CS9840
Machine Learning in Computer Vision
Olga Veksler

Lecture 6

Curse of Dimensionality

PCA

Outline

- Curse of Dimensionality
- Dimensionality reduction with PCA

Curse of Dimensionality

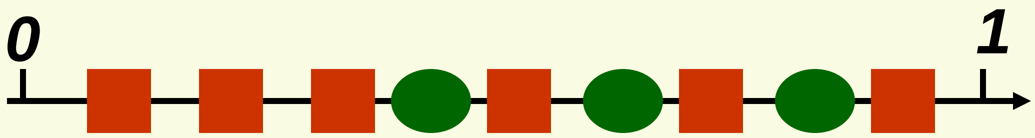
- Problems of high dimensional data, “the curse of dimensionality”
  - running time
  - overfitting
  - number of samples required
- Dimensionality Reduction Methods
  - Principle Component Analysis

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

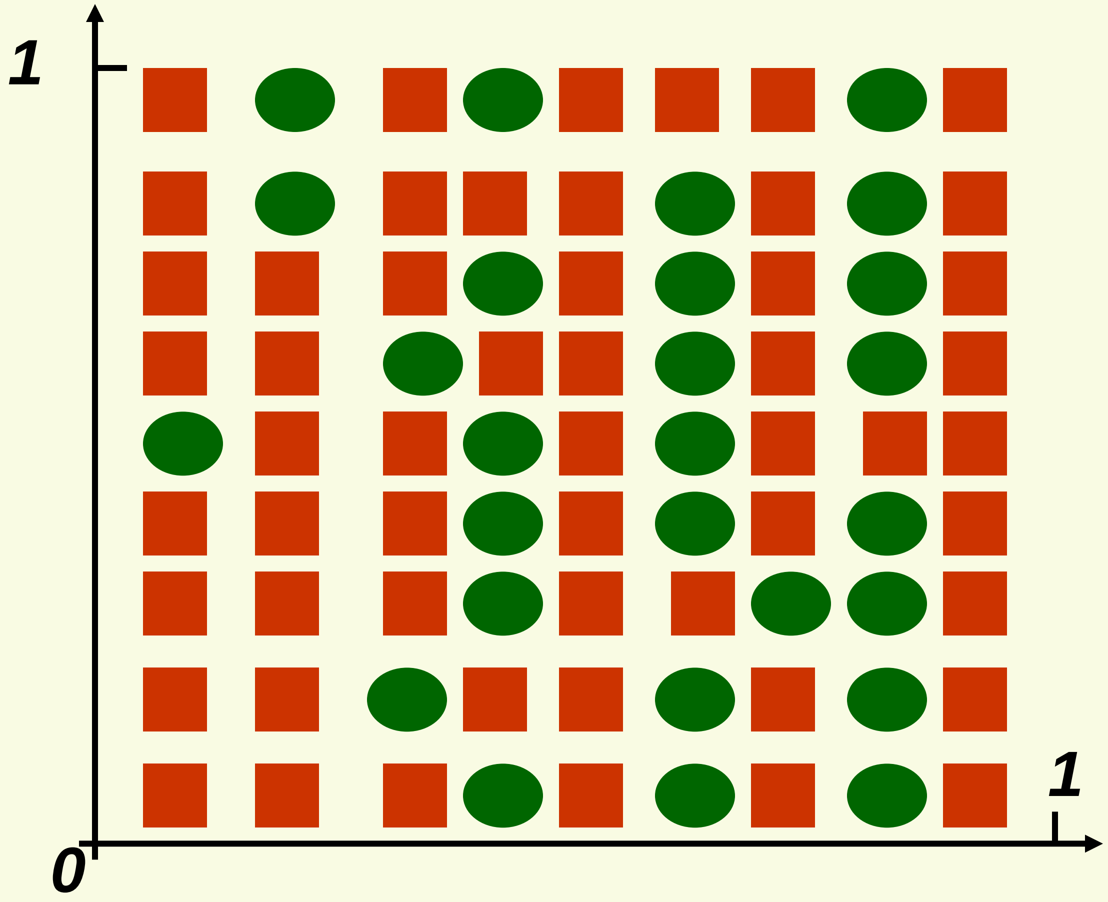
- Suppose we want to use the nearest neighbor approach with ***k*** = 1 (***1NN***)
- Suppose we start with only one feature



- 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?

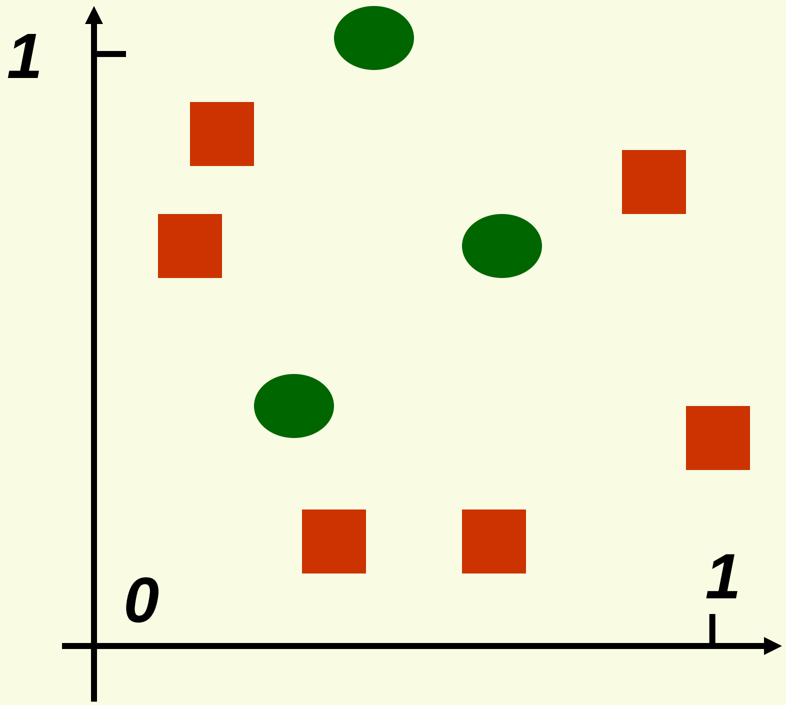
Curse of Dimensionality: Number of Samples

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



Curse of Dimensionality: Number of Samples

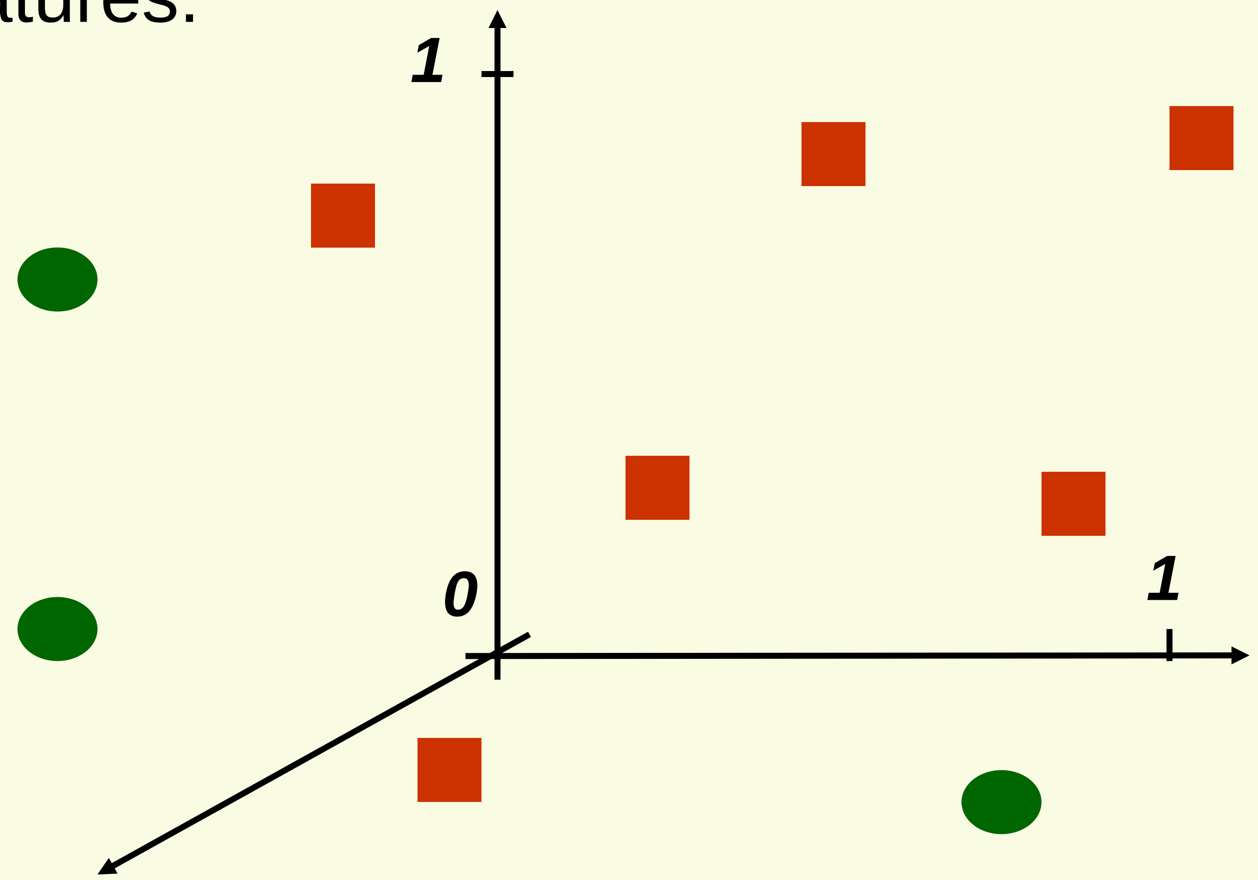
- Of course, when we go from 1 feature to 2, no one gives us more samples, we still have 9



- 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:



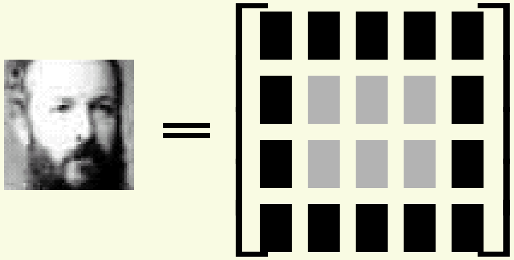
- If ***9*** was dense enough in 1D, in 3D we need $\boldsymbol{9^3=729}$ samples!

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

The Curse of Dimensionality

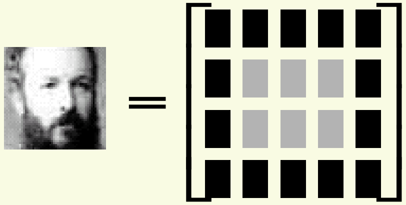
- We should try to avoid creating lot of features
- Often no choice, problem starts with many features
- Example: Face Detection
    - One sample point is ***k*** by ***m*** array of pixels



    - Feature extraction is not trivial
    - Say pixel intensities are 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***

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

- For example,

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

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

Dimensionality Reduction

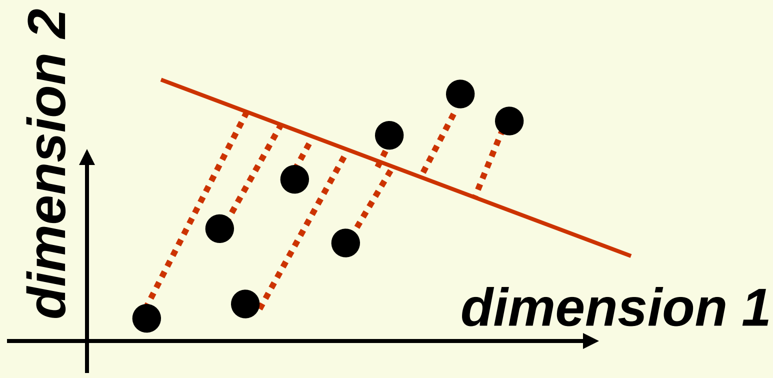
- The best $\boldsymbol{f}(\boldsymbol{x})$ is most likely a non-linear function
- Linear functions are easier to find though
- For now, assume that $\boldsymbol{f}(\boldsymbol{x})$ is a linear mapping
- Thus it can be represented by a matrix $\boldsymbol{W}$:

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

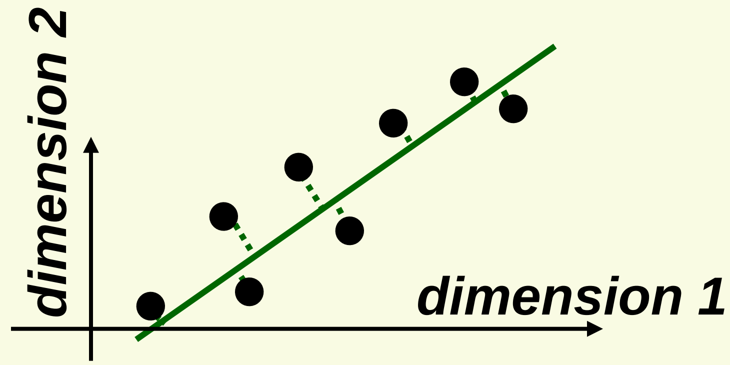
Principle Component Analysis (PCA)

- 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

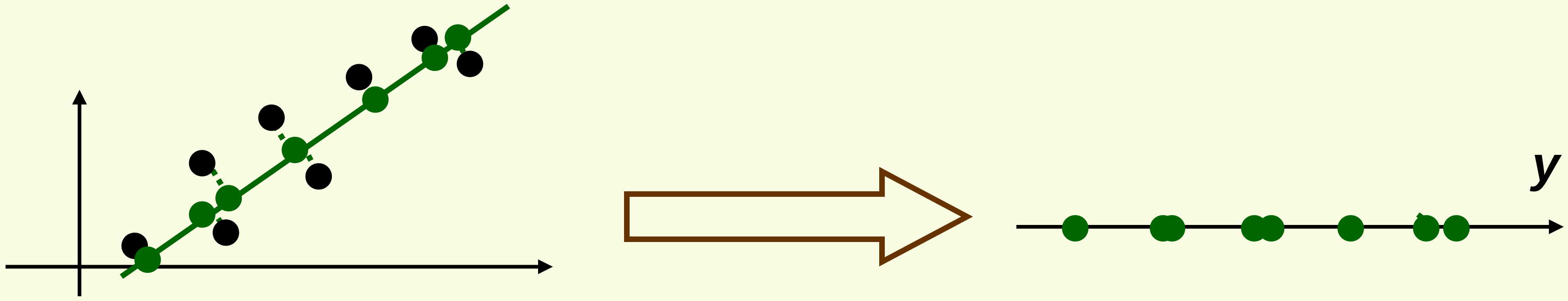


small projection errors, good line to project to

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

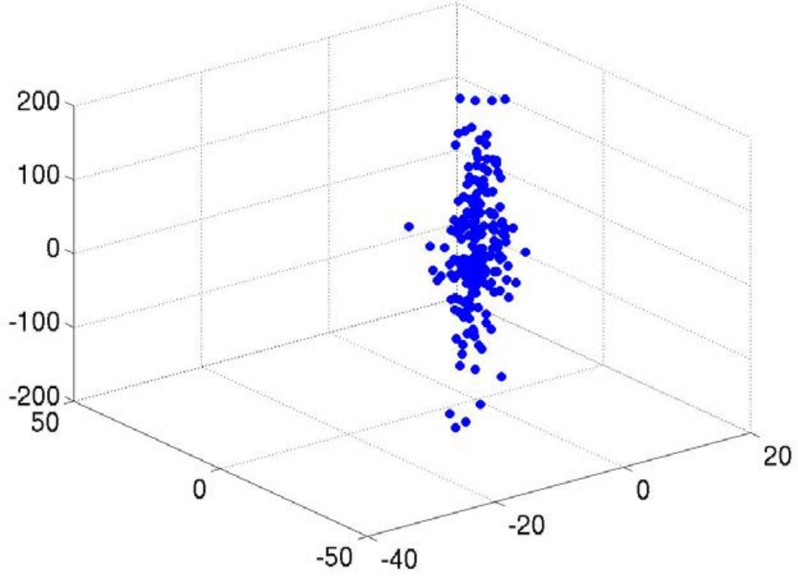
PCA

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

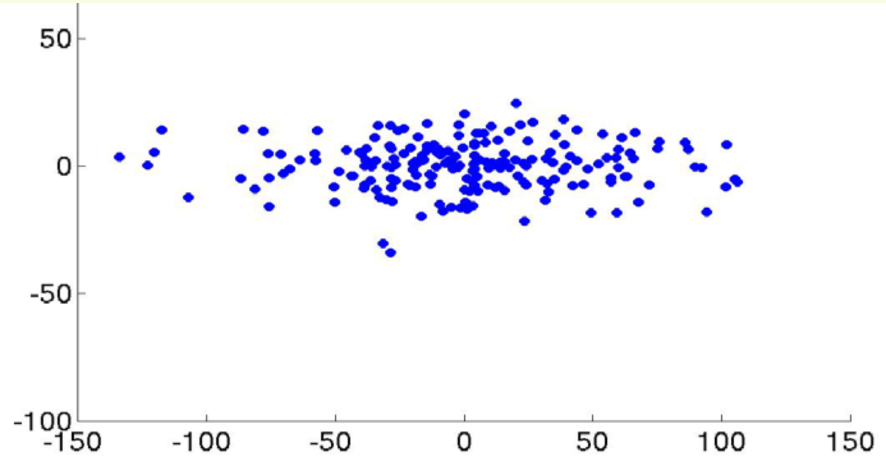


- Note that new data ***y*** has the same variance as old data ***x*** in the direction of the green line
- PCA preserves largest variances in the data

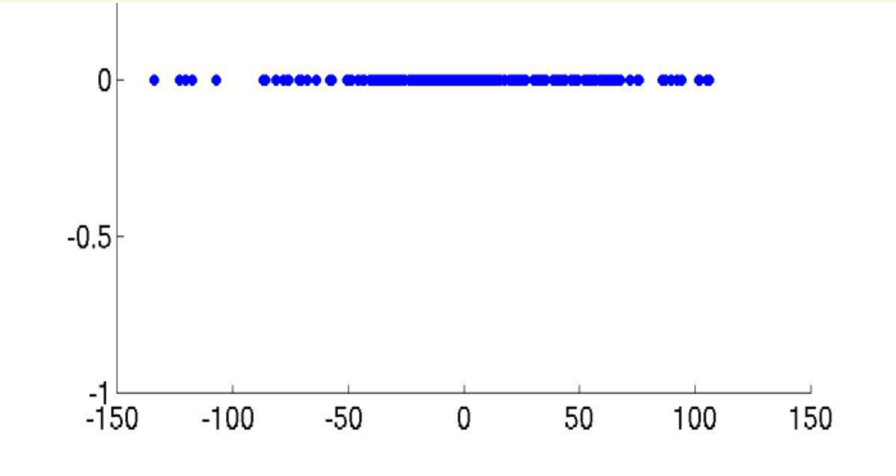
PCA: Approximation of Elliptical Cloud in 3D



best 2D approximation

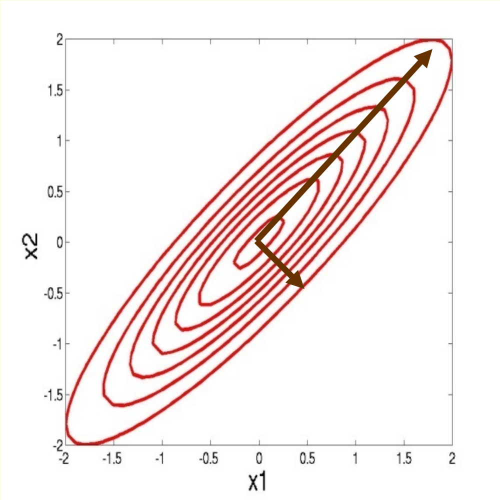


best 1D approximation



PCA

- What is the direction of largest variance in data?
- Recall that if $\boldsymbol{x}$ has multivariate distribution $N(\boldsymbol{\mu},\boldsymbol{\Sigma})$, direction of largest variance is given by eigenvector corresponding to the largest eigenvalue of $\boldsymbol{\Sigma}$

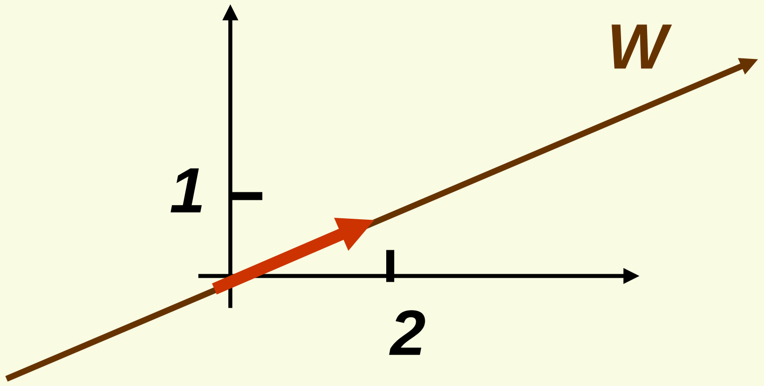


- This is a hint that we should be looking at the covariance matrix of the data (note that PCA can be applied to distributions other than Gaussian)

PCA: Linear Algebra Review

- 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 $\{\mathbf{e}_1,\mathbf{e}_2,\ldots,\mathbf{e}_k\}$ which forms an orthonormal basis for ***W***
  - $\langle \mathbf{e}_i,\mathbf{e}_j\rangle = 0$ if ***i*** is not equal to ***j*** and $\langle \mathbf{e}_i,\mathbf{e}_i\rangle = 1$
- Thus any vector in ***W*** can be written as

$$\alpha_1\mathbf{e}_1+\alpha_2\mathbf{e}_2+\ldots+\alpha_k\mathbf{e}_k=\sum_{i=1}^{k}\alpha_i\mathbf{e}_i \quad \textbf{\textit{for scalars}}\ \alpha_1,\ldots,\alpha_k$$

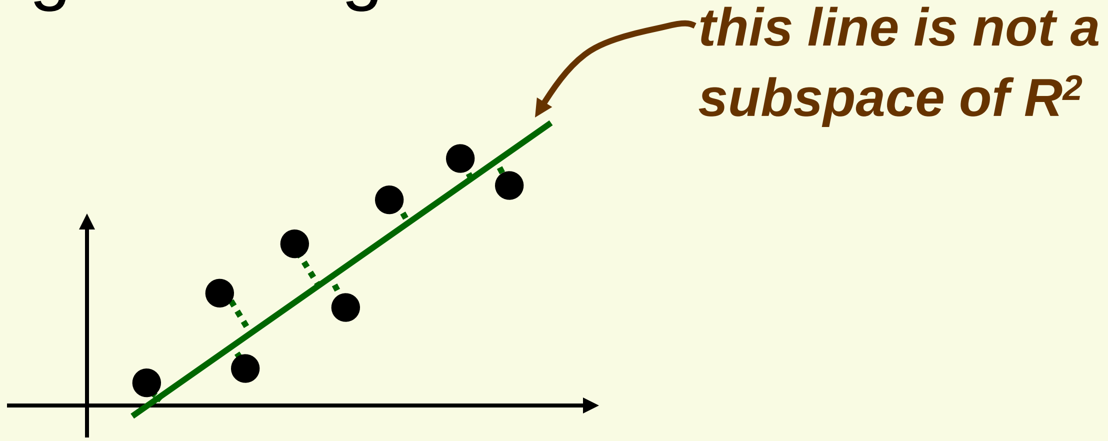


Let $V = R^2$ and ***W*** be the line x-2y=0. Then the orthonormal basis for W is

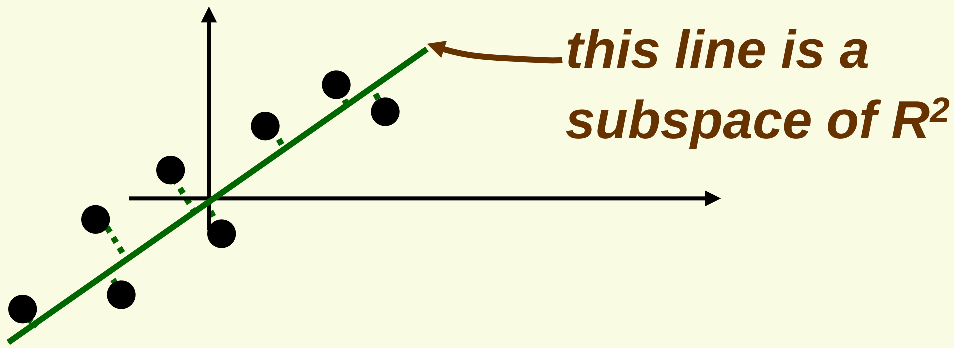
$$\left\{\begin{bmatrix}2/\sqrt{5}\\1/\sqrt{5}\end{bmatrix}\right\}$$

PCA: Linear Algebra

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



- It is convenient to project to subspace ***W***: thus we need to shift everything

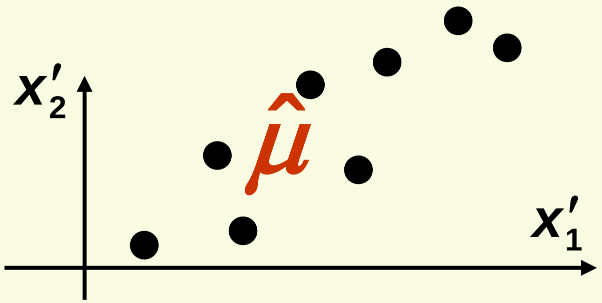


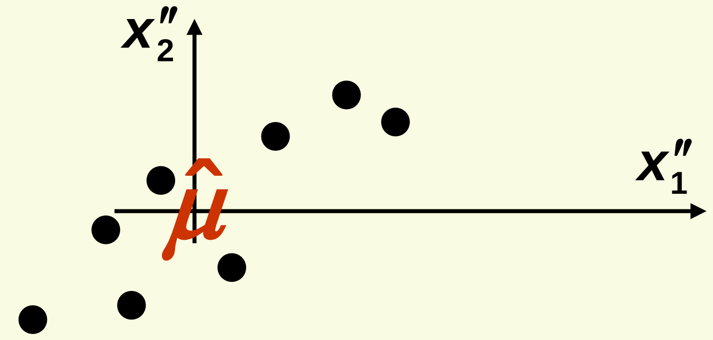
PCA Derivation: Shift by the Mean Vector

- 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***

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

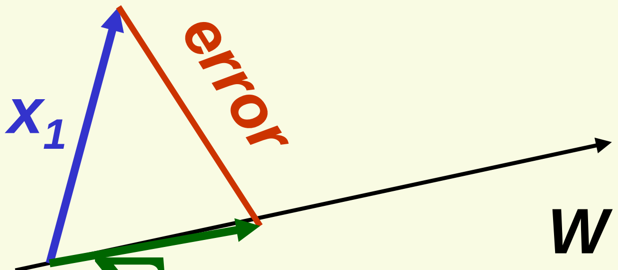
PCA: Derivation

- 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 $\mathbf{x}_j$'s
- Any $\mathbf{x}_j$ can be written as $\sum_{i=1}^{k} \alpha_{ji} \mathbf{e}_i$
- Thus the total error for representation of all data ***D*** is:

sum over all data points

$$J(\mathbf{e}_1,\ldots,\mathbf{e}_k,\alpha_{11},\ldots\alpha_{nk}) = \sum_{j=1}^{n} \left\| \mathbf{x}_j - \sum_{i=1}^{k} \alpha_{ji} \mathbf{e}_i \right\|^2$$

unknowns

error at one point

PCA: Derivation

- A lot of math…….to finally get:

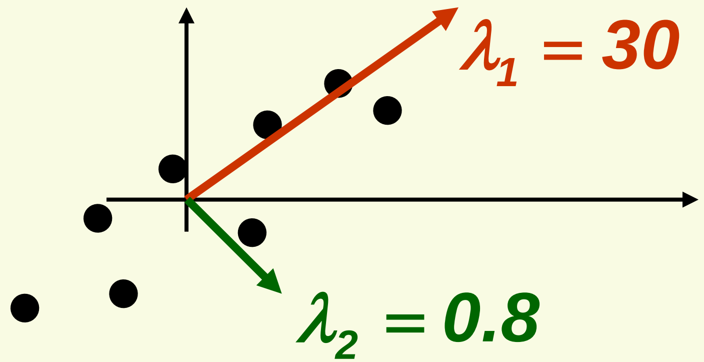
- Let ***S*** be the scatter matrix, it is just n-1 times the sample covariance matrix

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

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

PCA

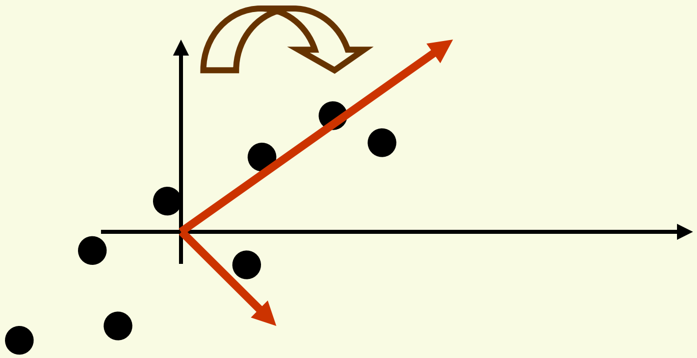
- The larger the eigenvalue of ***S***, the larger is the variance in the direction of corresponding eigenvector



- 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

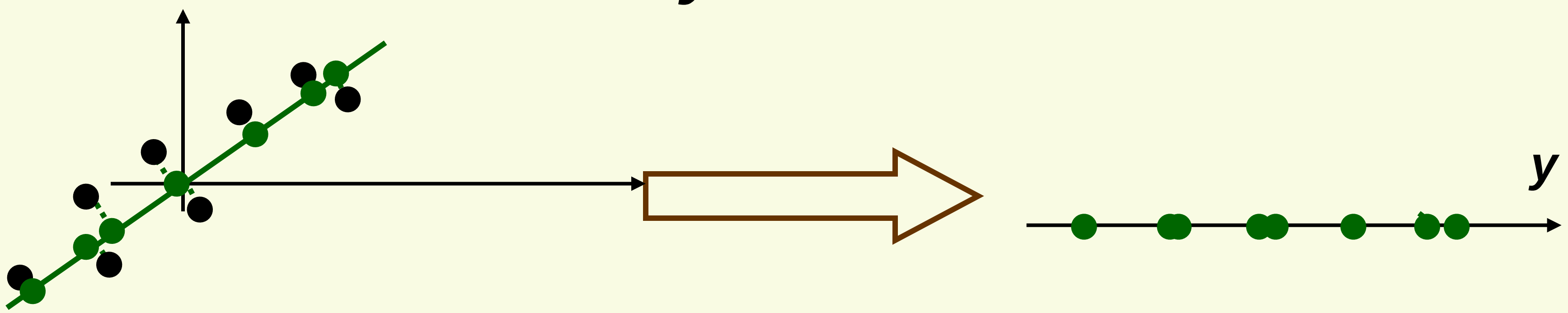
- Let $\{\boldsymbol{e_1},\boldsymbol{e_2},\ldots,\boldsymbol{e_d}\}$ be all ***d*** eigenvectors of the scatter matrix ***S***, sorted in order of decreasing corresponding eigenvalue
- Without any approximation, for any sample $\boldsymbol{x_i}$:

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

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



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

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

Recipe for Dimension Reduction with PCA

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

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 $\boldsymbol{e_1},\boldsymbol{e_2},\ldots,\boldsymbol{e_k}$ corresponding to the $\boldsymbol{k}$ largest eigenvalues of $\boldsymbol{S}$
5. Let $\boldsymbol{e_1},\boldsymbol{e_2},\ldots,\boldsymbol{e_k}$ be the columns of matrix $E = [e_1 \cdots e_k]$
6. The desired $\boldsymbol{y}$ which is the closest approximation to $\boldsymbol{x}$ is $y = E^t z$

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