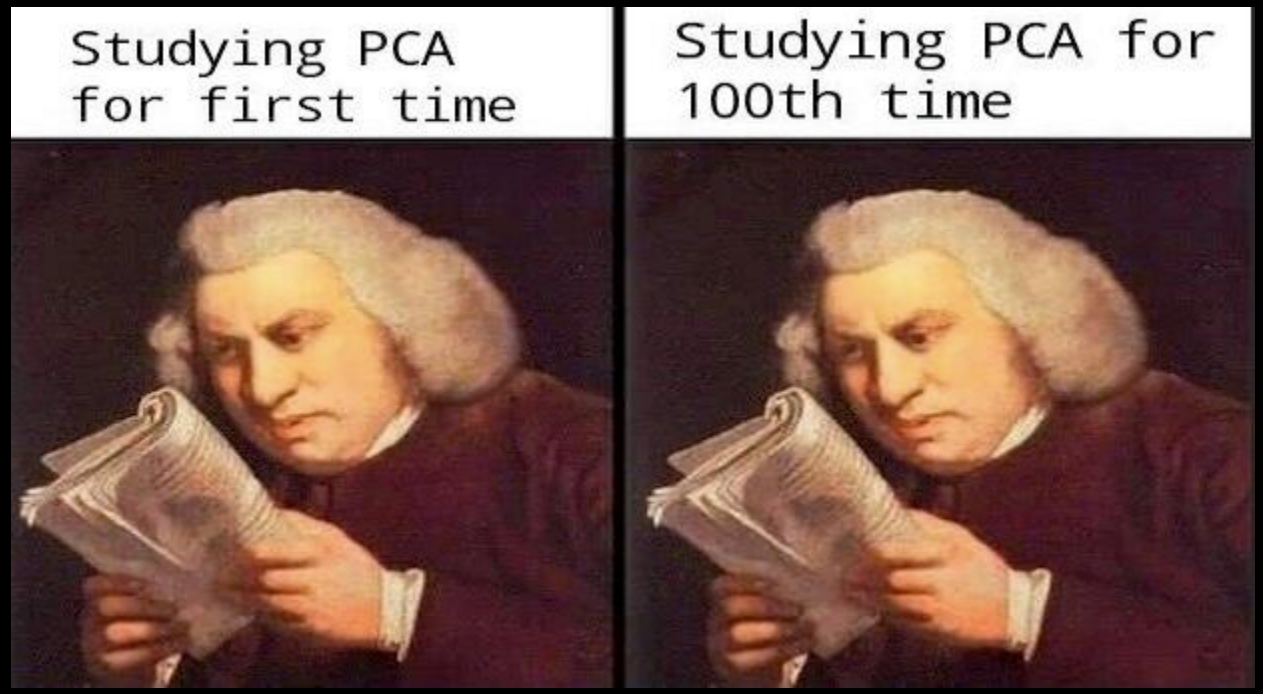


INFO 251: Applied Machine Learning

Dimensionality Reduction

Course Outline

- Causal Inference and Research Design
  - Experimental methods
  - Non-experiment methods
- **Machine Learning**
  - Design of Machine Learning Experiments
  - Linear Models and Gradient Descent
  - Non-linear models
  - Fairness and Bias in ML
  - Neural models
  - Deep Learning
  - Practicalities
  - **Unsupervised Learning**
- Special topics

Announcements

- Upcoming classes
  - Today: Dimensionality reduction
  - Thursday (April 24): ML + Causal Inference
  - Tuesday (April 29): ML "in the wild"
  - Thursday (May 1) : Final quiz
    - Email full teaching team (Suraj, Satej, Josh) if you have a conflict
- Final problem set
  - Available on bCourses
  - Due May 5

Last Lecture: Key concepts

- Supervised vs. unsupervised learning
- Why cluster?
- k-Means clustering
- Hierarchical clustering
- Distance and linkage functions
- Motivation for dimensionality reduction

Sample Quiz Questions

- True or False:
  - Cluster analysis is most commonly used for *unsupervised* learning
  - The "Euclidean distance" (L2 norm) is a metric that is non-negative, symmetric, and satisfies the triangle inequality
  - The "Manhattan distance" (L1 norm) is a metric that is non-negative, symmetric, and satisfies the triangle inequality
  - A common practice for determining how many clusters *k* to use with *k*-means clustering is to use cross-validation to choose value of *k* that minimizes error/loss

Outline

- **Multi-class classification**
- Interpreting models

- Dimensionality Reduction
  - Motivation & Intuition
  - Principal Component Analysis
  - Example: Eigenfaces
  - Other approaches to dimensionality reduction

Summary: Motivation

- **Statistical:** fewer dimensions can lead to better generalization, improve tractability
- **Visualization/Interpretation**: Visualize and understand structure of data
- **Computational:** compress data to increase time/space efficiency

Dimensionality Reduction: Goal

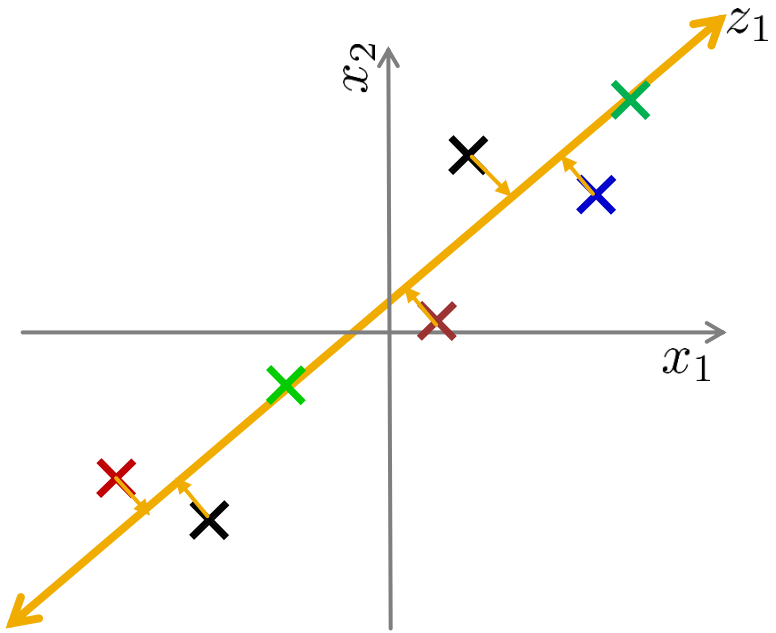
- **Idea:** Reduce each object from *m* dimensions to a corresponding object with *k* dimensions
  - $k<m$
  - Do this by "collapsing" dimensions (features)
  - $x_i \in \mathbb{R}^m \Rightarrow z_i \in \mathbb{R}^k$
- **Goal:** Minimal "loss of information"
- **Intuition:** find new dimensions that still accurately represent the variation in the original data

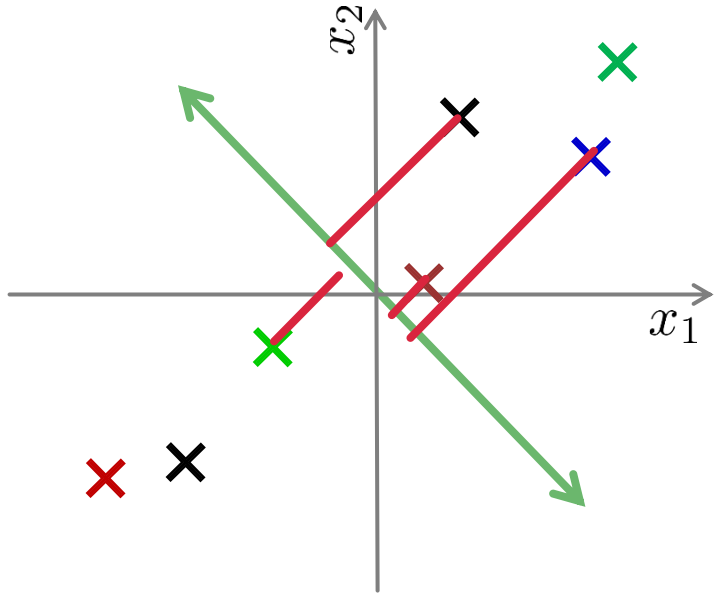
Outline

- Dimensionality Reduction
  - Motivation & Intuition
  - **Principal Component Analysis**
    - **Intuition**
    - Algorithm
    - Implementation
  - Example: Eigenfaces
  - Other approaches to dimensionality reduction

Principal Component Analysis

- Graphical intuition: Minimize projection error
- What if we chose a different line?
  1. *"Projection error" is much larger*
  2. *Projection captures less variance*
     - *Formally, we can prove these are equivalent statements*





Outline

- Dimensionality Reduction
  - Motivation & Intuition
  - **Principal Component Analysis**
    - Intuition
    - **Algorithm**
    - Implementation
  - Example: Eigenfaces
  - Other approaches to dimensionality reduction

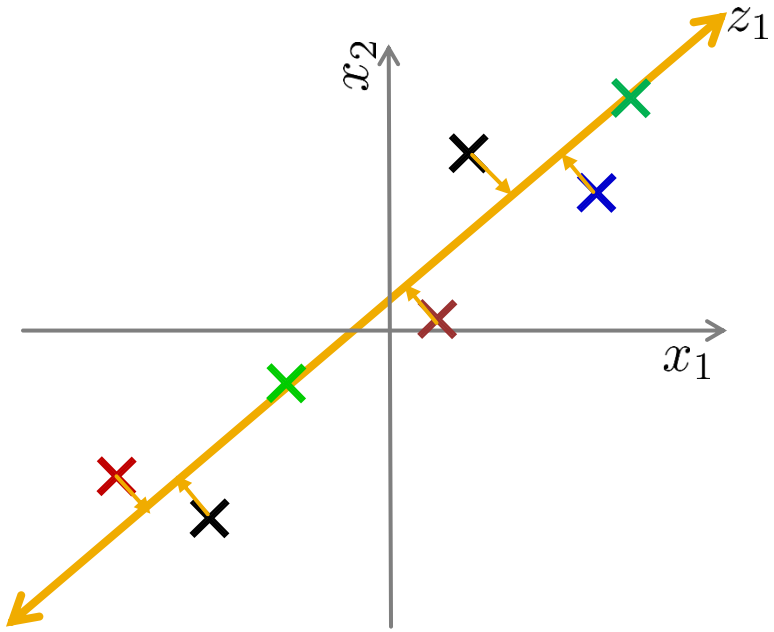
How does PCA work?

- Goal: find vectors (principal components) and compute/project old data onto new space
  - Project *m*-dimensional data (in $\mathbb{R}^m$) to *k* dimensions (in $\mathbb{R}^k$)
  - $x_i = (x_{11}, \dots, x_{1m}) \Rightarrow (z_{11}, \dots, z_{1k}), with\ k \leq m$
  - Generally, *z=f(x)*
  - In *linear* dimensionality reduction (matrix notation) $z = U^T x$
- How to choose transformation (U)?
  1. Intuitive, inefficient way
  2. Efficient, analytic solution

PCA Algorithm (the wrong way)

- Intuitive, brute force algorithm:

1. Fit a squared-distance minimizing line to the data
2. Compute residuals normal to the line
3. Iterate



PCA Algorithm (intuition)

- Just like OLS:
  - Gradient descent (MLE) yields "correct" solution
  - Analytic solution is far faster: $(X'X)^{-1}(X'Y)$
- With PCA
  - Analytic solution uses linear algebra on X (not Y!)
  - Formal derivation is tricky, but: the principal components of a set of data can be found from the (first *k*) eigenvalues and eigenvectors of the covariance matrix of the data

PCA algorithm (the right way)

- Summary:
  1. Compute **covariance matrix** of data
  2. Compute **eigenvectors** of covariance matrix
  3. Compute **projections**

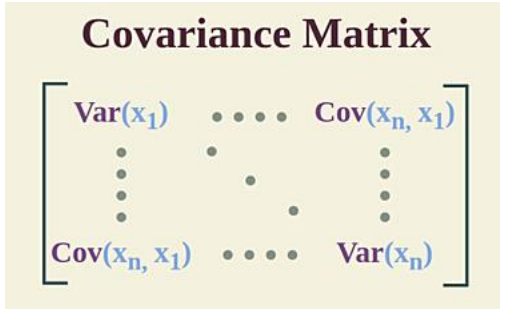
PCA algorithm

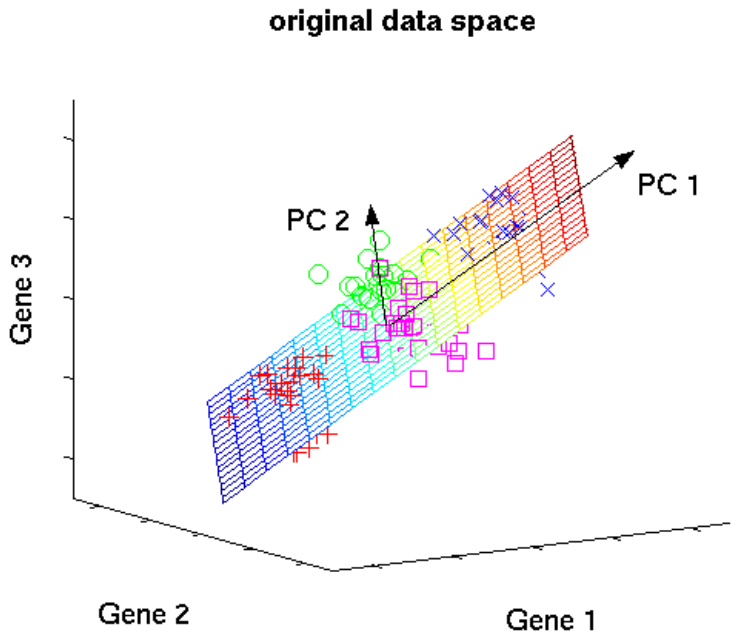
1. Compute **covariance matrix** of data X
   - Mean and covariance of data
   - Mean of data tells you "where" the cloud is
   - Covariance tells you "which way it's pointing"

$$Q = \frac{1}{N-1} X^T X$$

$$q_{j,m} = \frac{1}{n-1} \sum_{i=1}^{n} (x_{i,j} - \bar{x}_j)(x_{i,m} - \bar{x}_m)$$

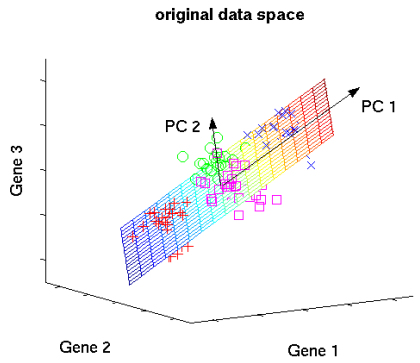
2. Compute **eigenvectors**
3. Compute **projections**





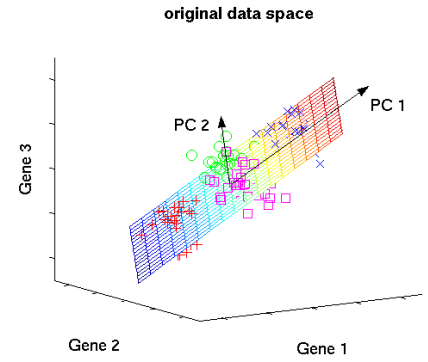
PCA algorithm

1. Compute **covariance matrix** of data
2. Compute **eigenvectors** of covariance matrix
   - Singular value decomposition (SVD) of (m x n) matrix Q is: $Q=U\sum V$
     - U is m x m unitary matrix; ∑ is m x n diagonal matrix of singular values; V is n x n unitary matrix
   - Columns of U are the eigenvectors (of QQ^T)
     - Columns of V are eigenvectors of Q^TQ
   - ∑ is diagonal matrix, with eigenvalues on the diagonal
   - **Eigenvectors** represent the *directions* of greatest variation of the data
   - **Eigenvalues** tell you how much variation is captured by the corresponding eigenvector
3. Compute **projections**



PCA algorithm (the right way)

1. Compute **covariance matrix** of data
2. Compute **eigenvectors** of covariance matrix
   - Singular value decomposition (SVD) of (m x n) matrix Q is: $Q=U\sum V$
   - Columns of U are the eigenvectors (of QQ^T)
3. Compute **projections**
   - Denote by U_k the matrix comprising the first *k* columns of U
   - The projection of X is $X U_k$ (we call this "Z")
     - X is (nXm), U_k is (mXk), projection is (nXk)
   - In other words, multiply the original data by the matrix of eigenvectors, this maps it to *k*-space



PCA algorithm (the right way)

- Summary:
  1. Compute **covariance matrix** of data
  2. Compute **eigenvectors**
  3. Compute **projections**

Outline

- Dimensionality Reduction
  - Motivation & Intuition
  - **Principal Component Analysis**
    - Intuition
    - Algorithm
    - **Implementation**
  - Example: Eigenfaces
  - Other approaches to dimensionality reduction

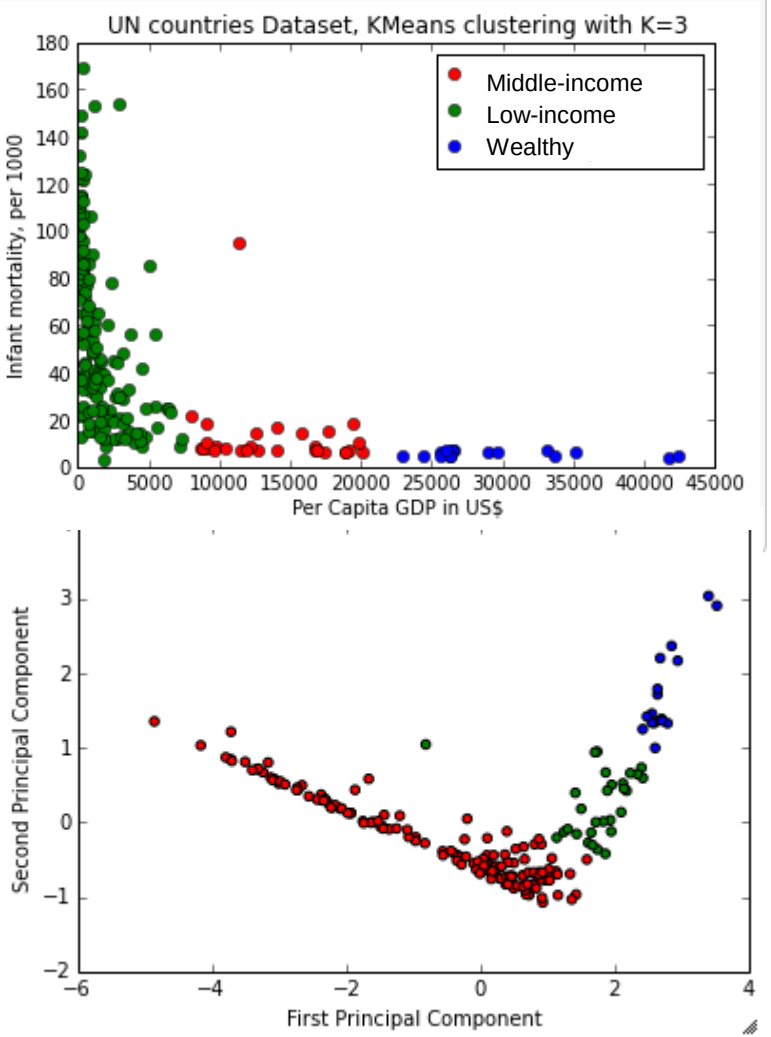
PCA Implementation & Use

- Before starting: Standardize your features!
  - Replace each x_{im} with $\frac{x_{im}-\mu_m}{s_m}$, where $\mu_m = \frac{1}{k}\sum_{i=1}^{k} x_{im}$

PCA in Python

```
from sklearn.decomposition import PCA
estimator = PCA(n_components=2)
X_pca = estimator.fit_transform(normalize(X))

colors = ['red', 'blue','green']
for i in xrange(len(colors)):
    px = X_pca[:, 0][c == i]
    py = X_pca[:, 1][c == i]
    plt.scatter(px, py, c=colors[i])
    plt.xlabel('First Principal Component')
    plt.ylabel('Second Principal Component')
```



How many components?

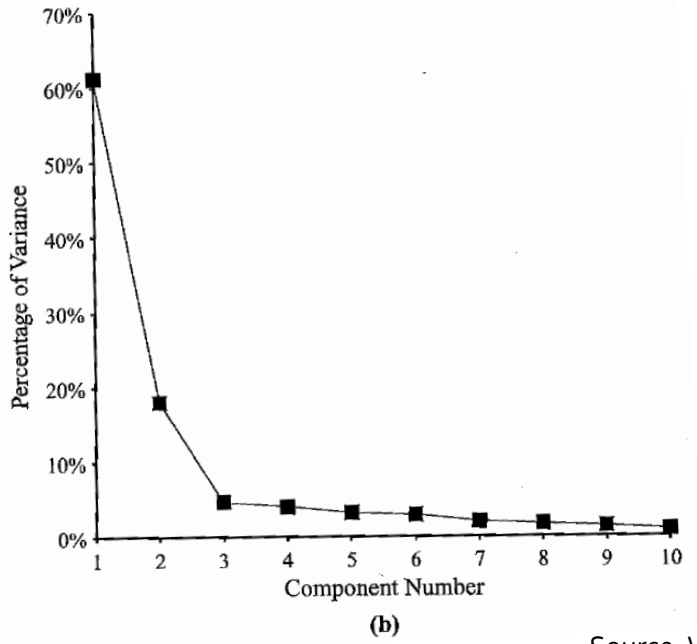
- How many components?
  - Some art and some science
  - Often there is a "natural" breaking point
- Depends on amount of variation you want to explain!
- "Retained variance"
  - Average squared projection error / total variation
  - Call $\hat{x}_i$ our reconstructed x_i (like predicted values from a regression)
    - Think of this as "inflation" of projected/reduced $z_i \Rightarrow U\ z_i$, where U are eigenvectors of the covariance matrix Q
    - Variance lost: $\frac{\frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x}_i|^2}{\frac{1}{n}\sum_{i=1}^{n}|x_i|}$
  - Common rule of thumb: retain 99% (or 95% or 90%) of variance

Example

- Example: Components of a 10-dimensional space
  - Each component is an axis, table indicates share of variance accounted for. 3 PC's account for roughly 84% of variance of original data

Axis	Variance	Cumulative
1	61.2%	61.2%
2	18.0%	79.2%
3	4.7%	83.9%
4	4.0%	87.9%
5	3.2%	91.1%
6	2.9%	94.0%
7	2.0%	96.0%
8	1.7%	97.7%
9	1.4%	99.1%
10	0.9%	100.0%

(a)



(b)

Source: Witten et al (2011)

Interpreting PCA

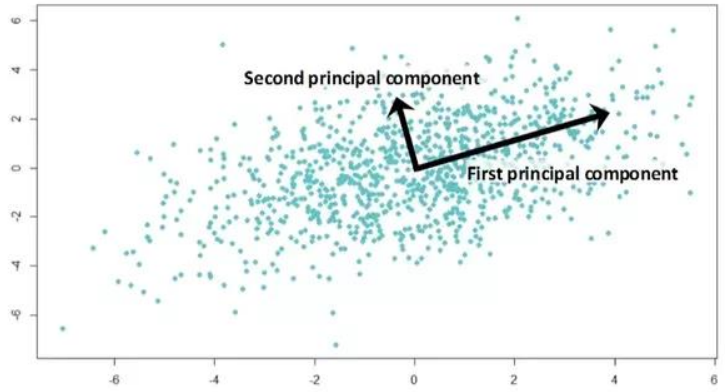
- Do the new features (the principal components) have any meaning?

$$PC_1 = U_{11}X_1 + \cdots + U_{1m}X_m$$

...

$$PC_k = U_{k1}X_1 + \cdots + U_{km}X_m$$

- Each component is a linear combination of the original features



Common applications of PCA

- Wealth indices
- Genetics
  - E.g. compress M=>K, then look at individual contributions to the principal components
- Outlier detection
- Data pre-processing and compression
- Facial recognition

Image Compression Example

Eigenvector Number	Cumulative proportion of variance
1	0.5160
2	0.6979
3	0.7931
4	0.8794
5	0.9130
6	0.9378
7	0.9494
8	0.9596
9	0.9678
10	0.9732

Table 2: Cumulative variance accounted for by PCs

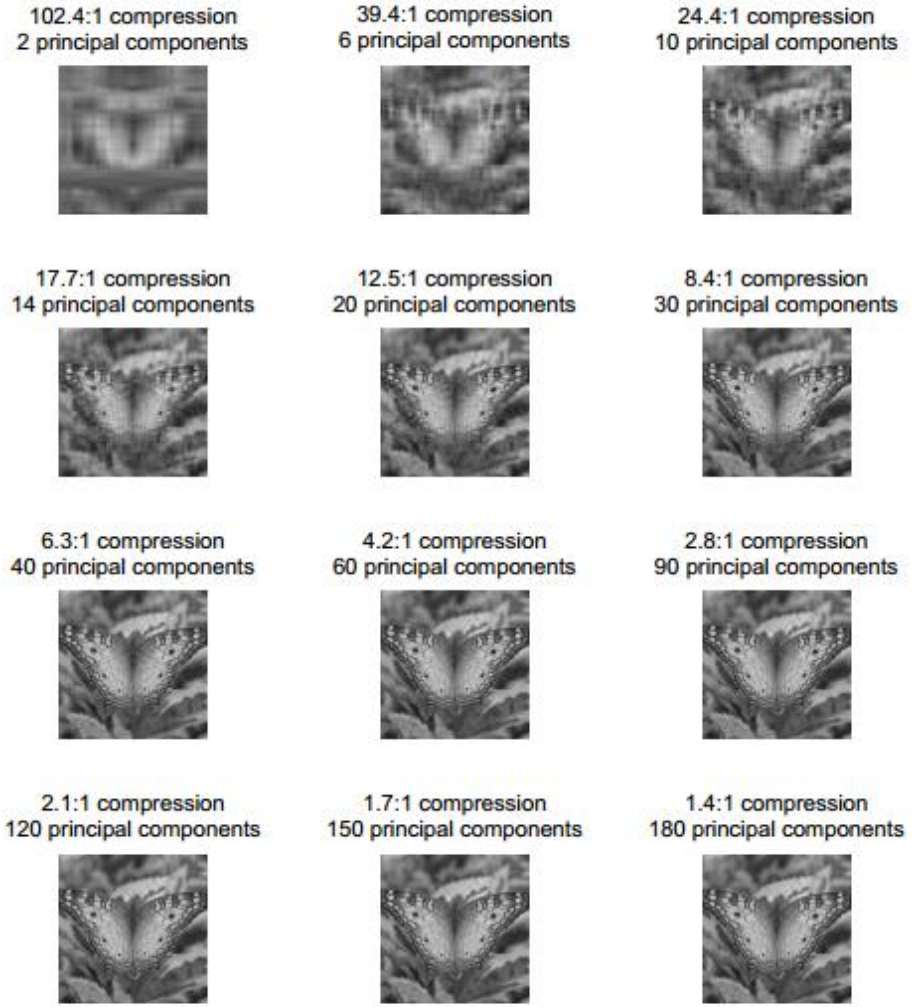


Figure 10: The visual effect of retaining principal components

Another important point

- One of most common uses of PCA is to increase efficiency of other learning algorithms, e.g. PCA then regression
- Important: Best practice is to train PCA algorithm on training data, not on validation/testing data!
  - The transformation matrix (U) is *learned* on the data, and will be different depending on the data used to train it

Outline

- Dimensionality Reduction
  - Motivation & Intuition
  - Principal Component Analysis
  - **Example: Eigenfaces**
  - Other methods for dimensionality reduction

PCA Example: Eigenfaces

- Turk and Pentland (1991), "Eigenfaces for Facial Recognition", *Journal of Cognitive Neuroscience* 3 (1)
- Goal: 2-D Face recognition
  - Efficient, simple, and accurate face recognition in a constrained environment

PCA Example 2: Eigenfaces

- Initialization (training):
  - Use initial set of facial images as training data
    - 256 pixel X 256 pixel array (65,536 pixels; 8-bit intensity)
  - Calculate PC's (eigenvectors, aka "eigenfaces"), retain components with largest eigenvalues
  - Project original data onto "face space"
- Recognition:
  - Project unknown image onto "face space"
  - Classify projection as known face, unknown face, or not face

Eigenfaces: examples

- Original Faces and the "average face"

Eigenfaces

- Eigenfaces and projections into "face space"
  - Each eigenvector/eigenface is a new dimension of the face space (65536 x 1)
  - The eigenvectors are used to transform a new face image into its eigenface components (in this case, a 7-dimensional space)
  - In other words: a "real" face is a linear combination of those 7 eigenfaces

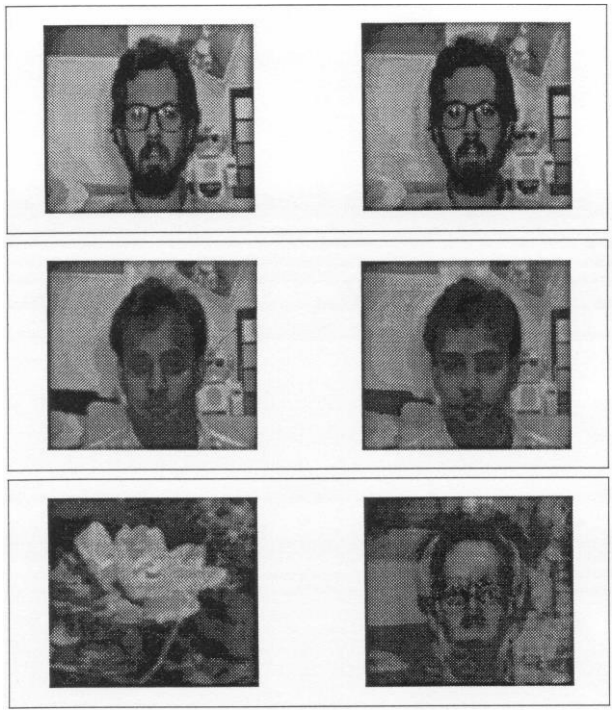


Figure 4. Three images and their projections onto the face space defined by the eigenfaces of Figure 2. The relative measures of distance from face space are **(a)** 29.8, **(b)** 58.5, **(c)** 5217.4. Images **(a)** and **(b)** are in the original training set.

Eigenfaces

- Recognition: Nearest neighbors!
  - Known images projected onto face space
    - Compute "average eigenface" for each person
  - Unknown images projected onto face space
    - Choose "average eigenface" that minimizes distance (in face space!) to projection
- Possible use cases?
  - Fast and simple: Real-time recognition
  - Facial reconstruction

Outline

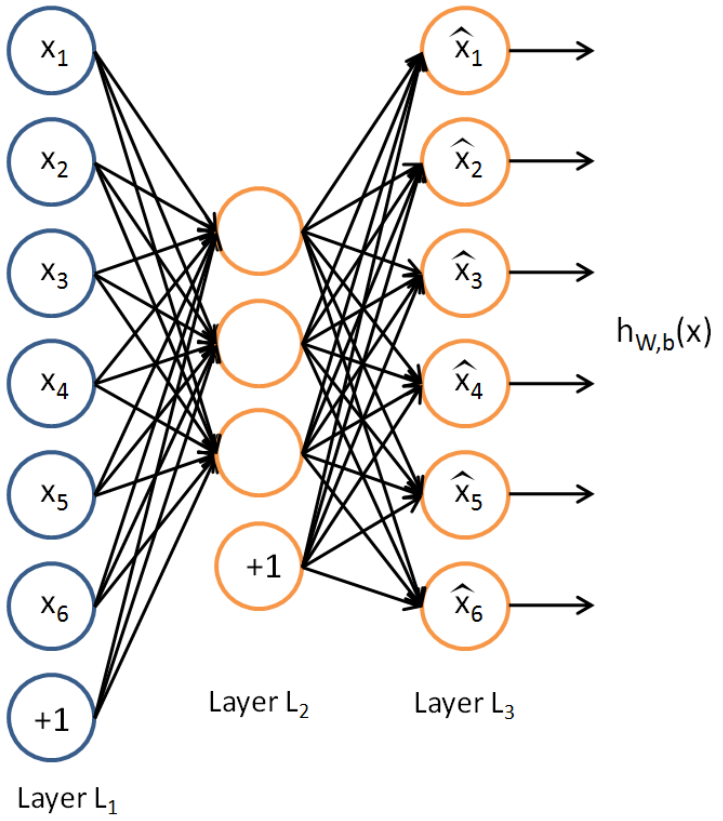
- Dimensionality Reduction
  - Motivation
  - Intuition
  - Principal Component Analysis
  - Example: Eigenfaces
  - **Other approaches to dimensionality reduction**

Other methods for dimensionality reduction

- PCA: Popular, fast, linear
- What else is out there?
  - Autoencoders
  - Kernel methods
  - Multi-dimensional scaling
  - Isomap
  - Etc.
    - Non-Negative Matrix Factorization
    - t-Distributed Stochastic Neighbor Embedding (t-SNE)
    - Uniform Manifold Approximation and Projection (Umap)

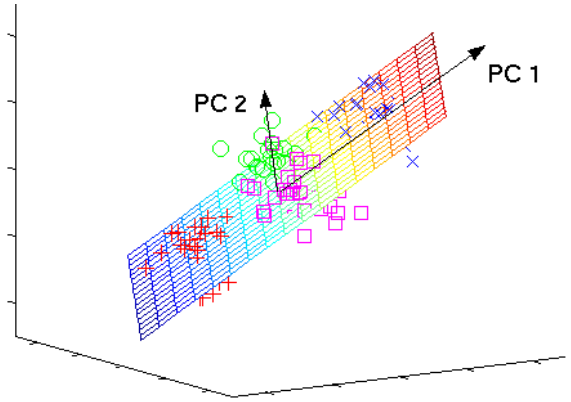
Autoencoders

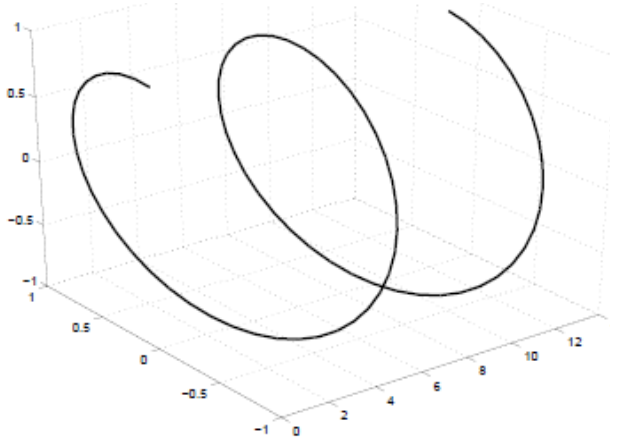
- Auto-encoders map inputs to inputs (i.e., it approximates the identity function): $\hat{y}_i(x_i) = x_i$
  - (See deep learning lecture notes)



Non-linear methods

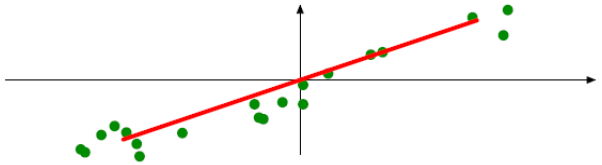
- PCA and MDS are linear
  - Work well when high-D data sit on a lower-D hyperplane
  - Not as effective if data sit on a curved manifold:



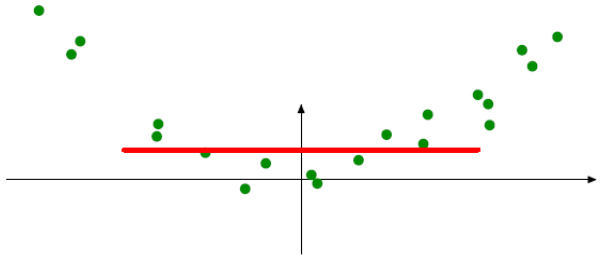


Kernel Methods

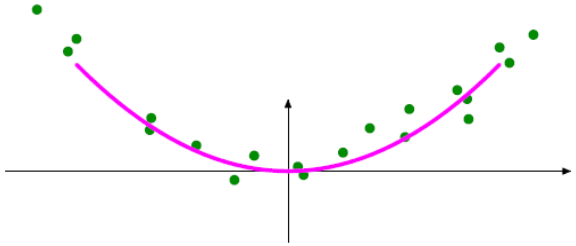
- Simple solution: Kernel
  - Apply kernel transformation to original data, then run PCA on resultant dataset



PCA is effective



PCA is ineffective

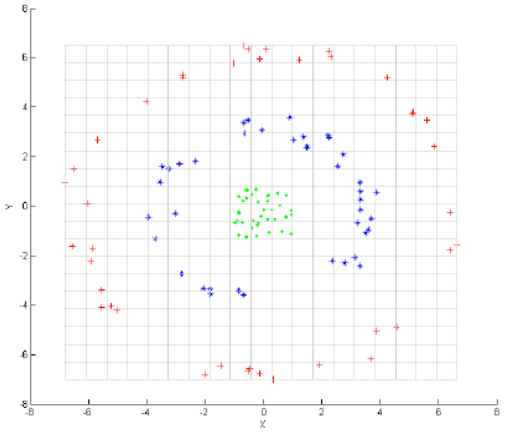


Desired solution

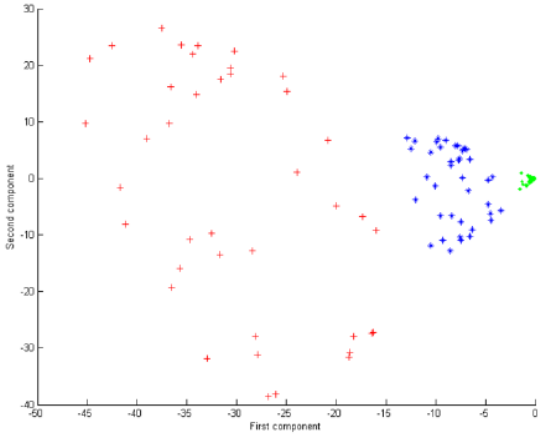
P. Liang

Kernel Methods

- Kernel PCA
  - PCA finds linear projections, but sometimes projection space is nonlinear
  - Kernel PCA applies kernels to allow for differently shaped subspaces



Original point set



After kernel-PCA with quadratic kernel

J. Canny

Multi-Dimensional Scaling

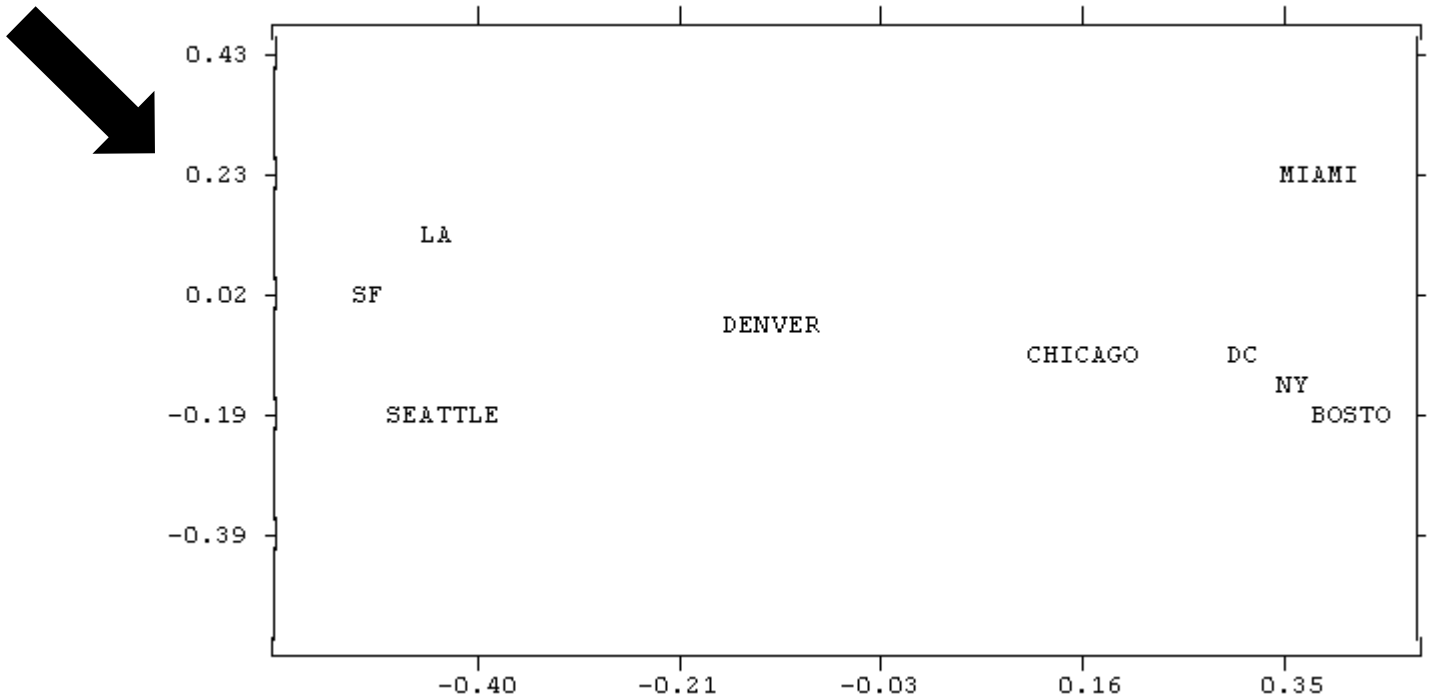
- Multi-Dimensional Scaling (also linear)
  - Idea: preserve inter-point distances, rather than preserving total variance
  - Formally: call d_{ij} the distance between two points in $\mathbb{R}^m$ and denote by δ_{ij} the distance in $\mathbb{R}^k$, goal is to minimize cost:

$$\sum_{i,j}\left(\frac{d_{ij}-\delta_{ij}}{\delta_{ij}}\right)^2$$

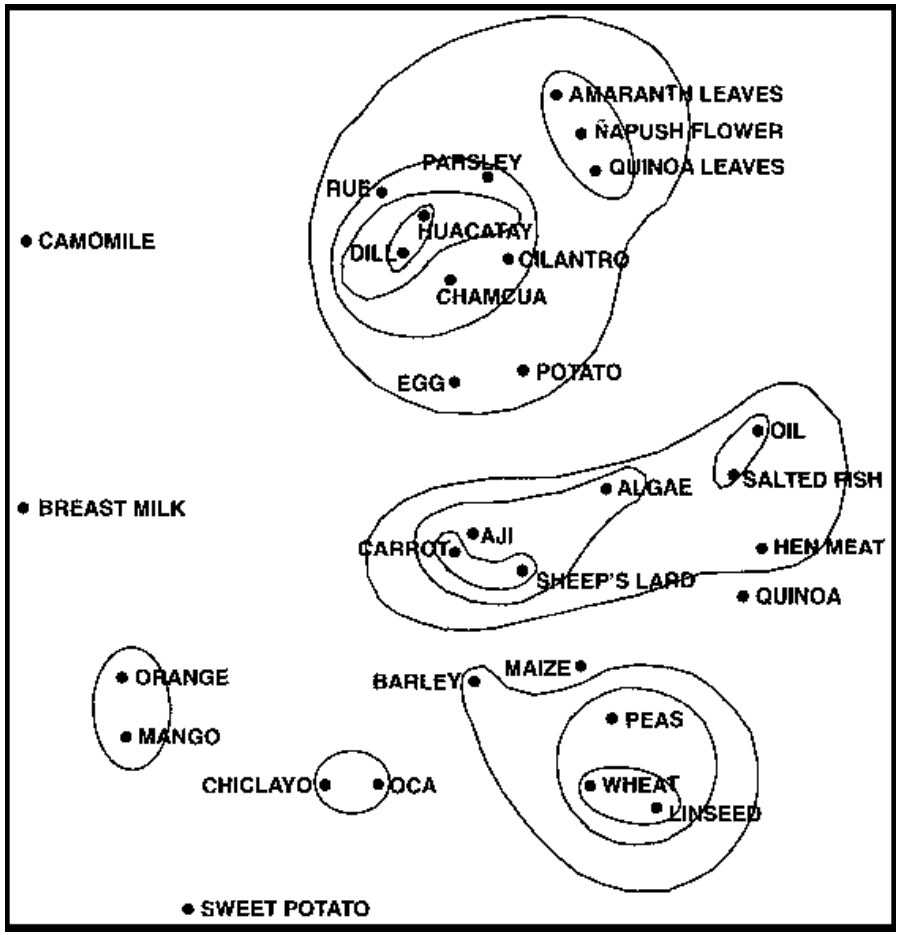
  - Useful in situations where similarity and distance is paramount, or when original coordinates of data are unknown
    - e.g. market research where subjects specify similarity between objects.
  - Really good for representing/visualizing data in 2D

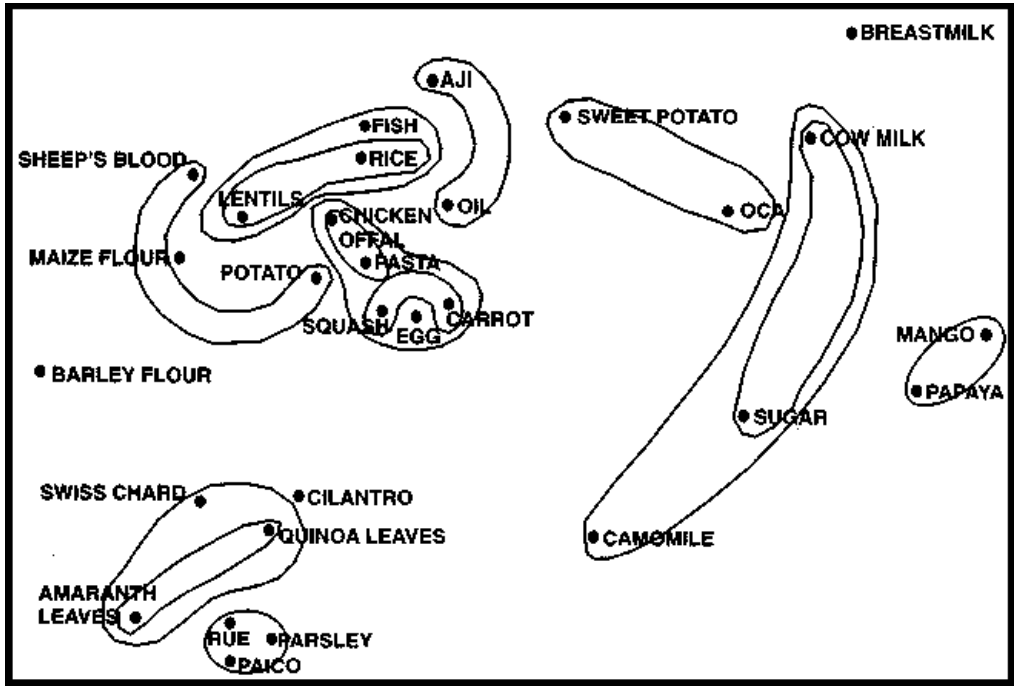
Multi-Dimensional Scaling

		1 BOST	2 NY	3 DC	4 MIAM	5 CHIC	6 SEAT	7 SF	8 LA	9 DENV
1	BOSTON	0	206	429	1504	963	2976	3095	2979	1949
2	NY	206	0	233	1308	802	2815	2934	2786	1771
3	DC	429	233	0	1075	671	2684	2799	2631	1616
4	MIAMI	1504	1308	1075	0	1329	3273	3053	2687	2037
5	CHICAGO	963	802	671	1329	0	2013	2142	2054	996
6	SEATTLE	2976	2815	2684	3273	2013	0	808	1131	1307
7	SF	3095	2934	2799	3053	2142	808	0	379	1235
8	LA	2979	2786	2631	2687	2054	1131	379	0	1059
9	DENVER	1949	1771	1616	2037	996	1307	1235	1059	0



Multidimensional Scaling



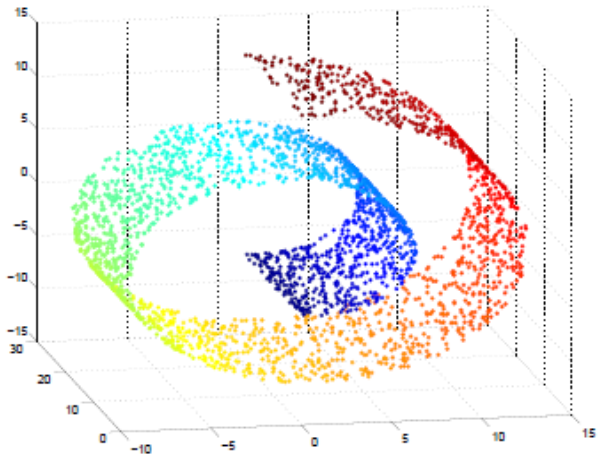


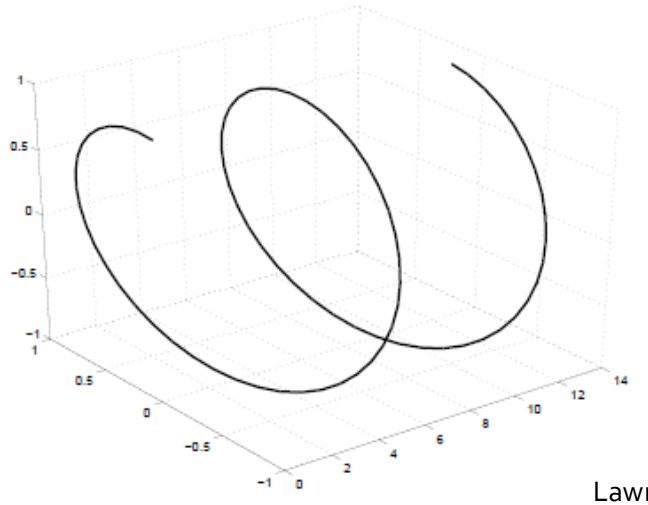
Non-linear methods

- Other methods
  - Fisher Discriminant Analysis
  - Also locally-linear embedding, Laplacian eigenmaps, self-organizing maps, manifold alignment, curvilinear component analysis, etc.
  - **Isomap**

Isomap

- Isometric Feature Mapping
  - Example manifold: the "Swiss Roll"
  - PCA finds hyperplane projections
  - Isomap projects original data onto a non-linear, high-dimensional manifold.

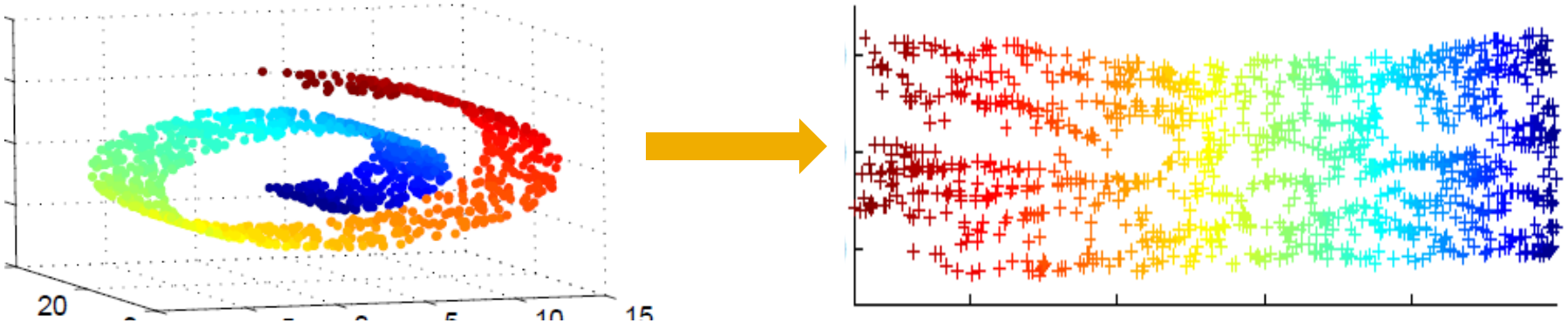




Lawrence Cayton

Isomap

- Isomap: "MDS with geodesic distance"
  1. Construct neighborhood graph **G**, only add edges if (Euclidean) distance < ε
  2. Compute distance matrix using geodesic (shortest path) distance in **G**, (using e.g. Floyd's or Djikstra's algorithm)
  3. Use MDS on **G** to find points in low-dimensional Euclidean space whose interpoint distances match the distances found in step 1



Etc

- Non-Negative Matrix Factorization
  - Similar to PCA, but data and components are non-negative
    - Useful for modelling non-negative data, e.g. images
- t-Distributed Stochastic Neighbor Embedding (t-SNE)
  - Very fast, non-linear method, typically used for 2D visualization
- Uniform Manifold Approximation and Projection (Umap)
  - Fast(er) and general non-linear dimensionality reduction
  - Models manifolds with "fuzzy" topological structure
  - Attempts to preserve global structure of data
    - Cool demo: https://pair-code.github.io/understanding-umap/