CS840a
Learning and Computer Vision
Prof. Olga Veksler

Lecture 4

Curse of Dimensionality,
Dimensionality Reduction with PCA

Today

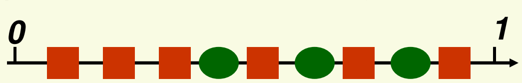
- Problems of high dimensional data, “the curse of dimensionality”
  - running time
  - overfitting
  - number of samples required
- Dimensionality Reduction Methods
  - Principle Component Analysis (today)

Curse of Dimensionality: Complexity

- Complexity (running time) increases with dimension d
- A lot of methods have at least $O(nd^2)$ complexity, where n is the number of samples
  - For example if we need to estimate covariance matrix
- So as d becomes large, $O(nd^2)$ complexity may be too costly

Curse of Dimensionality: Number of Samples

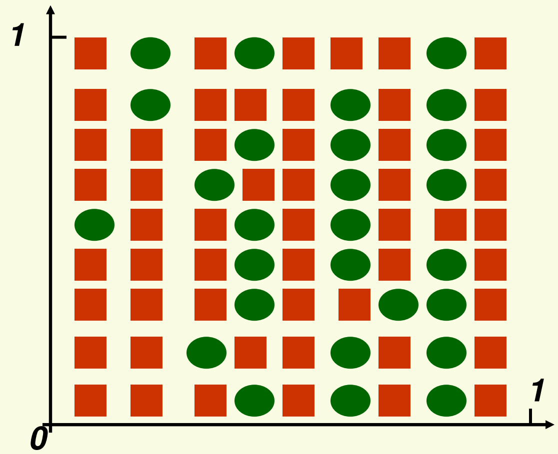
- Suppose we want to use the nearest neighbor approach with $k = 1$ (***1NN***)
- Suppose we start with only one feature



- This feature is not discriminative, i.e. it does not separate the classes well
- We decide to use 2 features. For the 1NN method to work well, need a lot of samples, i.e. samples have to be dense
- To maintain the same density as in 1D (9 samples per unit length), how many samples do we need?

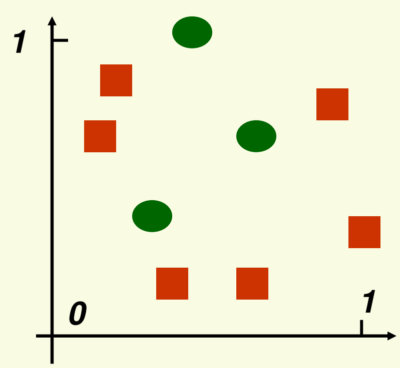
Curse of Dimensionality: Number of Samples

- We need 9^2 samples to maintain the same density as in ***1D***



Curse of Dimensionality: Number of Samples

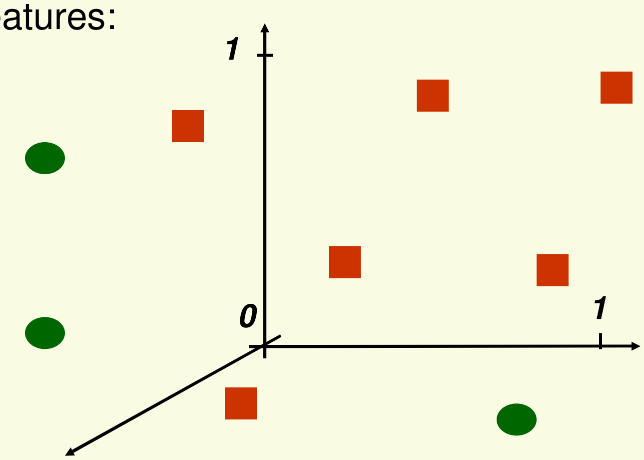
- Of course, when we go from 1 feature to 2, no one gives us more samples, we still have 9



- This is way too sparse for ***1NN*** to work well

Curse of Dimensionality: Number of Samples

- Things go from bad to worse if we decide to use 3 features:



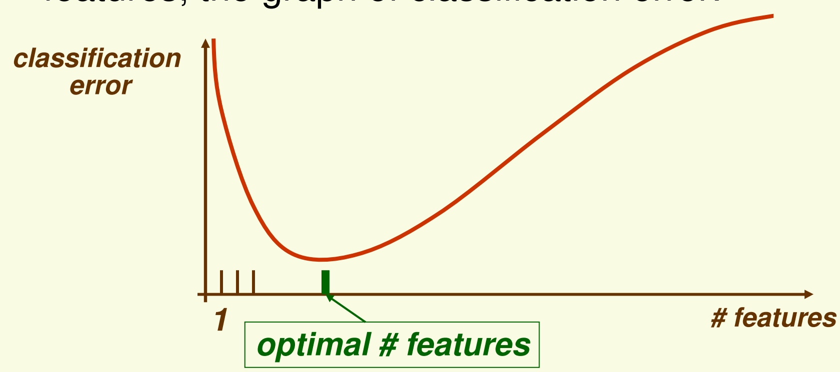
- If ***9*** was dense enough in 1D, in 3D we need 9^3=***729*** samples!

Curse of Dimensionality: Number of Samples

- In general, if ***n*** samples is dense enough in ***1D***
- Then in ***d*** dimensions we need n^d samples!
- And n^d grows really really fast as a function of ***d***
- Common pitfall:
  - If we can't solve a problem with a few features, adding more features seems like a good idea
  - However the number of samples usually stays the same
  - The method with more features is likely to perform worse instead of expected better

Curse of Dimensionality: Number of Samples

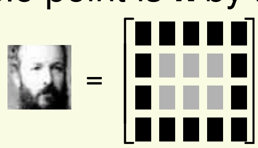
- For a fixed number of samples, as we add features, the graph of classification error:



- Thus for each fixed sample size ***n***, there is the optimal number of features to use

The Curse of Dimensionality

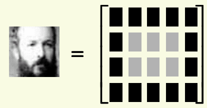
- We should try to avoid creating lot of features
- Often no choice, problem starts with many features
- Example: Face Detection
  - One sample point is ***k*** by ***m*** array of pixels



  - Feature extraction is not trivial, usually every pixel is taken as a feature
  - Typical dimension is 20 by 20 = 400
  - Suppose ***10*** samples are dense enough for 1 dimension. Need only $\boldsymbol{10^{400}}$ samples

The Curse of Dimensionality

- Face Detection, dimension of one sample point is ***km***



- The fact that we set up the problem with ***km*** dimensions (features) does not mean it is really a ***km***-dimensional problem
- Space of all ***k*** by ***m*** images has ***km*** dimensions
- Space of all ***k*** by ***m*** faces must be much smaller, since faces form a tiny fraction of all possible images
- Most likely we are not setting the problem up with the right features
- If we used better features, we are likely need much less than ***km***-dimensions

Dimensionality Reduction

- High dimensionality is challenging and redundant
- It is natural to try to reduce dimensionality
- Reduce dimensionality by feature combination: combine old features ***x*** to create new features ***y***

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \rightarrow f\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}\right) = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix} = y \quad \text{with } k < d$$

- For example,

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 + x_2 \\ x_3 + x_4 \end{bmatrix} = y$$

- Ideally, the new vector ***y*** should retain from ***x*** all information important for classification

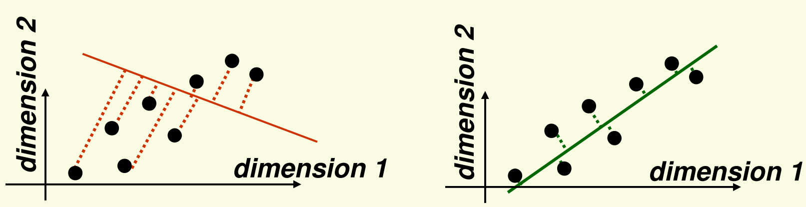
Dimensionality Reduction

- The best $\boldsymbol{f}(\boldsymbol{x})$ is most likely a non-linear function
- Linear functions are easier to find though
- For now, assume that $\boldsymbol{f}(\boldsymbol{x})$ is a linear mapping
- Thus it can be represented by a matrix $\boldsymbol{W}$:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \Rightarrow W \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} w_{11} & \cdots & w_{1d} \\ \vdots & & \vdots \\ w_{k1} & \cdots & w_{kd} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix} \quad \text{with } k < d$$

Principle Component Analysis (PCA)

- Main idea: seek most accurate data representation in a lower dimensional space
- Example in 2-D
  - Project data to 1-D subspace (a line) which minimize the projection error



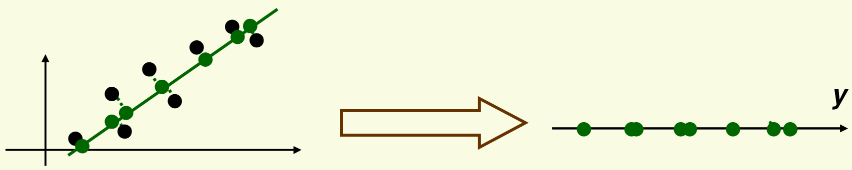
large projection errors, bad line to project to

small projection errors, good line to project to

- Notice that the the good line to use for projection lies in the direction of largest variance

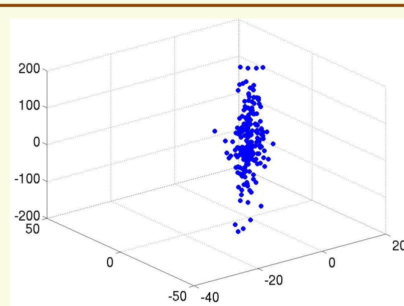
PCA

- After the data is projected on the best line, need to transform the coordinate system to get 1D representation for vector ***y***

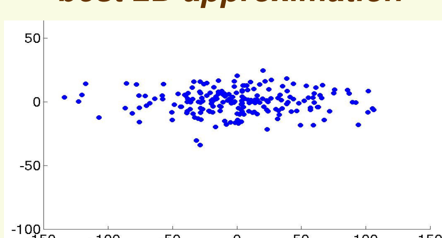


- Note that new data ***y*** has the same variance as old data ***x*** in the direction of the green line
- PCA preserves largest variances in the data. We will prove this statement, for now it is just an intuition of what PCA will do

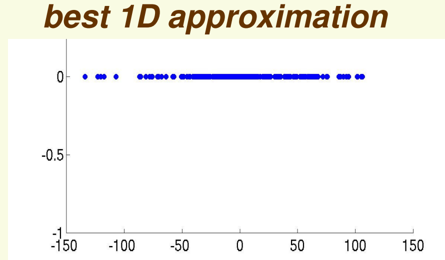
PCA: Approximation of Elliptical Cloud in 3D



best 2D approximation

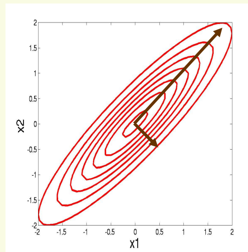


best 1D approximation



PCA

- What is the direction of largest variance in data?
- Recall that if ***x*** has multivariate distribution N(μ,Σ), direction of largest variance is given by eigenvector corresponding to the largest eigenvalue of Σ

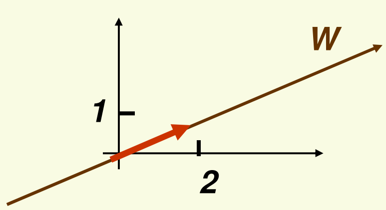


- This is a hint that we should be looking at the covariance matrix of the data (note that PCA can be applied to distributions other than Gaussian)

PCA: Linear Algebra for Derivation

- Let ***V*** be a ***d*** dimensional linear space, and ***W*** be a ***k*** dimensional linear subspace of ***V***
- We can always find a set of ***d*** dimensional vectors $\{e_1, e_2, \ldots, e_k\}$ which forms an orthonormal basis for ***W***
  - $\langle e_i, e_j \rangle = 0$ if ***i*** is not equal to ***j*** and $\langle e_i, e_i \rangle = 1$
- Thus any vector in ***W*** can be written as

$$\alpha_1 e_1 + \alpha_2 e_2 + \ldots + \alpha_k e_k = \sum_{i=1}^{k} \alpha_i e_i \quad \text{for scalars } \alpha_1, \ldots, \alpha_k$$

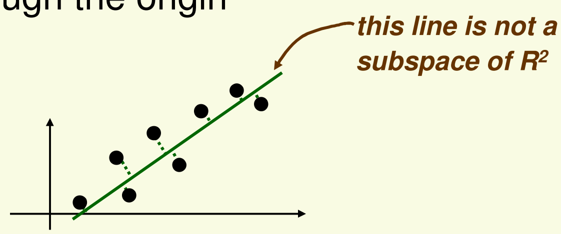


Let ***V*** = R^2 and ***W*** be the line x-2y=0. Then the orthonormal basis for W is

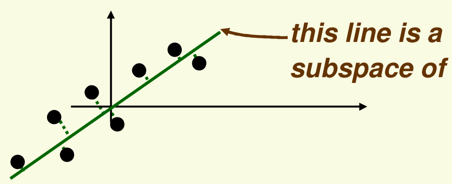
$$\left\{ \begin{bmatrix} 2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix} \right\}$$

PCA: Linear Algebra for Derivation

- Recall that subspace ***W*** contains the zero vector, i.e. it goes through the origin



- For derivation, it will be convenient to project to subspace ***W***: thus we need to shift everything

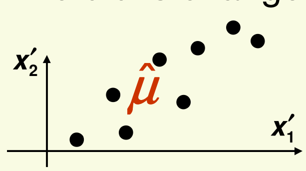


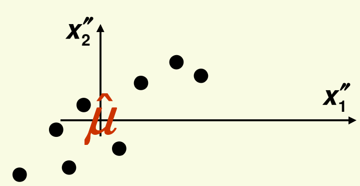
PCA Derivation: Shift by the Mean Vector

- Before PCA, subtract sample mean from the data

$$\boldsymbol{x} - \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i = \boldsymbol{x} - \hat{\mu}$$

- The new data has zero mean: E(***X***-E(***X***)) = E(***X***)-E(***X***) = 0
- All we did is change the coordinate system





- Another way to look at it:
  - first step of getting ***y*** is to subtract the mean of ***x***

$$\boldsymbol{x} \rightarrow \boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{x} - \hat{\mu})$$

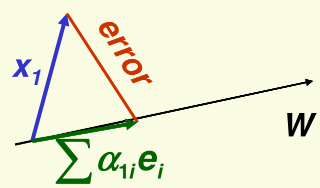
PCA: Derivation

- We want to find the most accurate representation of data $D=\{x_1,x_2,\ldots,x_n\}$ in some subspace W which has dimension $k < d$
- Let $\{e_1,e_2,\ldots,e_k\}$ be the orthonormal basis for W. Any vector in W can be written as $\sum_{i=1}^{k} \alpha_i e_i$
- Thus x_1 will be represented by some vector in W

$$\sum_{i=1}^{k} \alpha_{1i} e_i$$

- Error this representation:

$$error = \left\| x_1 - \sum_{i=1}^{k} \alpha_{1i} e_i \right\|^2$$



PCA: Derivation

- To find the total error, we need to sum over all x_j's
- Any x_j can be written as $\sum_{i=1}^{k} \alpha_{ji} e_i$
- Thus the total error for representation of all data D is:

sum over all data points

$$\underbrace{J(e_1,\ldots,e_k,\alpha_{11},\ldots\alpha_{nk})}_{\text{unknowns}} = \sum_{j=1}^{n} \underbrace{\left\| x_j - \sum_{i=1}^{k} \alpha_{ji} e_i \right\|^2}_{\text{error at one point}}$$

PCA: Derivation

- To minimize J, need to take partial derivatives and also enforce constraint that $\{e_1, e_2, \ldots, e_k\}$ are orthogonal

$$J(e_1,\ldots,e_k,\alpha_{11},\ldots\alpha_{nk}) = \sum_{j=1}^{n} \left\| x_j - \sum_{i=1}^{k} \alpha_{ji} e_i \right\|^2$$

- Let us simplify J first

$$J(e_1,\ldots,e_k,\alpha_{11},\ldots\alpha_{nk}) = \sum_{j=1}^{n} \left\| x_j \right\|^2 - 2\sum_{j=1}^{n} x_j^t \left(\sum_{i=1}^{k} \alpha_{ji} e_i \right) + \sum_{j=1}^{n} \sum_{i=1}^{k} \alpha_{ji}^2$$

$$= \sum_{j=1}^{n} \left\| x_j \right\|^2 - 2\sum_{j=1}^{n} \sum_{i=1}^{k} \alpha_{ji} x_j^t e_i + \sum_{j=1}^{n} \sum_{i=1}^{k} \alpha_{ji}^2$$

PCA: Derivation

$$J(e_1,\ldots,e_k,\alpha_{11},\ldots\alpha_{nk}) = \sum_{j=1}^{n} \left\| x_j \right\|^2 - 2\sum_{j=1}^{n} \sum_{i=1}^{k} \alpha_{ji} x_j^t e_i + \sum_{j=1}^{n} \sum_{i=1}^{k} \alpha_{ji}^2$$

- First take partial derivatives with respect to α_{ml}

$$\frac{\partial}{\partial \alpha_{ml}} J(e_1,\ldots,e_k,\alpha_{11},\ldots\alpha_{nk}) = -2 x_m^t e_l + 2\alpha_{ml}$$

- Thus the optimal value for α_{ml} is

$$-2 x_m^t e_l + 2\alpha_{ml} = 0 \Rightarrow \alpha_{ml} = x_m^t e_l$$

PCA: Derivation

$$J(e_1,\dots,e_k,\alpha_{11},\dots\alpha_{nk}) = \sum_{j=1}^{n}\|x_j\|^2 - 2\sum_{j=1}^{n}\sum_{i=1}^{k}\alpha_{ji}x_j^t e_i + \sum_{j=1}^{n}\sum_{i=1}^{k}\alpha_{ji}^2$$

- Plug the optimal value for $\alpha_{ml} = x^t_m e_l$ back into J

$$J(e_1,\dots,e_k) = \sum_{j=1}^{n}\|x_j\|^2 - 2\sum_{j=1}^{n}\sum_{i=1}^{k}(x_j^t e_i)x_j^t e_i + \sum_{j=1}^{n}\sum_{i=1}^{k}(x_j^t e_i)^2$$

- Can simplify J

$$J(e_1,\dots,e_k) = \sum_{j=1}^{n}\|x_j\|^2 - \sum_{j=1}^{n}\sum_{i=1}^{k}(x_j^t e_i)^2$$

PCA: Derivation

$$J(e_1,\dots,e_k) = \sum_{j=1}^{n}\|x_j\|^2 - \sum_{j=1}^{n}\sum_{i=1}^{k}(x_j^t e_i)^2$$

- Rewrite J using $(a^t b)^2 = (a^t b)(a^t b) = (b^t a)(a^t b) = b^t(aa^t)b$

$$J(e_1,\dots,e_k) = \sum_{j=1}^{n}\|x_j\|^2 - \sum_{i=1}^{k}e_i^t\left(\sum_{j=1}^{n}(x_j x_j^t)\right)e_i$$

$$= \sum_{j=1}^{n}\|x_j\|^2 - \sum_{i=1}^{k}e_i^t S e_i$$

- Where $S = \sum_{j=1}^{n} x_j x_j^t$
- S is called the scatter matrix, it is just n-1 times the sample covariance matrix we have seen before

$$\hat{\Sigma} = \frac{1}{n-1}\sum_{j=1}^{n}(x_j - \hat{\mu})(x_j - \hat{\mu})^t$$

PCA: Derivation

$$J(e_1,\dots,e_k) = \underbrace{\sum_{j=1}^{n} \|x_j\|^2}_{constant} - \sum_{i=1}^{k} e_i^t S e_i$$

- Minimizing J is equivalent to maximizing $\sum_{i=1}^{k} e_i^t S e_i$
- We should also enforce constraints $e_i^t e_i = 1$ for all i
- Use the method of Lagrange multipliers, incorporate the constraints with undetermined $\lambda_1, \dots, \lambda_k$
- Need to maximize new function u

$$u(e_1,\dots,e_k) = \sum_{i=1}^{k} e_i^t S e_i - \sum_{j=1}^{k} \lambda_j \left(e_j^t e_j - 1\right)$$

PCA: Derivation

- If x is a vector and $f(x) = f(x_1, \dots, x_d)$ is a function, to simplify notation, define

$$\frac{d}{dx} f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix}$$

- It can be shown that $\frac{d}{dx}\left(x^t x\right) = 2x$
- If A is a symmetric matrix, it can be shown that

$$\frac{d}{dx}\left(x^t A x\right) = 2Ax$$

PCA: Derivation

$$u(e_1,\ldots,e_k) = \sum_{i=1}^{k} e_i^t S e_i - \sum_{j=1}^{k} \lambda_j \left(e_j^t e_j - 1\right)$$

- Compute the partial derivatives with respect to e_m

$$\frac{\partial}{\partial e_m} u(e_1,\ldots,e_k) = 2Se_m - 2\lambda_m e_m = 0$$

Note: e_m *is a vector, what we are really doing here is taking partial derivatives with respect to each element of* e_m *and then arranging them up in a linear equation*

- Thus λ_m and e_m are eigenvalues and eigenvectors of scatter matrix S

$$Se_m = \lambda_m e_m$$

PCA: Derivation

$$J(e_1,\ldots,e_k) = \sum_{j=1}^{n} \left\| x_j \right\|^2 - \sum_{i=1}^{k} e_i^t S e_i$$

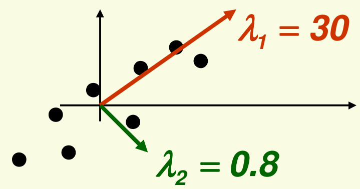
- Let's plug e_m back into J and use $Se_m = \lambda_m e_m$

$$J(e_1,\ldots,e_k) = \sum_{j=1}^{n} \left\| x_j \right\|^2 - \sum_{i=1}^{k} \lambda_i \left\| e_i \right\|^2 = \underbrace{\sum_{j=1}^{n} \left\| x_j \right\|^2}_{\text{constant}} - \sum_{i=1}^{k} \lambda_i$$

- Thus to minimize J take for the basis of W the k eigenvectors of S corresponding to the k largest eigenvalues

PCA

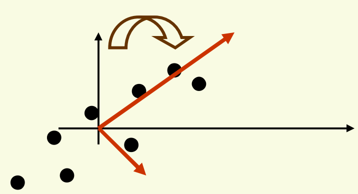
- The larger the eigenvalue of ***S***, the larger is the variance in the direction of corresponding eigenvector



- This result is exactly what we expected: project ***x*** into subspace of dimension ***k*** which has the largest variance
- This is very intuitive: restrict attention to directions where the scatter is the greatest

PCA

- Thus PCA can be thought of as finding new orthogonal basis by rotating the old axis until the directions of maximum variance are found



PCA as Data Approximation

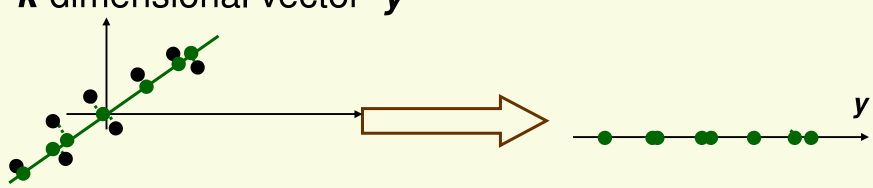
- Let $\{\boldsymbol{e_1}, \boldsymbol{e_2}, \ldots, \boldsymbol{e_d}\}$ be all $\boldsymbol{d}$ eigenvectors of the scatter matrix $\boldsymbol{S}$, sorted in order of decreasing corresponding eigenvalue
- Without any approximation, for any sample $\boldsymbol{x_i}$:

$$\boldsymbol{x}_i = \sum_{j=1}^{d} \alpha_j \boldsymbol{e}_j = \underbrace{\alpha_1 \boldsymbol{e}_1 + \ldots + \alpha_k \boldsymbol{e}_k}_{\text{approximation of } x_i} + \overbrace{\alpha_{k+1} \boldsymbol{e}_{k+1} \ldots + \alpha_d \boldsymbol{e}_d}^{\text{error of approximation}}$$

- coefficients $\alpha_m = \boldsymbol{x}^t_i \boldsymbol{e}_m$ are called *principle components*
  - The larger $\boldsymbol{k}$, the better is the approximation
  - Components are arranged in order of importance, more important components come first
- Thus PCA takes the first $\boldsymbol{k}$ most important components of $\boldsymbol{x_i}$ as an approximation to $\boldsymbol{x_i}$

PCA: Last Step

- Now we know how to project the data
- Last step is to change the coordinates to get final $\boldsymbol{k}$-dimensional vector $\boldsymbol{y}$



- Let matrix $\boldsymbol{E} = [\boldsymbol{e}_1 \cdots \boldsymbol{e}_k]$
- Then the coordinate transformation is $\boldsymbol{y} = \boldsymbol{E}^t \boldsymbol{x}$
- Under $\boldsymbol{E}^t$, the eigenvectors become the standard basis:

$$\boldsymbol{E}^t \boldsymbol{e}_i = \begin{bmatrix} \boldsymbol{e}_1 \\ \vdots \\ \boldsymbol{e}_i \\ \vdots \\ \boldsymbol{e}_k \end{bmatrix} \boldsymbol{e}_i = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

Recipe for Dimension Reduction with PCA

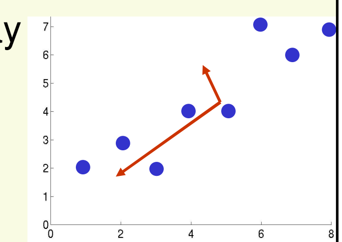
Data $D=\{x_1,x_2,\ldots,x_n\}$. Each x_i is a d-dimensional vector. Wish to use PCA to reduce dimension to k

1. Find the sample mean $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$
2. Subtract sample mean from the data $z_i = x_i - \hat{\mu}$
3. Compute the scatter matrix $S = \sum_{i=1}^{n} z_i z_i^t$
4. Compute eigenvectors $e_1,e_2,\ldots,e_k$ corresponding to the k largest eigenvalues of S
5. Let $e_1,e_2,\ldots,e_k$ be the columns of matrix $E = [e_1 \cdots e_k]$
6. The desired y which is the closest approximation to x is $y = E^t z$

PCA Example Using Matlab

- Let $D = \{(1,2),(2,3),(3,2),(4,4),(5,4),(6,7),(7,6),(9,7)\}$
- Convenient to arrange data in array

$$X = \begin{bmatrix} 1 & 2 \\ \vdots & \vdots \\ 9 & 7 \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_8 \end{bmatrix}$$



- Mean $\mu = mean(X) = [4.6 \;\; 4.4]$
- Subtract mean from data to get new data array Z

$$Z = X - \begin{bmatrix} \mu \\ \vdots \\ \mu \end{bmatrix} = X - repmat(\mu,8,1) = \begin{bmatrix} -3.6 & -4.4 \\ \vdots & \vdots \\ 4.4 & 2.6 \end{bmatrix}$$

- Compute the scatter matrix S

$$S = 7 * cov(Z) = [-3.6 \;\; -4.4]\begin{bmatrix} -3.6 \\ -4.4 \end{bmatrix} + \ldots + [4.4 \;\; 2.6]\begin{bmatrix} 4.4 \\ 2.6 \end{bmatrix} = \begin{bmatrix} 57 & 40 \\ 40 & 34 \end{bmatrix}$$

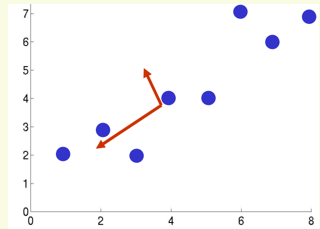
*matlab uses unbiased estimate for covariance, so S=(n-1)*cov(Z)*

PCA Example Using Matlab

- Use [V,D] =eig(***S***) to get eigenvalues and eigenvectors of ***S***

$$\lambda_1 = 87 \text{ and } e_1 = \begin{bmatrix} -0.8 \\ -0.6 \end{bmatrix}$$

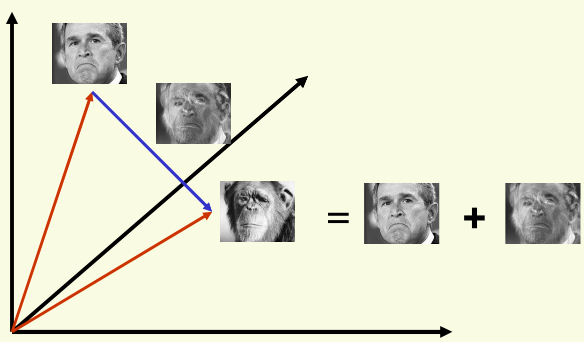
$$\lambda_2 = 3.8 \text{ and } e_2 = \begin{bmatrix} 0.6 \\ -0.8 \end{bmatrix}$$



- Projection to 1D space in the direction of e_1

$$Y = e_1^t Z^t = \left(\begin{bmatrix} -0.8 & -0.6 \end{bmatrix} \begin{bmatrix} -3.6 & \cdots & 4.4 \\ -4.4 & \cdots & 2.6 \end{bmatrix} \right) = \begin{bmatrix} 4.3 & \cdots & -5.1 \end{bmatrix}$$
$$= \begin{bmatrix} y_1 & \cdots & y_8 \end{bmatrix}$$

The Space of Faces



- An image is a point in a high dimensional space
  - An N x M image is a point in R^{NM}
  - We can define vectors in this space as we did in the 2D case

[Thanks to Chuck Dyer, Steve Seitz, Nishino]

Eigenfaces

Eigenfaces look somewhat like generic faces.

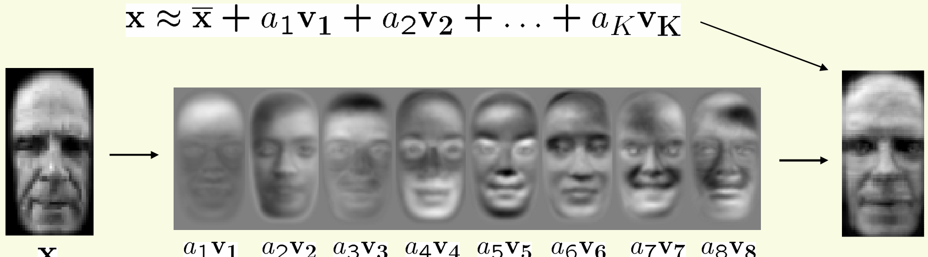
Thanks to S. Narasimhan

Projecting onto the Eigenfaces

- The eigenfaces $\mathbf{v_1}$, ..., $\mathbf{v_K}$ span the space of faces
  - A face is converted to eigenface coordinates by

$$\mathbf{x} \rightarrow (\underbrace{(\mathbf{x}-\overline{\mathbf{x}})\cdot\mathbf{v}_1}_{a_1},\ \underbrace{(\mathbf{x}-\overline{\mathbf{x}})\cdot\mathbf{v}_2}_{a_2},\ldots,\ \underbrace{(\mathbf{x}-\overline{\mathbf{x}})\cdot\mathbf{v}_K}_{a_K})$$

$$\mathbf{x} \approx \overline{\mathbf{x}} + a_1\mathbf{v_1} + a_2\mathbf{v_2} + \ldots + a_K\mathbf{v_K}$$



Thanks to S. Narasimhan

Drawbacks of PCA

- PCA was designed for accurate *data representation*, not for data classification
  - Preserves as much variance in data as possible
  - If directions of maximum variance is important for classification, will work
- However the directions of maximum variance may be useless for classification