Clustering and Component Analysis

Terminology

- each data source (image, sound clip, etc.) can be represented by a single vector $\boldsymbol{d}$ of length N
- N is the dimension in which the data lies
- let $\boldsymbol{D} = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_M\}$ denote the set of M such data in the database

Clustering

- Given **D**, group the data points $\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_M$ in clusters according to a criterion of similarity

Component Analysis

- each data vector ***d*** is of length (dimension) *N*
- determine which dimensions are most important to represent the data of ***D***

Dimensionality Reduction

- use component analysis to find an optimal subspace of dimension S, often where $S << N$
  - e.g., compress a colour image to store it in the minimum number of bits with sufficient quality to identify who is who

Clustering with *k*-means

- partitions the data points into *k* disjoint subsets based on a distance measure between instances
- advantage: easy to implement
- input: a set of *N*-dimensional vectors $\{\mathbf{d}_1, \mathbf{d}_2, \ldots \mathbf{d}_m\}$
- output: a mapping $\boldsymbol{C} = \{C_1, C_2, \ldots C_k\}$ of the vectors $\boldsymbol{d_i}$ into *k* disjoint clusters

k-means algorithm

- initialize *C* randomly
- repeat
  - compute centroid of each cluster
  - assign each vector to the closest centroid using Euclidean distance
- until *C* unchanged

Termination of *k*-means clustering

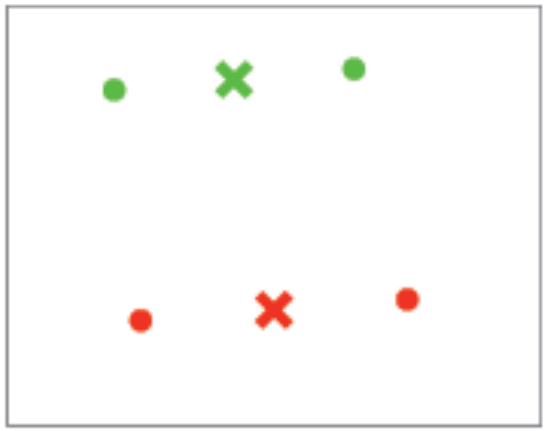
- for given data $\{\mathbf{d}_1, \mathbf{d}_2, \ldots \mathbf{d}_m\}$ and clustering C, consider the sum of the Euclidean distances between each vector and the center of its cluster:

$$J = \sum_{i=1}^{m} \left\| d_i - \mu_{C(i)} \right\|$$

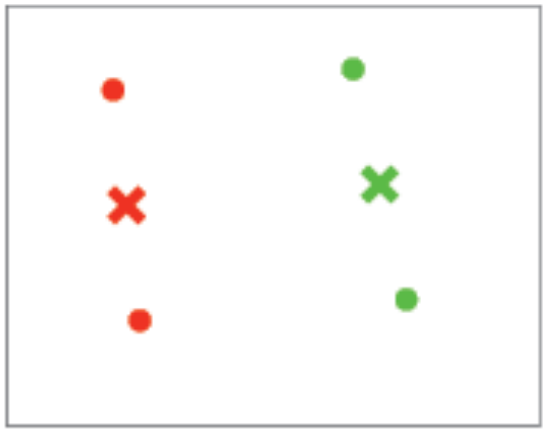
  - finitely many possible clusterings: at most k^m
  - each time we reassign a vector to a cluster with a nearer centroid, J decreases
  - each time we recompute the centroids of each cluster, J decreases (or stays the same) => algorithm must terminate!

Does k-means always find same answer?

- solution is *locally optimal*
- error function has many local minima
- depends on initial assignment of instances to clusters



$J = 0.2287$

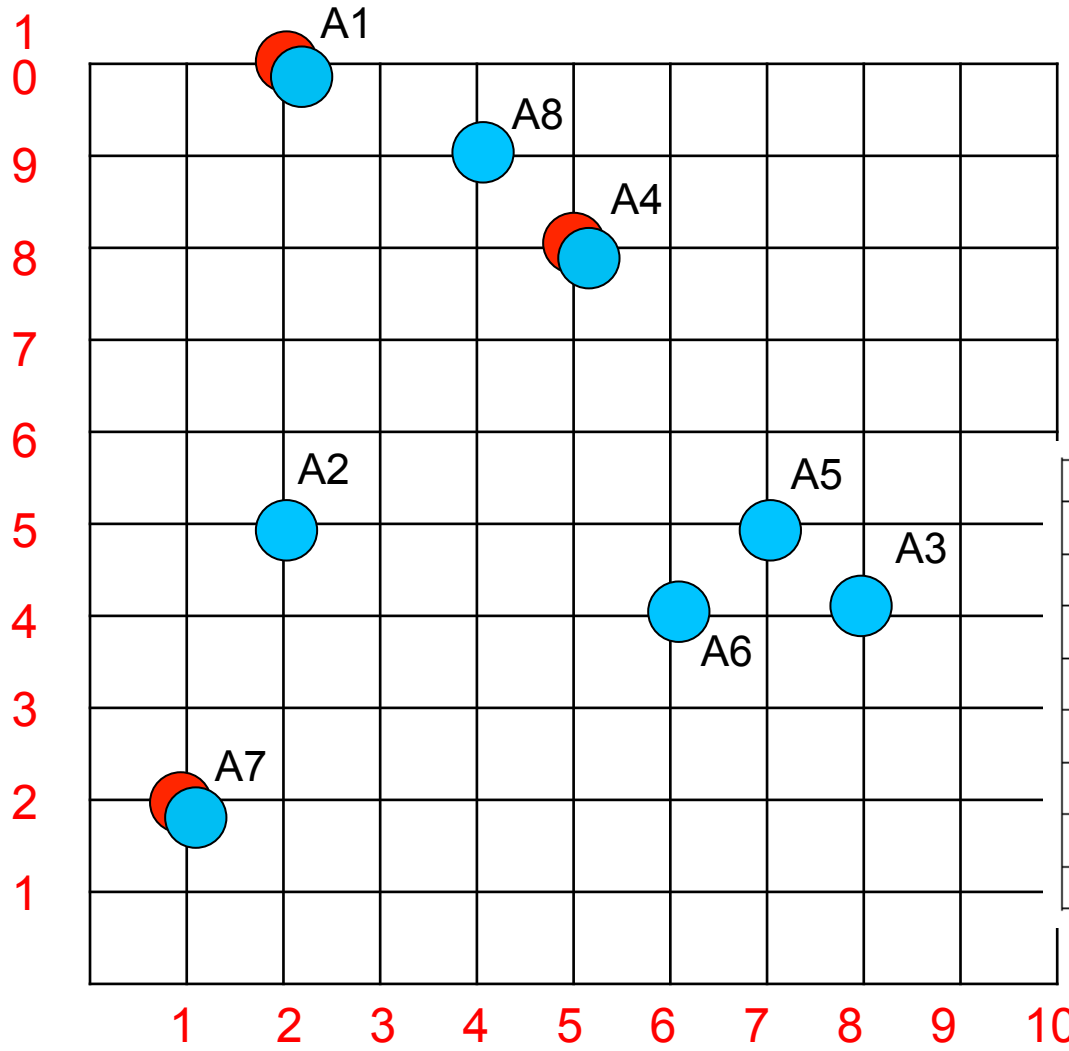


$J = 0.3088$

Finding good initial configurations

- place first center on a randomly chosen data point
- place second center on a data point as far away as possible from first
- place the i^{th} center as far away from the closest of center 1 through i-1

K-means exercise



Data points

Initial cluster seeds

Euclidean Distance Matrix

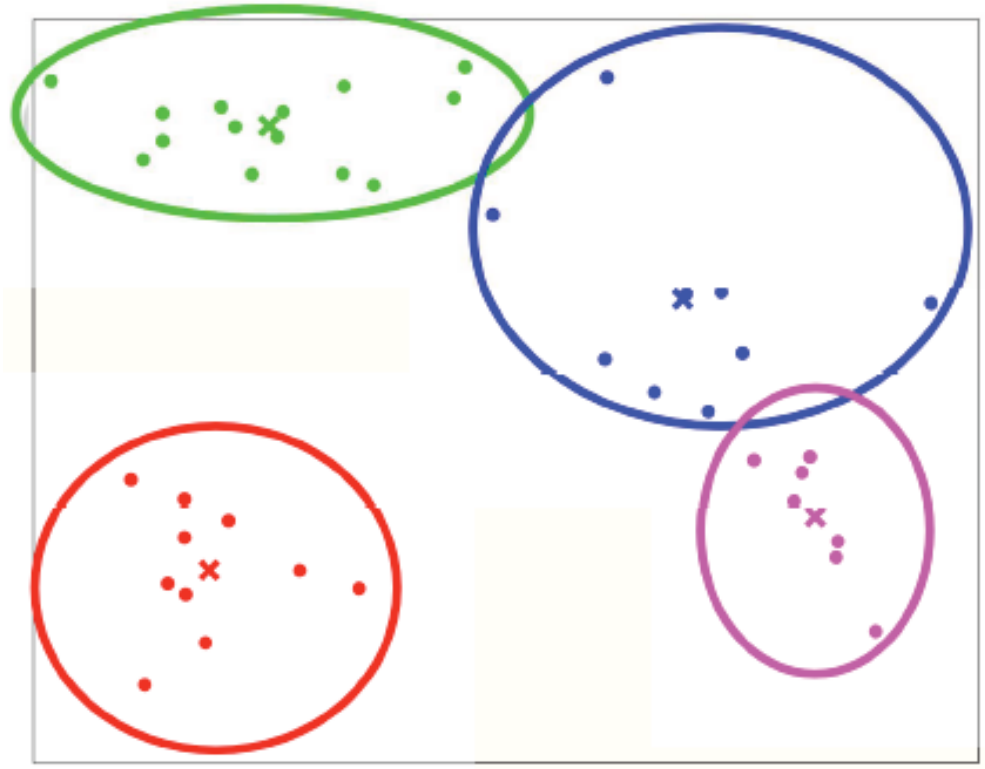
	A1	A2	A3	A4	A5	A6	A7	A8
A1	0	$\sqrt{25}$	$\sqrt{36}$	$\sqrt{13}$	$\sqrt{50}$	$\sqrt{52}$	$\sqrt{65}$	$\sqrt{5}$
A2		0	$\sqrt{37}$	$\sqrt{18}$	$\sqrt{25}$	$\sqrt{17}$	$\sqrt{10}$	$\sqrt{20}$
A3			0	$\sqrt{25}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{53}$	$\sqrt{41}$
A4				0	$\sqrt{13}$	$\sqrt{17}$	$\sqrt{52}$	$\sqrt{2}$
A5					0	$\sqrt{2}$	$\sqrt{45}$	$\sqrt{25}$
A6						0	$\sqrt{29}$	$\sqrt{29}$
A7							0	$\sqrt{58}$
A8								0

Choosing number of clusters

- delete clusters that cover too few points
- split clusters that cover too many points
- add extra clusters for “outliers”
- *minimum description length* principle:
  minimize loss + complexity of the clustering
- perceptual or other application-specific criteria

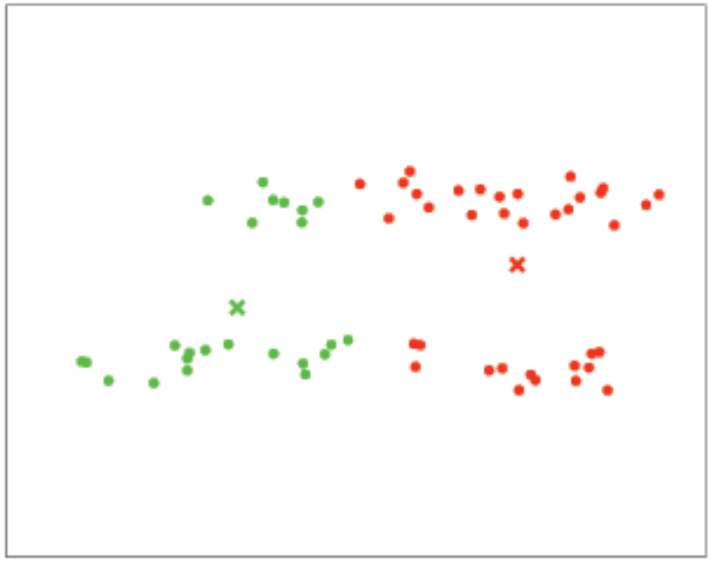
Why use sum of squared Euclidian distances?

- subjective: produces nice round clusters

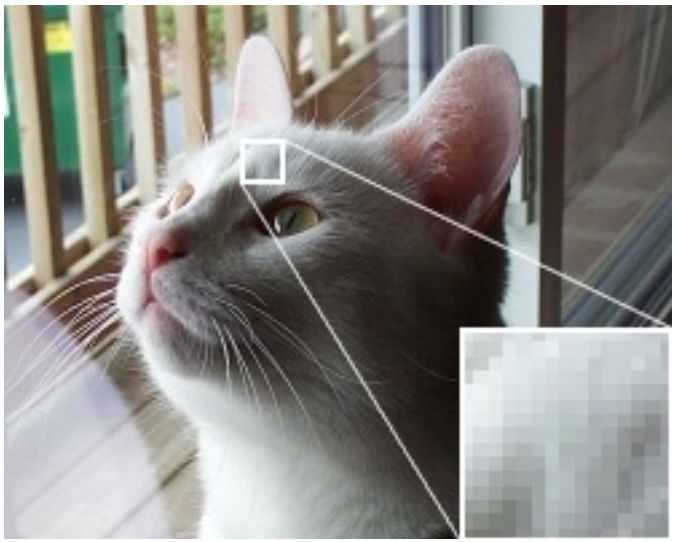


Why not use sum of squared Euclidian distances?

- produces nice round clusters
- differently scaled axes can dramatically affect results
- symbolic attributes may have to be treated differently



Example application: Color Quantization



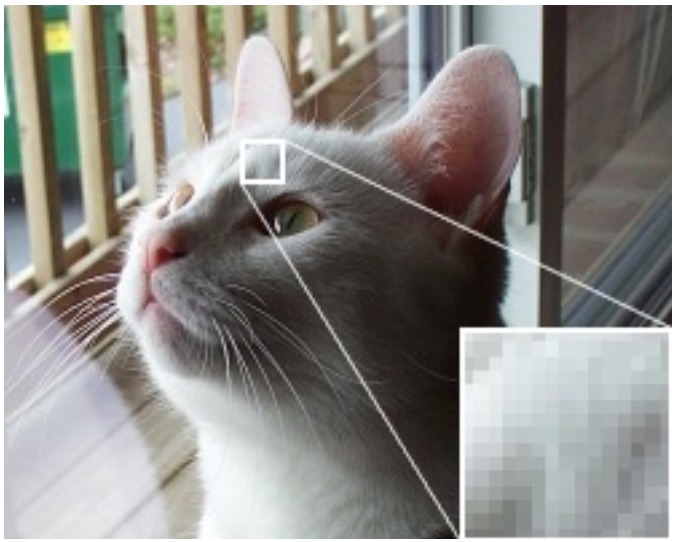
original: 24-bit RGB color

- want compressed version to look as similar as possible to original
- can transmit only compressed version plus color map
- want to minimize reconstruction error

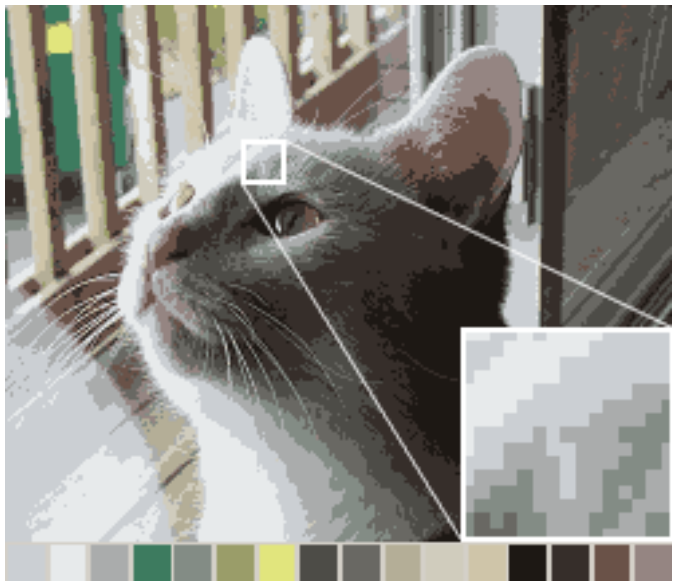
Review of Clustering

- Clustering of data using k-means
- How to choose initial centroids
- Pro and con of Euclidean distance
- Dimensionality reduction as an application

Example application: Color Quantization



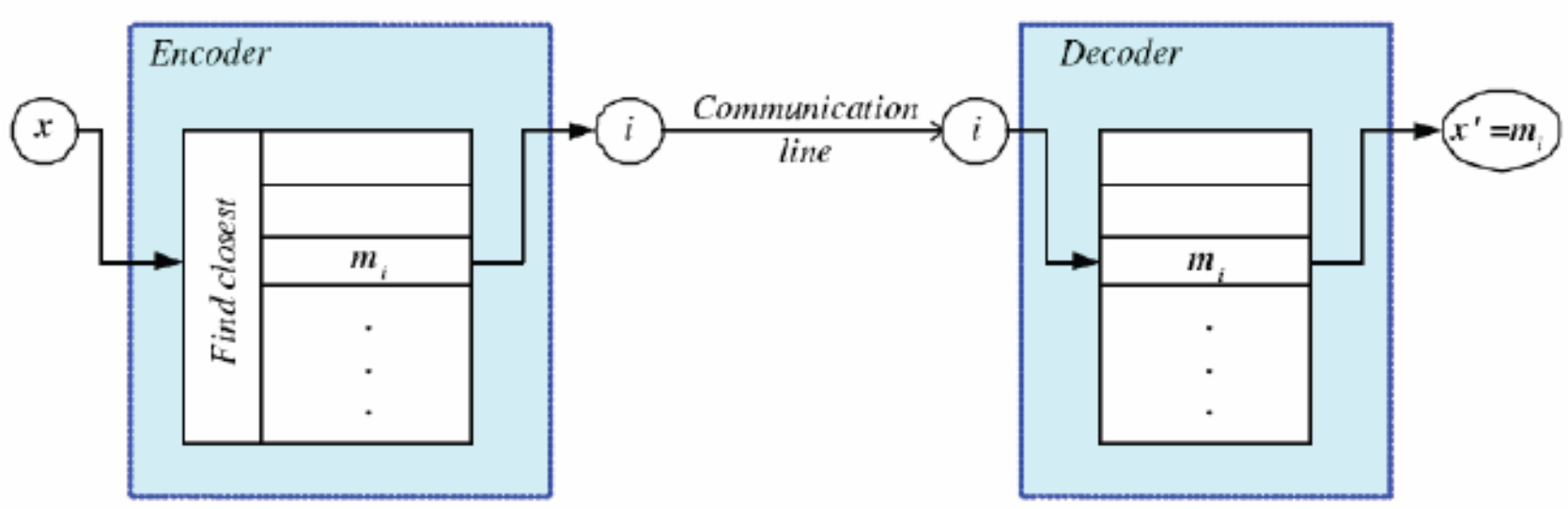
original: 24-bit RGB color



reduced to 4-bits (16 colors)

Example application: Speech Coder

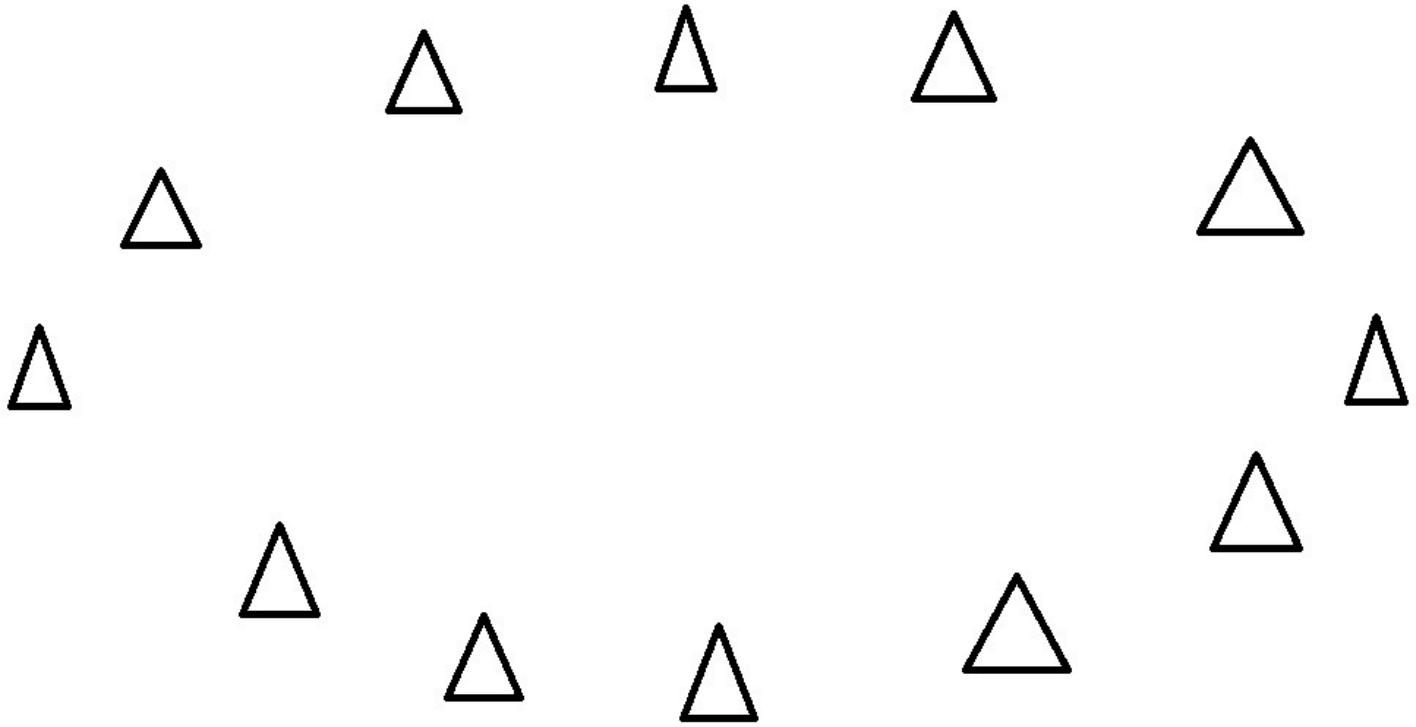
- determine a compact (low bits/sec) representation of speech, possibly for transmission over communication line



Today's agenda

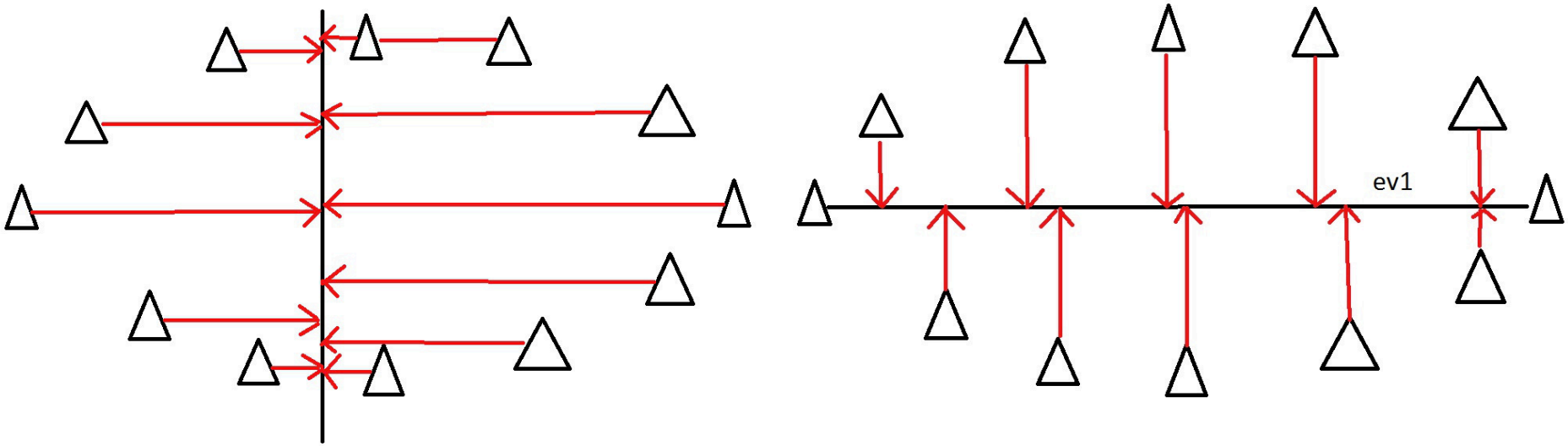
- Principal Component Analysis
- (and possibly, intro to Machine Learning)

Principal Components Analysis

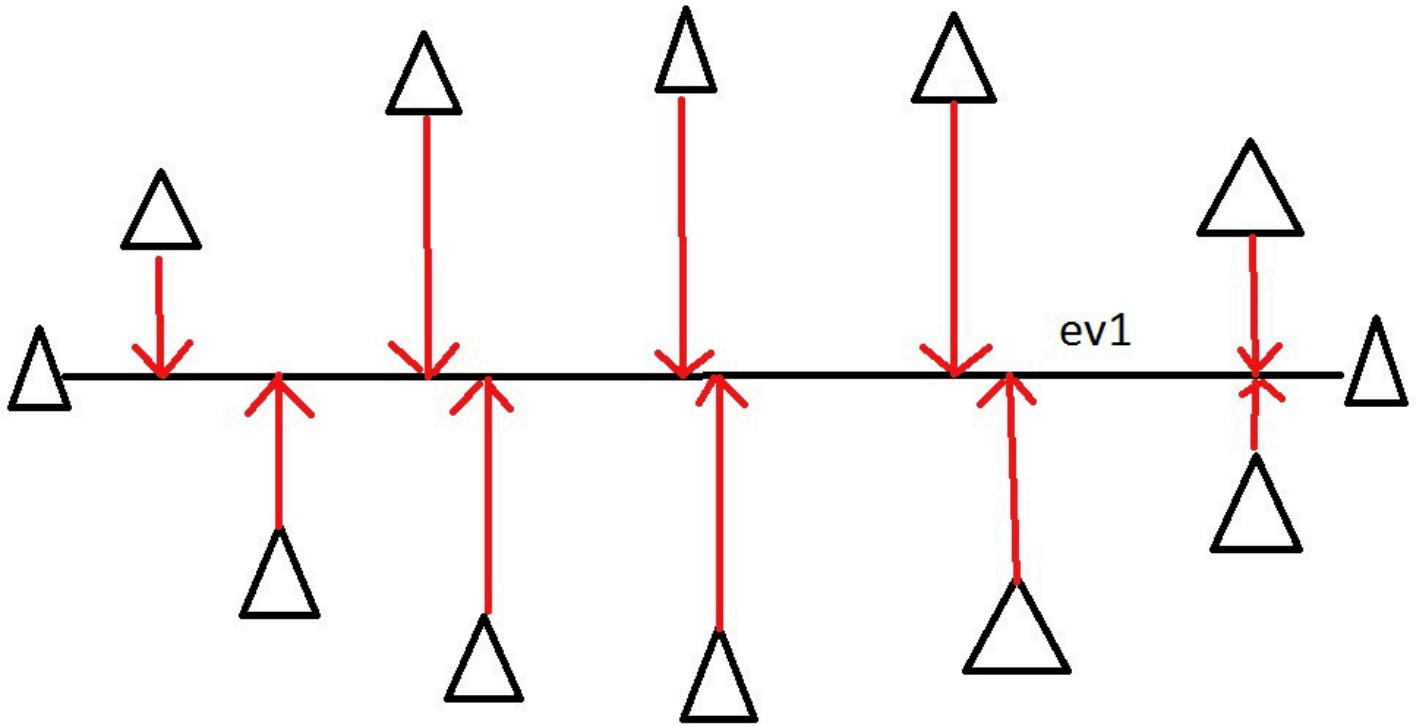


Credit: https://georgemdallas.wordpress.com/2013/10/30/principal-component-analysis-4-dummies-eigenvectors-eigenvalues-and-dimension-reduction/

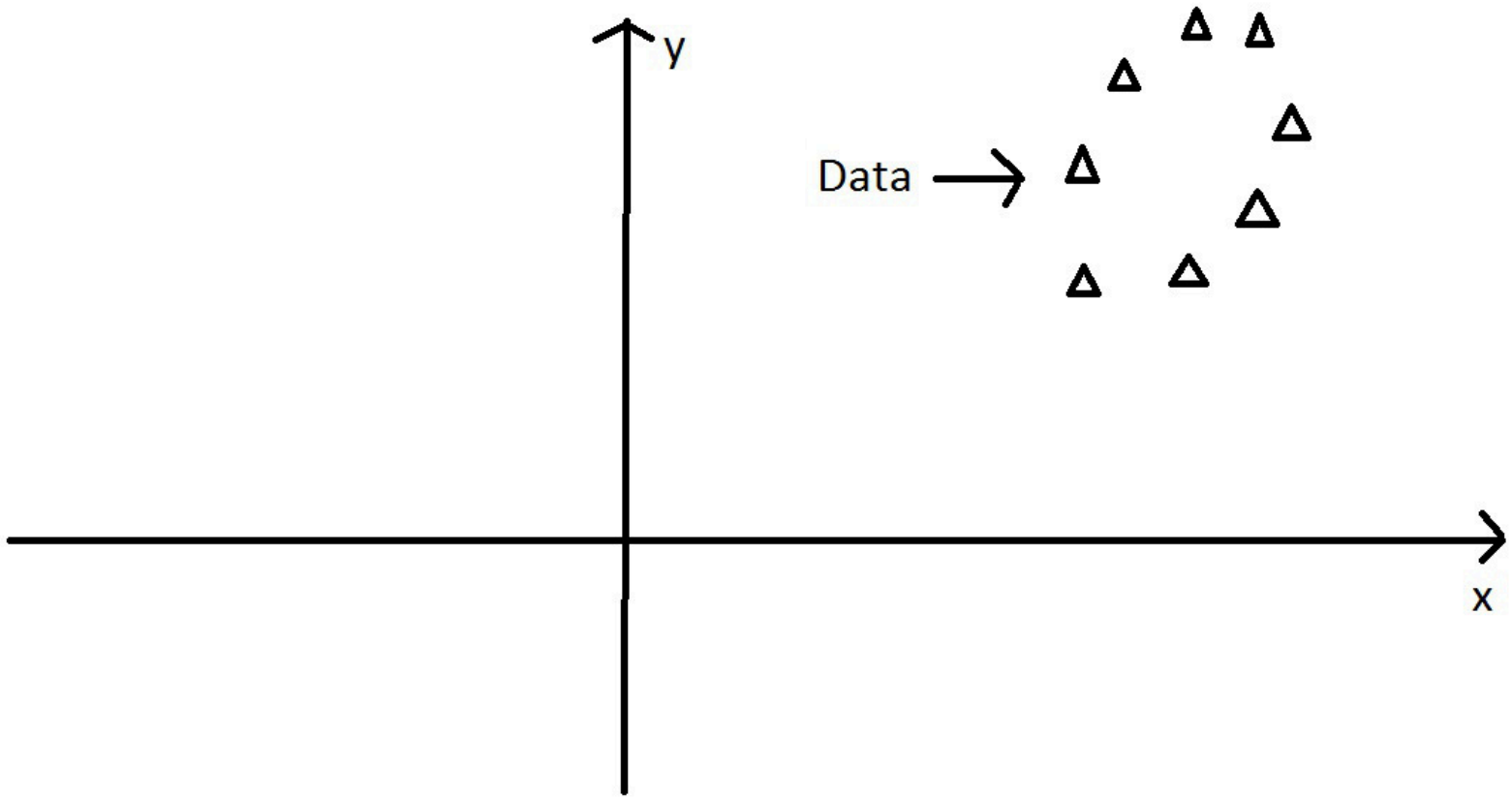
Along which axis does the data exhibit the greatest variance?



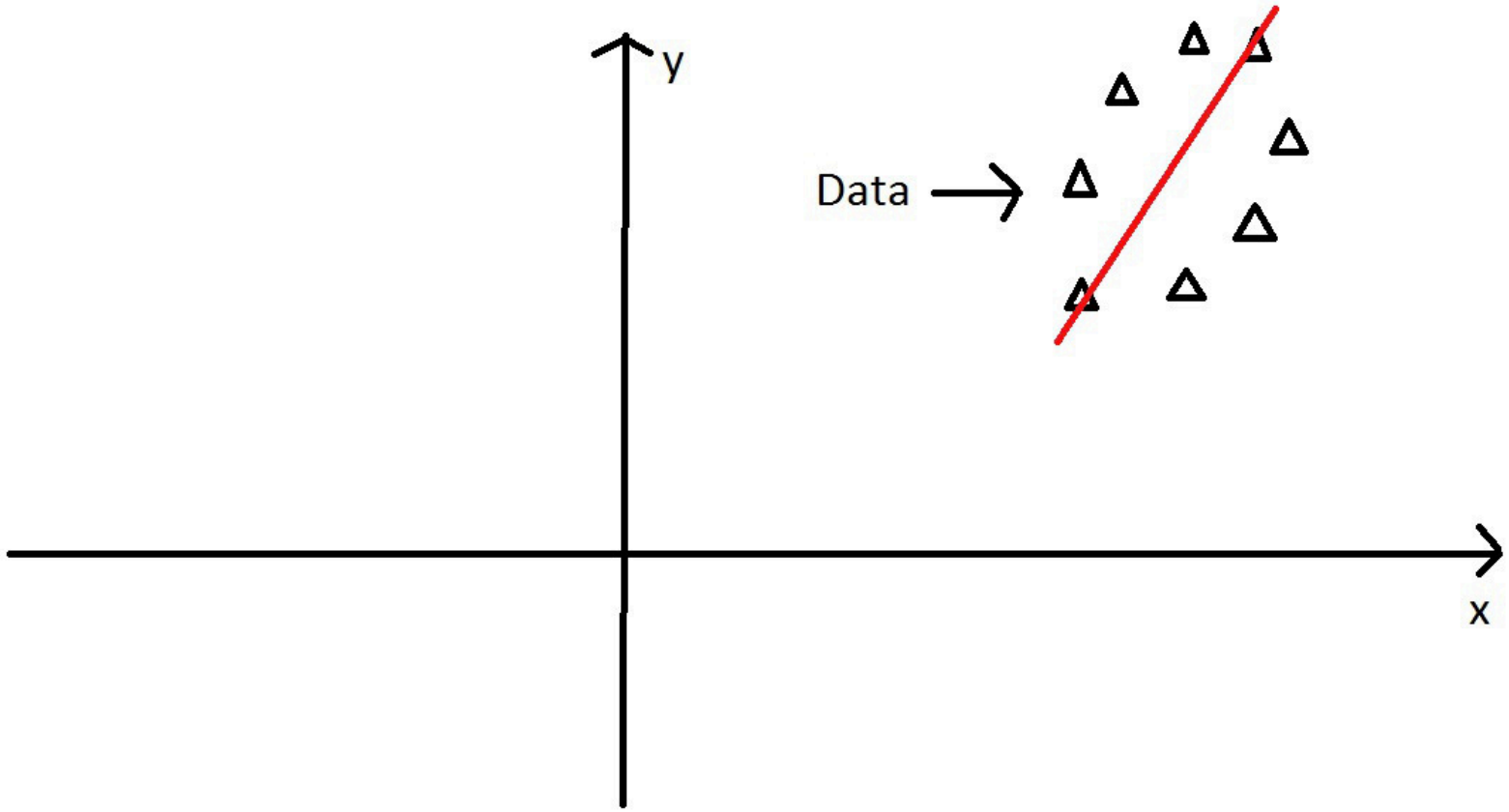
So that's your principal component



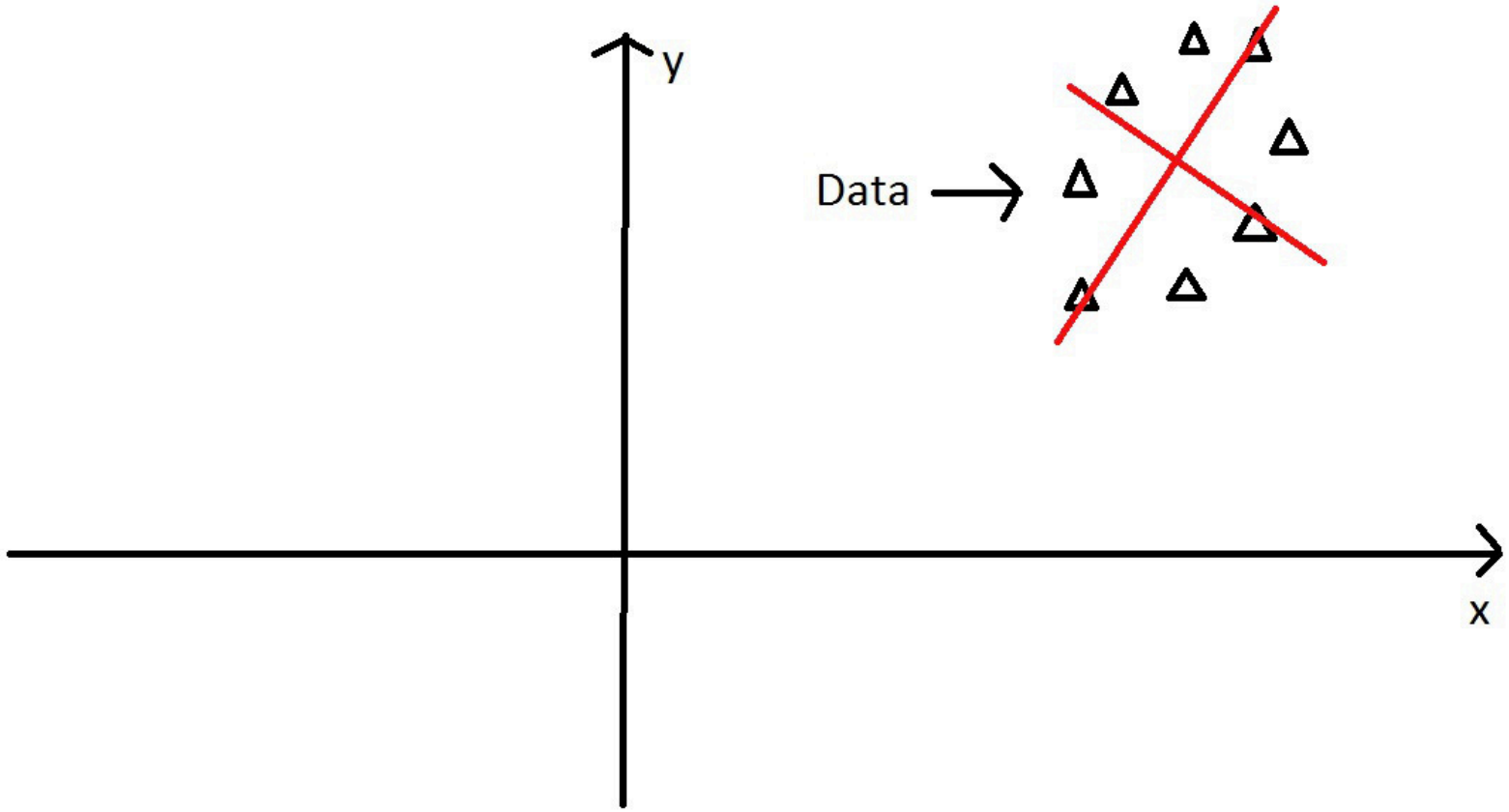
Where's the principal component now?



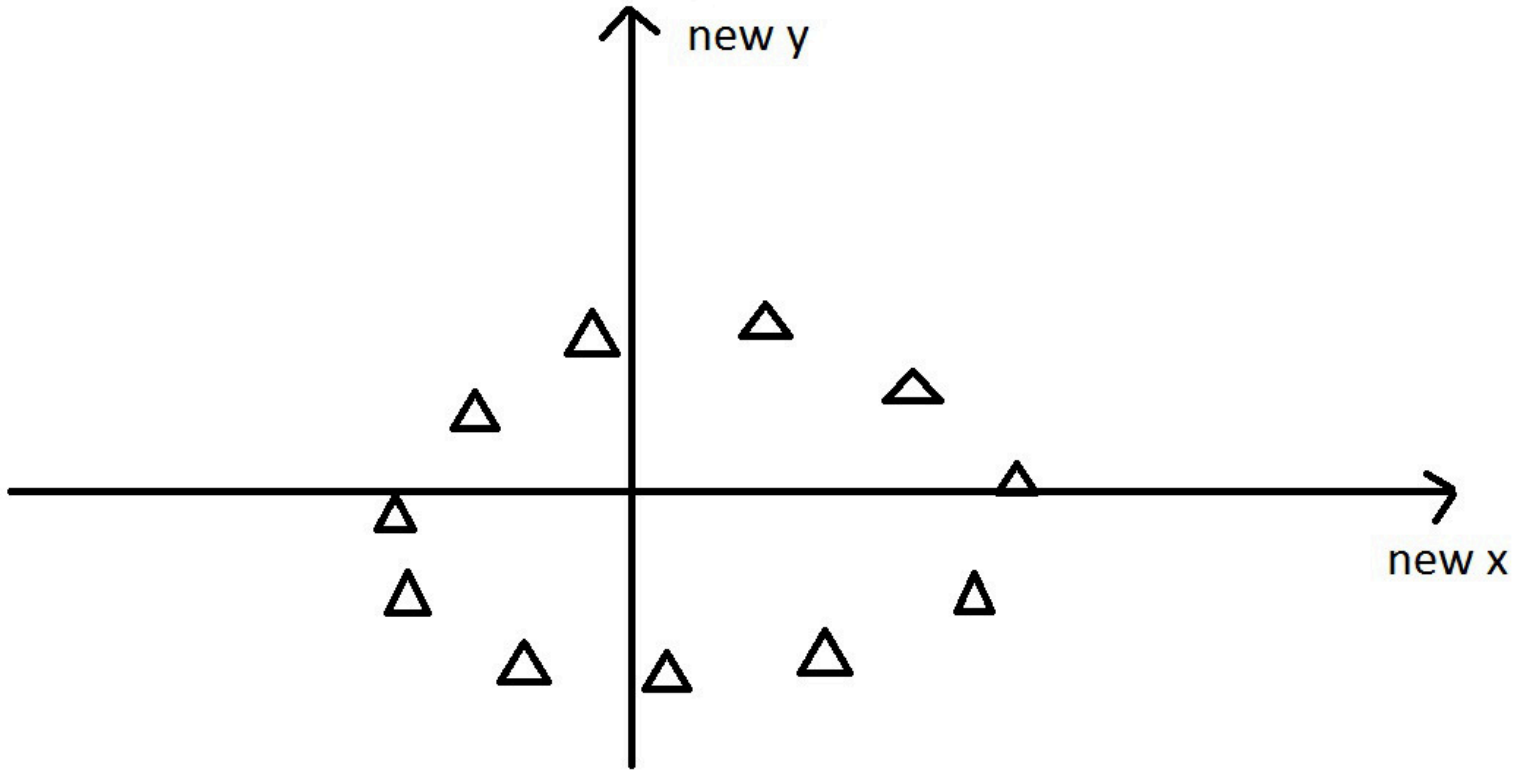
Principal component doesn't need to be aligned with coordinate axes



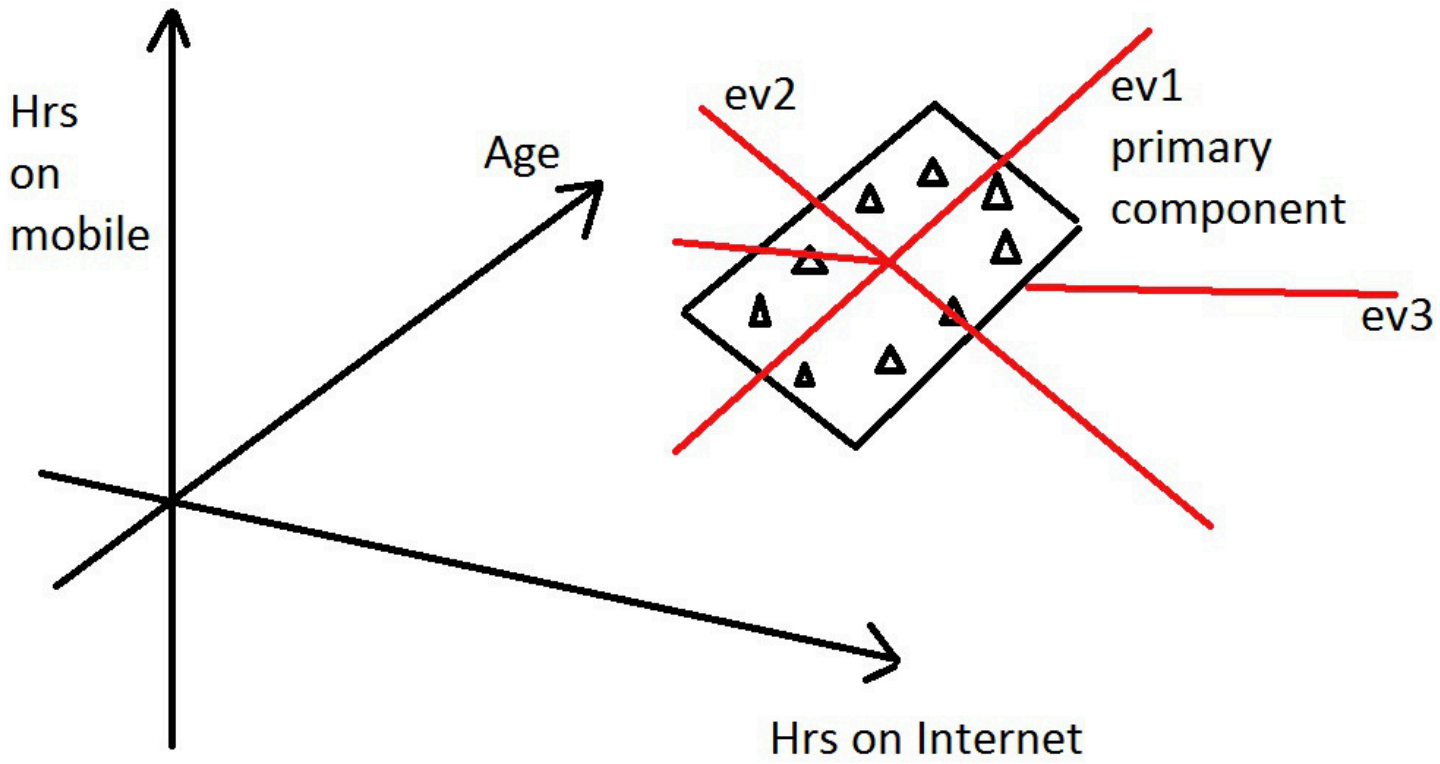
For 2-D data, the second principal component is perpendicular to the first



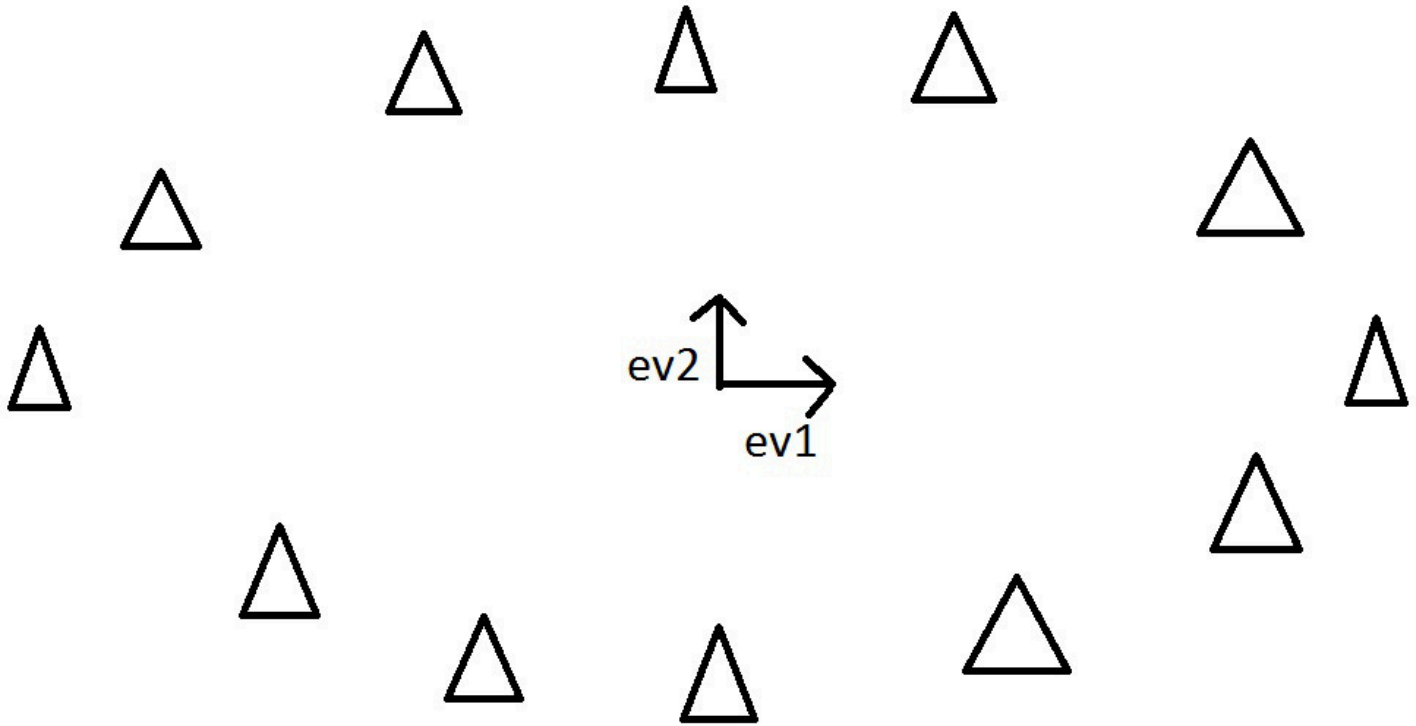
We can now use those two principal components to define a new basis for the data



What about in higher dimensions?



If the third eigenvalue is small, could ignore it to simplify the data



Determining the components

- find first axis v_1 that best models the data points d_i
- this gives a one-dimensional model for the data
- next, find another axis v_2 (orthogonal to v_1) that best models the deviation of data points from v_1
- repeat as desired for v_j up to dimensionality of the data space

Finding the axes *v*: two interpretations

1. minimize the square modeling error: $E_R = \sum \left\| d_i - v^T d_i v \right\|^2$
   where $v^T d_i v$ represents projection of the data onto axis *v*

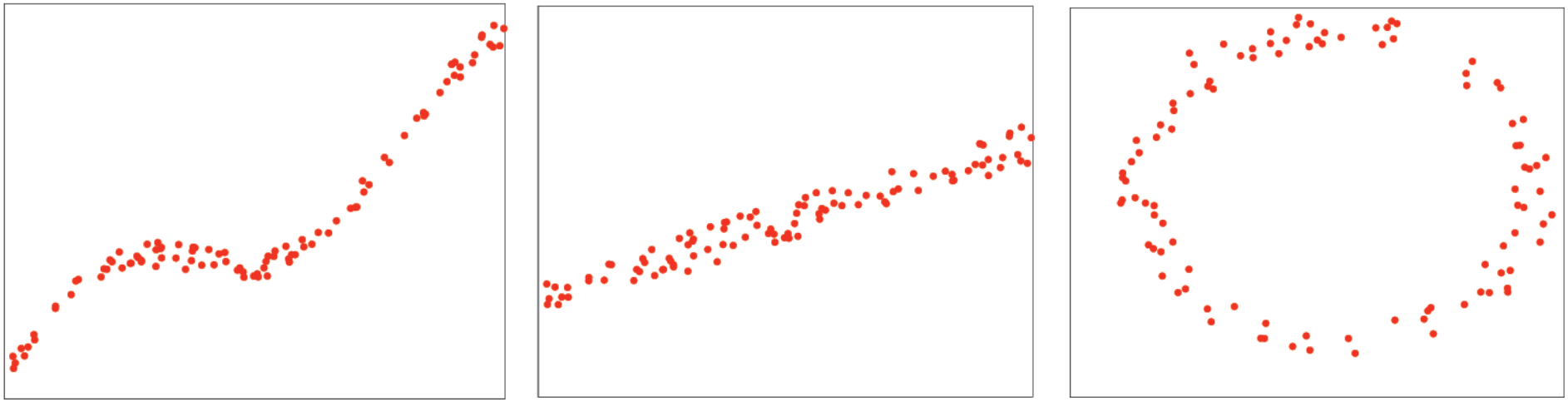
2. maximize the variance of the data after projection onto the axis *v* (assuming the projection has zero mean):

$$\sigma^2 = \frac{1}{N-1} \sum \left(d_i^T v \right)^2$$

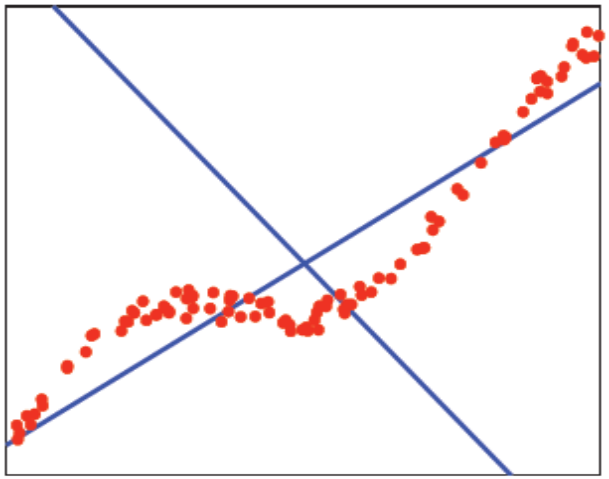
How to find them

- Principal components are simply the eigenvectors of the covariance matrix
  - 1st principal component is the eigenvector with greatest eigenvalue
  - 2nd principal component is the eigenvector with second greatest eigenvalue
  - …

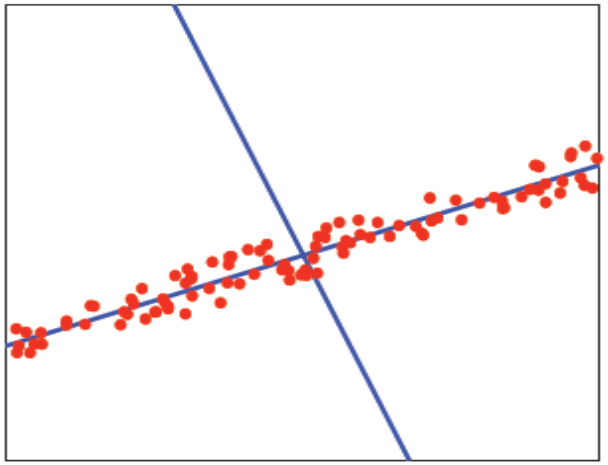
How many dimensions do you need?
Depends on the data



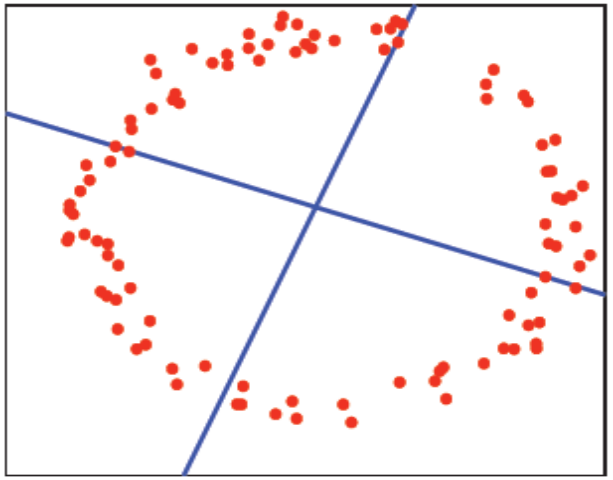
First two principal components



λ_1 = 0.0938, λ_2=0.0007



λ_1 = 0.1260, λ_2=0.0054



λ_1 = 0.0884, λ_2=0.0725

$$\mathbf{w}_1 = \arg\max_{\|\mathbf{w}\|=1} \mathrm{var}\{\mathbf{w}^T\mathbf{x}\} = \arg\max_{\|\mathbf{w}\|=1} E\left\{\left(\mathbf{w}^T\mathbf{x}\right)^2\right\}$$

Face Recognition

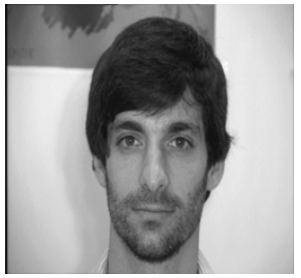
- *Identify face from a database of known faces.*

?

From MIT database







Why is this hard?

- faces are similar, same set of features (e.g. eyes, nose, mouth) in same configuration
- have to find representation that captures enough features to identify face from database of faces.
- early attempts – model relations between face features
- ignores important information about texture and shape

Face Recognition using PCA

[Sirovich and Kirby, 1987, 1990]

- PCA provides a set of standard ingredients to model faces
- good results using simple recognition algorithms with eigenface decomposition, e.g. nearest neighbor classifier
- found that 50 Principal Components give an average modeling error of 3.68% over ten test images
- better results by performing PCA on local features of an image

Claimed advantages

- good for vague recognition that face is “known” or “unknown”
- capture global, non-subjective features of faces

Eigenfaces
[Turk & Pentland, 1991]