Lecture 20: Dimensionality Reduction

- Overview
- Self-organizing maps
- Principal component analysis

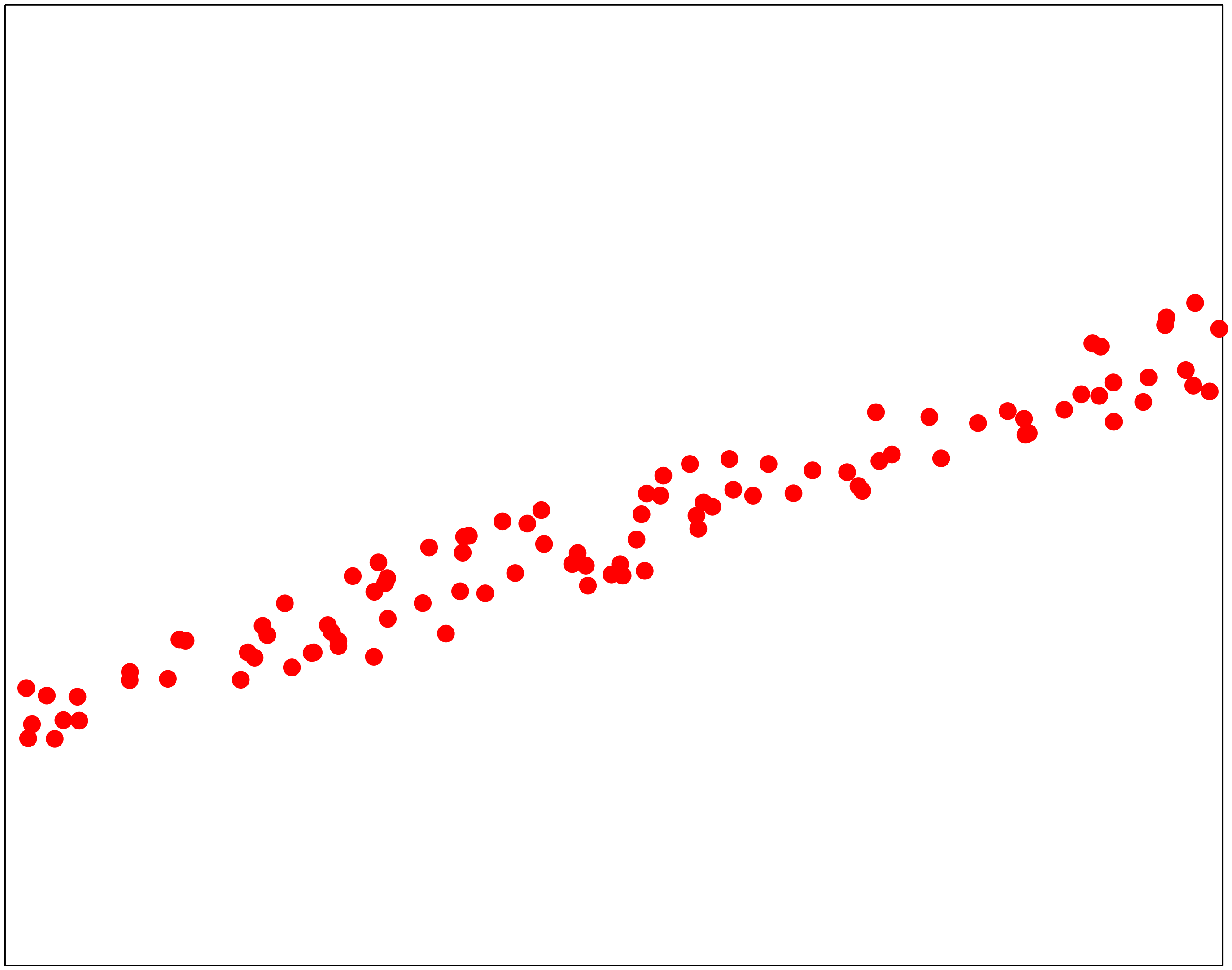
Motivation for dimensionality reduction

- Clustering, flat or hierarchical, can group the data according to similarity, helping visualization and discovery.
- But we still cannot *plot* high-dimensional (or non-numeric) data.
- We also have to guess at the number of clusters.
- We may want to “understand” better how the data was generated or how “variable” it is
- Dimensionality reduction (or embedding) techniques:
  - Assign instances to new coordinates, in a space that is much smaller-dimensional (even 2D or 3D for visualization).
  - Approximately preserve similarity/distance relationships between instances.
  - Allow us to “see” distance relationships more directly.

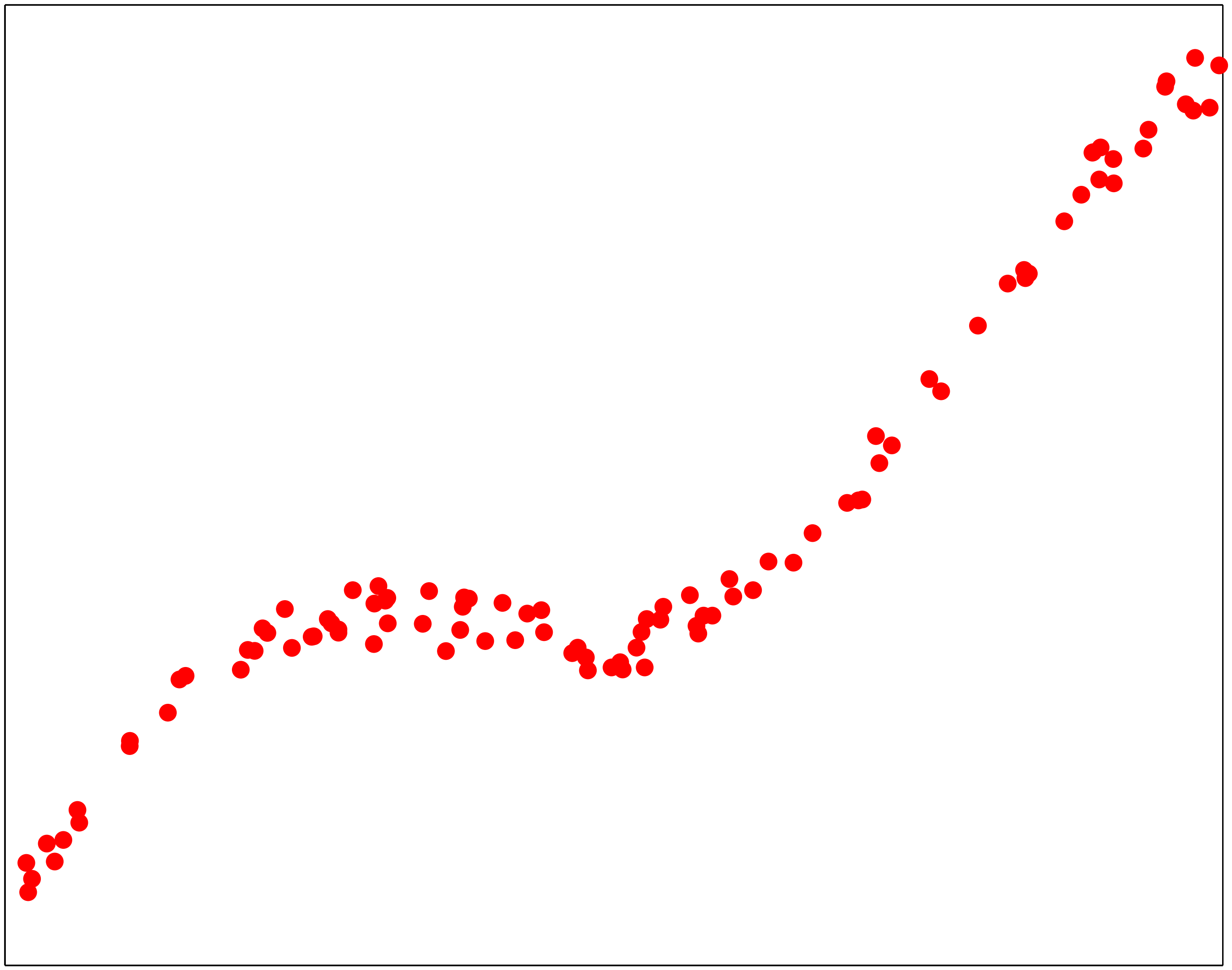
What is dimensionality reduction?

- Mapping instances to real vectors, usually with few dimensions
- Possible uses:
  - Visualization, comparison
  - Outlier detection
  - Further machine learning
- Some techniques:
  - Principal components analysis (linear)
  - Independent components analysis (linear or nonlinear)
  - Self-organizing maps (nonlinear)
  - Multi-dimensional scaling (nonlinear, allows non-numeric data objects)

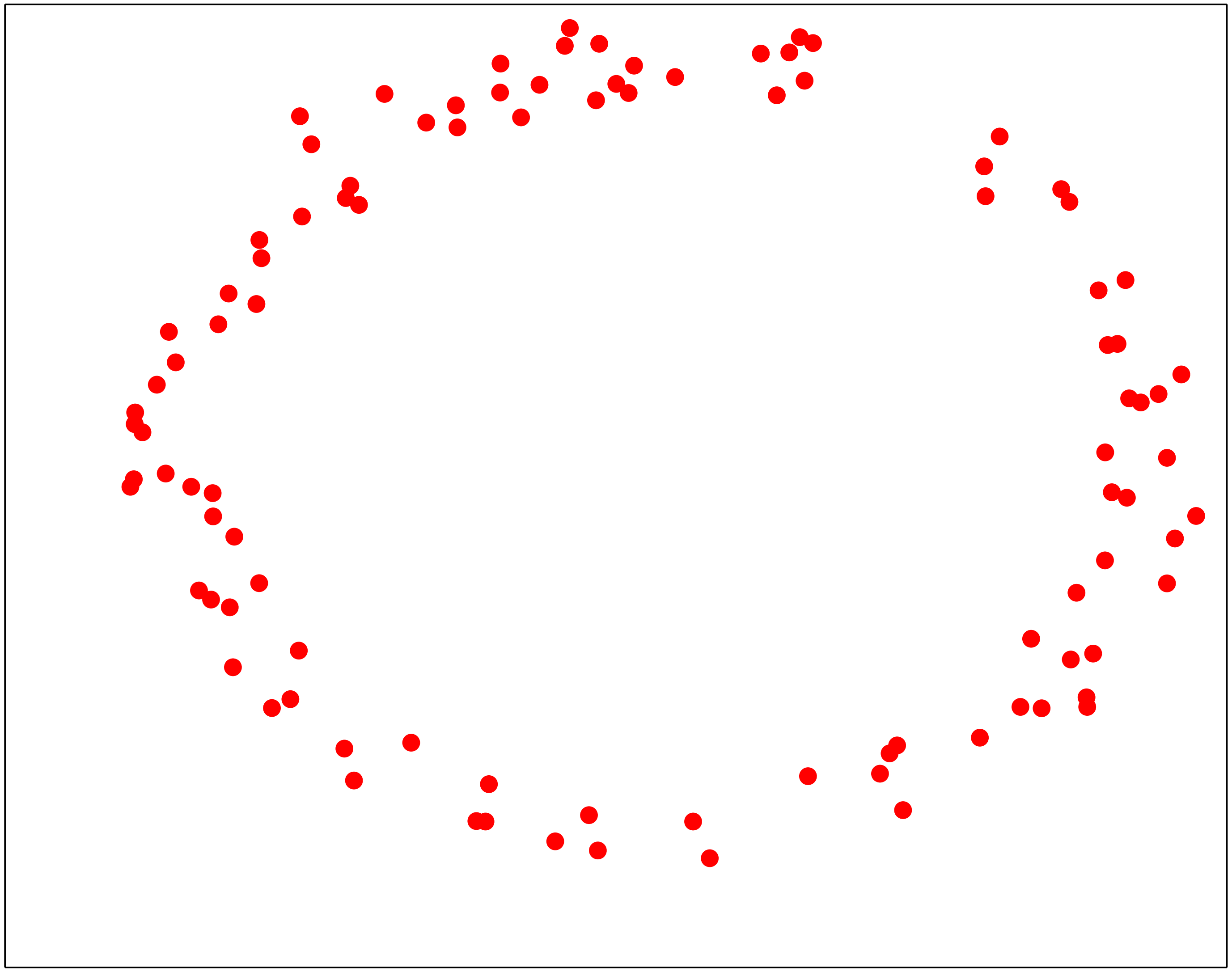
What is the true dimensionality of this data?



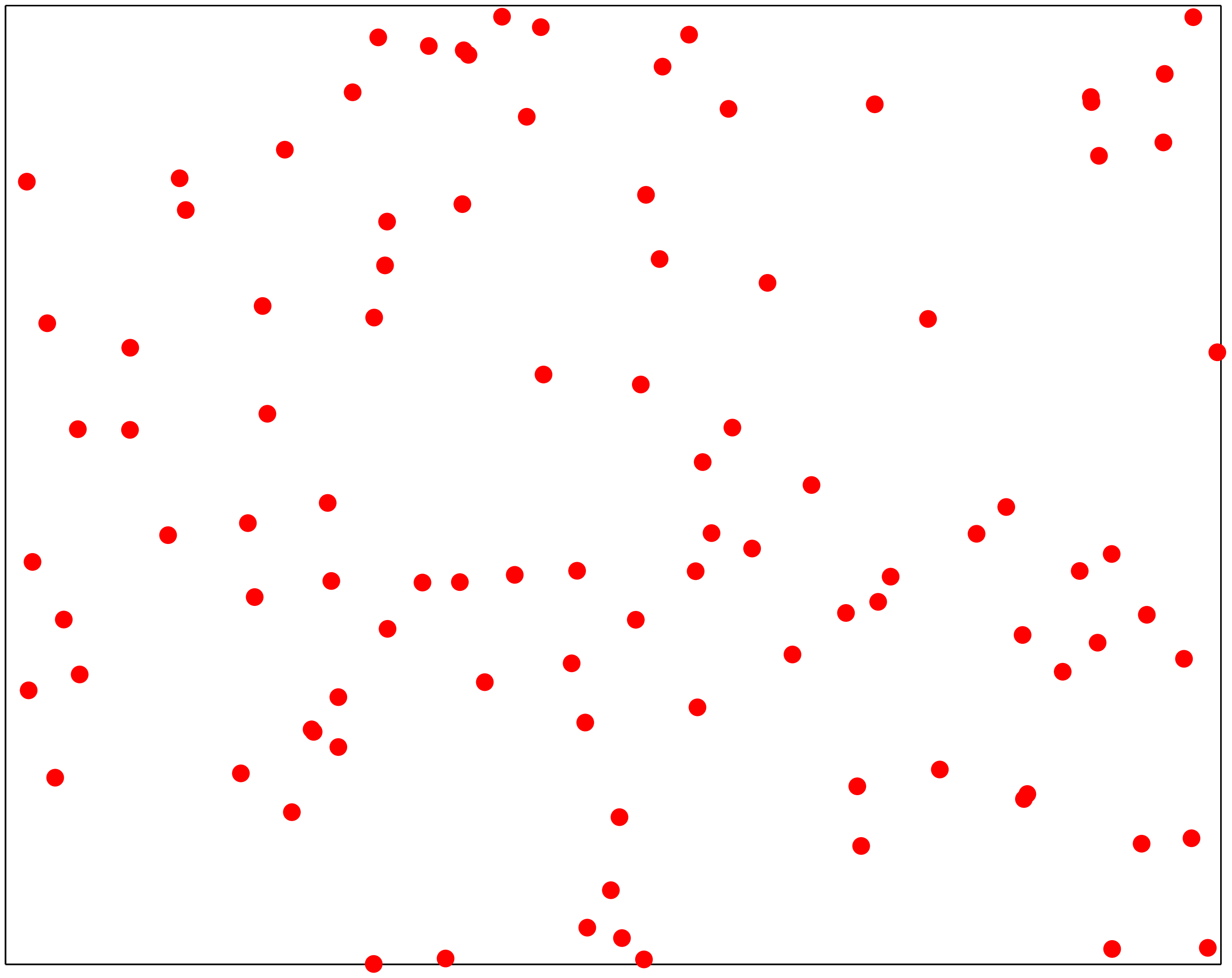
What is the true dimensionality of this data?



What is the true dimensionality of this data?



What is the true dimensionality of this data?



Remarks

- All dimensionality reduction techniques are based on an implicit assumption that the data lies along some *low-dimensional manifold*
- This is the case for the first three examples, despite being plotted in 2D
- In the last example, the data has been generated randomly in 2D, so no dimensionality reduction is possible without losing information
- The first three cases are in increasing order of difficulty, from the point of view of existing techniques.

Self-organizing maps

- Assume the data objects are real vectors of length n.
- Try to stretch a “grid” of points in n-dimensional space to approximate the data.
- The indices of the grid points indicate *neighborhood* relationships
- E.g., in 2D, $G(i, j)$ is neighbor with $G(i - 1, j)$, $G(i + 1, j)$, $G(i, j - 1)$, $G(i, j + 1)$.
- The grid points are iteratively moved, ”pulled”, by data points, similar to how the centroids of K-means clustering move around.
- The data can then be visualized by mapping each object to the nearest grid point.

Self-organizing maps

- Inputs:
  - A set $D = \{\mathbf{x_1}, \ldots, \mathbf{x_m}\}$ of n-dimensional real vectors.
  - A dimension for the grid (1,2 or 3 if we want to plot it.)
  - Number of grid points along each dimension.
- Output: Coordinates G in $\Re^n$ for each grid-point.

  E.g., for the 2D grid case, $G(i, j) \in \Re^n$ specifies the coordinates of grid-point (i, j).

SOM learning algorithm

- Initialize the grid points.
- Repeat
  - Choose a data point $\mathbf{x}$ at random.
  - Find the nearest grid point; e.g., in 2D:

$$\mathbf{G}^* = G(i^*, j^*) = \arg\min_{i,j} \|G(i,j) - \mathbf{x}\|$$

  - Find the "neighborhood" of $\mathbf{G}^*$
  - Move all points $\mathbf{G}$ in the neighborhood towards $\mathbf{x}$:

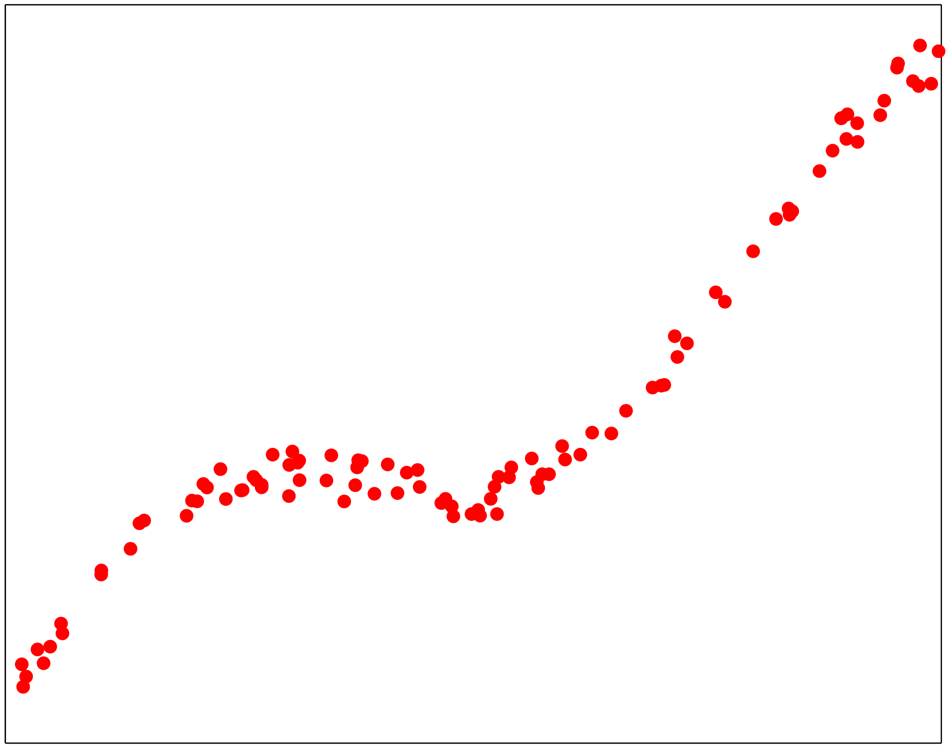
$$\mathbf{G} \leftarrow \mathbf{G} + \alpha e(\mathbf{x}, \mathbf{G})(\mathbf{x} - \mathbf{G})$$

    where $e(\mathbf{x}, \mathbf{G})$ is a similarity function, equal to 1 if $\mathbf{x} = \mathbf{G}$ and decreasing with $|\mathbf{x} - \mathbf{G}|$ (e.g. Gaussian)

Remarks

- DHS has nice pictures of SOMs at work
- Typically the learning rate $\alpha \to 0$ with time
- The SOM builds a *topographical map* of the input space, putting more points where the data is dense
- Instances that are close in the input space will be mapped to units which are neighbors in the grid.
- If the data approximately lies on a curve or surface, the SOM may capture that structure, but:
  - Different runs can find different solutions.
  - If we try to fit data on a 2D surface with a 1D grid, well...
- More sophisticated versions of SOMs use different updating rules, different neighboring functions

Back to the example

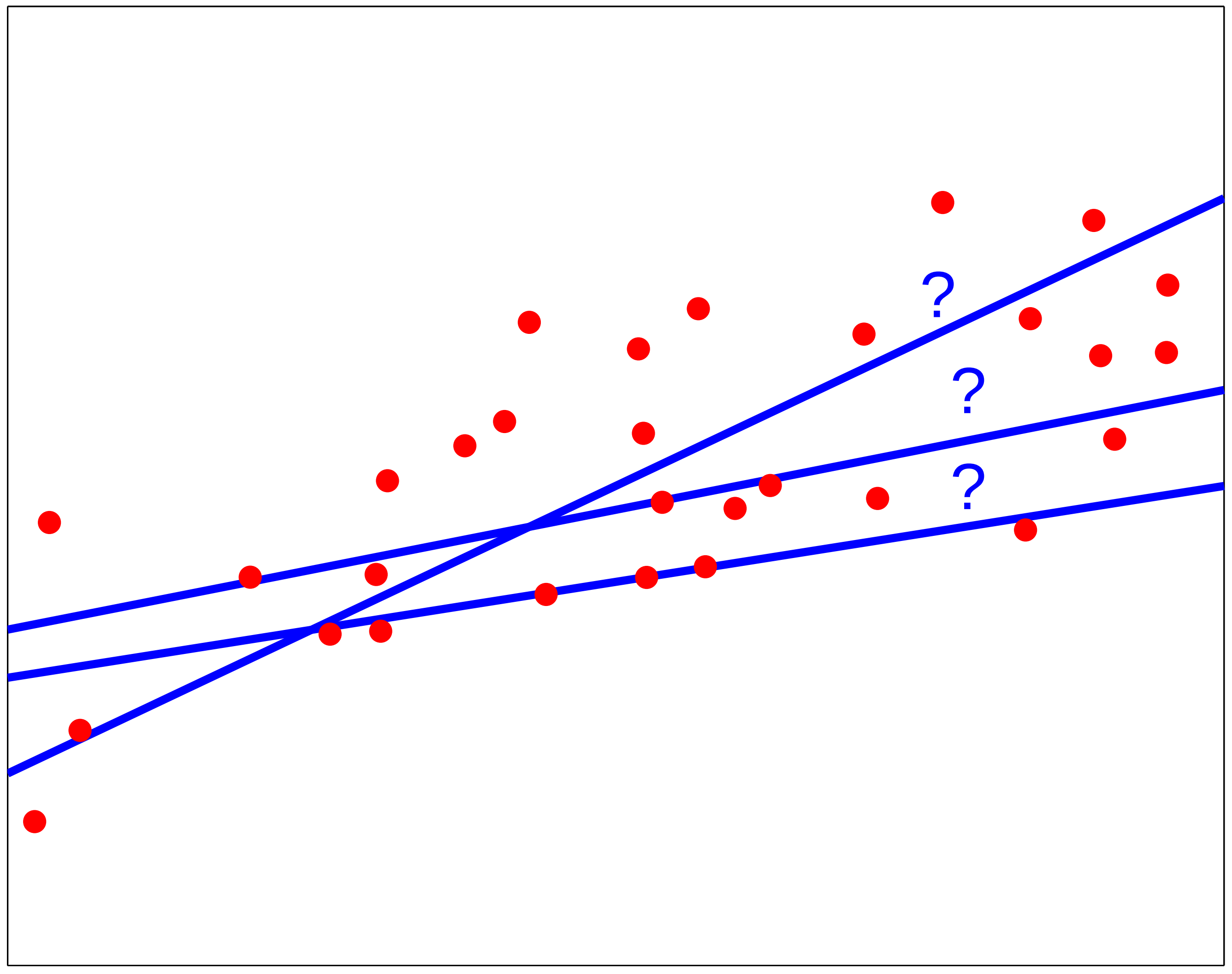


- SOMs will put units along the curve, where the data lies
- But there are other interesting things about this data!
- Can we find a good model for it?

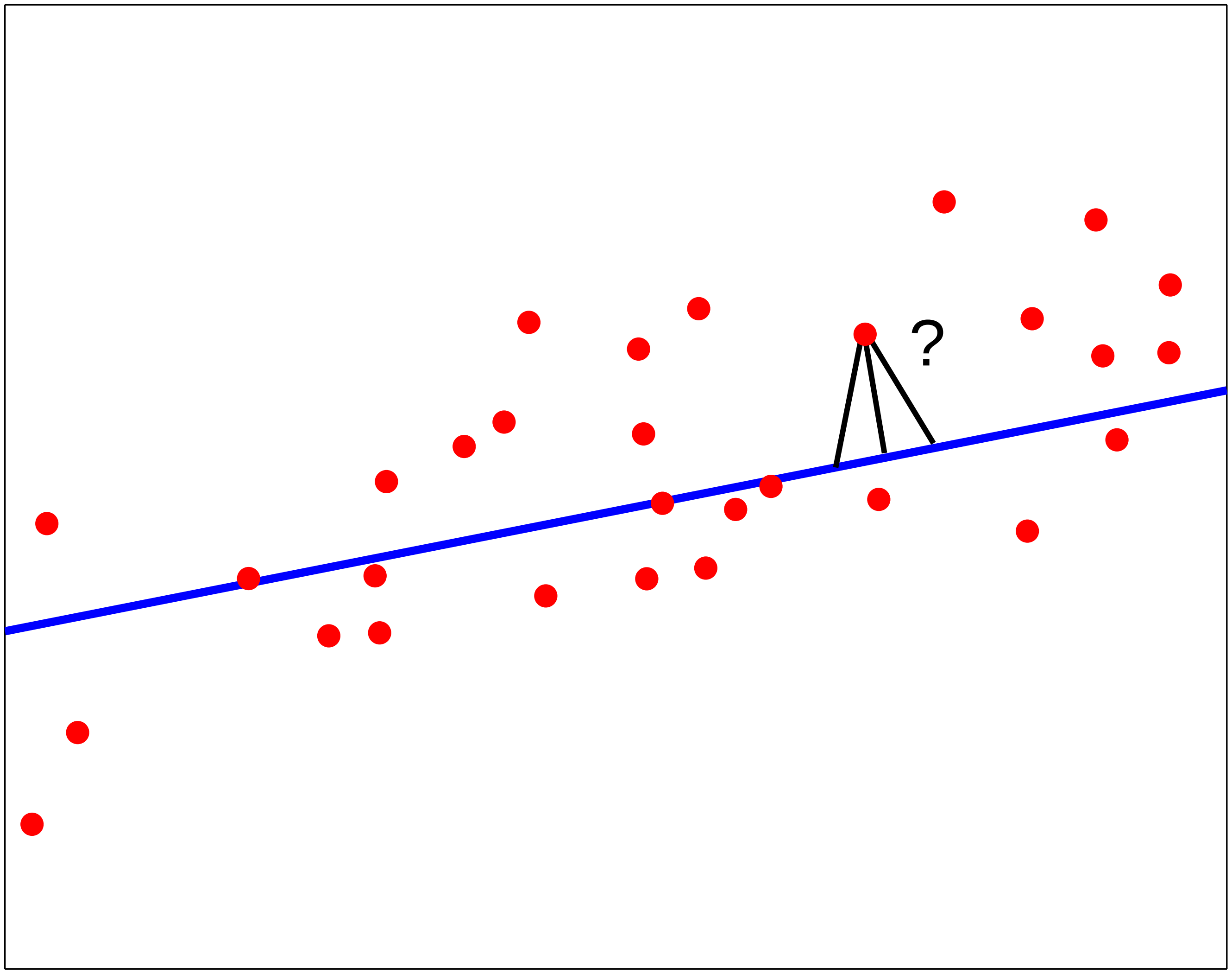
Simple Principal Component Analysis (PCA)

- Given: m data objects, each a length-n real vector.
- Suppose we want a 1-dimensional representation of that data, instead of n-dimensional.
- Specifically, we will:
  - Choose a line in $\Re^n$ that “best represents” the data.
  - Assign each data object to a point along that line.

Which line is best?



How do we assign points to lines?



Recall a useful tool: Covariance

- Covariance quantifies a linear relationship (if any) between two random variables X and Y.

$$Cov(X,Y) = E\{(X - E(X))(Y - E(Y))\}$$

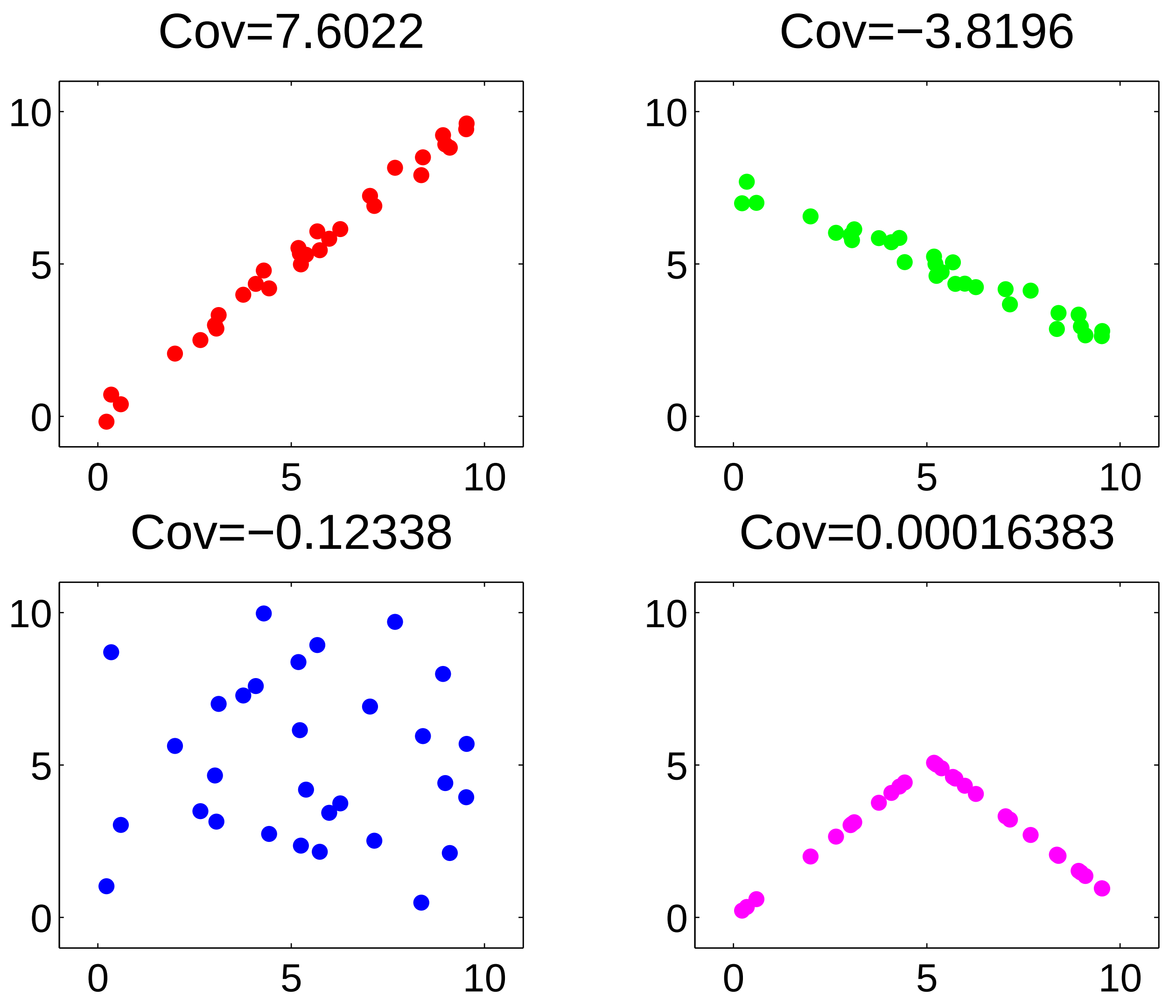
- Given m samples of X and Y, covariance can be estimated as

$$\frac{1}{m-1}\sum_{i=1}^{m}(x_i - \mu_X)(y_i - \mu_Y) \ ,$$

where $\mu_X = \sum_{i=1}^{m} x_i$ and $\mu_Y = \sum_{i=1}^{m} y_i$.

- Note: $Cov(X,X) = Var(X)$.

Examples



Reconstruction error

- Let our line be represented as $\mathbf{b} + \alpha\mathbf{v}$ for $\mathbf{b}, \mathbf{v} \in \Re^n$, $\alpha \in \Re$. For later convenience, assume $\|\mathbf{v}\| = 1$.
- Each instance $\mathbf{x}_i$ is assigned a point on the line $\hat{\mathbf{x}}_i = \mathbf{b} + \alpha_i\mathbf{v}$.
- The (squared Euclidean) reconstruction error for instance i is

$$\|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 = \sum_{j=1}^{n} (\mathbf{x}_i(j) - \hat{\mathbf{x}}_i(j))^2$$

- We want to choose $\mathbf{b}$, $\mathbf{v}$, and the α_i to minimize the total reconstruction error over all data points:

$$R = \sum_{i=1}^{m} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2$$

A constrained optimization problem!

$$\begin{array}{rl} \text{min} & \|\mathbf{x}_i - (\mathbf{b} + \alpha_i \mathbf{v})\|^2 \\ \text{w.r.t.} & \mathbf{b}, \mathbf{v}, \alpha_i, i = 1, \ldots n \\ \text{s.t.} & \|v\|^2 = 1 \end{array}$$

We can write down the Lagrangian and try to solve directly, but this gets a bit difficult... (see homework 7)

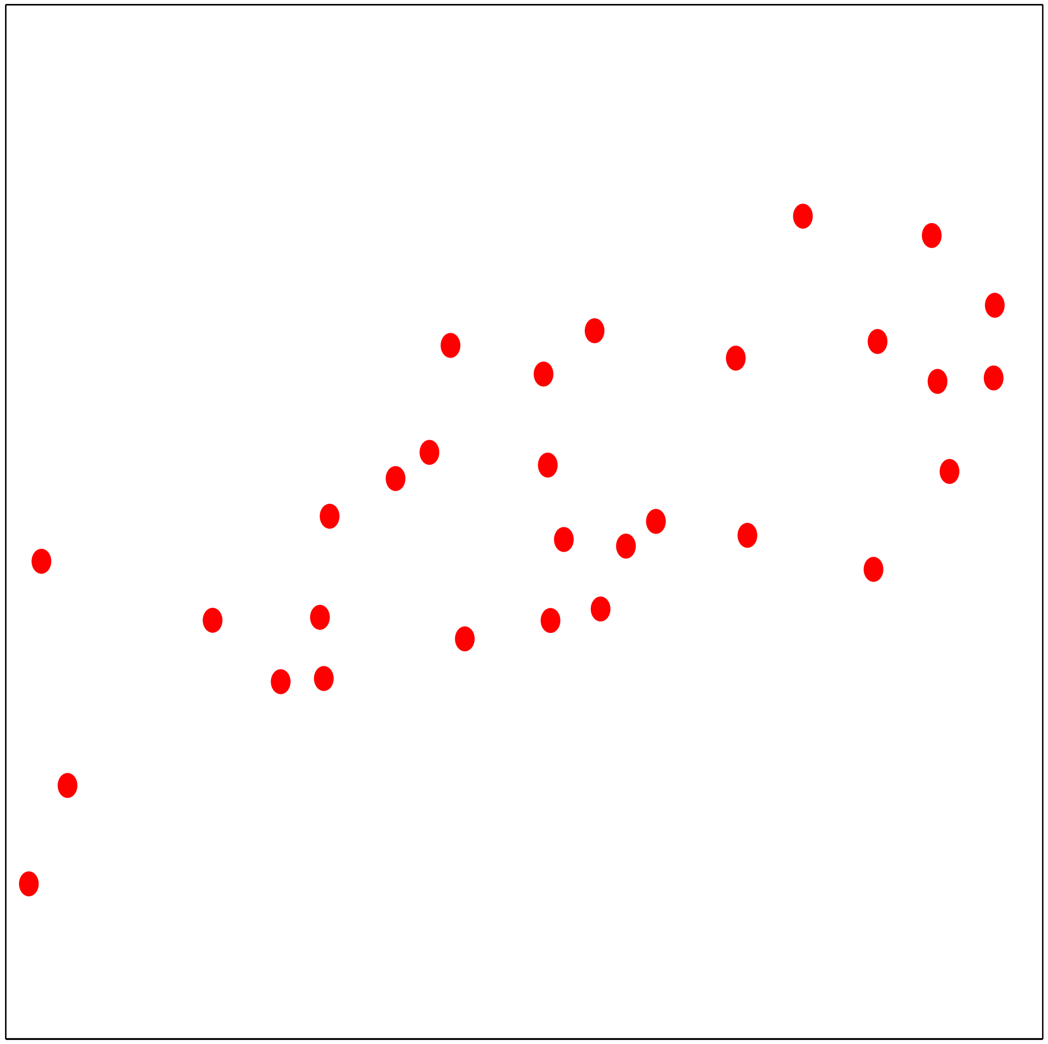
Minimizing reconstruction error: b and the α_i

- Suppose we fix $\mathbf{v}$. Now we have an unconstrained optimization problem!
- By taking the gradient of the reconstruction error and setting it to 0 we get that an optimal choice for $\mathbf{b}$ is

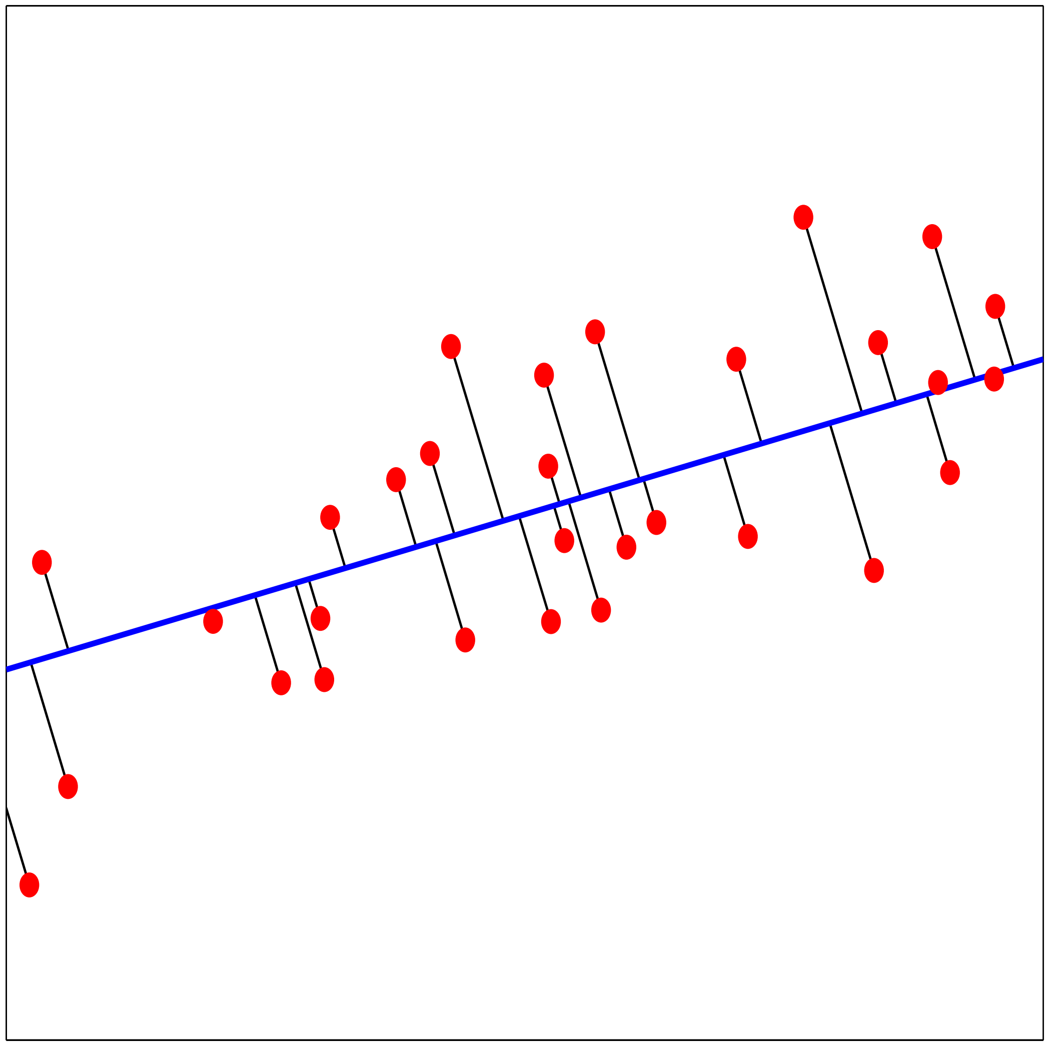
$$\mathbf{b} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i ,$$

- From this, we get $\alpha_i = \mathbf{v}\cdot(\mathbf{x}_i - \mathbf{b})$
- By substituting, we get: $\hat{\mathbf{x}}_i = \mathbf{b} + \mathbf{v}\cdot(\mathbf{x}_i - \mathbf{b})$.
- Intuitively:
  - The line goes through the centroid of the data.
  - Instances are projected orthogonally on the line to get the associated point.

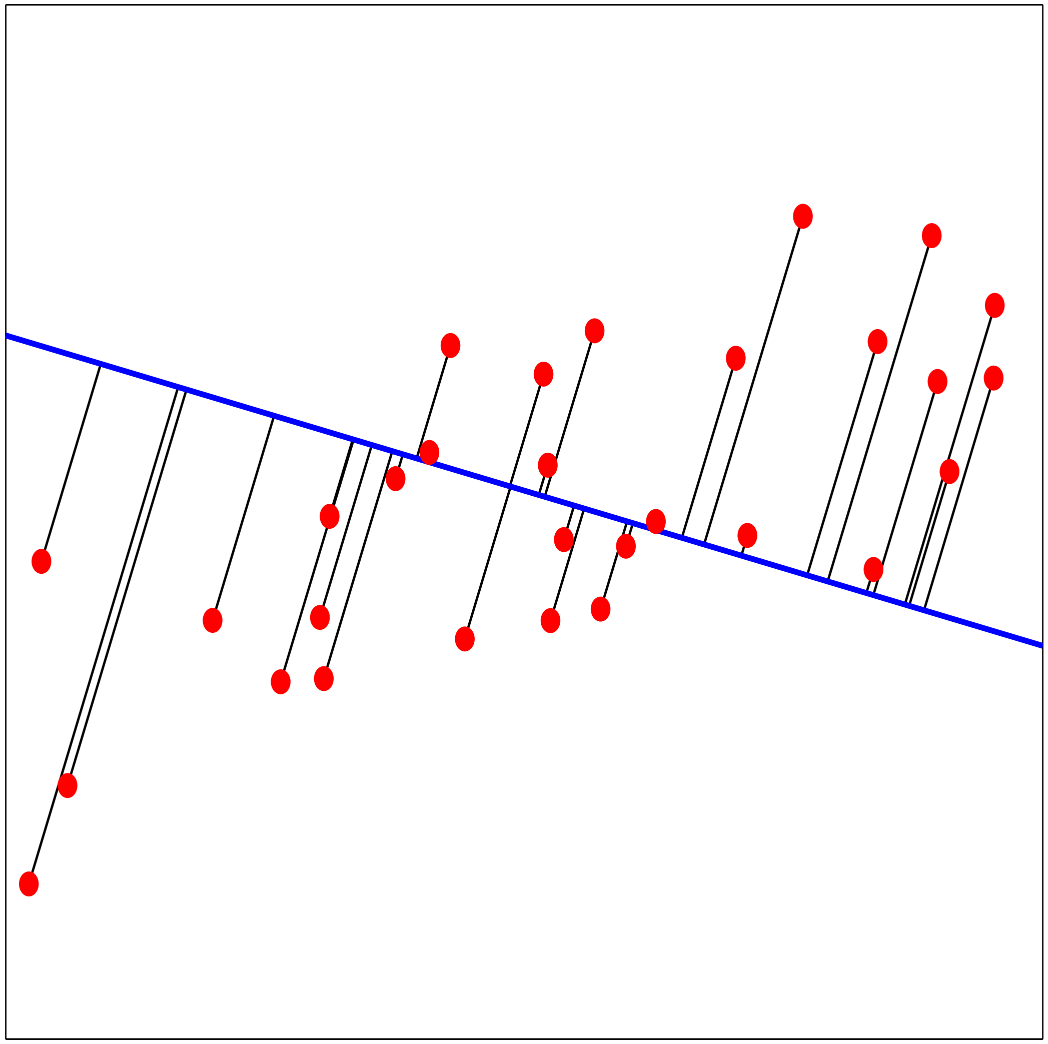
Example data



Example with $\mathbf{v} \propto (1, 0.3)$



Example with $\mathbf{v} \propto (1, -0.3)$



Minimizing reconstruction error: the scatter matrix

- Substituting back into the formula for the reconstruction error, we get that $\mathbf{v}$ should maximize

$$\mathbf{v}^T S \mathbf{v} \ ,$$

where S is an $n \times n$ matrix with

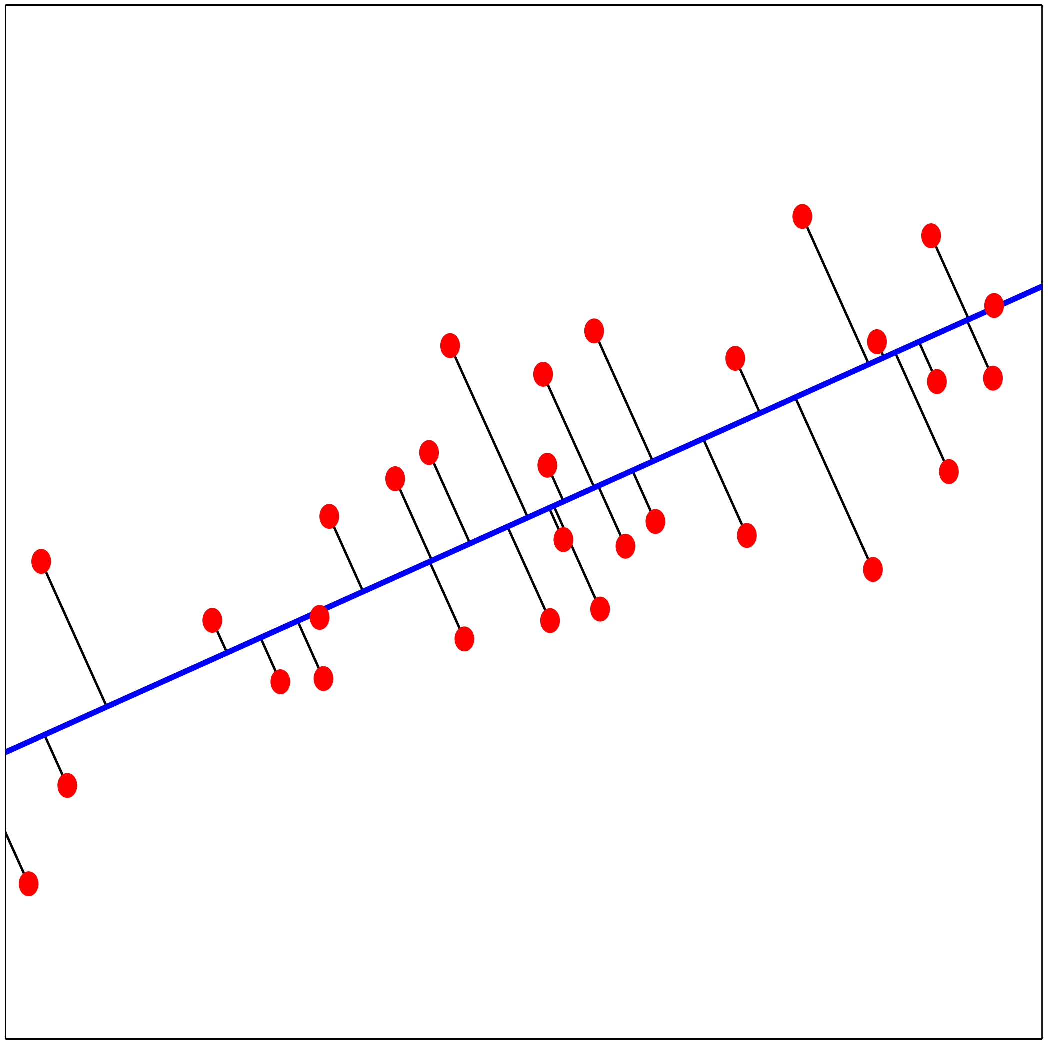
$$S(k,l) = \sum_{i=1}^{m} (\mathbf{x}_i(k) - \mathbf{b}(k))(\mathbf{x}_i(l) - \mathbf{b}(l))$$

- Note that $S(k,l)$ is proportional to the estimated covariance between the kth and lth dimension in the data.
- S is called the **scatter matrix**.

Optimal choice of $\mathbf{v}$

- Recall: an **eigenvector** $\mathbf{u}$ of a matrix A satisfies $A\mathbf{u} = \lambda\mathbf{u}$, where $\lambda \in \Re$ is the **eigenvalue**.
- Fact: the scatter matrix, S, has n non-negative eigenvalues and n orthogonal eigenvectors.
- The $\mathbf{v}$ that maximizes $\mathbf{v}^T S\mathbf{v}$ is the eigenvector of S with the largest eigenvalue (homework 7)

Example with optimal line: $\mathbf{b} = (0.54, 0.52)$, $\mathbf{v} \propto (1, 0.45)$



Remarks

- The line $\mathbf{b} + \alpha\mathbf{v}$ is the **first principal component**.
- The variance of the data along the line $\mathbf{b} + \alpha\mathbf{v}$ is as large as along any other line.
- $\mathbf{b}$, $\mathbf{v}$, and the α_i can be computed easily in polynomial time.

Reduction to d dimensions

- More generally, we can create a d-dimensional representation of our data by projecting the instances onto a hyperplane $bfb + \alpha^1 \mathbf{v}_1 + \ldots + \alpha^d \mathbf{v}_d$.
- If we assume the $\mathbf{v}_j$ are of unit length and orthogonal, then the optimal choices are:
  - $\mathbf{b}$ is the mean of the data (as before)
  - The $\mathbf{v}_j$ are orthogonal eigenvectors of S corresponding to its d largest eigenvalues.
  - Each instance is projected orthogonally on the hyperplane.

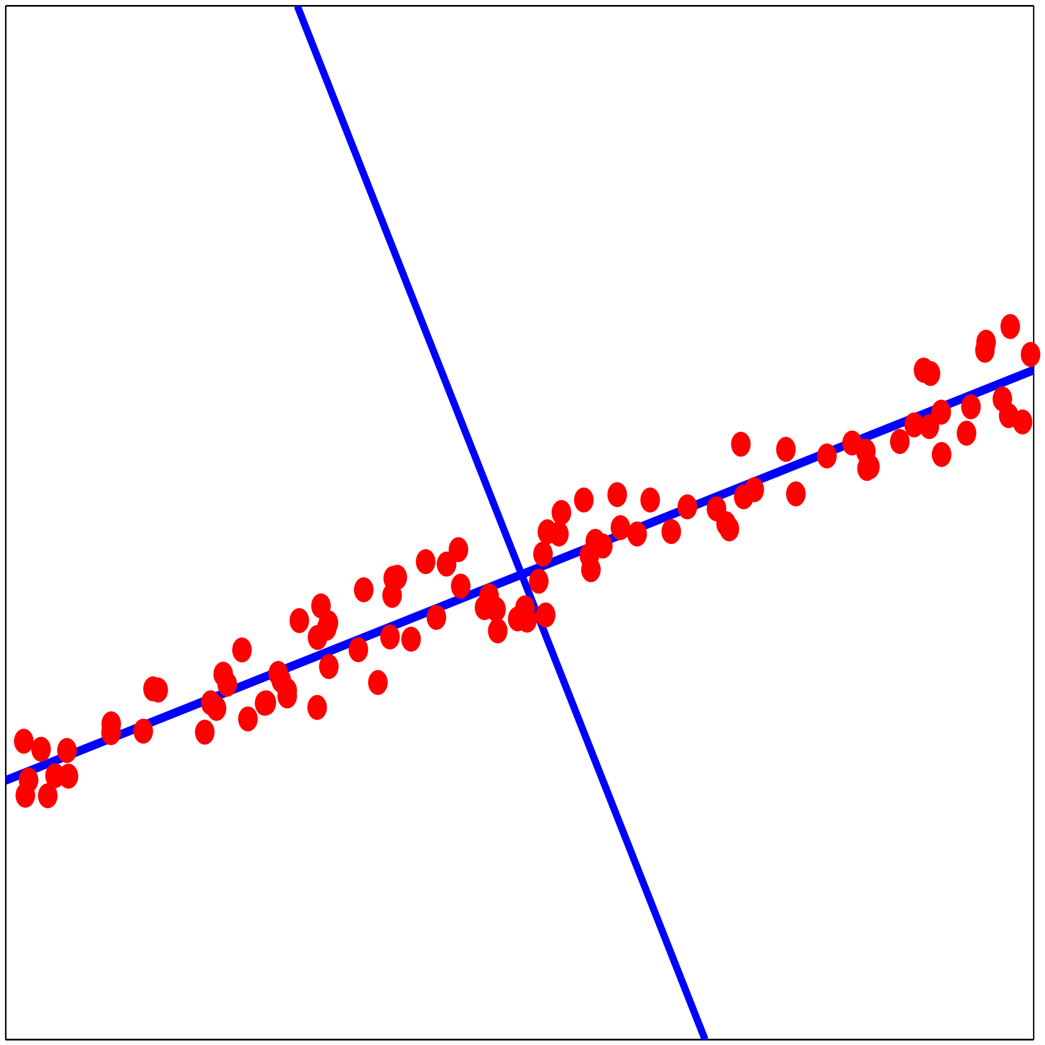
Remarks

- $\mathbf{b}$, the eigenvalues, the $\mathbf{v}_j$, and the projections of the instances can all be computing in polynomial time.
- The magnitude of the j^{th}-largest eigenvalue, λ_j, tells you how much variability in the data is captured by the j^{th} principal component
- So you have feedback on how to choose d!
- When the eigenvalues are sorted in decreasing order, the proportion of the variance captured by the first d components is:

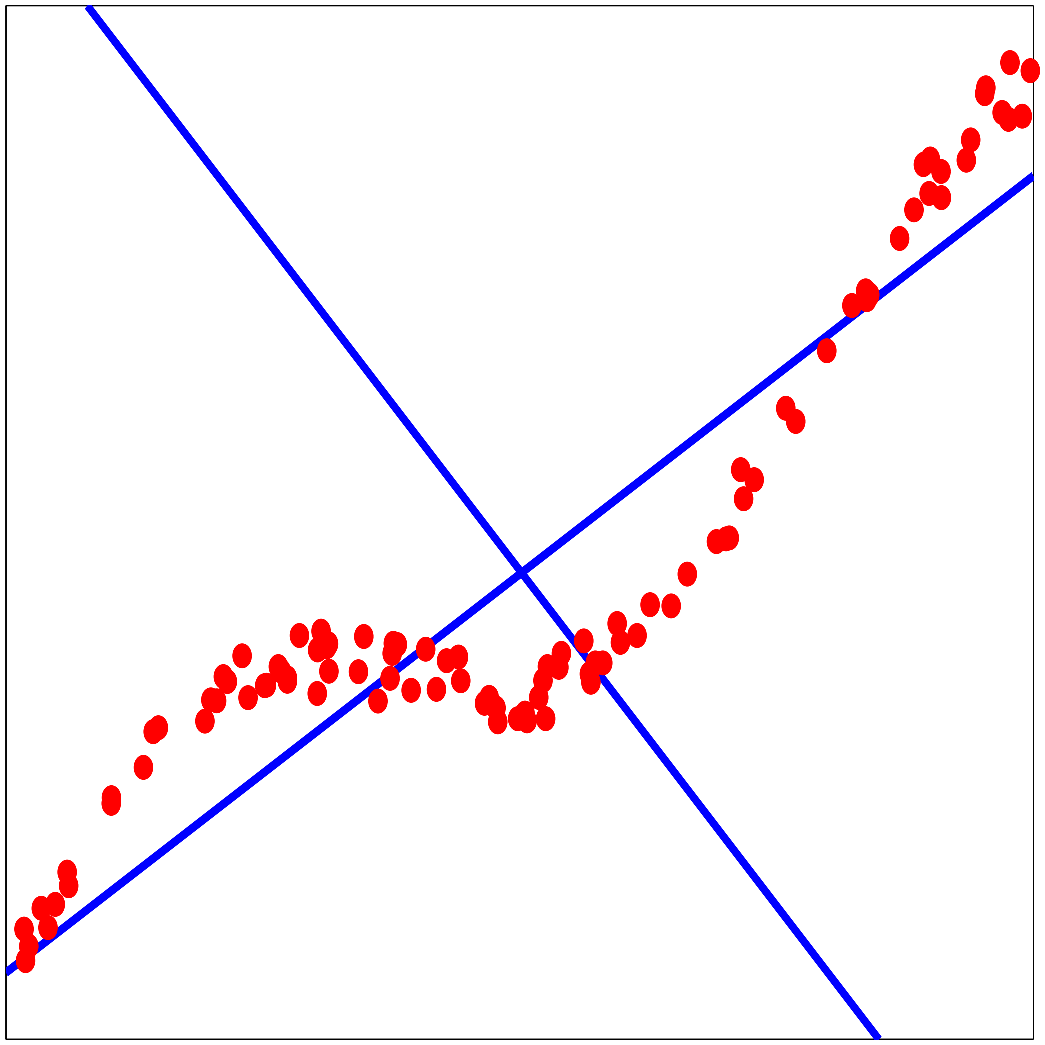
$$\frac{\lambda_1 + \cdots + \lambda_d}{\lambda_1 + \cdots + \lambda_d + \lambda_{d+1} + \cdots + \lambda_n}$$

- So if a “big” drop occurs in the eigenvalues at some point, that suggests a good dimension cutoff

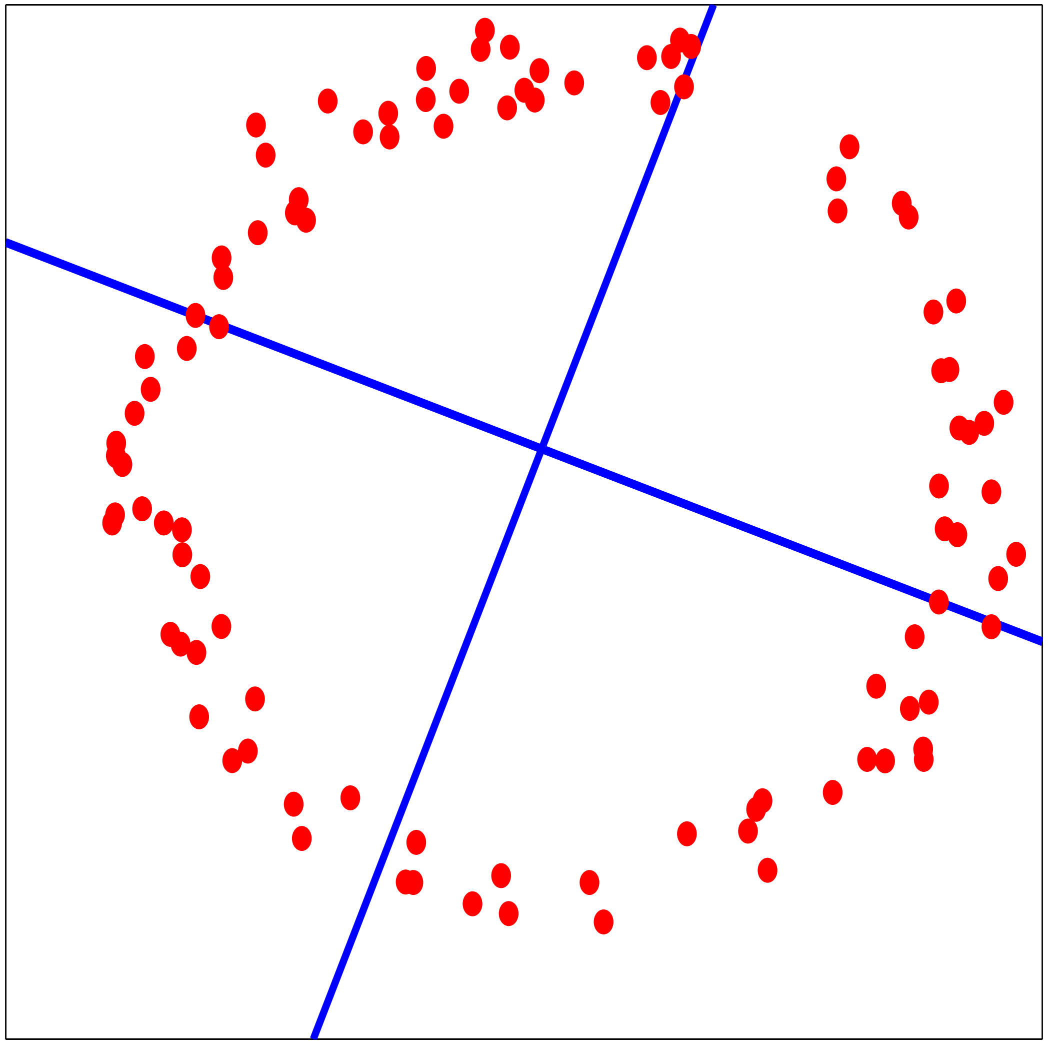
Example: $\lambda_1 = 0.0938, \lambda_2 = 0.0007$



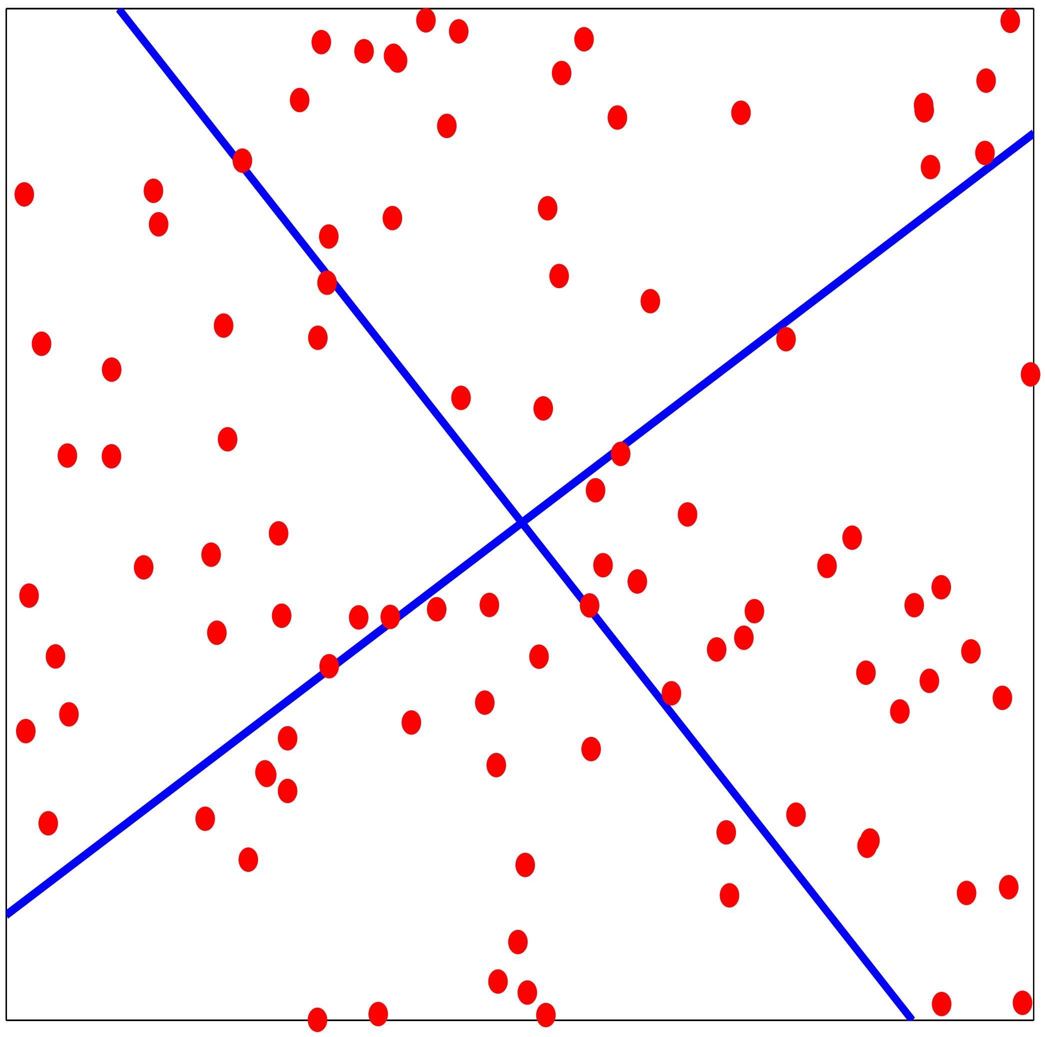
Example: $\lambda_1 = 0.1260, \lambda_2 = 0.0054$



Example: $\lambda_1 = 0.0884, \lambda_2 = 0.0725$



Example: $\lambda_1 = 0.0881, \lambda_2 = 0.0769$



More remarks

- Outliers have a big effect on the covariance matrix, so they can affect the eignevectors quite a bit
- A simple examination of the pairwise distances between instances can help discard points that are very far away (for the purpose of PCA)
- If the variances in the original dimensions vary considerably, they can “muddle” the true correlations. There are two solutions:
  - work with the correlation of the original data, instead of covariance matrix
  - normalize the input dimensions individually before PCA
- In certain cases, the eigenvectors are meaningful; e.g. in vision, they can be displayed as images (“eigenfaces”)

Uses of PCA

- Pre-processing for a supervised learning algorithm, e.g. for image data, robotic sensor data
- Used with great success in image and speech processing
- Visualization
- Exploratory data analysis
- Removing the linear component of a signal (before fancier non-linear models are applied)