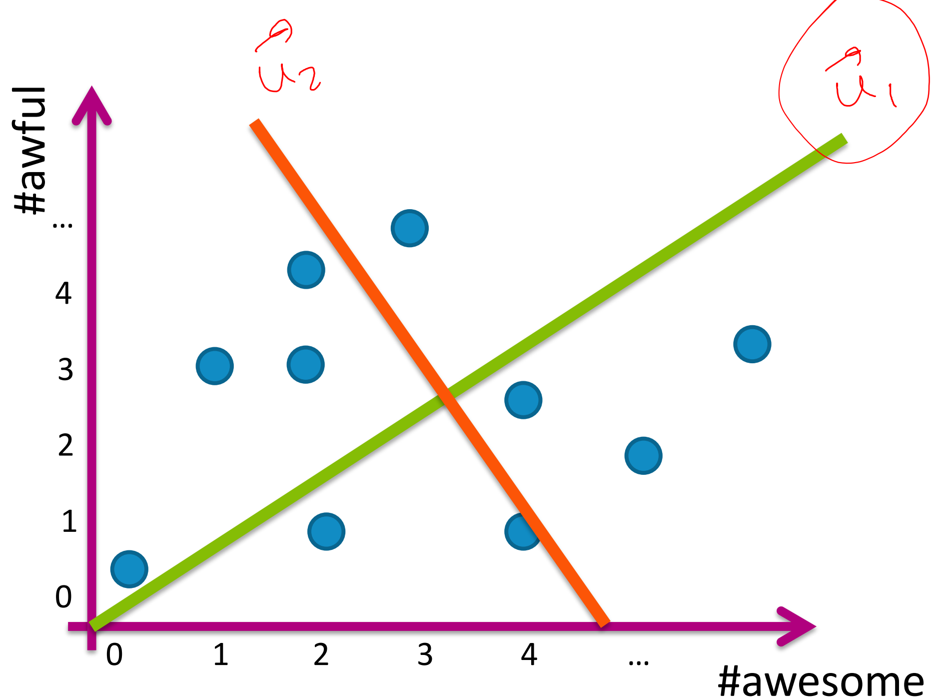




Perfect reconstruction!

If I had to choose one of these vectors, which do I prefer?

best projection



Principal component analysis (PCA) – Basic idea

- Project d-dimensional data into k-dimensional space while preserving as much information as possible:
  - e.g., project space of 10000 words into 3-dimensions
  - e.g., project 3-d into 2-d

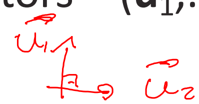
- Choose projection with minimum reconstruction error

"PCA explained visually"

http://setosa.io/ev/principal-component-analysis/

Linear projections, a review

- Project a point into a (lower dimensional) space:
  - **point**: $\mathbf{x} = (x_1,\dots,x_d)$ 100,000
  - **select a basis** – set of basis vectors – $(\mathbf{u}_1,\dots,\mathbf{u}_k)$ 100
    - we consider orthonormal basis:



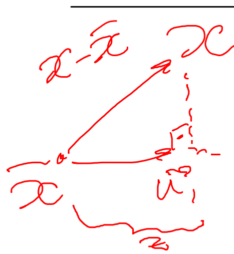
      - $\mathbf{u}_i \bullet \mathbf{u}_i = 1$, and $\mathbf{u}_i \bullet \mathbf{u}_j = 0$ for $i \neq j$

  mean of x
  - **select a center** – $\overline{\mathbf{x}}$, defines offset of space
  - **best coordinates** in lower dimensional space defined by dot-products: $(z_1,\dots,z_k)$, $z_i = (\mathbf{x}-\overline{\mathbf{x}}) \bullet \mathbf{u}_i$
    - minimum squared error

$$\hat{x} = \bar{x} + \sum_{j=1}^{k} z_j u_j$$

if $k = d$:

$$\hat{x} = x$$



$$z_1 = (x-\bar{x})\cdot \vec{u}_1 \quad z_1 = \arg\min_{z} \left((x-\bar{x}) - z\vec{u}_1\right)^2$$

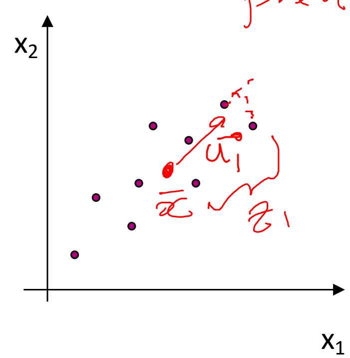
PCA finds projection that minimizes reconstruction error

- Given N data points: $\mathbf{x}^i = (x_1^i,\ldots,x_d^i)$, i=1…N
- Will represent each point as a projection:

index of datapoint

$$\underbrace{\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j}_{} \quad \text{and} \quad \underbrace{\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^i}_{\text{avg over data}} \qquad \underbrace{z_j^i = (\mathbf{x}^i - \bar{\mathbf{x}}) \cdot \mathbf{u}_j}_{\text{projection from previous slide}}$$

projection direction



- PCA:
  - Given k<<d, find $(\mathbf{u}_1,\ldots,\mathbf{u}_k)$ minimizing reconstruction error:

want to min $u_1,\ldots u_k$

$$error_k = \sum_{i=1}^{N} (\mathbf{x}^i - \hat{\mathbf{x}}^i)^2$$

Understanding the reconstruction error

- Note that $\mathbf{x}^i$ can be represented exactly by d-dimensional projection:

$$\mathbf{x}^i = \bar{\mathbf{x}} + \sum_{j=1}^{d} z_j^i \mathbf{u}_j$$

- Rewriting error:

$$\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j$$

$$z_j^i = (\mathbf{x}^i - \bar{\mathbf{x}}) \cdot \mathbf{u}_j$$

☐Given k<<d, find ($\mathbf{u}_1,\ldots,\mathbf{u}_k$) minimizing reconstruction error:

$$error_k = \sum_{i=1}^{N} (\mathbf{x}^i - \hat{\mathbf{x}}^i)^2$$

$$\min_u : \; error_k = \sum_{i=1}^{N} (x^i - \hat{x}^i)^2 = \sum_{i=1}^{N}\left[\bar{x} + \sum_{j=1}^{d} z_j^i u_j - \left(\bar{x} + \sum_{j=1}^{k} z_j^i u_j\right)\right]^2 = \sum_{i=1}^{N}\left[\sum_{j=k+1}^{d} z_j^i u_j\right]^2$$

$$= \sum_{i=1}^{N}\left[\sum_{j=k+1}^{d} z_j^i \, \cancel{u_j \cdot u_j}^{\,1} \, z_j^i + \sum_{j=k+1}^{d} \sum_{j' \neq j} z_j^i \, \cancel{u_j \cdot u_{j'}}^{\,0} \, z_{j'}^i\right]$$

$$= \sum_{i=1}^{N} \sum_{j=k+1}^{d} (z_j^i)^2 \leftarrow$$

minimizing reconstruction error $\equiv$ min square of thrown out coefficients !! ☺

Reconstruction error and covariance matrix

$$error_k = \sum_{i=1}^{N} \sum_{j=k+1}^{d} [\overbrace{\mathbf{u}_j \cdot (\mathbf{x}^i - \bar{\mathbf{x}})}^{z_j^i}]^2$$

$$= \sum_{i=1}^{N} \sum_{j=k+1}^{d} u_j^T (x^i - \bar{x})(x^i - \bar{x})^T u_j$$

push sum over i in, because u_j doesn't depend on i

$$= \sum_{j=k+1}^{d} u_j^T \underbrace{\left[\sum_{i=1}^{N} (x^i - \bar{x})(x^i - \bar{x})^T \right]}_{N\Sigma} u_j$$

$$= N \sum_{j=k+1}^{d} u_j^T \Sigma u_j$$

choose u_j that minimizes this error

$$\Sigma \overset{MLE}{=} \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}^i - \bar{\mathbf{x}})(\mathbf{x}^i - \bar{\mathbf{x}})^T$$

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \cdots \\ \vdots & \sigma_2^2 & \cdots \\ \vdots & & \ddots \end{pmatrix}$$

$$\sigma_{rs} \overset{MLE}{=} \frac{1}{N} \sum_{i=1}^{N} (x_r^i - \bar{x}_r)(x_s^i - \bar{x}_s)$$

matrix notation

Minimizing reconstruction error and eigen vectors

- Minimizing reconstruction error equivalent to picking orthonormal basis ($\mathbf{u}_1,\ldots,\mathbf{u}_d$) minimizing:

$$error_k = N \sum_{j=k+1}^{d} \mathbf{u}_j^T \Sigma \mathbf{u}_j = N \sum_{j=k+1}^{d} \lambda_j$$

if u_j are eigen vectors

$$error_k = N \sum_{j=k+1}^{d} u_j^T \Sigma u_j$$

- Eigen vector:

memory Lane:

$$\Sigma u = \lambda u$$

eigen vector

eigen value

if u_j is an eigen vector

$$u_j^T \Sigma u_j = \lambda_j u_j^T u_j = \lambda_j$$

- Minimizing reconstruction error equivalent to picking ($\mathbf{u}_{k+1},\ldots,\mathbf{u}_d$) to be eigen vectors with smallest eigen values

ignored dimensions

$\min_{u_1 \ldots u_k} error_k \equiv$ throwing out $u_{k+1}, \ldots, u_d$ with smallest eigen values of Σ

$\equiv$ keeping $u_1, \ldots, u_k$ with largest eigen values of Σ

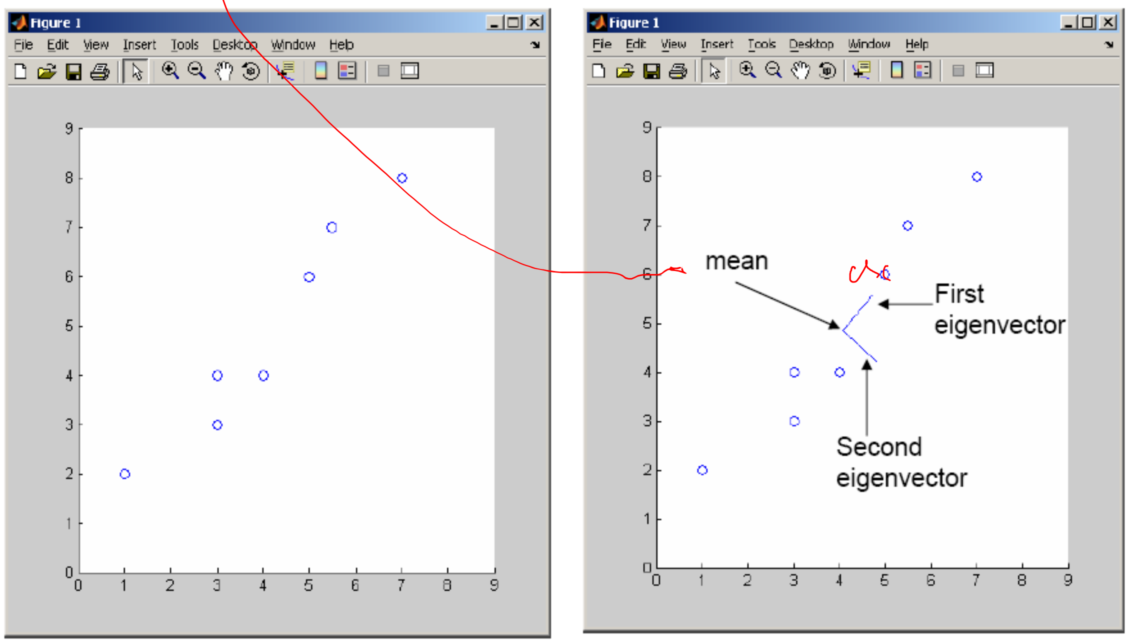
Basic PCA algoritm

- Start from N by d data matrix **X**
- **Recenter**: subtract mean from each row of **X**
  - $\mathbf{X_c} \leftarrow \mathbf{X} - \overline{\mathbf{X}}$
- **Compute covariance matrix**:
  - $\Sigma \leftarrow 1/N\ \mathbf{X_c}^T\ \mathbf{X_c}$
- Find **eigen vectors and values** of Σ
- **Principal components:** k eigen vectors with highest eigen values

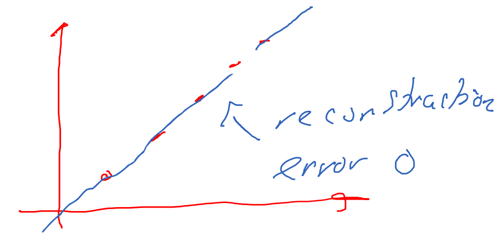
$$X_c = N \overset{k}{\left(\quad\right)} \quad x^i - \bar{x}$$

PCA example

$$\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j$$

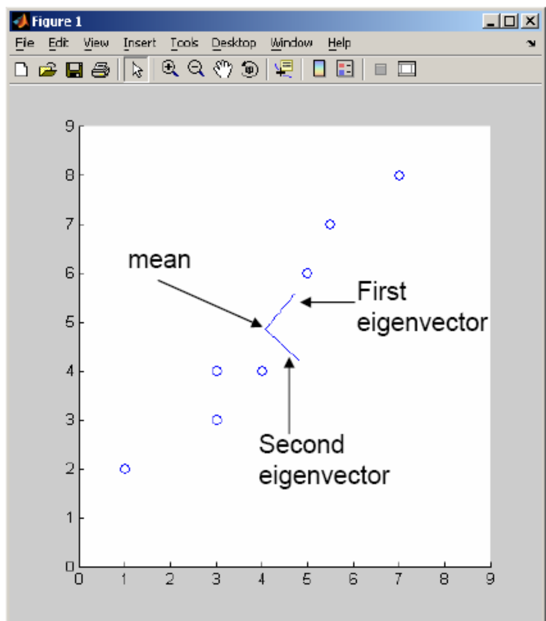


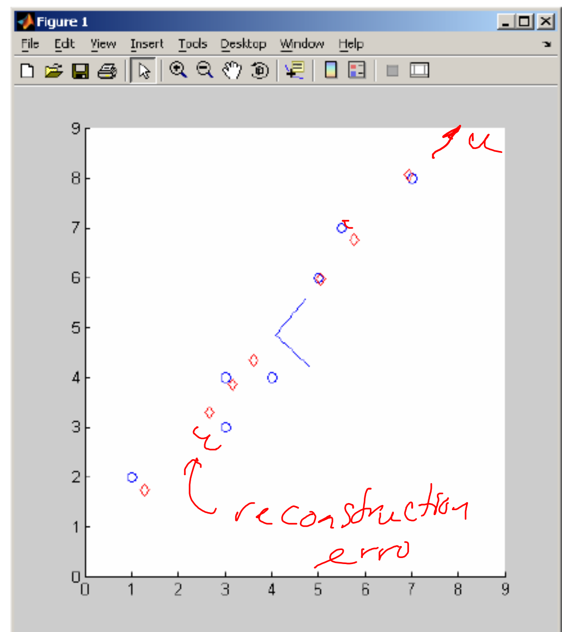
PCA example – reconstruction



$$\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j$$

only used first principal component





Eigenfaces [Turk, Pentland ’91]

- Input images:

$\bar{x}$

- Principal components:

short or longer

glass

Eigenfaces reconstruction

- Each image corresponds to adding 8 principal components:

mean
8 components 16 24 32

target face x

Scaling up

- Covariance matrix can be really big!
  - Σ is d by d ← 100 000 000 entries
  - Say, only 10000 features
  - finding eigenvectors is very slow…
- Use singular value decomposition (SVD)
  - finds to k eigenvectors, without forming Σ explicitly
  - great implementations available, e.g., python, R, Matlab svd

SVD

- Write $\mathbf{X} = \mathbf{W}\,\mathbf{S}\,\mathbf{V}^T$
  - $\mathbf{X} \leftarrow$ data matrix, one row per datapoint
  - $\mathbf{W} \leftarrow$ weight matrix, one row per datapoint – coordinate of $\mathbf{x}^i$ in eigenspace
  - $\mathbf{S} \leftarrow$ singular value matrix, diagonal matrix
    - in our setting each entry is eigenvalue λ_j
  - $\mathbf{V}^T \leftarrow$ singular vector matrix
    - in our setting each row is eigenvector $\mathbf{v}_j$

PCA using SVD algoritm

- Start from m by n data matrix **X**
- **Recenter**: subtract mean from each row of **X**
  - $\mathbf{X_c} \leftarrow \mathbf{X} - \overline{\mathbf{X}}$
- Call SVD algorithm on $\mathbf{X_c}$ – ask for k singular vectors
- **Principal components:** k singular vectors with highest singular values (rows of $\mathbf{V}^T$)
  - **Coefficients** become:

What you need to know

- Dimensionality reduction
  - why and when it's important
- Simple feature selection
- Principal component analysis
  - minimizing reconstruction error
  - relationship to covariance matrix and eigenvectors
  - using SVD