CS 277, Data Mining

Dimension Reduction Methods

Padhraic Smyth
Department of Computer Science
Bren School of Information and Computer Sciences
University of California, Irvine

Today's lecture

- Dimension reduction methods
  - Motivation
  - Variable selection methods
  - Linear projection techniques
  - Non-linear embedding methods

Dimension Reduction methods

- Dimension reduction
  - From d-dimensional **x** to k-dimensional **x**' , $k < d$

- Techniques
  - Variable selection:
    - e.g., for predictive modeling: use an algorithm to find individual variables in x that are relevant to the problem and discard the rest
    - e.g,. Use domain knowledge to discard irrelevant variables
  - Linear projections
    - Linearly project data to a lower-dimensional space
    - e.g., principal components
  - Non-linear embedding
    - Use a non-linear mapping to "embed" data in a lower-dimensional space
    - e.g., multidimensional scaling

Dimension Reduction: why is it useful?

- In general, incurs loss of information about **x**
  - so why do this?

- If dimensionality p is very large (e.g., 1000's), representing the data in a lower-dimensional space may make learning more reliable,
  - e.g., clustering example
    - 100 dimensional data
    - but cluster structure is only present in 2 of the dimensions, the others are just noise
    - if other 98 dimensions are just noise (relative to cluster structure), then clusters will be much easier to discover if we just focus on the 2d space

- Dimension reduction can also provide interpretation/insight
  - e.g., 2d visualization purposes, e.g., very useful for scientific data analysis

- Caveat:
  - Consider 2-step approach of (1) dimension reduction (2) followed by learning, e.g., principal components followed by classification)
  - In theory this may be suboptimal

Dimension Reduction with Variable Selection

Stepwise/Greedy Approaches to Variable Selection

- Consider building a predictive model with k variables from d variables
  - We can evaluate the quality of any model by

    (a) fitting it to the training data, and

    (b) evaluating its error E (e.g., squared error) on a validation data set

- Forward-variable selection
  - Train a model with each variable on its own and compute E each time
  - Select the variable that gives the lowest error E (out of the d candidates)
  - Now evaluate (train, compute E) adding each of the d-1 other variables to the model
  - Select the pair with the lowest error E out of the d-1 candidates
  - Continue adding variables in this manner until E starts to increase

Effectively this is an "outside loop" over our training algorithm, looping over different subsets of variables…sometimes referred to as "wrapper" methods.

Stepwise/Greedy Approaches to Variable Selection

- Backward-variable selection
  - Same procedure but in reverse
  - Start with all d variables in the model and at each iteration consider removing a single variable at a time.
- Limitations of these stepwise/greedy approaches?
  - Greedy search is not necessarily optimal (the usual limitation of local search)
  - Computational: may require training the model $O(d^2)$ times
    - Could be very expensive for large d
  - Results may be dependent on the particular validation set being used
- Alternatives
  - Linear projection and non-linear embedding methods (upcoming slides)
  - Algorithms that simultaneously do variable selection and model training, e.g.,
    - Decision trees
    - Regression models with penalty functions that drive weights to 0

Dimension Reduction with Linear Projections

Basic Principles of Linear Projection

x = d-dimensional (d x 1) vector of data measurements

Let **a** = weight vector, also dimension d x 1

Assume $\mathbf{a}^T \mathbf{a} = 1$ (i.e., unit norm)

$\mathbf{a}^T \mathbf{x} = \Sigma\, a_j\, x_j$

= projection of **x** onto vector **a**,

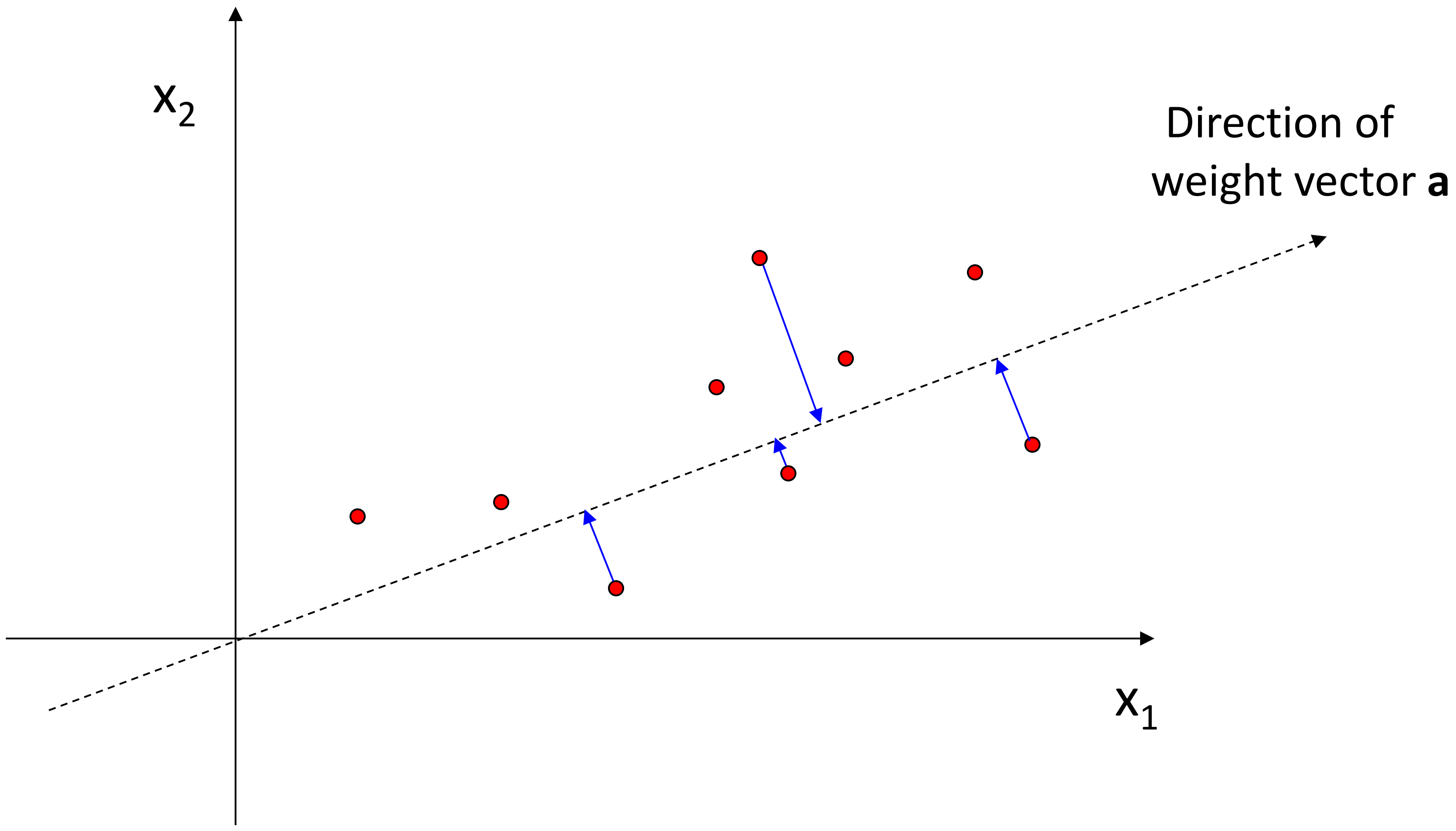
gives distance of projected **x** along **a**

e.g., $\mathbf{a}^T = [1\ 0]$ -> projection along 1st dimension

$\mathbf{a}^T = [0\ 1]$ -> projection along 2nd dimension

$\mathbf{a}^T = [0.71, 0.71]$ -> projection along diagonal

Example of projection from 2d to 1d



Projections to more than 1 dimension

Multidimensional projections:

e.g., if x is 4-dimensional and we want to project to 2 dimensions

$\mathbf{a_1}^T = [\ 0.71 \quad 0.71 \quad 0 \quad 0\ ]$

$\mathbf{a_2}^T = [\ 0 \quad 0 \quad 0.71 \quad 0.71\ ]$

$\mathbf{A}^T \mathbf{x}$ -> coordinates of $\mathbf{x}$ in 2dim space spanned by columns of $\mathbf{A}$

-> linear transformation from 4dim to 2dim space

where $\mathbf{A} = [\ \mathbf{a_1}\ \mathbf{a_2}\ ]$ with dimensions 4 x 2

Projections to more than 1 dimension

Multidimensional projections:

e.g., if x is 4-dimensional and we want to project to 2 dimensions

$\mathbf{a_1}^T$ = [0.71 0.71 0 0]

$\mathbf{a_2}^T$ = [0 0 0.71 0.71]

$\mathbf{A}^T \mathbf{x}$ -> coordinates of **x** in 2dim space spanned by columns of **A**

-> linear transformation from 4dim to 2dim space

where **A** = [$\mathbf{a_1}$ $\mathbf{a_2}$] with dimensions 4 x 2

More generally, to go from d dimensions to k dimensions, we can specify k linear projections, $\mathbf{a_1}$,…. $\mathbf{a_k}$, each of length d, and **A** is size d x k

Principal Components Analysis (PCA)

$\mathbf{X}$ = d times N data matrix: columns = d-dim data vectors

Let $\mathbf{a}$ = weight vector, also dimension d

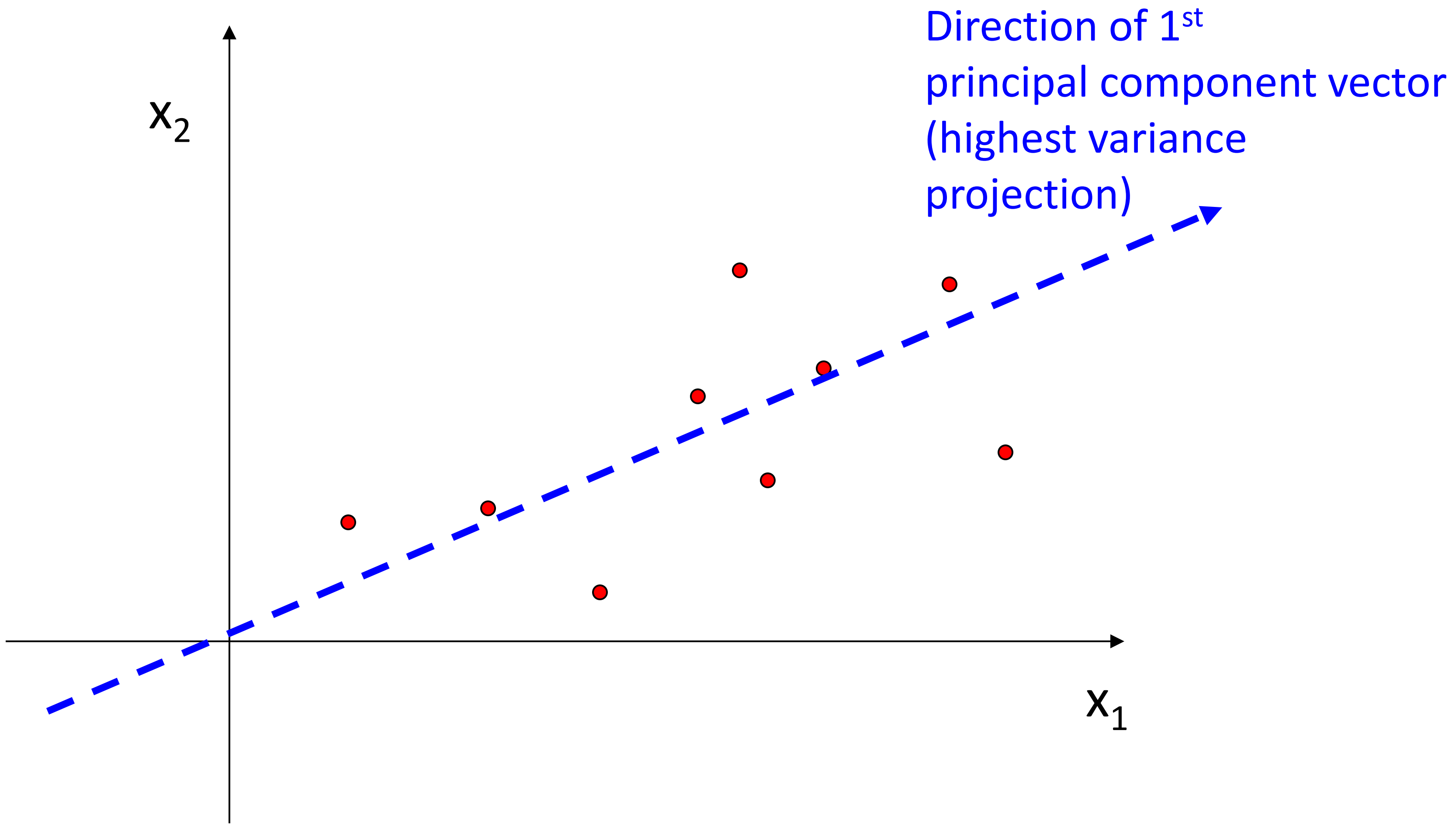
Assume $\mathbf{a}^T \mathbf{a} = 1$ (i.e., unit norm)

$\mathbf{a}^T \mathbf{X}$ = projection of each column $\mathbf{x}$ onto vector $\mathbf{a}$,
= vector of distances of projected $\mathbf{x}$ vectors along $\mathbf{a}$

PCA: find vector $\mathbf{a}$ such that var($\mathbf{a}^T \mathbf{X}$) is maximized
i.e., find linear projection with maximal variance

More generally:
$\mathbf{A}^T\mathbf{X}$ = k times N data matrix, with $\mathbf{x}$ vectors projected to k-dimensional space,
where size($\mathbf{A}$) = d x k
PCA: find k orthogonal columns of $\mathbf{A}$ such that variance in the
k-dimensional projected space is maximized, $k < d$

PCA Example


Direction of 1st principal component vector (highest variance projection)
x_2
x_1

Principal Components Analysis (PCA)

Given the first principal component a_1, define the 2nd principal component as the orthogonal direction a_2, that has the maximal variance

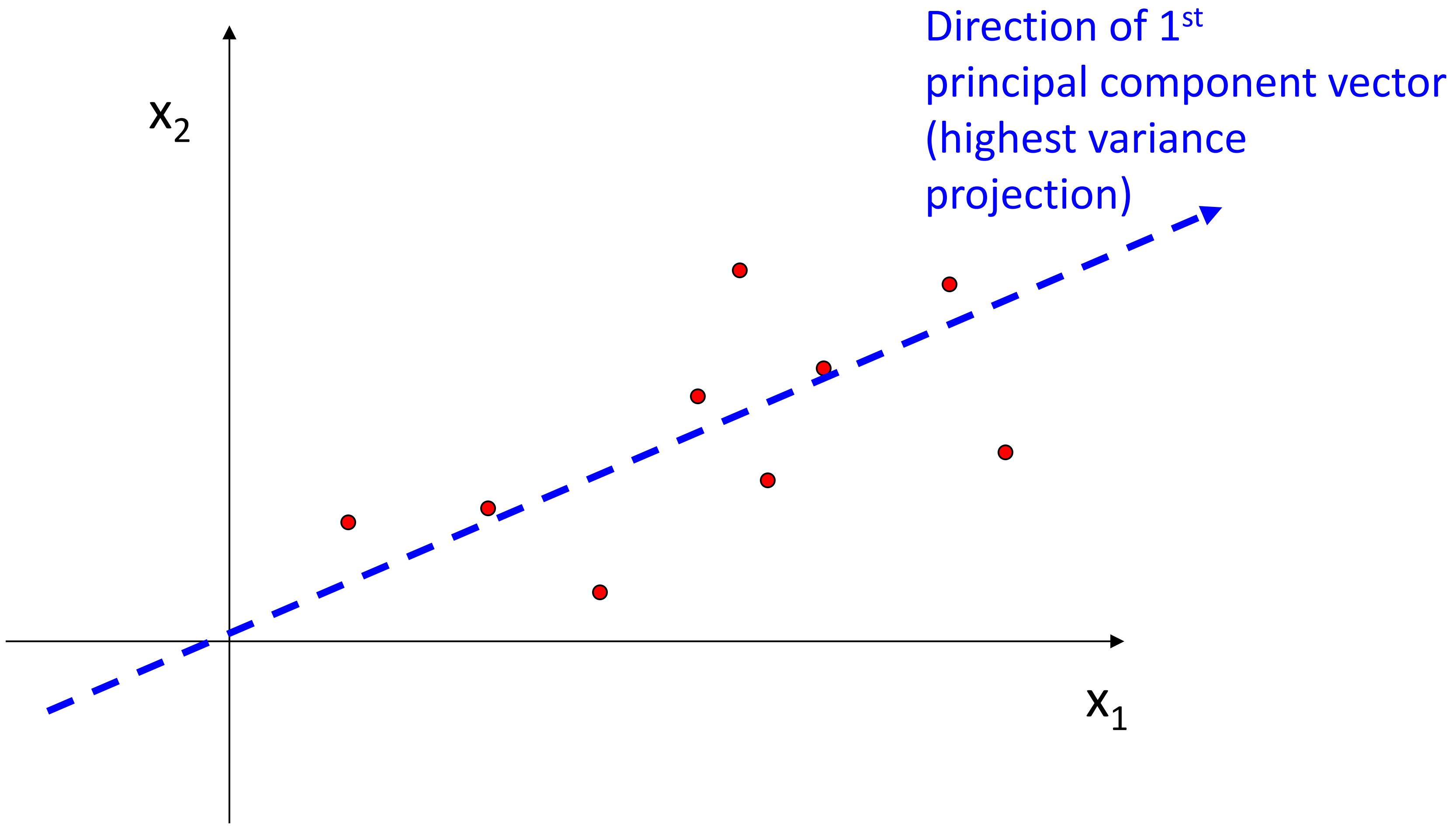
Continue in this fashion finding the first k components.

This yields a k-dimensional projection that has the property that it is the optimal k-dimensional projection in a squared-error sense

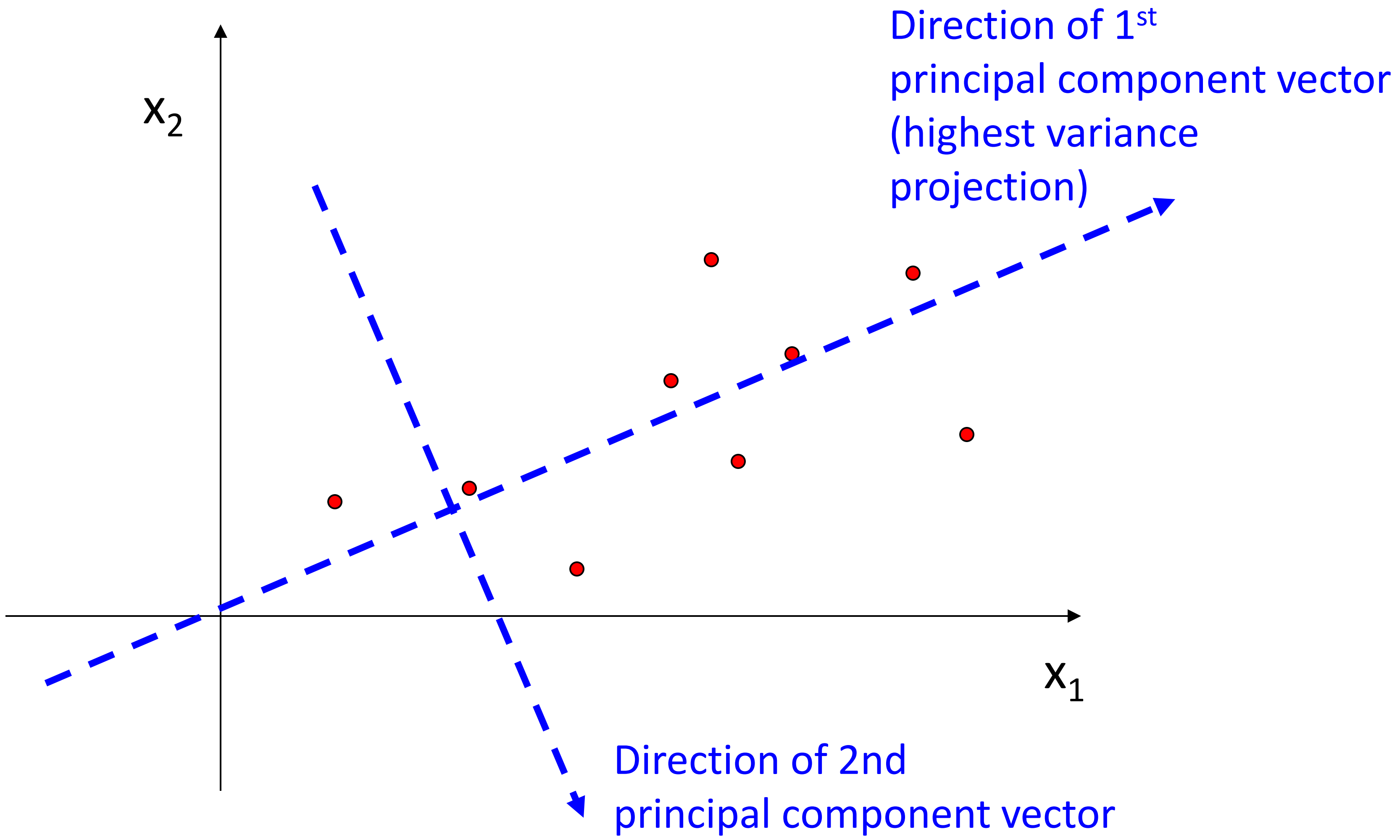
More generally:
$\mathbf{A}^T\mathbf{x} = \mathbf{x}'$, a vector of size k by 1,
representing the original d-dim **x** vector projected to a k-dimensional space, where size(**A**) = d by k

PCA: a k-dimensional projection where we find
k orthogonal columns of A such that variance in the k-dimensional projected space is maximized, $k < d$

PCA Example


Direction of 1st
principal component vector
(highest variance
projection)
x_2
x_1

PCA Example


Direction of 1st principal component vector (highest variance projection)
x_2
x_1
Direction of 2nd principal component vector

How do we compute the principal components?

Let C be the symmetric d x d empirical covariance matrix (from N x d data matrix) where entry(i,j) = cov(x_i, x_j) be the empirical covariance of (original) variables x_i and x_j

Basic result from linear algebra:

C has d eigenvectors $\mathbf{a}_1$,…. $\mathbf{a}_d$ each with a real-valued eigenvalue λ_1,…. λ_d > 0

Say (without loss of generality) that the eigenvalues are ordered by size, i.e.,

$\lambda_1 > \lambda_2 \ldots. > \lambda_d > 0$

How do we compute the principal components?

Let C be the symmetric d x d empirical covariance matrix (from N x d data matrix) where entry(i,j) = cov(x_i, x_j) be the empirical covariance of (original) variables x_i and x_j

Basic result from linear algebra:

C has d eigenvectors $\mathbf{a}_1$,…. $\mathbf{a}_d$ each with a real-valued eigenvalue λ_1,…. λ_d > 0

Say (without loss of generality) that the eigenvalues are ordered by size, i.e.,

$\lambda_1 > \lambda_2 \ldots. > \lambda_d > 0$

Result: the ith principal components of the N x d data matrix

= the ith eigenvectors $\mathbf{a}_i$

and the amount of variance accounted for by each component i is λ_i

So: to compute the principal components, we need to compute the eigenvectors of the empirical covariance matrix

Complexity of computing Principal Components

Step 1: given an N x d data matrix, compute the empirical covariance matrix C

-> For each entry $cov(x_i, x_j)$, sum over N elements -> O(N)

-> there are d(d+1)/2 such entries, -> $O(d^2)$ entries

-> thus, $O(N\, d^2)$ overall to compute C

Step 2: compute the eigenvalues and eigenvectors of d x d matrix C

-> in general scales as $O(d^3)$, same as matrix inversion

-> Overall complexity = $O(N\, d^2 + d^3)$

(a) if N >> d, $O(N\, d^2)$ will dominate (many more data points than dimensions)

(b) if d >> N, $O(d^3)$ will dominate (many more dimensions than data points)

Complexity of computing Principal Components

Step 1: given an N x d data matrix, compute the empirical covariance matrix C

-> For each entry cov(x_i, x_j) , sum over N elements -> O(N)

-> there are d(d+1)/2 such entries, -> O(d^2) entries

-> thus, O(N d^2) overall to compute C

Step 2: compute the eigenvalues and eigenvectors of d x d matrix C

-> in general scales as O(d^3), same as matrix inversion

-> Overall complexity = O(N d^2 + d^3)

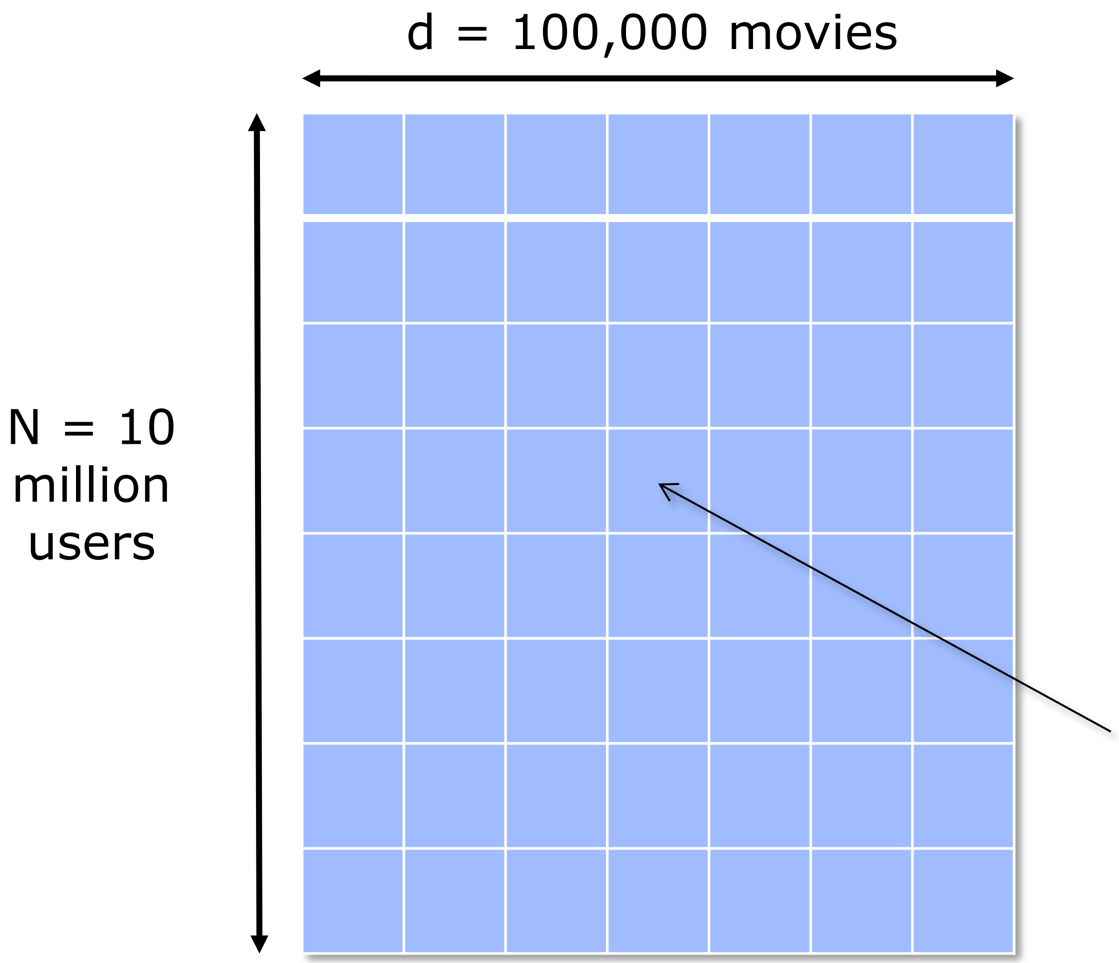
(a) if N >> d, O(N d^2) will dominate (many more data points than dimensions)

(b) if d >> N, O(d^3) will dominate (many more dimensions than data points)

Speed up strategies:

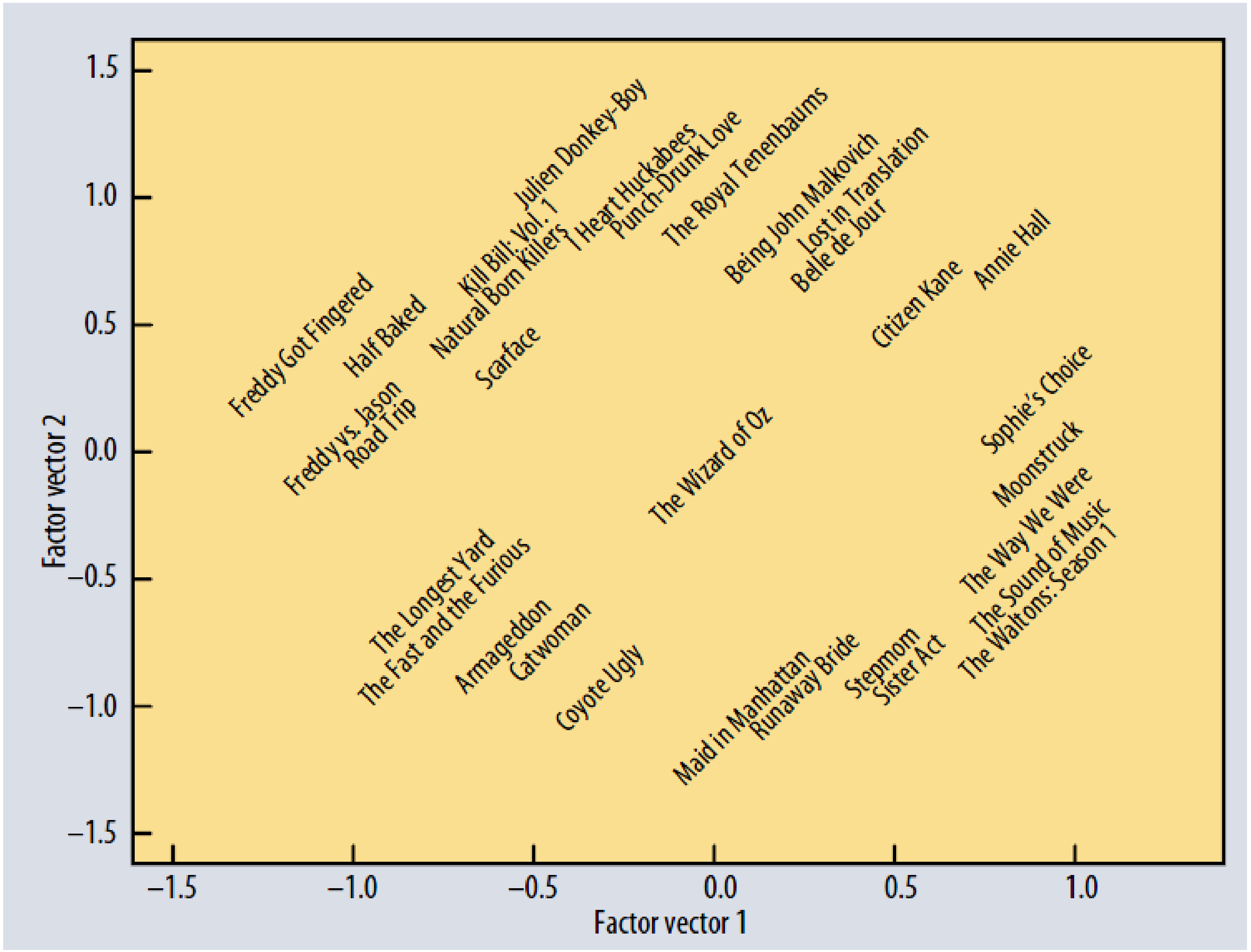
- for sparse data matrix, we can be much faster than O(d^3)
- only want the first k principal components? k << d, we can use this fact

Example: Customer Movie Ratings Data

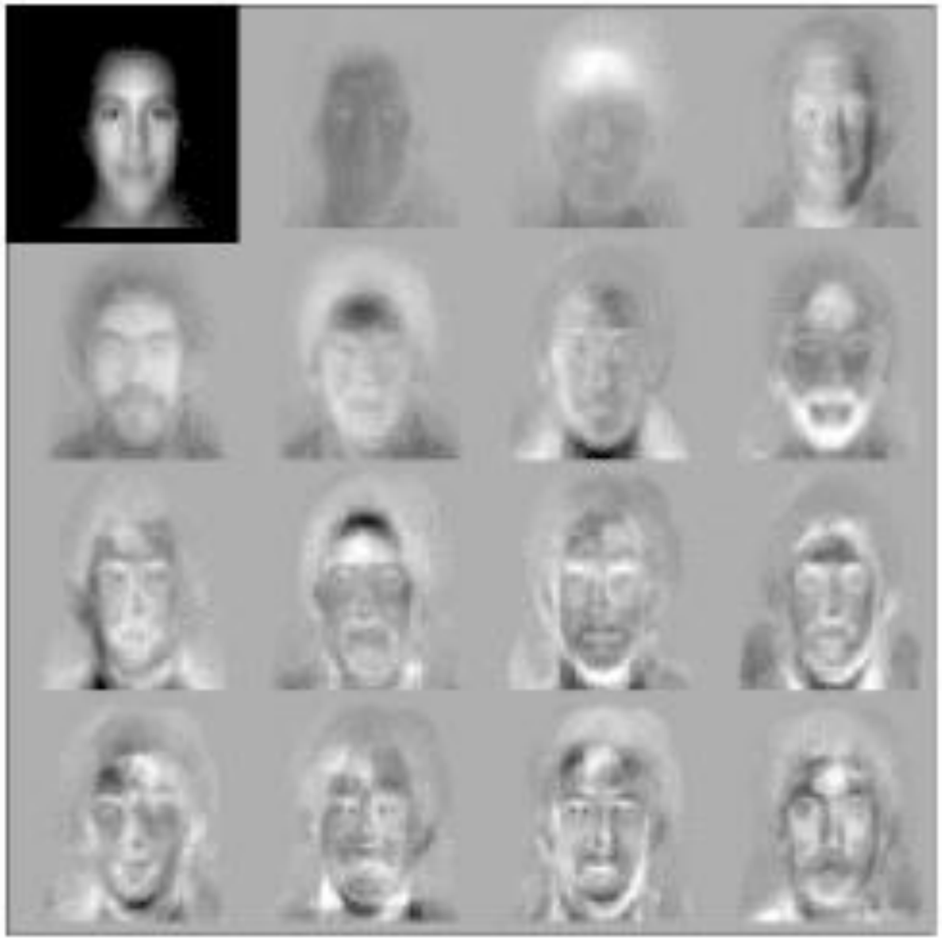


Each cell is either blank (customer did not rate this movie) or has a customer rating value

PCA applied to Netflix Movie Data

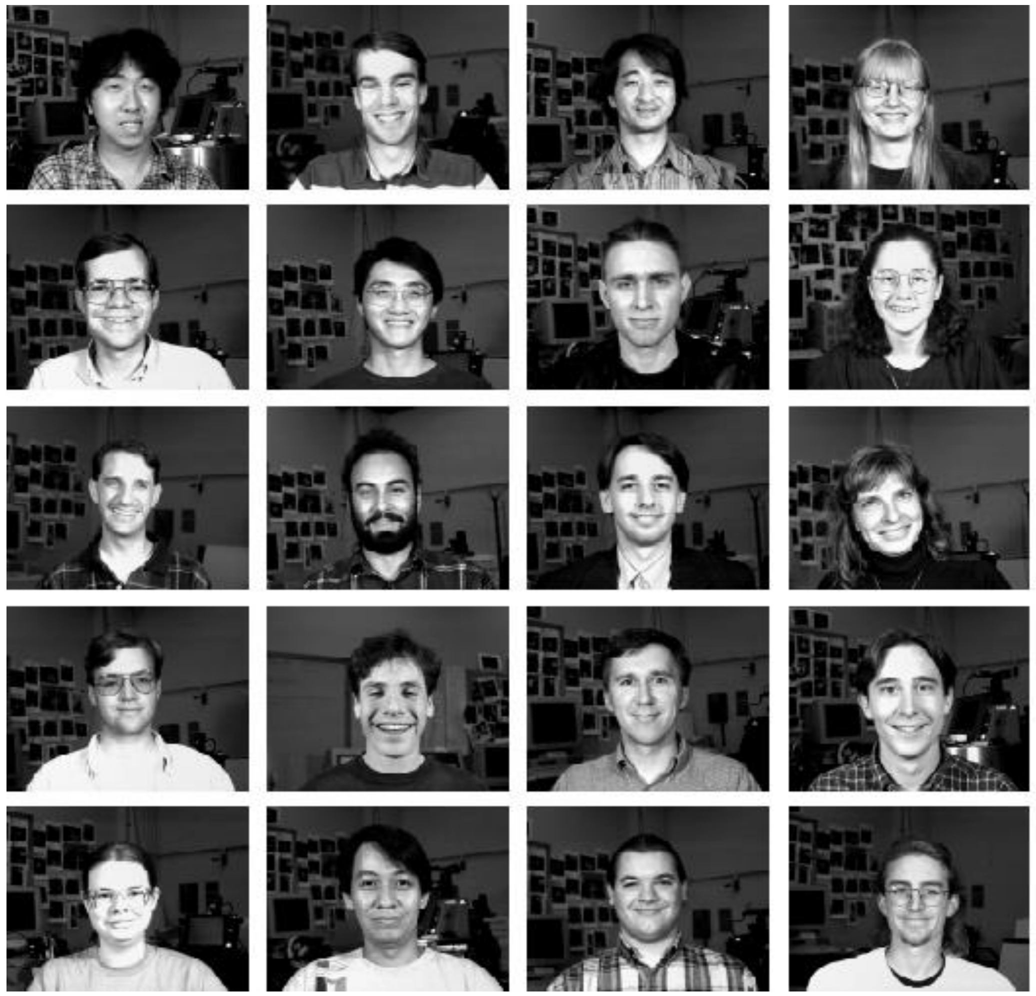


Basis images (eigenimages) of faces



Courtesy of Matthew Turk and Alex Pentland

20 face images

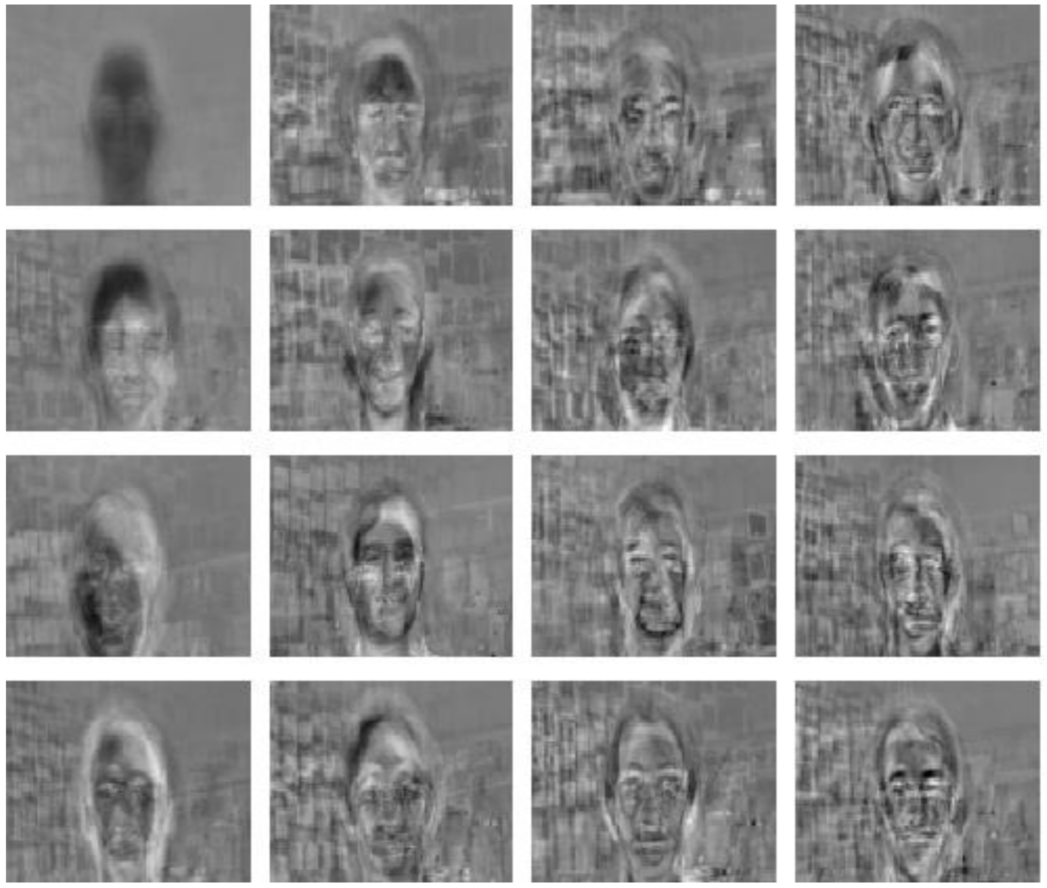


We can represent these images as an N x d matrix where

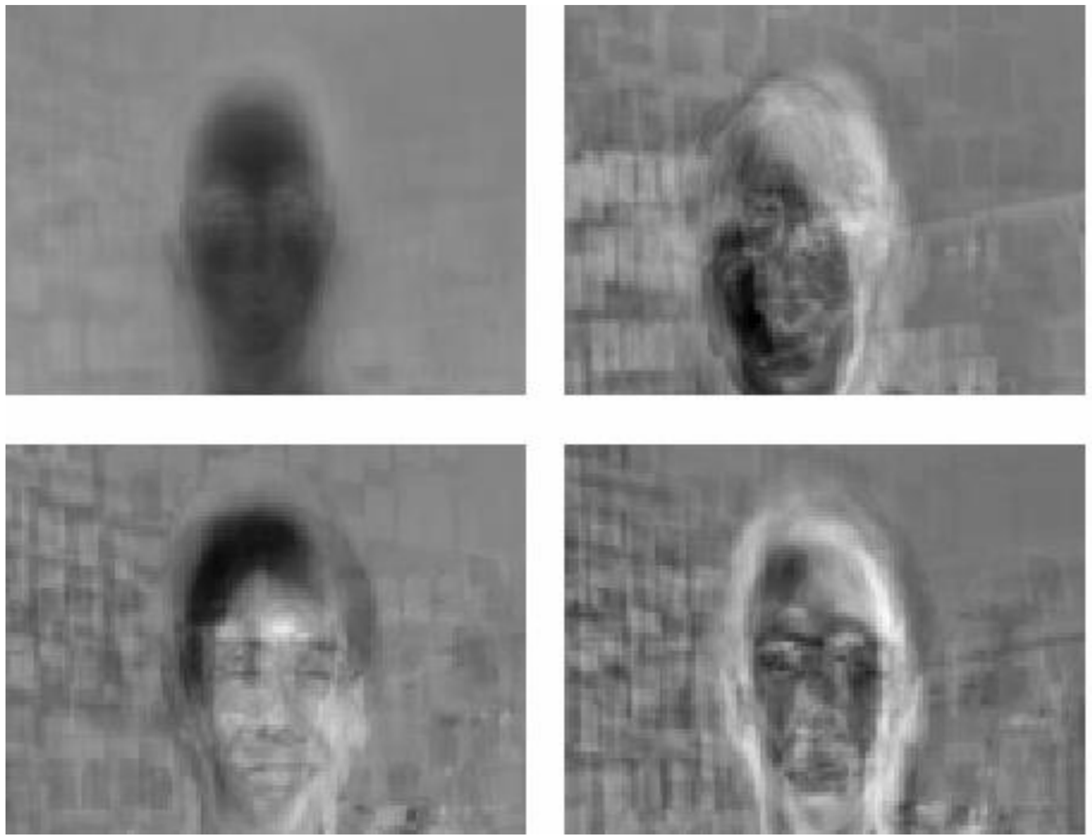
- each row is an individual image
- each column is a pixel value

(note: the spatial information is not being represented)

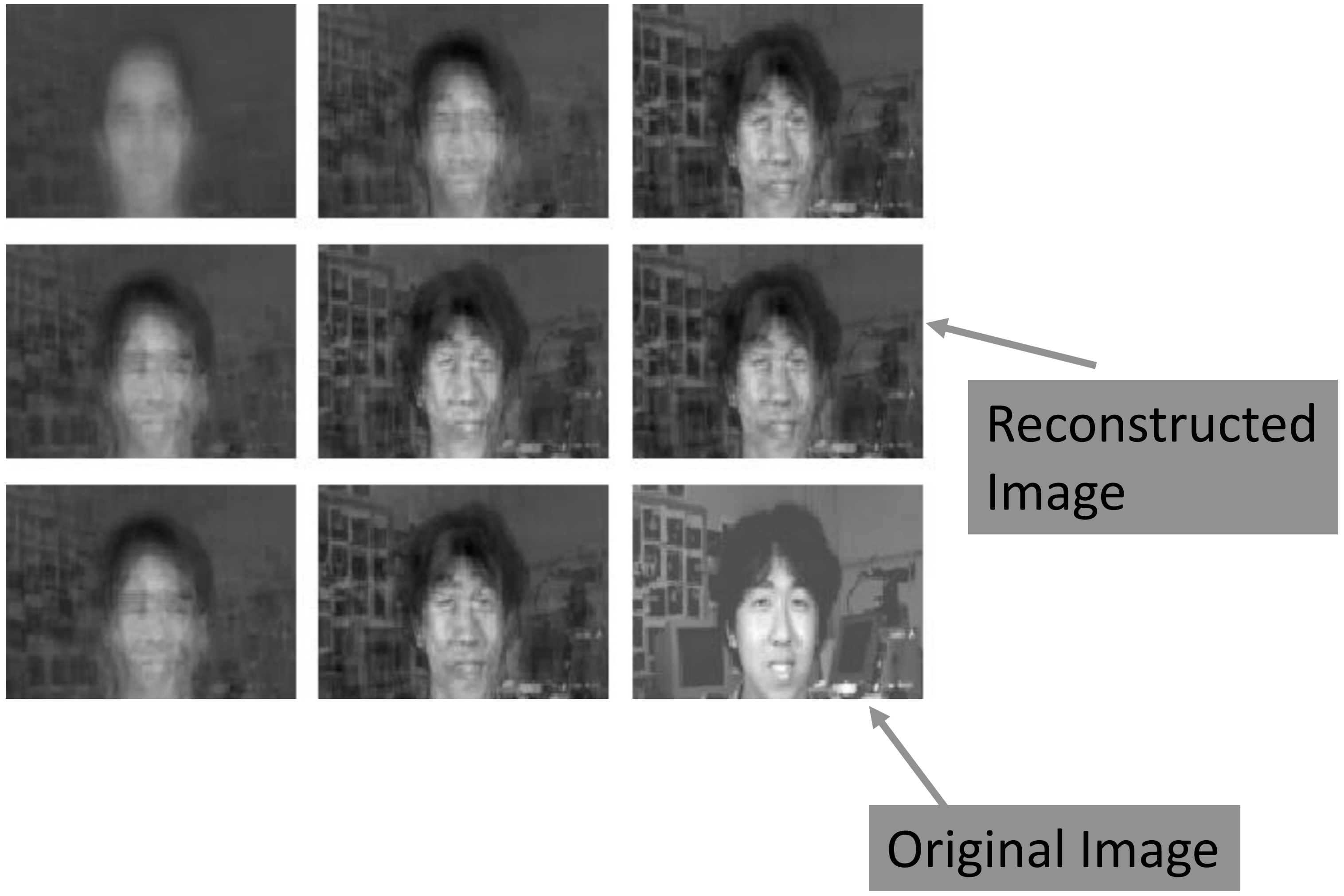
First 16 Eigenimages (eigenvectors plotted as images)



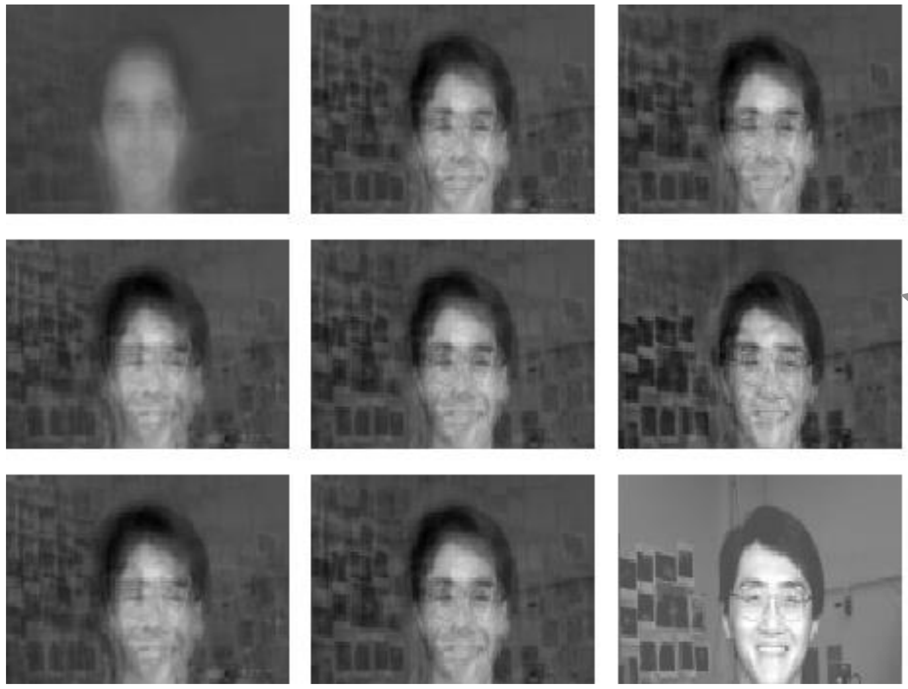
First 4 eigenimages



Reconstruction of First Image with 8 eigenimages



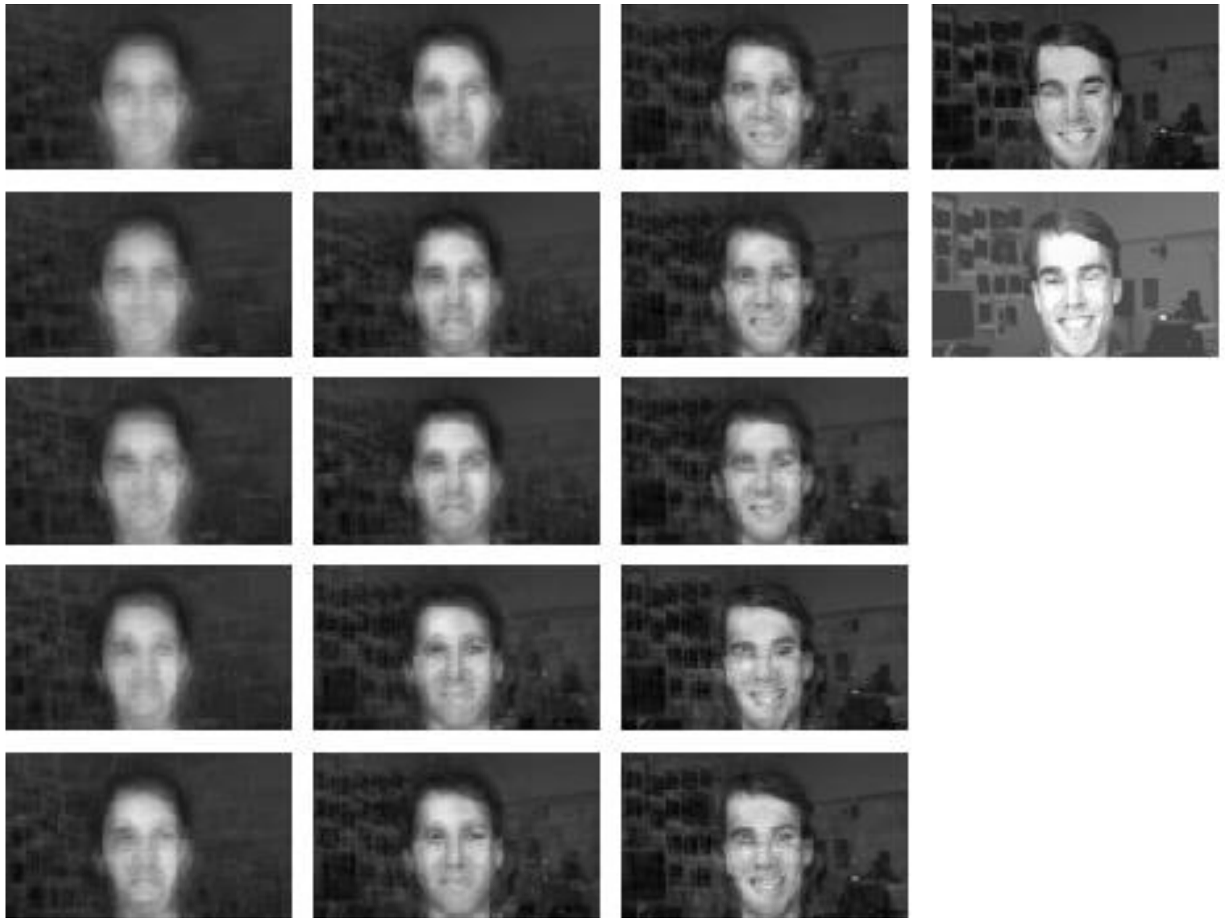
Reconstruction of another image with eigenimages



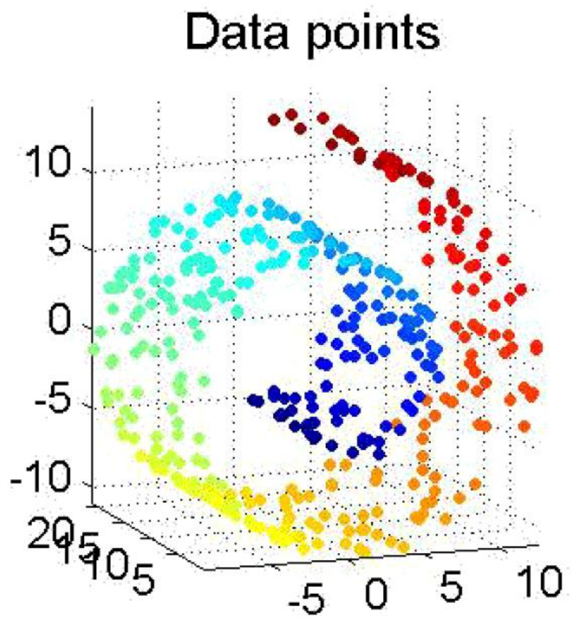
Reconstructed Image

Original Image

Reconstructing an Image with 16 eigenimages



Limitations of Local Projections



Any linear projection will do poorly….

… but it is clear that the data “live on” a lower-dimensional manifold

Dimension Reduction with Non-Linear Embeddings

Multidimensional Scaling (MDS)

- Say we have data on N objects in the form of an N x N matrix of dissimilarities
  - 0's on the diagonal
  - Symmetric
  - Could either be given data in this form, or we can create such a dissimilarity matrix from our data vectors

- **Examples**
  - Perceptual dissimilarity of N objects in cognitive science experiments
  - String-edit distance between N protein sequences

Multidimensional Scaling (MDS)

- Say we have data on N objects in the form of an N x N matrix of dissimilarities
  - 0's on the diagonal
  - Symmetric
  - Could either be given data in this form, or we can create such a dissimilarity matrix from our data vectors
- **Examples**
  - Perceptual dissimilarity of N objects in cognitive science experiments
  - String-edit distance between N protein sequences
- **Basic Idea of MDS**
  - Find k-dimensional coordinates for each of the N objects such that Euclidean distances in "embedded" space matches N x N matrix of dissimilarities as closely as possible
  - For k =2 (typical choice) we can plot our N points as a scatter plot

Multidimensional Scaling (MDS)

- Objective function for MDS is “stress” S:

$$S = \sum_{i,j} \big(\underset{\uparrow}{d(i,j)} - \underset{\nwarrow}{\delta(i,j)}\big)^2$$

Euclidean distance in new “embedded” k-dim space

Original dissimilarities

- N points embedded in k-dimensions -> N x k locations or parameters
  - To find the N x k locations?
    - Solve optimization problem -> minimize S function

- Often used for visualization, e.g., k=2, 3

Solving the MDS Optimization Problem

$$S = \sum_{i,j} (d(i,j) - \delta(i,j))^2$$

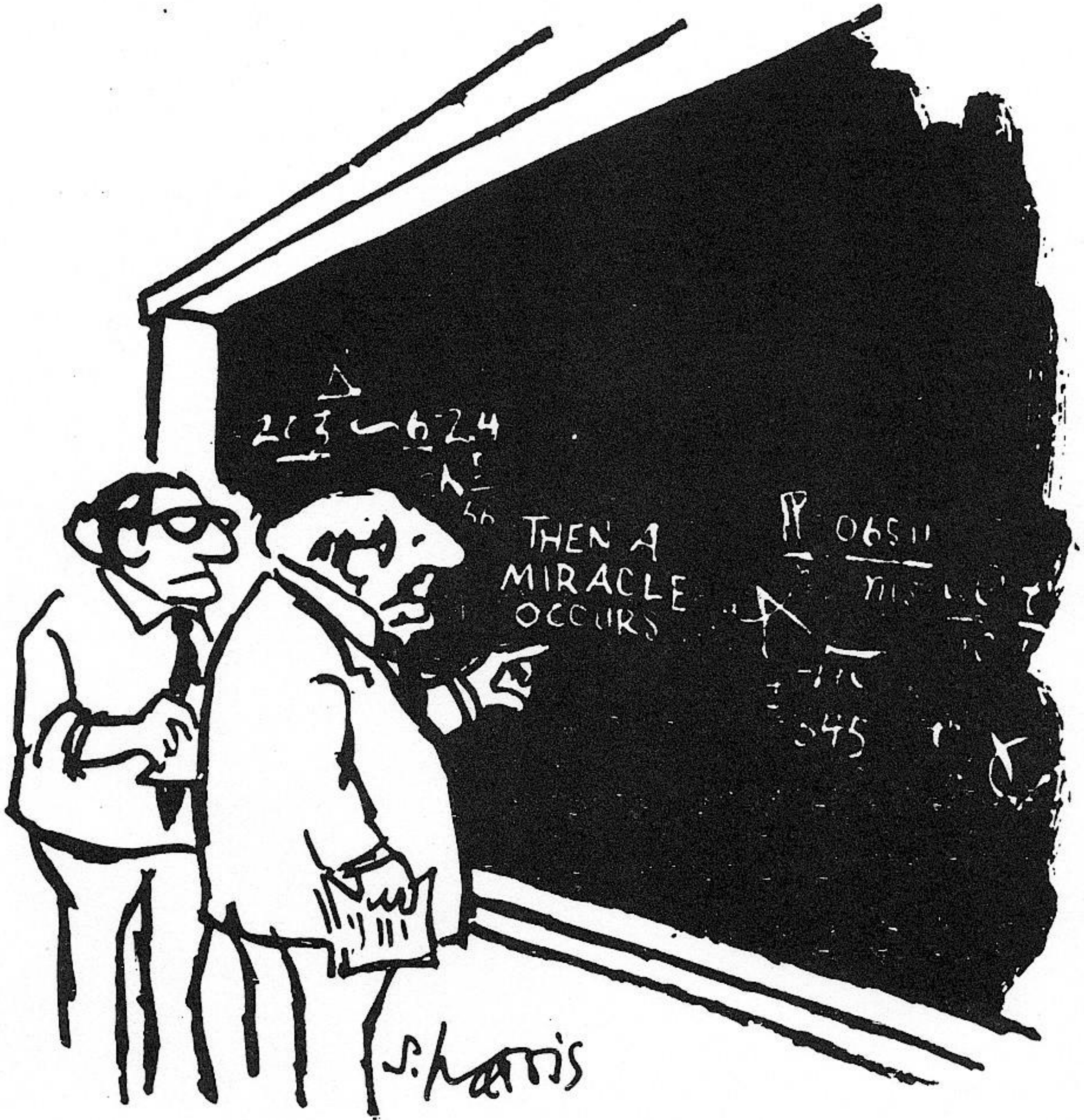
- Optimization problem:
  - S is a function of N times k parameters
    - Find the set of N k-dimensional positions that minimize S
    - Note that location and rotation are arbitrary

Solving the MDS Optimization Problem

$$S = \sum_{i,j} (d(i,j) - \delta(i,j))^2$$

- Optimization problem:
  - S is a function of N times k parameters
    - Find the set of N k-dimensional positions that minimize S
    - Note that location and rotation are arbitrary

- Gradient-based Optimization
  - Local iterative hill-descending, e.g., move each point to decrease S, repeat
  - Non-trivial optimization, can have local minima, etc
  - Initialization: either random or heuristically (e.g., by PCA)
  - Complexity is $O(N^2 k)$ per iteration
    - Evaluate gradient in k-dimensions based on $O(N^2)$ distances
    - iteration = move all points locally
  - Fast approximate algorithms exist
    - eg., Landmark MDS, de Silva and Tenenbaum (2003),
      - works with q x N distance matrix, q << N



"I think you should be more explicit here in step two."

MDS 2-dim representation of voting records in the US House of Representatives

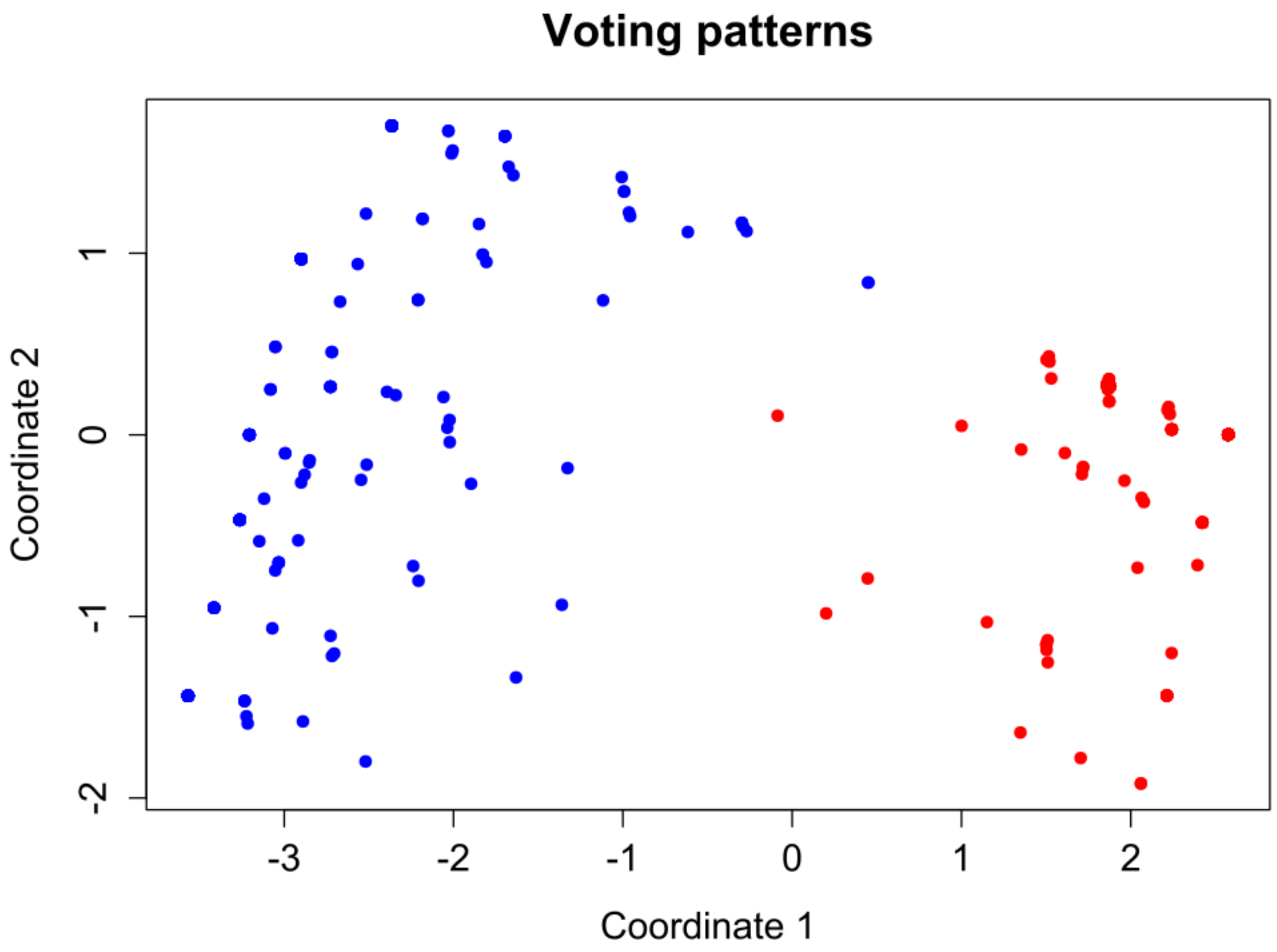
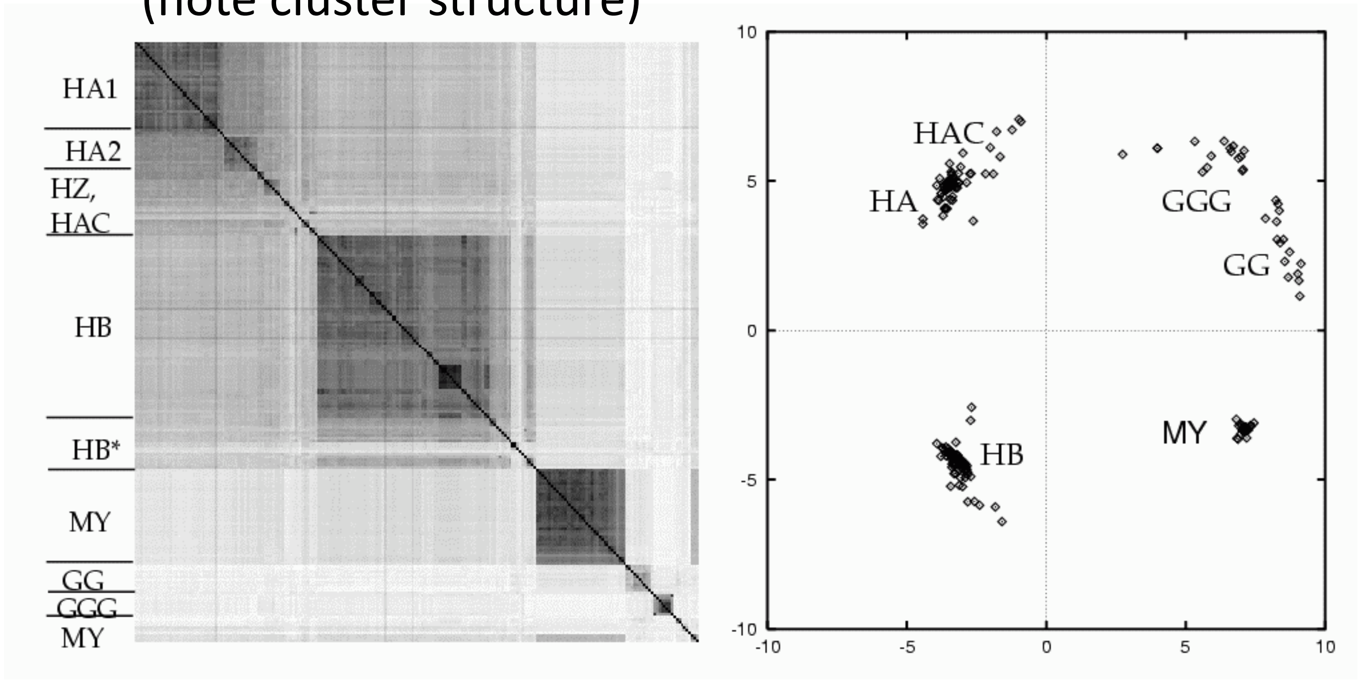


Figure from http://en.wikipedia.org/wiki/Multidimensional_scaling

MDS for protein sequences

Sequence similarity matrix (note cluster structure) → MDS embedding



226 protein sequences of the Globin family (from Klock & Buhmann 1997).

MDS from human judgements of emotion similarity

FIGURE 1
Multidimensional Scaling Solution for 28 Experiential Emotions

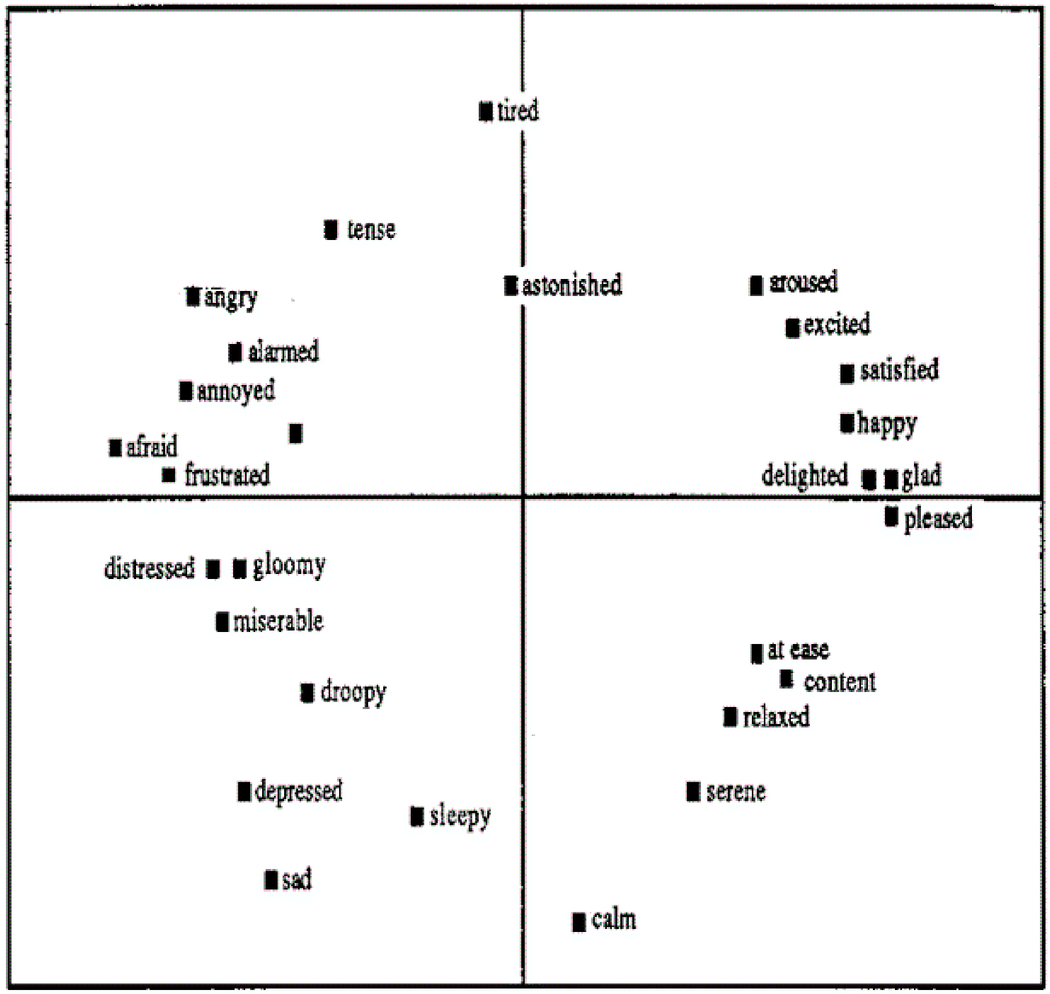


Figure from http://www.acrwebsite.org/

MDS of European Countries based on genetic similarity

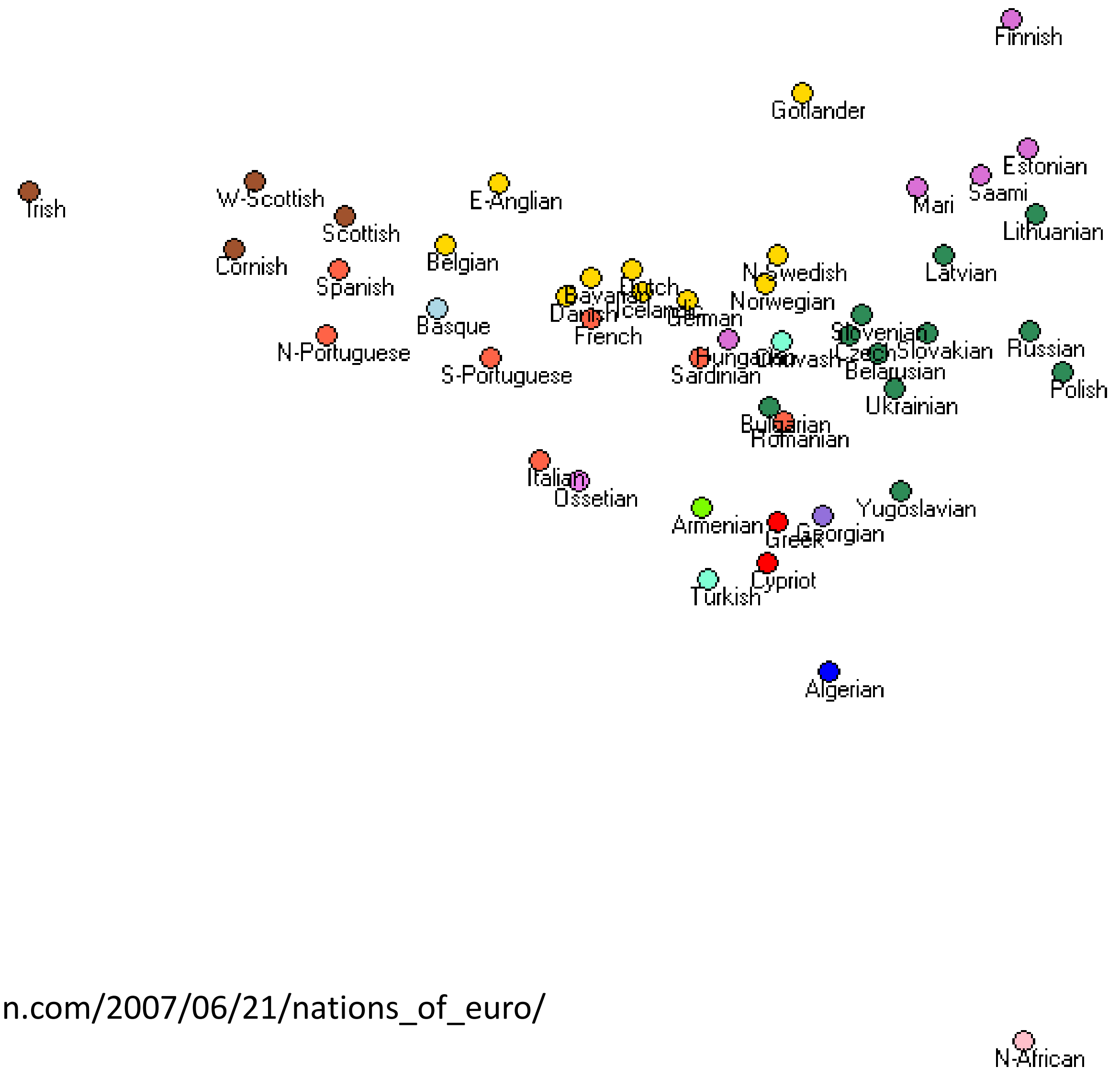
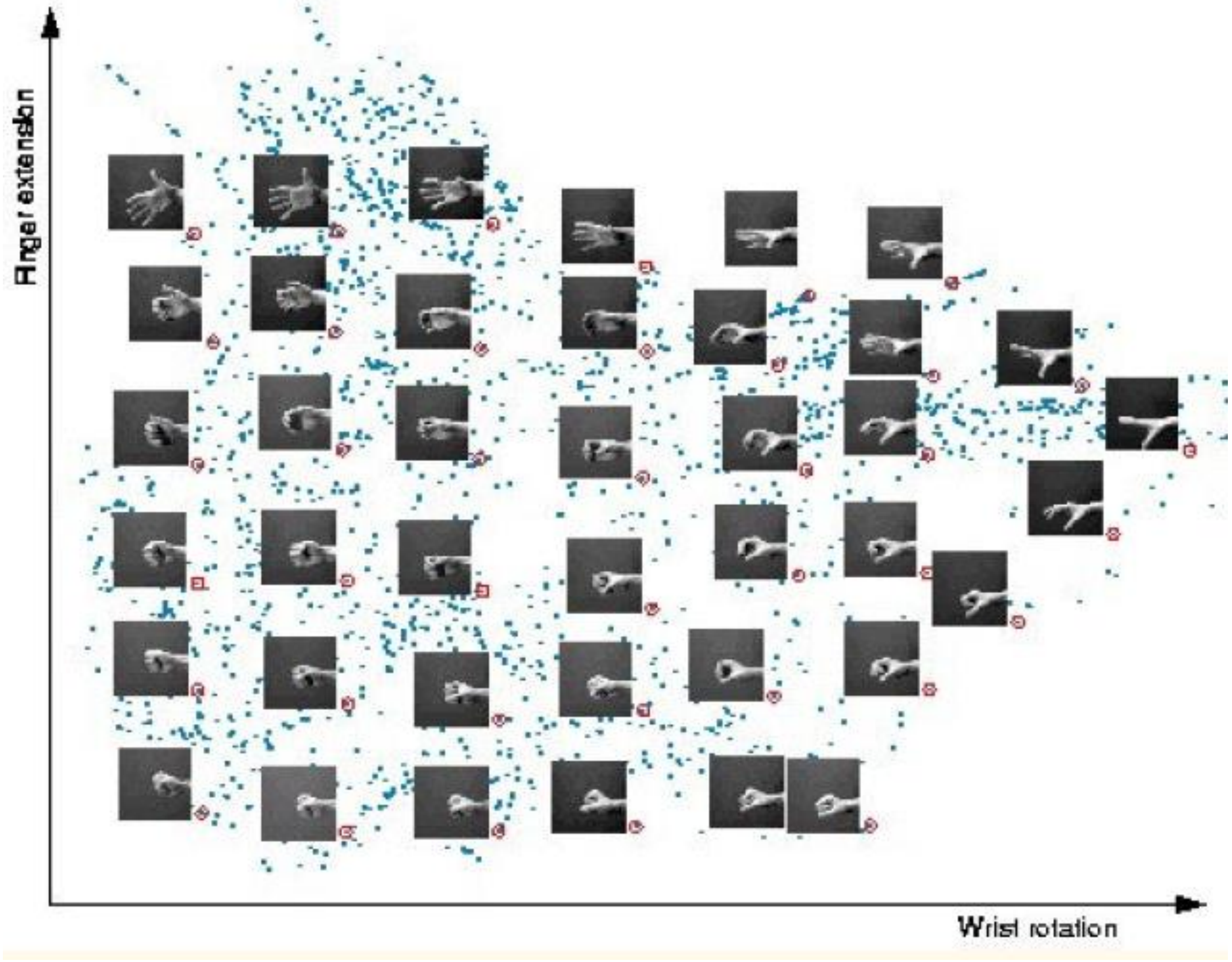


Figure from http://andrewgelman.com/2007/06/21/nations_of_euro/

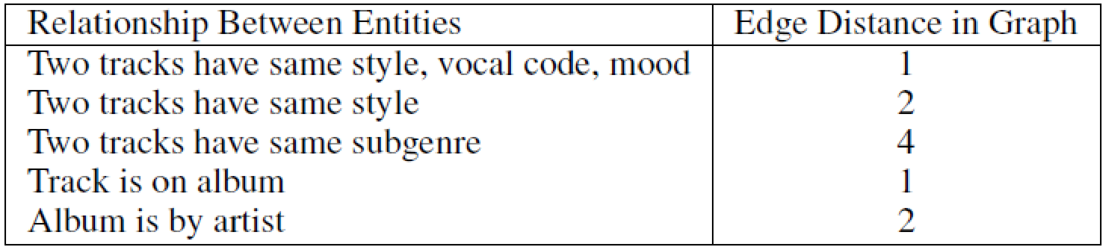
MDS of wrist images



Application of MDS to Music Playlists

Fast embedding of sparse music similarity graphs, J. Platt, 2004

- Music database
  - 10k artists, 68k albums, 189k tracks
  - Human-assigned metadata in terms of style/subgenre/vocal-code/mood

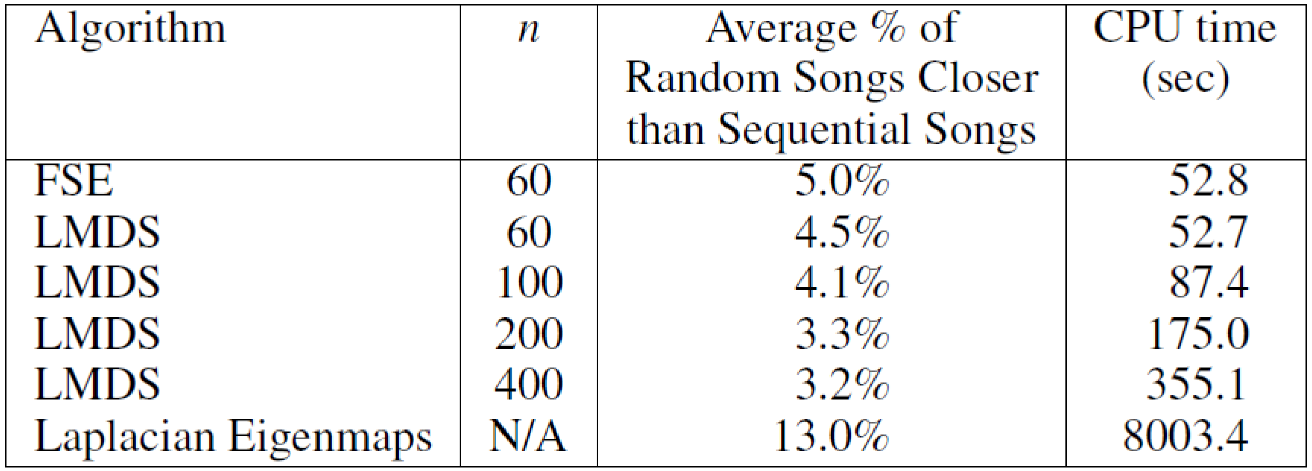


Relationship Between Entities	Edge Distance in Graph
Two tracks have same style, vocal code, mood	1
Two tracks have same style	2
Two tracks have same subgenre	4
Track is on album	1
Album is by artist	2

Table 1: Mapping of relationship to edge distance.

  - Results in graph with 267k vertices and 3.2 million edges
  - Complexity of MDS, $O(N^2 k)$ per iteration, is impractical here
  - Used fast sparse embedding (FSE) and Landmark MDS algorithms to embed data in 20 dimensions

Accuracy versus Human Playlists



Algorithm	n	Average % of Random Songs Closer than Sequential Songs	CPU time (sec)
FSE	60	5.0%	52.8
LMDS	60	4.5%	52.7
LMDS	100	4.1%	87.4
LMDS	200	3.3%	175.0
LMDS	400	3.2%	355.1
Laplacian Eigenmaps	N/A	13.0%	8003.4

Table 2: Speed and accuracy of music embedding for various algorithms.

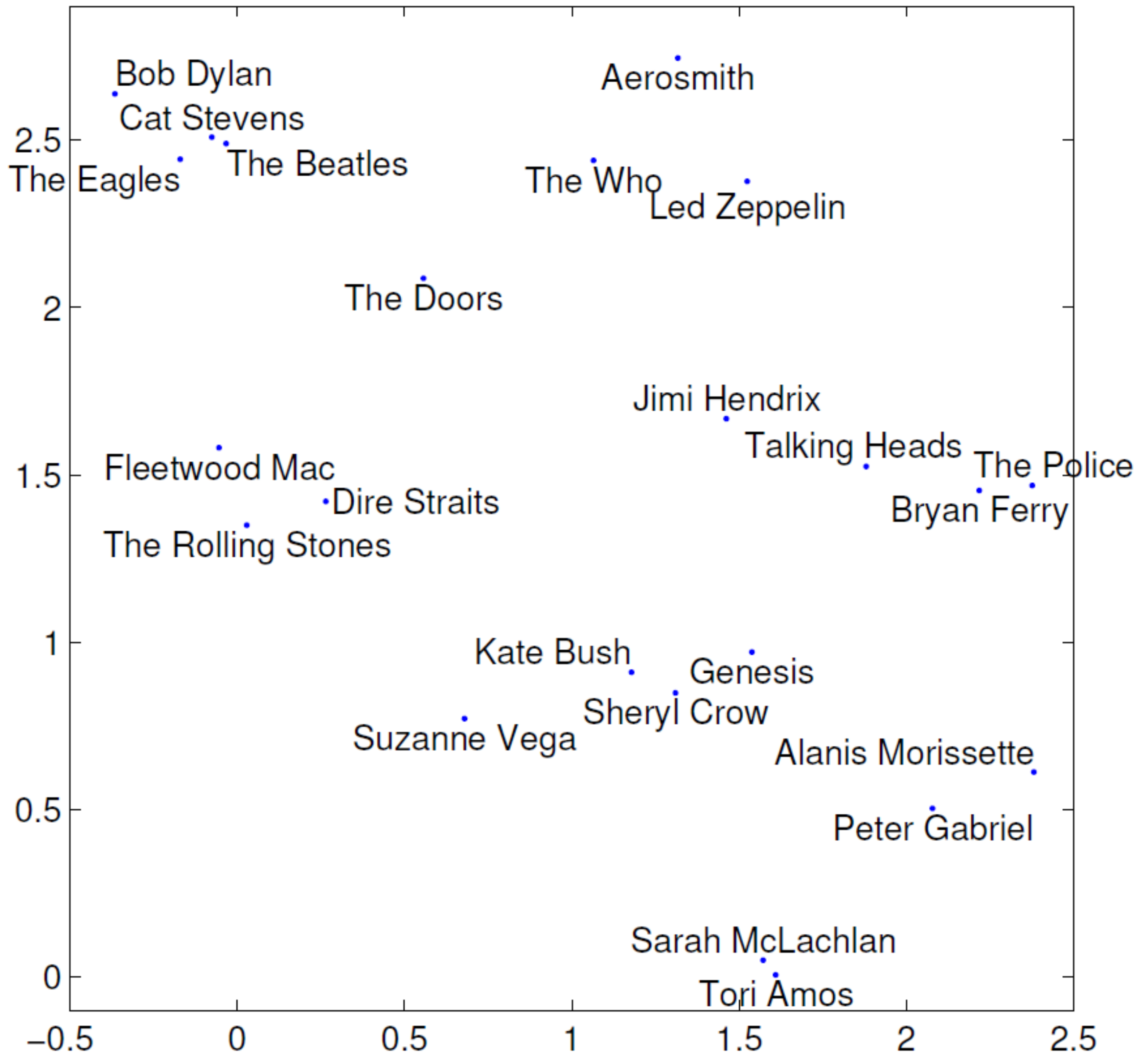
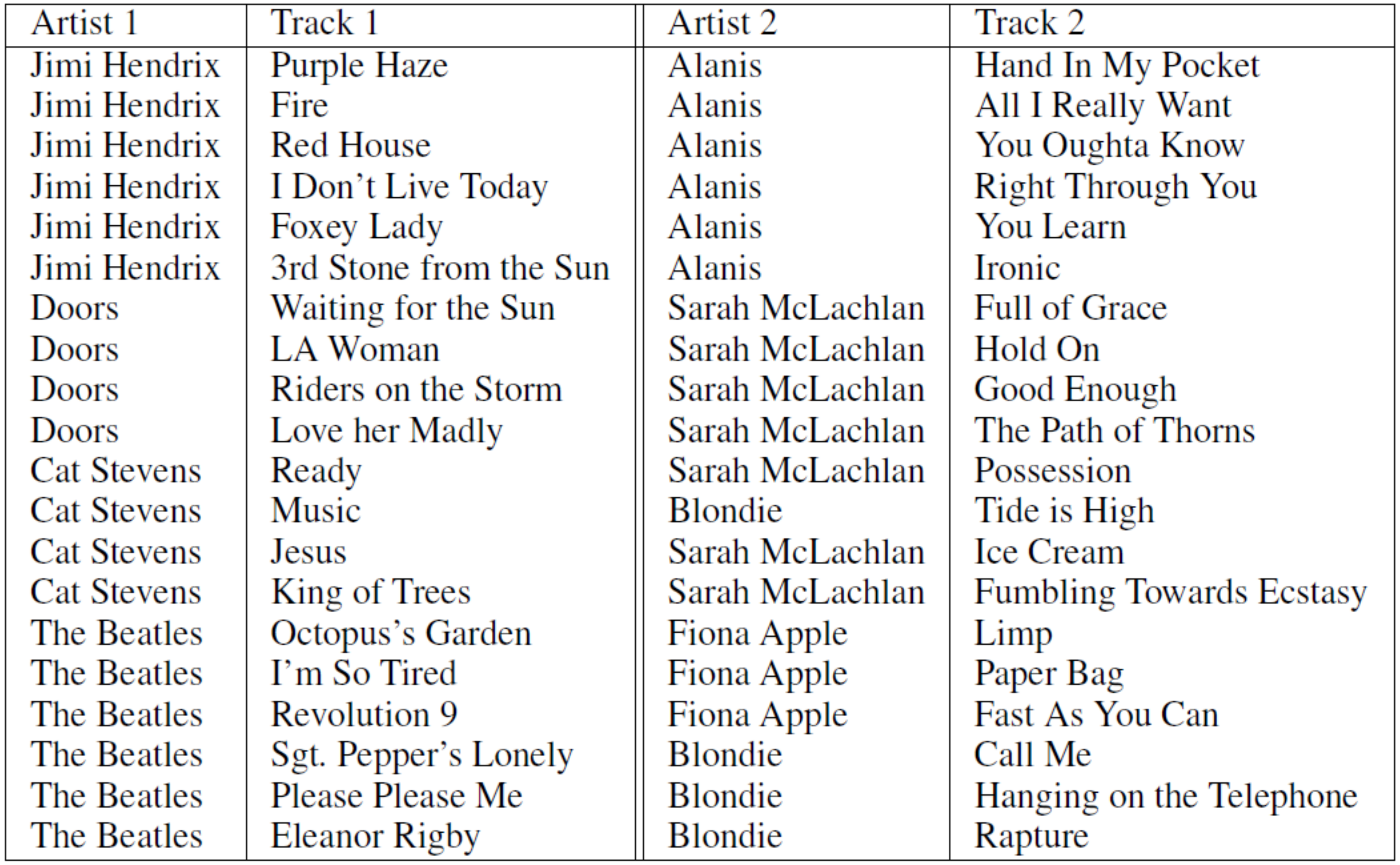


Figure 2: LMDS Projection of the entire music dissimilarity graph into 2D. The coordinates of 23 artists are shown.

Automated Playlist Generation



Artist 1	Track 1	Artist 2	Track 2
Jimi Hendrix	Purple Haze	Alanis	Hand In My Pocket
Jimi Hendrix	Fire	Alanis	All I Really Want
Jimi Hendrix	Red House	Alanis	You Oughta Know
Jimi Hendrix	I Don't Live Today	Alanis	Right Through You
Jimi Hendrix	Foxey Lady	Alanis	You Learn
Jimi Hendrix	3rd Stone from the Sun	Alanis	Ironic
Doors	Waiting for the Sun	Sarah McLachlan	Full of Grace
Doors	LA Woman	Sarah McLachlan	Hold On
Doors	Riders on the Storm	Sarah McLachlan	Good Enough
Doors	Love her Madly	Sarah McLachlan	The Path of Thorns
Cat Stevens	Ready	Sarah McLachlan	Possession
Cat Stevens	Music	Blondie	Tide is High
Cat Stevens	Jesus	Sarah McLachlan	Ice Cream
Cat Stevens	King of Trees	Sarah McLachlan	Fumbling Towards Ecstasy
The Beatles	Octopus's Garden	Fiona Apple	Limp
The Beatles	I'm So Tired	Fiona Apple	Paper Bag
The Beatles	Revolution 9	Fiona Apple	Fast As You Can
The Beatles	Sgt. Pepper's Lonely	Blondie	Call Me
The Beatles	Please Please Me	Blondie	Hanging on the Telephone
The Beatles	Eleanor Rigby	Blondie	Rapture

Table 3: Two playlists produced by the system. Each playlist reads top to bottom. The playlists interpolate between the first and last songs.

Relation between MDS and PCA

- Say our N x N matrix of dissimilarities actually represent Euclidean distances
  - e.g., matrix was computed based on d-dimensional distances of all pairs of rows in an N x d data matrix
  - This is referred to as "classic multidimensional scaling"

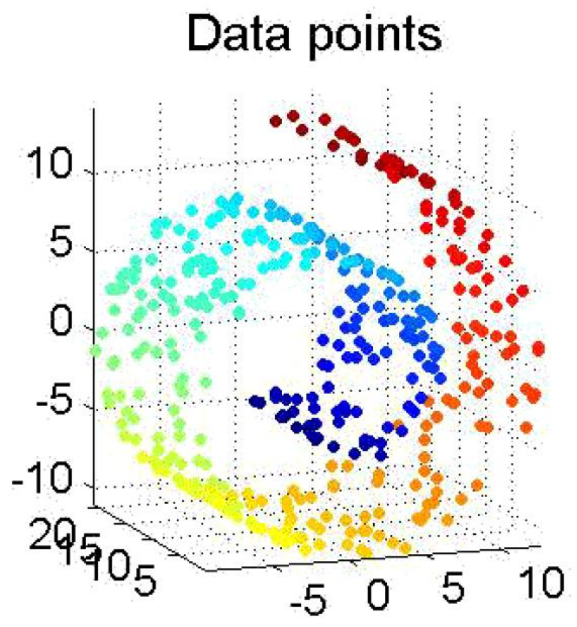
- In this case one can show that the optimal k-dimensional MDS solution is exactly the same as the k principal components of the N x d data matrix

- This suggests that MDS is doing something similar to PCA
  - i.e., if N x N dissimilarities are close to being Euclidean, we should expect a solution close to PCA

- Other variants of MDS try to relax the Euclidean/linear nature of MDS
  - E.g., "non-metric" scaling where we minimize S' where f is some monotonic function of the distances

$$S' = \sum_{i,j} (d(i,j) - f[\delta(i,j)])^2$$

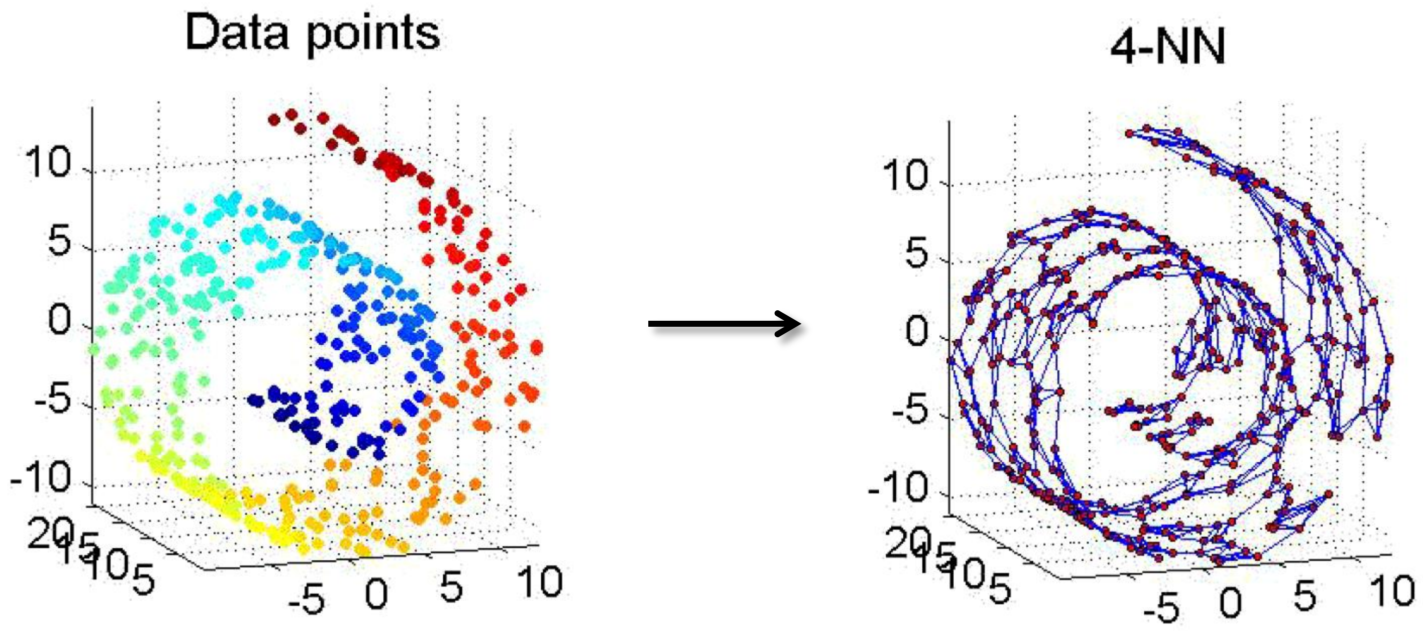
Limitations of Local Projections



Any linear projection will do poorly….
(think about what PCA will do to this data)

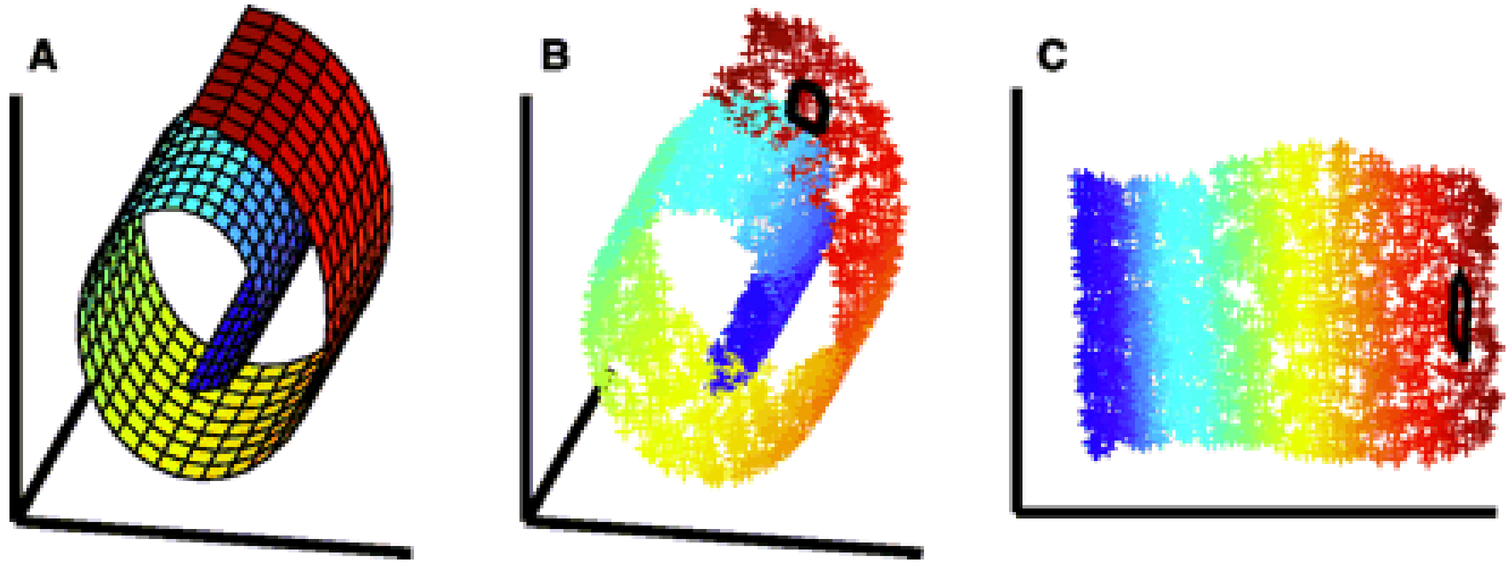
… but it is clear that the data “live on”
a lower-dimensional manifold

Using Local Embedding to find Non-Linear Structure



Local Linear Embedding (LLE)

(Roweis and Saul, *Science*, 2000)

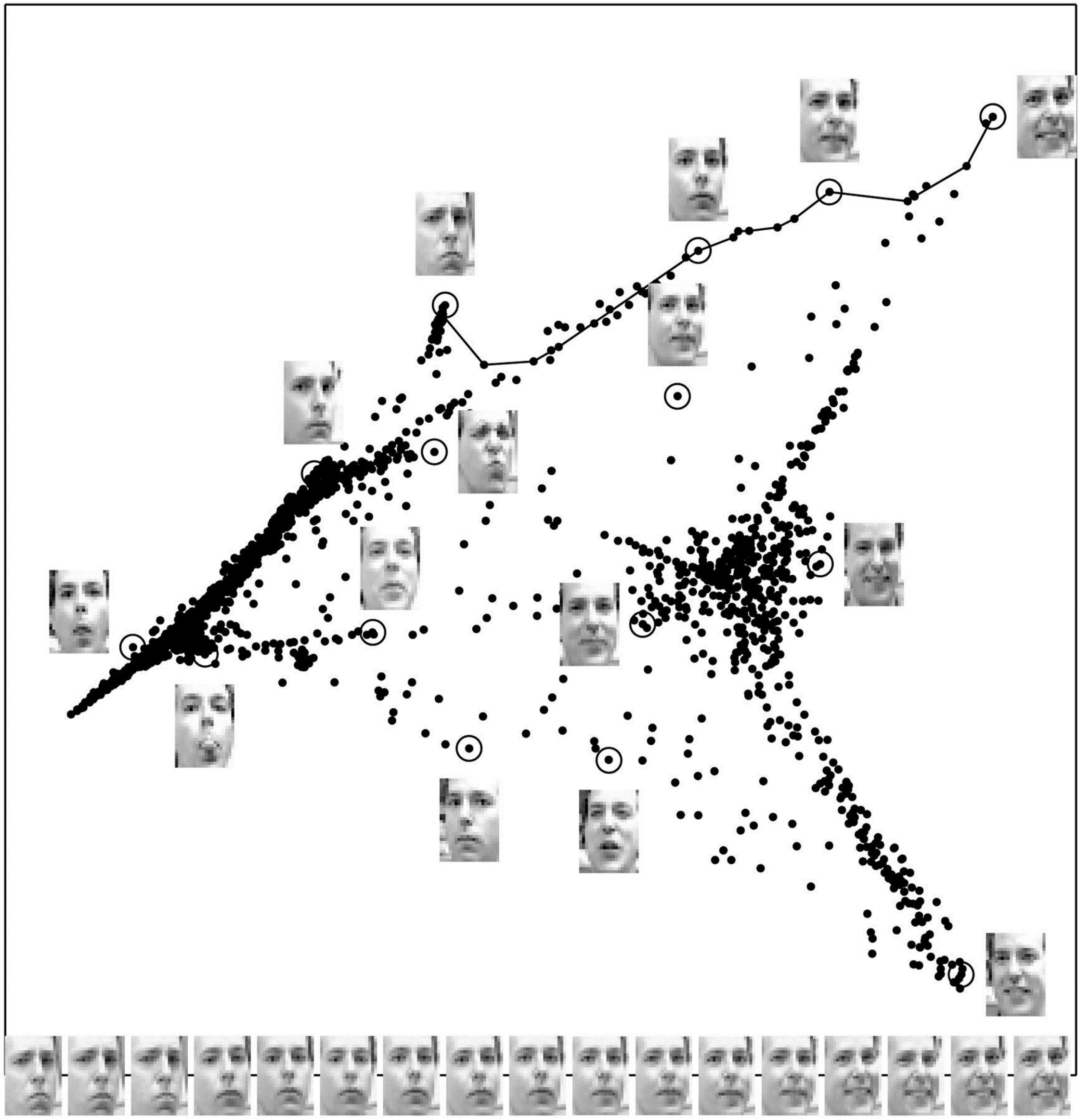


Note how points that are far away on the 3d manifold (e.g., red and blue) in “manifold distance” would be mapped as being close together by MDS or PCA but are kept “far apart” by LLE. LLE emphasizes local relationships

LLE Algorithm

- N points in dimension d: wish to reduce to dimension p, $p < d$
- Step 1:
  - Select K nearest neighbors for each point in training data
  - Represent each point as X = a weighted linear combination of its K neighbor points
  - Find best K weights for each of the X vectors (least squares fitting)
- Step 2:
  - Fix the weights from part 1
  - For each K-dim vector X, find a p-dimensional Y vector that is closest to its reconstructed approximation based on d-dim neighbors and weights
  - Reduces to another linear algebra/eigenvalue problem, $O(N^3)$ complexity

LLE applied to a set of face images

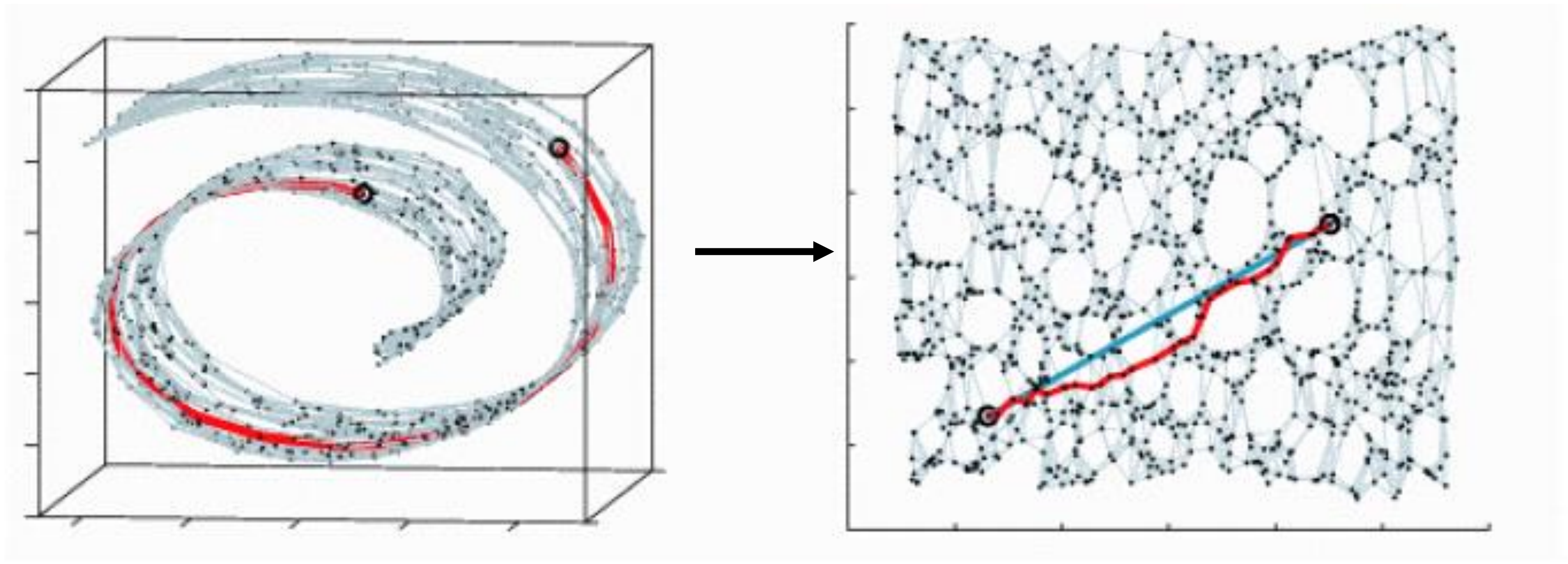


ISOMAP

Tenenbaum, de Silva, Langford, *Science*, 2000

Key idea

Distance between 2 points in the original space is their “geodesic path distance” on a hidden manifold in the original data space



ISOMAP

Tenenbaum, de Silva, Langford, *Science*, 2000

Key idea
Distance between 2 points in the original space is their "geodesic path distance" on a hidden manifold in the original data space

Outline of the ISOMAP Algorithm

- Create a graph of the N data points where data points are nodes
- Create an edge between them if they are "close" in original data space
- Approximate the geodesic distance by shortest paths in this graph
    - Use Floyd's algorithm: computationally intensive, $O(N^3)$
- Now use these N x N distances to run MDS to generate an embedding

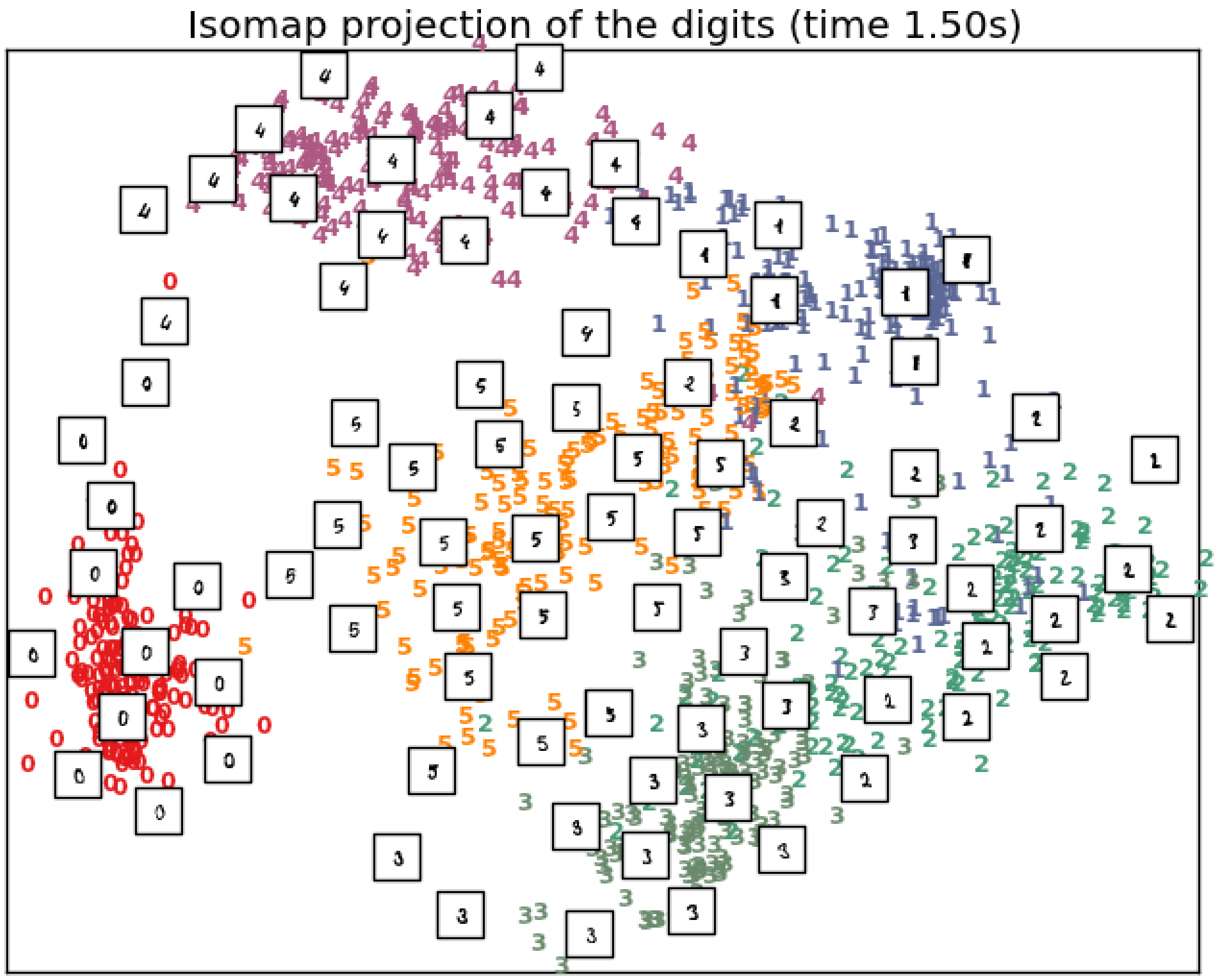


Figure from http://scikit-learn.org/0.12/_images/plot_lle_digits_51.png

Comparison of Different Methods

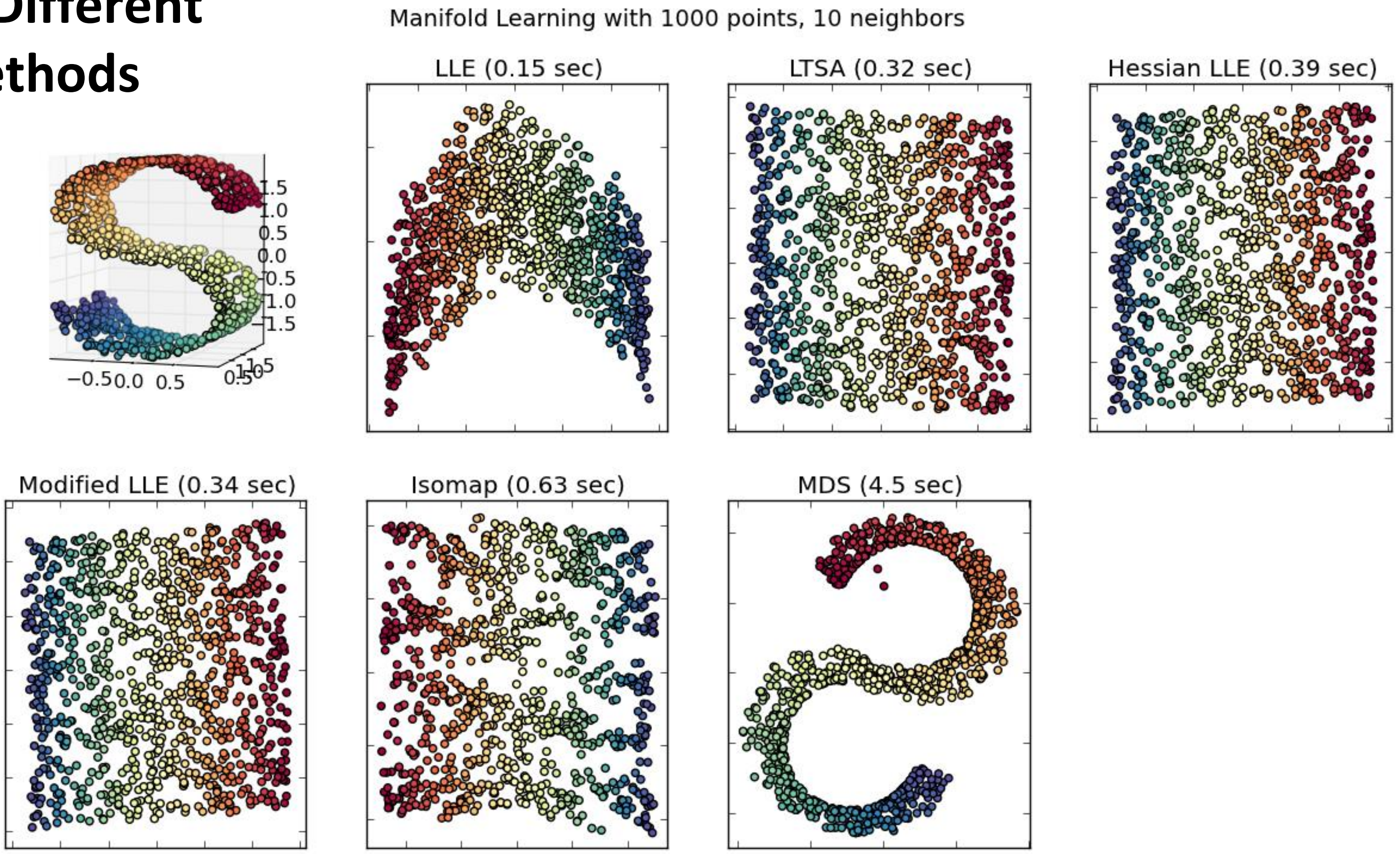


Figure from http://scikit-learn.org/0.12/modules/manifold.html

Summary on Dimension Reduction

- Useful for defining a new set of (lower-dimensional) variables
  - for modeling or for visualization/insight

- Three general approaches
  - Variable selection (only select “relevant” variables)
  - Linear projections (e.g., PCA)
  - Non-linear embedding methods (e.g., LLE, ISOMAP)

- Usual advice
  - These techniques can be useful, e.g., for visualization
  - Many different algorithms: PCA and MDS are best known “classics”
  - All algorithms have their built-in assumptions and limitations
  - There is no “universally-optimal” dimension reduction method.