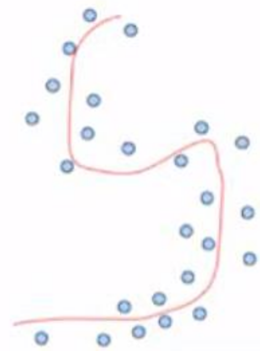
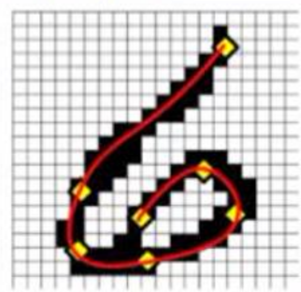
Principle Component Analysis

Curse of dimensionality

- Datasets typically high dimensional
  - vision: 10^4 pixels, text: 10^6 words
    - the way we observe / record them
  - true dimensionality often much lower
    - a manifold (sheet) in a high-d space
- Example: handwritten digits
  - 20 x 20 bitmap: $\{0,1\}^{400}$ possible events
    - will never see most of these events
    - actual digits: tiny fraction of events
  - true dimensionality:
    - possible variations of the pen-stroke





Curse of dimensionality (2)

- Machine learning methods are statistical by nature
  - count observations in various regions of some space
  - use counts to construct the predictor f(x)
  - e.g. decision trees: p_+/p_- in {o=rain,w=strong,T>28°}
  - text: #documents in {"hp" and "3d" and not "$" and …)
- As dimensionality grows: fewer observations per region
  - 1d: 3 regions, 2d: 3^2 regions, 1000d – hopeless
  - statistics need repetition
    - flip a coin once → head
    - P(head) = 100%?

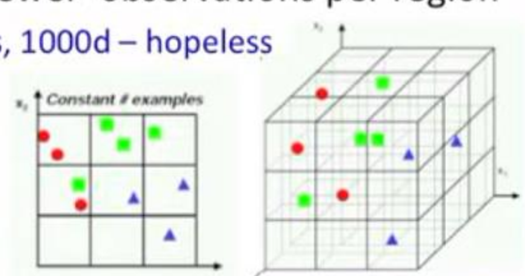
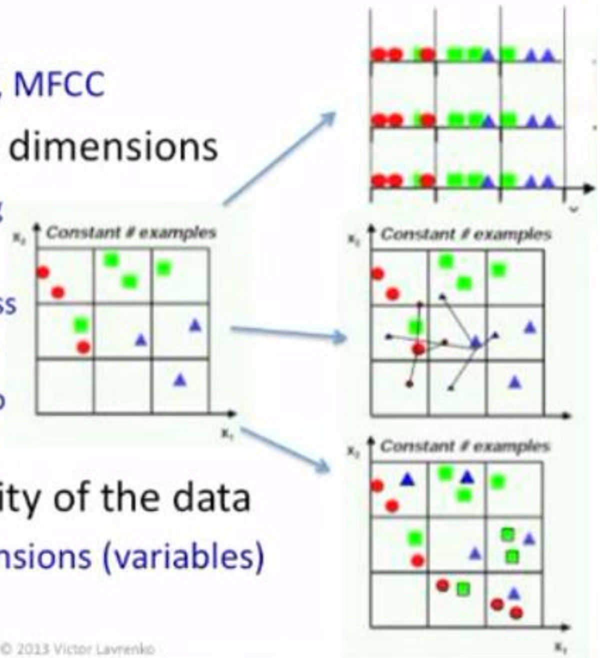


Figure © Ricardo Gutierrez-Osuna

Dealing with high dimensionality

- Use domain knowledge
  - feature engineering: SIFT, MFCC
- Make assumption about dimensions
  - independence: count along each dimension separately
  - smoothness: propagate class counts to neighboring regions
  - symmetry: e.g. invariance to order of dimensions: $x_1 \Leftrightarrow x_2$
- Reduce the dimensionality of the data
  - create a new set of dimensions (variables)



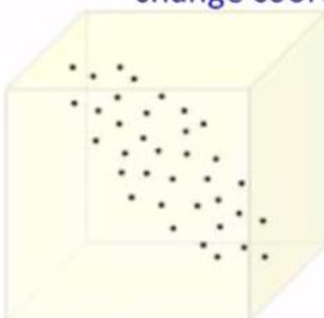
Dimensionality reduction

- Goal: represent instances with fewer variables
  - try to preserve as much structure in the data as possible
  - discriminative: only structure that affects class separability
- Feature selection
  - pick a subset of the original dimensions $X_1\ X_2\ X_3 \dots X_{d-1}\ X_d$
  - discriminative: pick good class "predictors" (e.g. gain)
- Feature extraction
  - construct a new set of dimensions $E_1\ E_2 \dots E_m$

    $E_i = f(X_1 \dots X_d)$
  - (linear) combinations of original $X_1\ X_2\ X_3 \dots X_d$

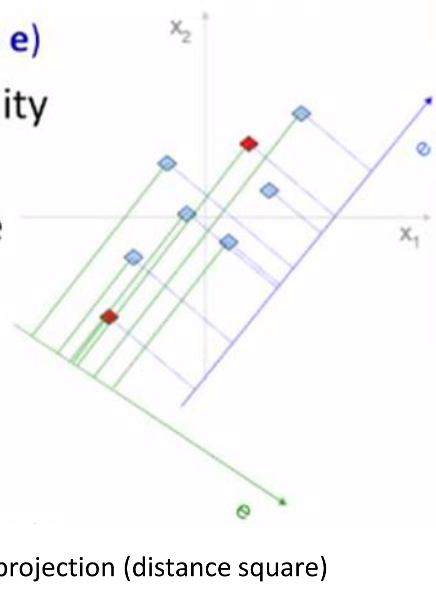
Principal Components Analysis

- Defines a set of principal components
  - 1^{st}: direction of the greatest variability in the data
  - 2^{nd}: perpendicular to 1^{st}, greatest variability of what's left
  - ... and so on until d (original dimensionality)
- First $m<<d$ components become m new dimensions
  - change coordinates of every data point to these dimensions



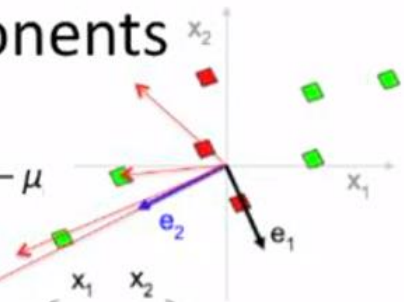
Why greatest variability?

- Example: reduce 2-dimensional data to 1-d
  - $\{x_1,x_2\} \rightarrow e'$ (along new axis $\mathbf{e}$)
- Pick e to maximize variability
- Reduces cases when two points are close in **e**-space but very far in (x,y)-space
- Minimizes distances between original points and their projections



Blue projection better than green projection (distance square)

Principal components



- "Center" the data at zero: $x_{i,a} = x_{i,a} - \mu$
  - subtract mean from each attribute
- Compute covariance matrix Σ
  - covariance of dimensions x_1 and x_2:
    - do x_1 and x_2 tend to increase together?
    - or does x_2 decrease as x_1 increases?

$$\begin{array}{ccc} & x_1 & x_2 \\ x_1 & 2.0 & 0.8 \\ x_2 & 0.8 & 0.6 \end{array} \qquad \mathrm{var}(a) = \frac{1}{n}\sum_{i=1}^{n} x_{ia}^2 \qquad \mathrm{cov}(b,a) = \frac{1}{n}\sum_{i=1}^{n} x_{ib} x_{ia}$$

- Multiply a vector by Σ: $\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix}\begin{pmatrix} -1 \\ +1 \end{pmatrix} \to \begin{pmatrix} -1.2 \\ -0.2 \end{pmatrix}$ again $\to \begin{pmatrix} -2.5 \\ -1.0 \end{pmatrix} \to \begin{pmatrix} -6.0 \\ -2.7 \end{pmatrix} \to \begin{pmatrix} -14.1 \\ -6.4 \end{pmatrix} \to \begin{pmatrix} -33.3 \\ -15.1 \end{pmatrix}$
  - turns towards direction of variance

  slope: 0.400 0.450 0.454 0.454
- Want vectors **e** which aren't turned: Σ **e** = λ **e**
  - e … eigenvectors of Σ, λ … corresponding eigenvalues
  - principal components = eigenvectors w. largest eigenvalues

Finding Principal Components

1. find eigenvalues by solving: det(Σ – λI) = 0

$$\det\begin{pmatrix} 2.0-\lambda & 0.8 \\ 0.8 & 0.6-\lambda \end{pmatrix} = (2-\lambda)(0.6-\lambda) - (0.8)(0.8) = \lambda^2 - 2.6\lambda + 0.56 = 0$$

$$\{\lambda_1, \lambda_2\} = \tfrac{1}{2}\left(2.6 \pm \sqrt{2.6^2 - 4 * 0.56}\right) = \{2.36, 0.23\}$$

2. find i^{th} eigenvector by solving: Σ $\mathbf{e}_i = \lambda_i \mathbf{e}_i$

$$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix}\begin{pmatrix} e_{1,1} \\ e_{1,2} \end{pmatrix} = 2.36\begin{pmatrix} e_{1,1} \\ e_{1,2} \end{pmatrix} \Rightarrow \begin{matrix} 2.0e_{1,1} + 0.8e_{1,2} = 2.36e_{1,1} \\ 0.8e_{1,1} + 0.6e_{1,2} = 2.36e_{1,2} \end{matrix} \Rightarrow e_{1,1} = 2.2e_{1,2}$$

$$e_1 \sim \begin{bmatrix} 2.2 \\ 1 \end{bmatrix}$$

want: $\|e_1\| = 1$

$$e_1 = \begin{bmatrix} 0.91 \\ 0.41 \end{bmatrix}$$

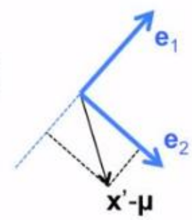
slope: 0.454

$$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix}\begin{pmatrix} e_{2,1} \\ e_{2,2} \end{pmatrix} = 0.23\begin{pmatrix} e_{2,1} \\ e_{2,2} \end{pmatrix} \Rightarrow e_2 = \begin{bmatrix} -0.41 \\ 0.91 \end{bmatrix}$$

3. 1^{st} PC: $\begin{bmatrix} 0.91 \\ 0.41 \end{bmatrix}$, 2^{nd} PC: $\begin{bmatrix} -0.41 \\ 0.91 \end{bmatrix}$

Projecting to new dimensions

- $\mathbf{e}_1 \ldots \mathbf{e}_m$ are new dimension vectors
- Have instance $\mathbf{x} = \{x_1 \ldots x_d\}$ (original coordinates)
- Want new coordinates $\mathbf{x}' = \{x'_1 \ldots x'_m\}$:
  1. "center" the instance (subtract the mean): $\mathbf{x}'-\boldsymbol{\mu}$
  2. "project" to each dimension: $(\mathbf{x}'-\boldsymbol{\mu})^T\mathbf{e}_j$ for j=1…m

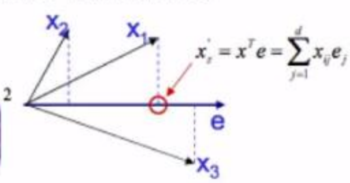


$$(\vec{x}-\vec{\mu}) = \qquad \left[(x_1-\mu_1)\ \ (x_2-\mu_2)\ \ \cdots\ \ (x_d-\mu_d)\right]$$

$$\begin{bmatrix} x_1' \\ x_2' \\ \vdots \\ x_m' \end{bmatrix} = \begin{bmatrix} (\vec{x}-\vec{\mu})^T\vec{e}_1 \\ (\vec{x}-\vec{\mu})^T\vec{e}_2 \\ \vdots \\ (\vec{x}-\vec{\mu})^T\vec{e}_m \end{bmatrix} = \begin{bmatrix} (x_1-\mu_1)e_{1,1}+(x_2-\mu_2)e_{1,2}+\cdots+(x_d-\mu_d)e_{1,d} \\ (x_1-\mu_1)e_{2,1}+(x_2-\mu_2)e_{2,2}+\cdots+(x_d-\mu_d)e_{2,d} \\ \vdots \\ (x_1-\mu_1)e_{m,1}+(x_2-\mu_2)e_{m,2}+\cdots+(x_d-\mu_d)e_{m,d} \end{bmatrix}$$

Direction of greatest variability

- Select dimension **e** which maximizes the variance
- Points $\boldsymbol{x}_i$ "projected" onto vector **e**:



- Variance of projections: $\frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{d} x_{ij}e_j - \mu\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{d} x_{ij}e_j\right)^2$
- Maximize variance
  - want unit length: $||\mathbf{e}||=1$
  - add Lagrange multiplier

$$V = \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{d} x_{ij}e_j\right)^2 - \lambda\left(\left(\sum_{k=1}^{d} e_j^2\right)-1\right)$$

$$\frac{\partial V}{\partial e_a} = \frac{2}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{d} x_{ij}e_j\right)x_{ia} - 2\lambda e_a = 0$$

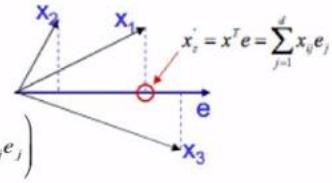
d

$$\Sigma e = \lambda e \Leftarrow \begin{Bmatrix} \sum_{j=1}^{d}\mathrm{cov}(1,j)e_j = \lambda e_1 \\ \vdots \\ \sum_{j=1}^{d}\mathrm{cov}(d,j)e_j = \lambda e_d \end{Bmatrix} \Leftarrow \text{hold for } a=1..d \quad 2\sum_{j=1}^{d} e_j \underbrace{\left(\frac{1}{n}\sum_{i=1}^{n} x_{ia}x_{ij}\right)}_{\text{covariance of a,j}} = 2\lambda e_a$$

e must be an eigenvector

a

Variance along eigenvector



Variance of projected points ($\mathbf{x}^T\mathbf{e}$):

$$\frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{d}x_{ij}e_j-\mu\right)^2=\frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{d}x_{ij}e_j\right)^2$$

$$\mu=\frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{d}x_{ij}e_j\right)=\sum_{j=1}^{d}\left(\frac{1}{n}\sum_{i=1}^{n}x_{ij}\right)e_j$$

$$=\frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{d}x_{ij}e_j\right)\left(\sum_{a=1}^{d}x_{ia}e_a\right)$$

$$=\sum_{a=1}^{d}\sum_{j=1}^{d}\left(\frac{1}{n}\sum_{i=1}^{n}x_{ia}x_{ij}\right)e_je_a$$

$$=\sum_{a=1}^{d}\left(\sum_{j=1}^{d}\operatorname{cov}(a,j)e_j\right)e_a \qquad \operatorname{cov}(a,j)=\frac{1}{n}\sum_{i=1}^{n}x_{ia}x_{ij}$$

$$=\sum_{a=1}^{d}(\lambda e_a)e_a \qquad \sum_{j=1}^{d}\operatorname{cov}(a,j)e_j=\lambda e_a$$

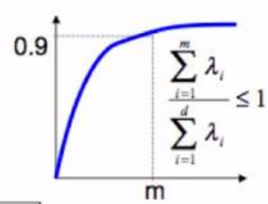
e is an eigenvector of the covariance matrix

$$=\lambda\|e\|^2=\lambda$$

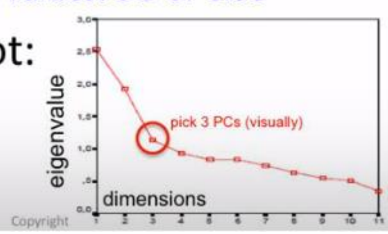
Biggest eigen value captures the maximum variance

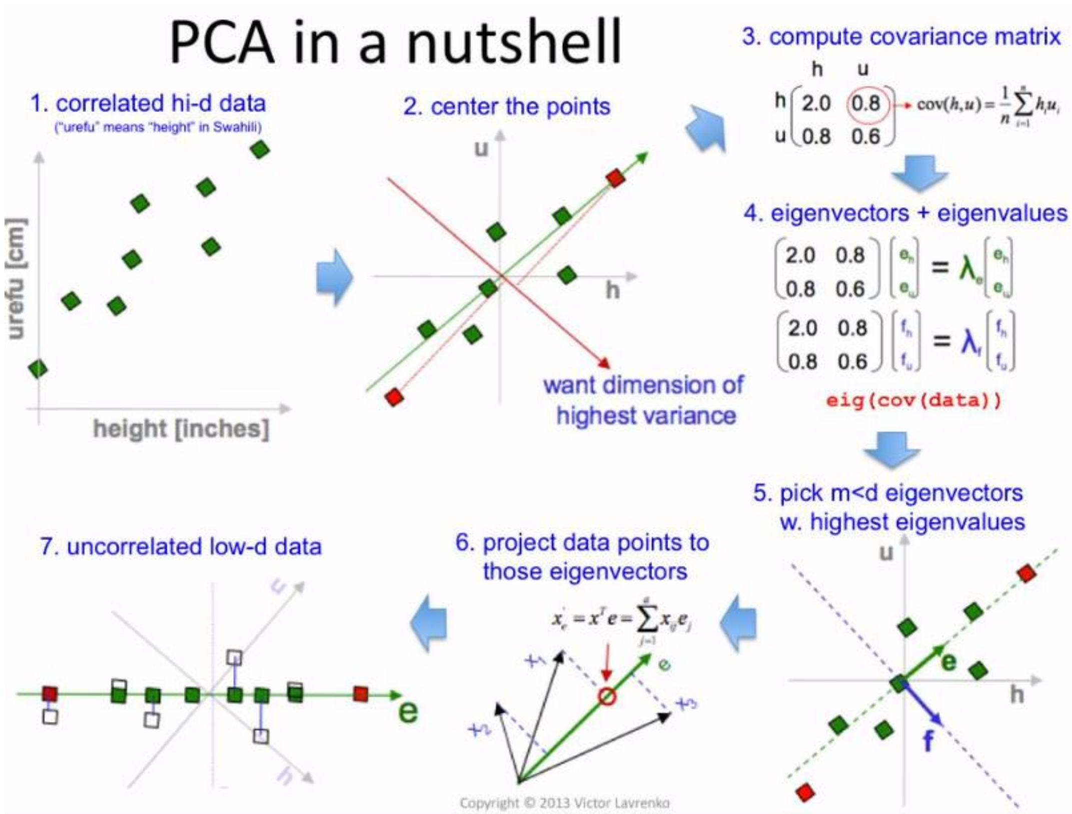
How many dimensions?

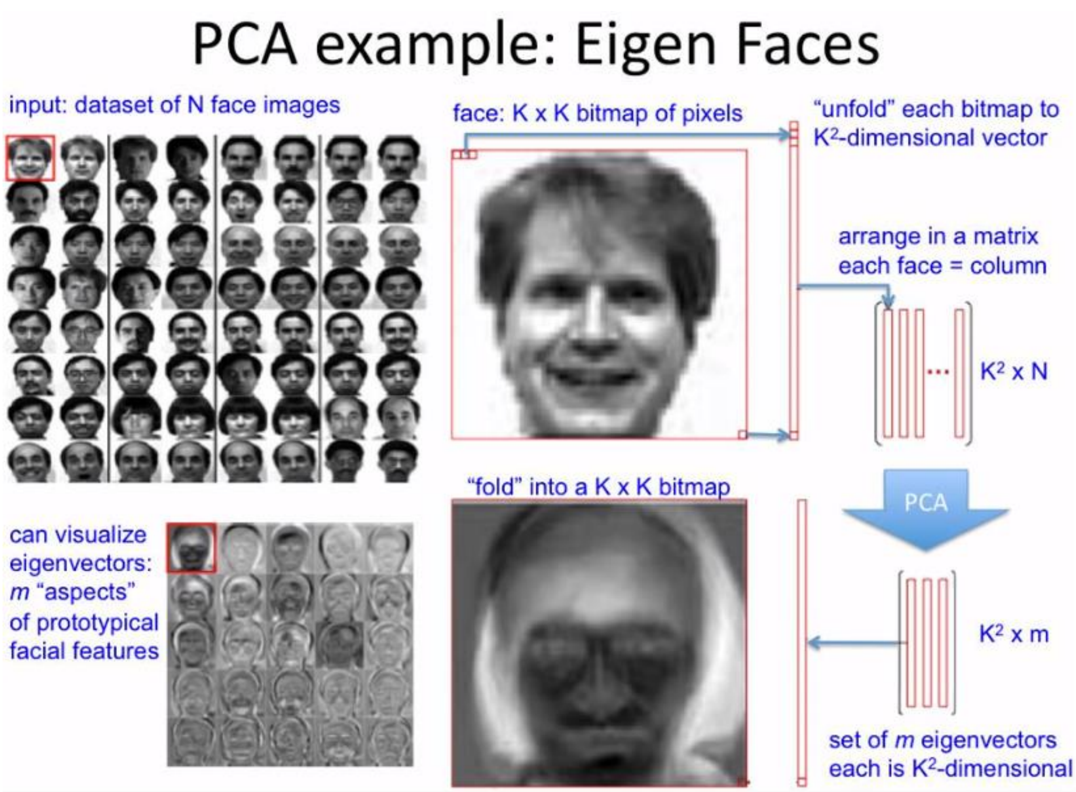
- Have: eigenvectors $\mathbf{e}_1 \ldots \mathbf{e}_d$ want: $m \ll d$
- Proved: eigenvalue λ_i = variance along $\mathbf{e}_i$
- Pick $\mathbf{e}_i$ that "explain" the most variance
  - sort eigenvectors s.t. $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$
  - pick first m eigenvectors which explain 90% or the total variance
    - typical threshold values: 0.9 or 0.95



- Or use a scree plot:
  - like K-means

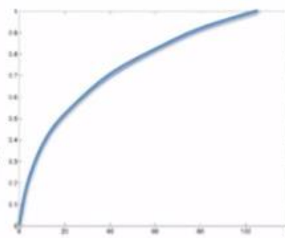



PCA in a nutshell
1. correlated hi-d data
("urefu" means "height" in Swahili)
urefu [cm]
height [inches]
2. center the points
u
h
want dimension of highest variance
3. compute covariance matrix
h u
h 2.0 0.8
u 0.8 0.6
$\mathrm{cov}(h,u) = \frac{1}{n}\sum_{i=1}^{n} h_i u_i$
4. eigenvectors + eigenvalues
$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix} \begin{pmatrix} e_h \\ e_u \end{pmatrix} = \lambda_e \begin{pmatrix} e_h \\ e_u \end{pmatrix}$
$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix} \begin{pmatrix} f_h \\ f_u \end{pmatrix} = \lambda_f \begin{pmatrix} f_h \\ f_u \end{pmatrix}$
eig(cov(data))
5. pick m<d eigenvectors w. highest eigenvalues
u
e
h
f
6. project data points to those eigenvectors
$x_e' = x^T e = \sum_{j=1}^{d} x_{ij} e_j$
e
7. uncorrelated low-d data
e
Copyright © 2013 Victor Lavrenko


PCA example: Eigen Faces
input: dataset of N face images
face: K x K bitmap of pixels
"unfold" each bitmap to K²-dimensional vector
arrange in a matrix
each face = column
K² x N
PCA
"fold" into a K x K bitmap
K² x m
set of m eigenvectors
each is K²-dimensional
can visualize eigenvectors: m "aspects" of prototypical facial features

Eigen Faces: Projection

- Project new face to space of eigen-faces
- Represent vector as a linear combination of principal components
- How many do we need?



(Eigen) Face Recognition

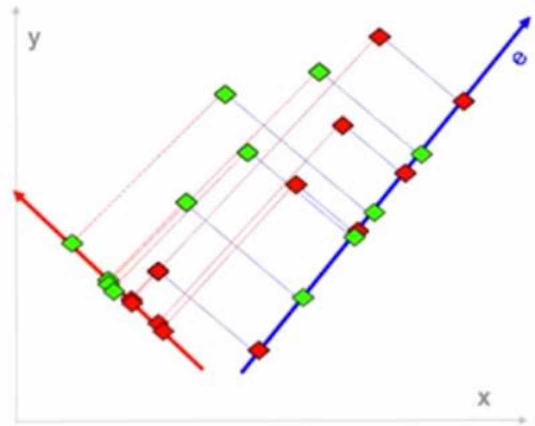
- Face similarity
  - in the reduced space
  - insensitive to lighting expression, orientation
- Projecting new "faces"
  - everything is a face

new face

projected to eigenfaces

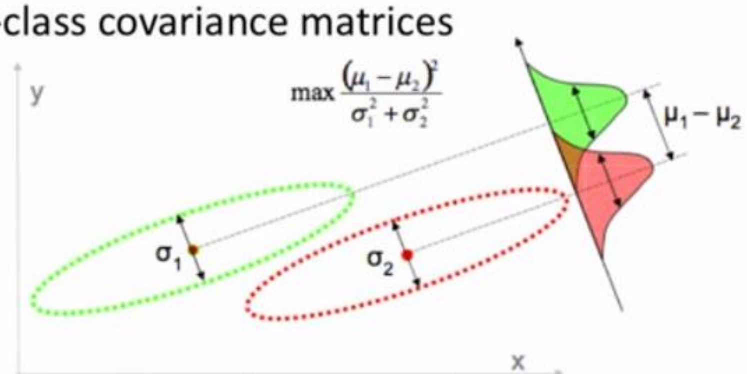
PCA and classification

- PCA is unsupervised
  - maximizes overall variance of the data along a small set of directions
  - does not know anything about class labels
  - can pick direction that makes it hard to separate classes
- Discriminative approach
  - look for a dimension that makes it easy to separate classes



Linear Discriminant Analysis

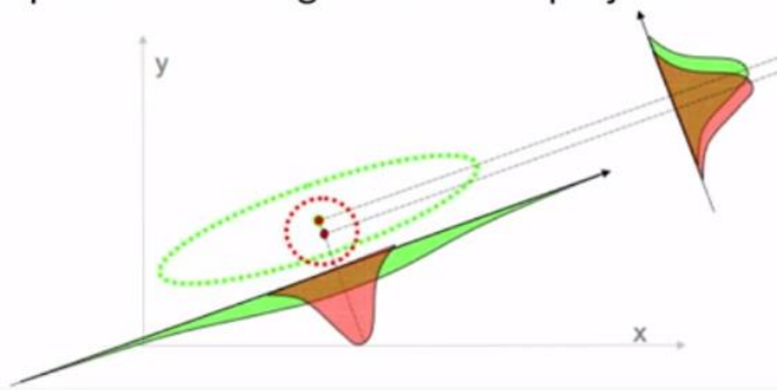
- LDA: pick a new dimension that gives:
  - maximum separation between means of projected classes
  - minimum variance within each projected class
- Solution: eigenvectors based on between-class and within-class covariance matrices



Useful for classification task

PCA vs. LDA

- LDA not guaranteed to be better for classification
  - assumes classes are unimodal Gaussians
  - fails when discriminatory information is not in the mean, but in the variance of the data
- Example where PCA gives a better projection:



Dimensionality reduction

- Pros
  - reflects our intuitions about the data
  - allows estimating probabilities in high-dimensional data
    - no need to assume independence etc.
  - dramatic reduction in size of data
    - faster processing (as long as reduction is fast), smaller storage
- Cons
  - too expensive for many applications (Twitter, web)
  - disastrous for tasks with fine-grained classes
  - understand assumptions behind the methods (linearity etc.)
    - there may be better ways to deal with sparseness

Summary

- True dimensionality << observed dimensionality
- High dimensionality → sparse, unstable estimates
- Dealing with high dimensionality:
  - use domain knowledge
  - make an assumption: independence / smoothness / symmetry
  - dimensionality reduction: feature selection / feature extraction
- Principal Components Analysis (PCA)
  - picks dimensions that maximize variability
    - eigenvectors of the covariance matrix
  - examples: Eigen Faces
  - variant for classification: Linear Discriminant Analysis