CSE 564
Visualization & Visual Analytics

Dimension reduction

Klaus Mueller

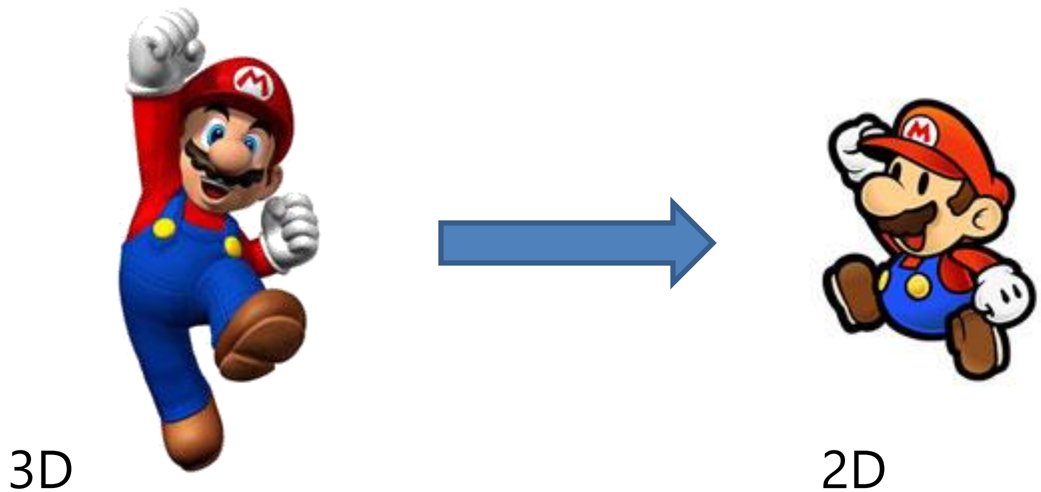
Computer Science Department
Stony Brook University

Lecture	Topic	Projects
1	Intro, schedule, and logistics	
2	Applications of visual analytics, basic tasks, data types	
3	Introduction to D3, basic vis techniques for non-spatial data	Project #1 out
4	Data assimilation and preparation	
5	Data reduction and notion of similarity and distance	
6	Visual perception and cognition	
7	Visual design and aesthetics	Project #1 due
8	Dimension reduction	Project #2 out
9	Data mining techniques: clusters, text, patterns, classifiers	
10	Data mining techniques: clusters, text, patterns, classifiers	
11	Computer graphics and volume rendering	
12	Techniques to visualize spatial (3D) data	Project #2 due
13	Scientific and medical visualization	Project #3 out
14	Scientific and medical visualization	
15	Midterm #1	
16	High-dimensional data, dimensionality reduction	Project #3 due
17	Big data: data reduction, summarization	
18	Correlation and causal modeling	
19	Principles of interaction	
20	Visual analytics and the visual sense making process	Final project proposal due
21	Evaluation and user studies	
22	Visualization of time-varying and time-series data	
23	Visualization of streaming data	
24	Visualization of graph data	Final Project preliminary report due
25	Visualization of text data	
26	Midterm #2	
27	Data journalism	
	Final project presentations	Final Project slides and final report due

Last Lecture's Theme

Data Reduction

This Lecture's Theme



Dimension Reduction

Measure of Attribute Similarity

Are there attributes that “go together”?

Can you name a few?

socrative
by MasteryConnect

Feature Vector (1)

Physical attributes

- color
- number of doors
- number of wheels
- retractable roof
- height
- length
- frames around side windows

Which attributes are useful to distinguish SUVs from convertibles?

- number of doors (4 vs. 2) --> numerical, two levels
- retractable roof (no vs. yes) --> categorical, two levels
- frames around side windows (yes vs. no) --> categorical, two levels
- height (higher vs. lower) --> numerical, many levels

Feature Vector (2)

Which attributes are not so useful?

- number of wheels (constant 4) --> no discriminative power
- length (short and long SUVs, convertibles) --> confounding
- color (colors are seemingly random, or are they?)

Is color useful?

- the convertibles seem to have more vibrant colors (red, yellow, …)
- so maybe we made a discovery

Attribute Space

Need to consider more than two attributes

- *height* attribute would have distinguished the Range Rover from the convertibles and caused it to be and outlier

Attribute Space

why can empty feature spaces be interesting or useful?

socrative by MasteryConnect

New classes are constantly evolving over time

- this is known as *cluster evolution*
- measuring more features will increase the chance of discovery

How Many Data Do We Need?

The more data (examples) the better

- increases the chances to discover the rare specimen

- but some attributes are useless
- we can cull them away
- perform attribute reduction or *dimension reduction*

Dimensionality Reduction

By axis rotation

- determine a more efficient basis
- Principal Component Analysis (PCA)
- Singular value decomposition (SVD)
- Latent semantic analysis (LSA)

By type transformation

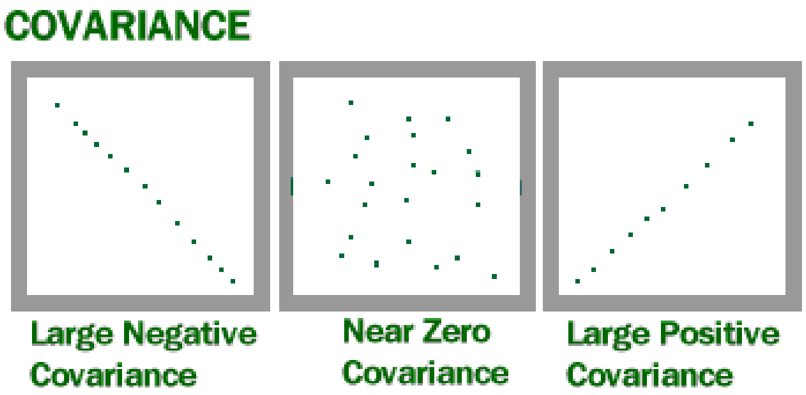
- determine a more efficient data type
- Fourier analysis and Wavelets for grids
- Multidimensional scaling (MSD) for graphs
- Locally Linear Embedding
- Isomap
- Self Organizing Maps (SOM)
- Linear Discriminant Analysis (LDA)

Principal Component Analysis (PCA)

Some Theory is Needed

Covariance

- measures how much two random variables change together



For N variable we have N^2 variable pairs

- we can write them in a matrix of size N^2 → the *covariance matrix*
- for two variables X_1 and X_2

$$\mathrm{Var}[X] = \begin{bmatrix} \mathrm{Var}[X_1] & \mathrm{Cov}[X_1, X_2] \\ \mathrm{Cov}[X_2, X_1] & \mathrm{Var}[X_2] \end{bmatrix}$$

FORMULAE

Covariance cov(X,Y)

mean of all data item values x_i and y_i for attributes X and Y, resp.

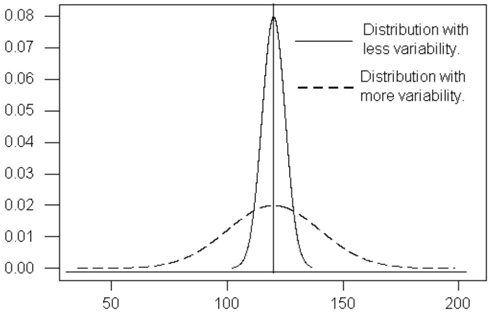
$$COV(X,Y) = \frac{\sum_{i=1}^{n}\left(X_i - \bar{X}\right)\left(Y_i - \bar{Y}\right)}{n-1}$$

Pearson's correlation r

- is covariance normalized by the individual variances for X and Y

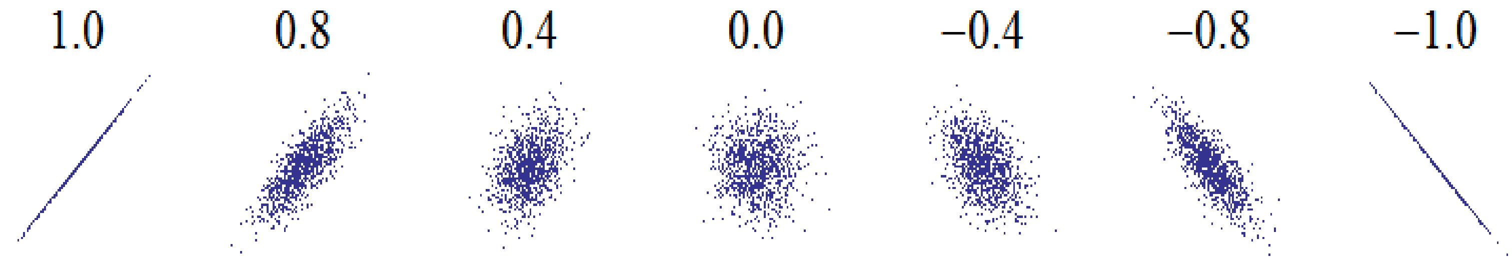
$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{x})^2}}$$

individual variances for attributes X and Y



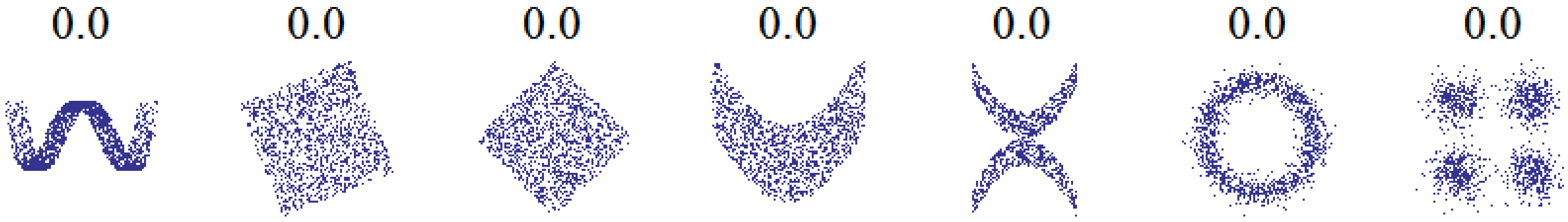
Correlation Patterns

Correlation rates between -1 and 1:



Important to note:

- correlation is defined for linear relationships
- visualization can help
- none of these point distributions have correlations:



Covariance Matrix

Analytical: $$Cov(X,Y) = E[(X-\mu_x)(Y-\mu_y)]$$

Samples: $$\sigma_{xy} = \mathrm{cov}_{xy} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

An n-D dataset has *n* variables $x_1, x_2, \ldots x_n$

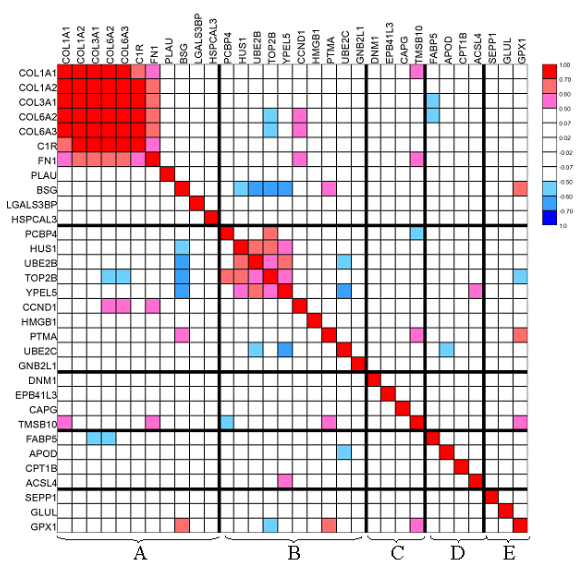
- define pairwise covariance among all of these variables
- construct a covariance matrix

$$\Sigma = \mathrm{Cov}(X) = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1n} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{n1} & \sigma_{n2} & \cdots & \sigma_{nn} \end{bmatrix}$$

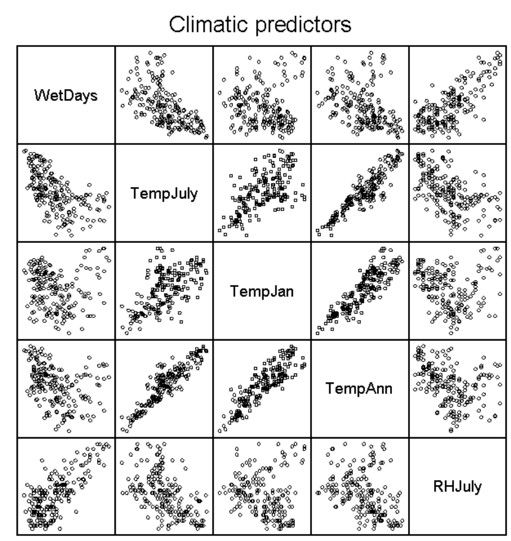
- a correlation matrix would just list the correlations instead

Correlation Matrix

	MO	FP	MP	IM	IC	FM	FE	FI	SPC	DSC	DST
MO	1.00										
FP	0.31[a]	1.00									
MP	0.32[a]	0.71[a]	1.00								
IM	0.36[a]	0.12[c]	0.14[c]	1.00							
IC	0.39[a]	0.18[b]	0.21[a]	0.62[a]	1.00						
FM	0.26[a]	0.21[a]	0.14[c]	0.30[a]	0.27[a]	1.00					
FE	0.47[a]	0.21[a]	0.18[b]	0.38[a]	0.28[a]	0.24[a]	1.00				
FI	0.53[a]	0.26[a]	0.22[a]	0.36[a]	0.37[a]	0.29[a]	0.47[a]	1.00			
SPC	0.32[a]	0.22[a]	0.31[a]	0.51[a]	0.47[a]	0.32[a]	0.37[a]	0.35[a]	1.00		
DSC	−0.12[c]	0.03[c]	0.05[c]	0.17[b]	0.08[c]	0.18[b]	−0.05[c]	0.06[c]	0.01[c]	1.00	
DST	−0.02[c]	−0.01[c]	0.05[c]	0.24[a]	0.14[c]	0.05[c]	−0.05[c]	0.05[c]	0.05[c]	0.56[a]	1.00
DM	0.05[c]	0.144	0.136[c]	0.199[a]	0.169[b]	0.247[a]	0.08[c]	0.11[c]	0.14[c]	0.46[a]	0.71[a]



just value

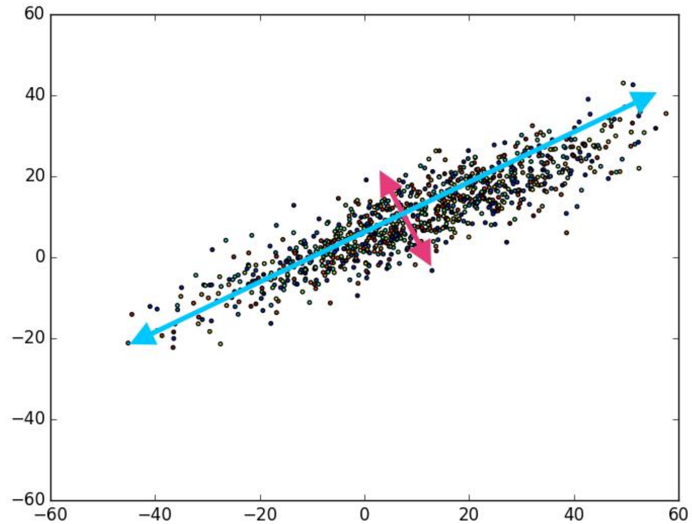


distribution (scatterplot matrix)

Principal Component Analysis

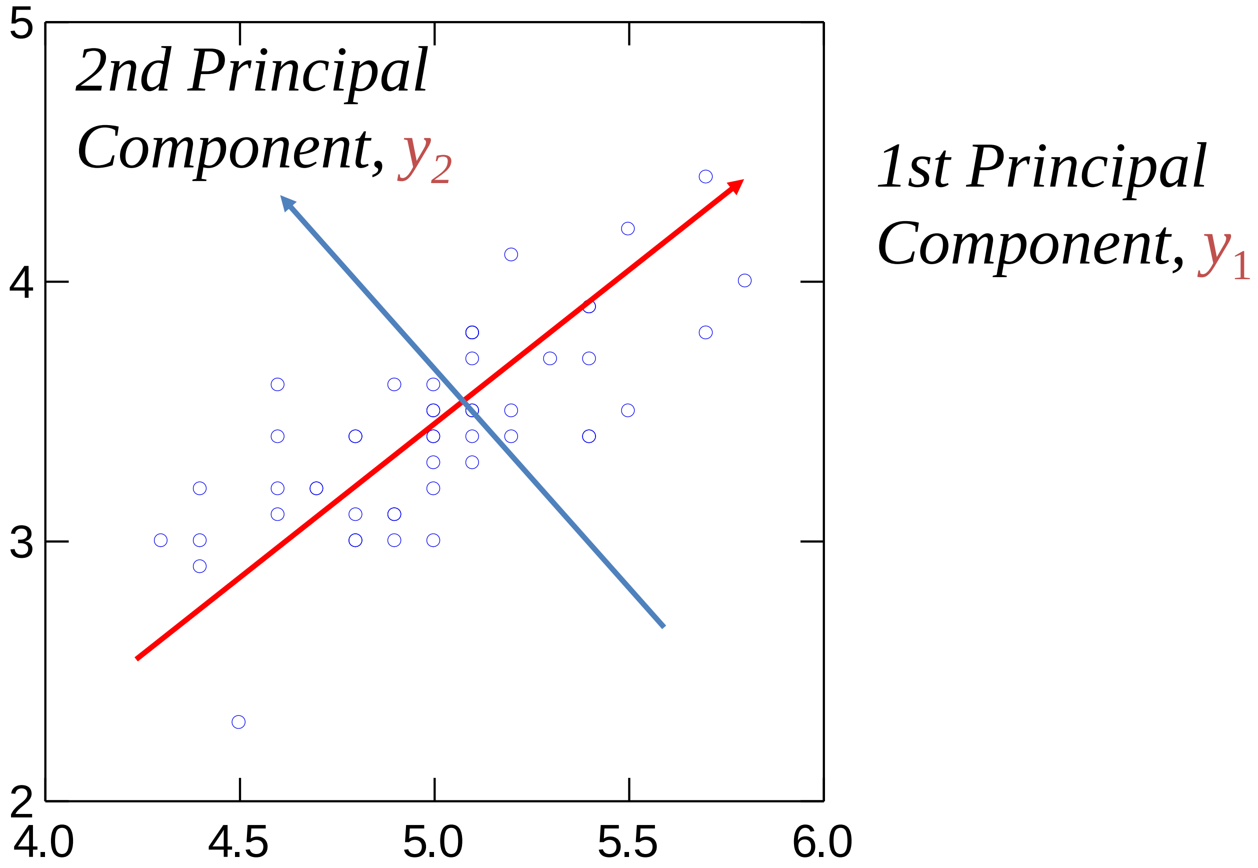
Ultimate goal:

- find a coordinate system that can represent the variance in the data with as few axes as possible



- rank these axes by the amount of variance (blue, red)
- drop the axes that have the least variance (red)

Principal Components



PCA – How To Do

Find the principal components (factors) of a distribution

First characterize the distribution by

- covariance matrix Cov
- correlation matrix Corr
- lets call it C

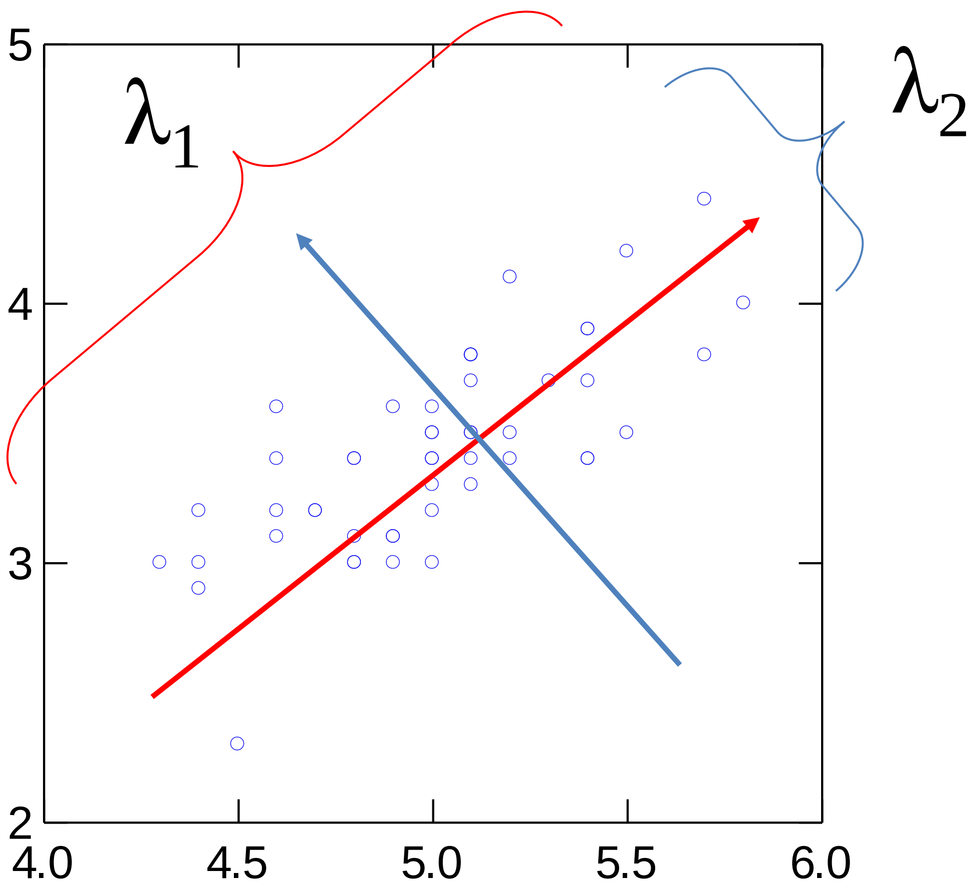
- perform QR factorization or LU decomposition to get

$$\boldsymbol{C} = Q\Lambda Q^{-1}$$

Q: matrix with Eigenvectors
Λ: diagonal matrix with Eigenvalues λ

- now order the Eigenvectors in terms of their Eigenvalues λ

Eigenvectors and values



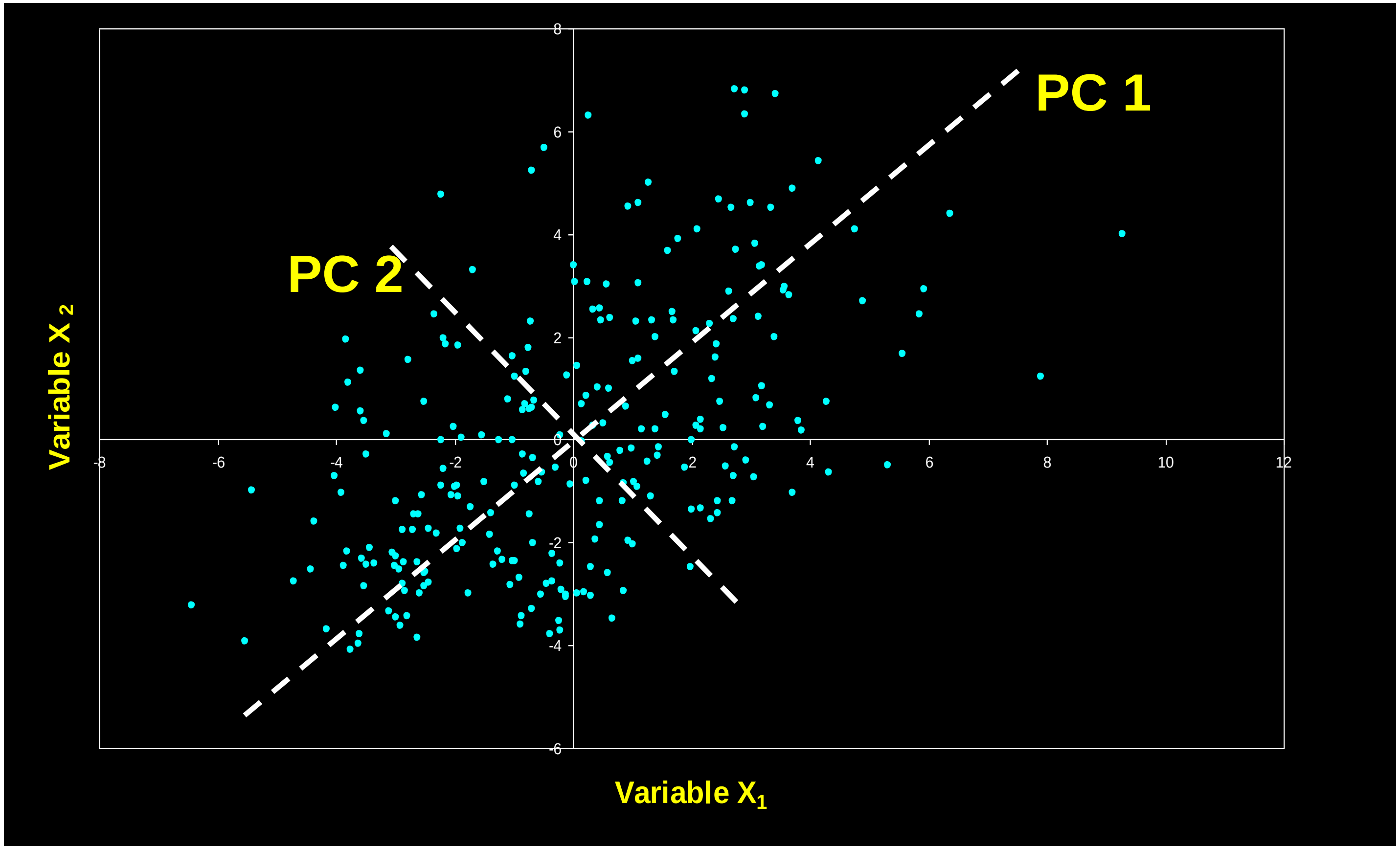
Covariance vs. Correlation

When to use what?

- use the covariance matrix when the variable scales are similar
- use the correlation matrix when the variables are on different scales
- the correlation matrix *standardizes* the data
- in general they give different results, especially when the scales are different

EXAMPLE

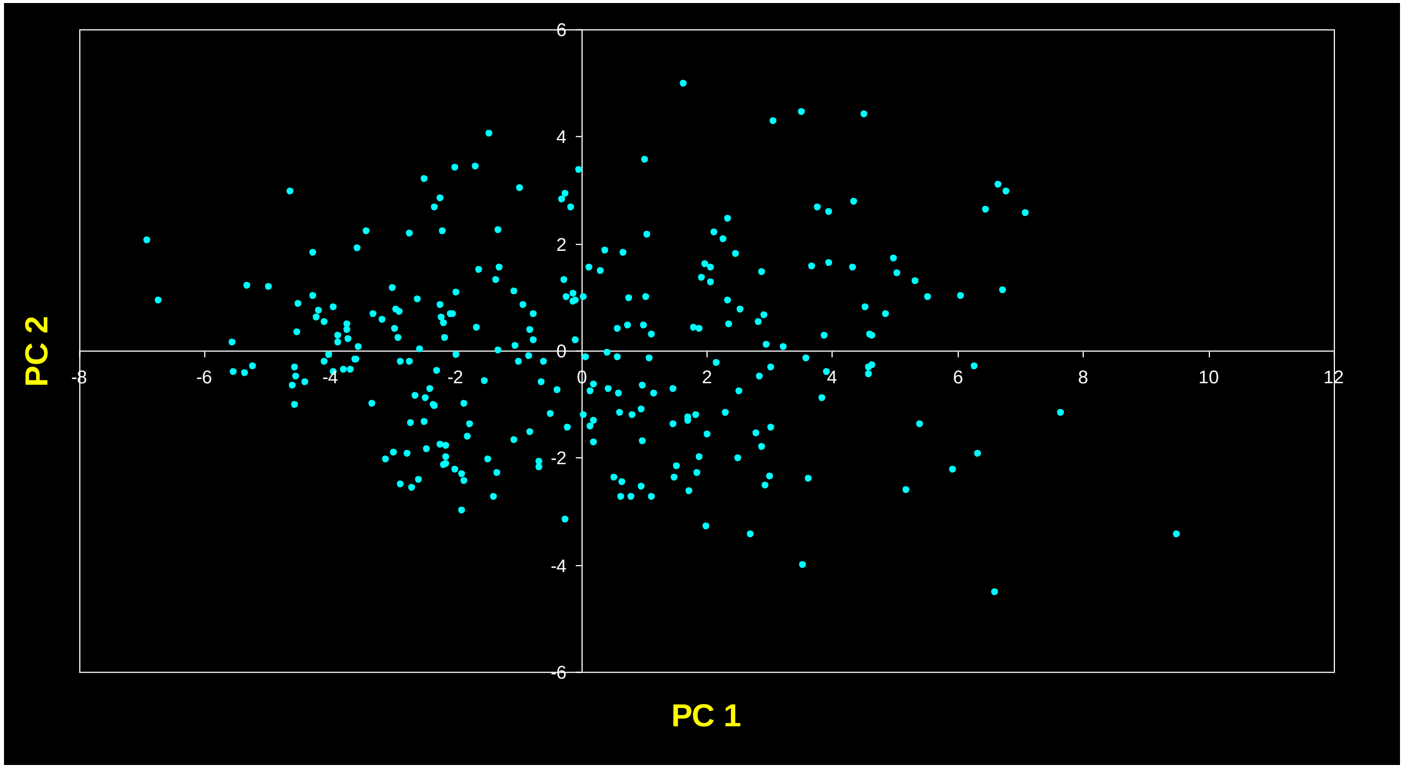
Before PCA



Example

After PCA

- λ_1 = 9.8783 λ_2 = 3.0308 Trace = 12.9091
- PC 1 displays ("explains") 9.8783/12.9091 = 76.5% of total variance



More Info on PCA

See other slide sets posted on the course website

Principal Component Analysis (PCA)

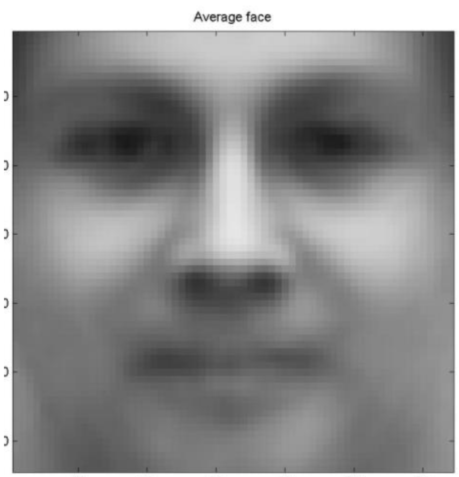
- Theory, Practice, and Examples
- PCA loadings, and what they mean for analysis

PCA Applied To Faces

Some familiar faces…

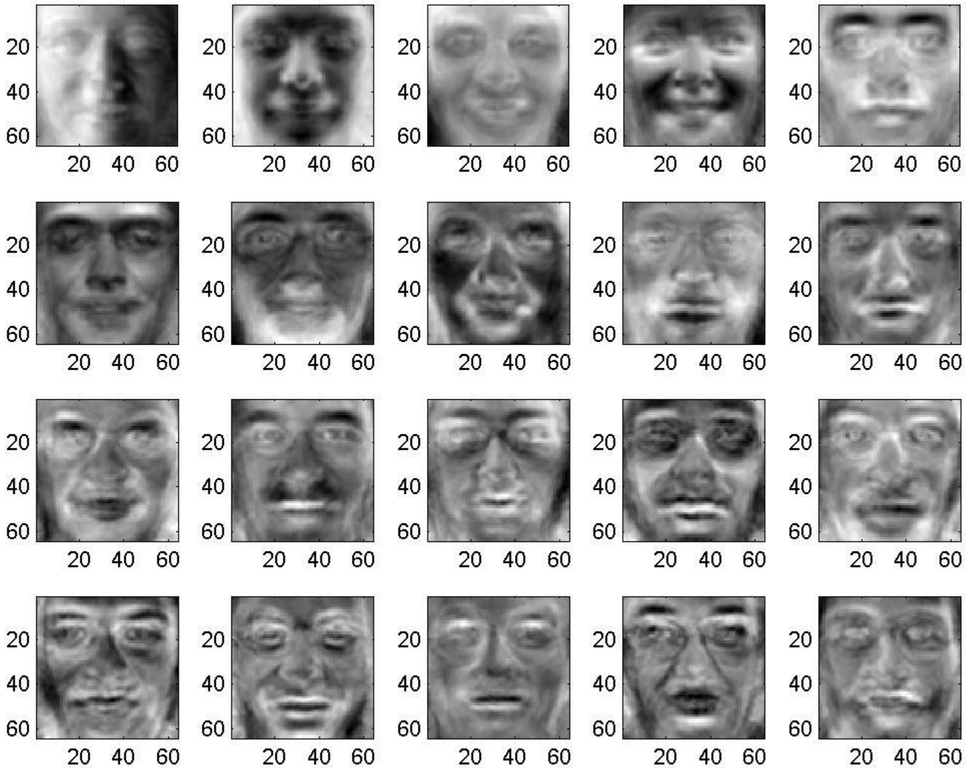
PCA Applied To Faces

We can reconstruct each face as a linear combination of "basis" faces, or Eigenfaces [M. Turk and A. Pentland (1991)]



Average Face

+



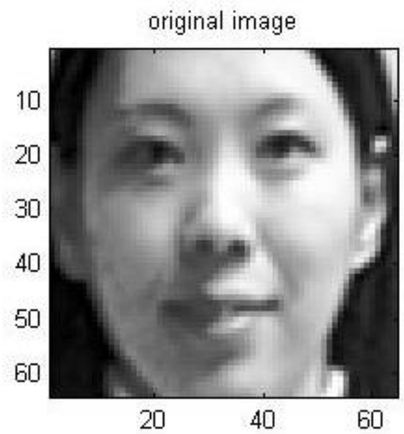
Eigenfaces

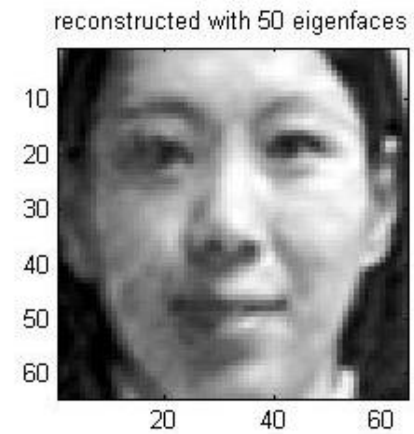
Reconstruction Using PCA

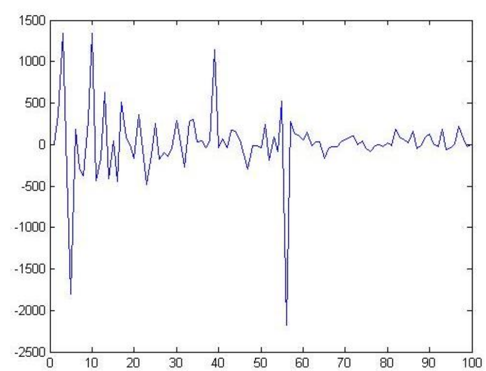
90% variance is captured by the first 50 eigenvectors

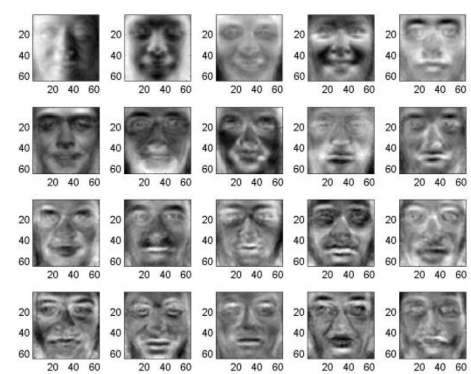
Reconstruct existing faces using only 50 basis images

We can also generate new faces by combining eigenvectors with different weights









Transformations

Multidimensional Scaling (MDS)

MDS is for irregular structures

- scattered points in high-dimensions (N-D)
- adjacency matrices

Maps the distances between observations from N-D into low-D (say 2D)

- attempts to ensure that differences between pairs of points in this reduced space match as closely as possible

Distance Matrix

MDS turns a distance matrix into a network or point cloud

- correlation, cosine, Euclidian, and so on

Suppose you know a matrix of distances among cities

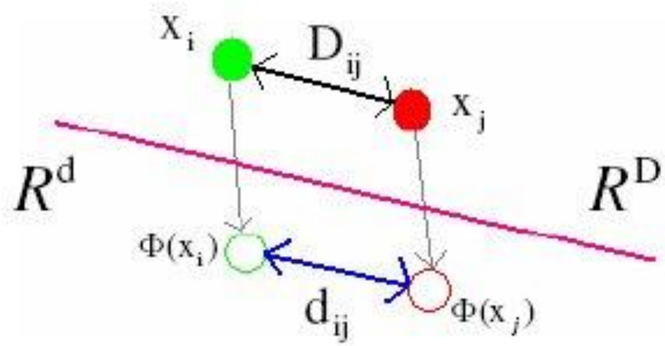
	Chicago	Raleigh	Boston	Seattle	S.F.	Austin	Orlando
Chicago	0						
Raleigh	641	0					
Boston	851	608	0				
Seattle	1733	2363	2488	0			
S.F.	1855	2406	2696	684	0		
Austin	972	1167	1691	1764	1495	0	
Orlando	994	520	1105	2565	2458	1015	0

Result of MDS

Compare with real Map

MDS Algorithm

- Task:
  - Find that configuration of image points whose pairwise distances are most similar to the original inter-point distances !!!
- Formally:
  - Define: $D_{ij} = \| x_i - x_j \|_D \qquad d_{ij} = \| y_i - y_j \|_d$
  - Claim: $D_{ij} \equiv d_{ij} \qquad \forall i, j \in [1, n]$
- In general: an exact solution is not possible !!!
- Inter Point distances → invariance features



MDS Algorithm

<u>Strategy (of metric MDS):</u>

- iterative procedure to find a good configuration of image points
  - 1) Initialization
    → Begin with some (arbitrary) initial configuration
  - 2) Alter the image points and try to find a configuration of points that minimizes the following sum-of-squares error function:

MDS Algorithm

Strategy (of metric MDS):

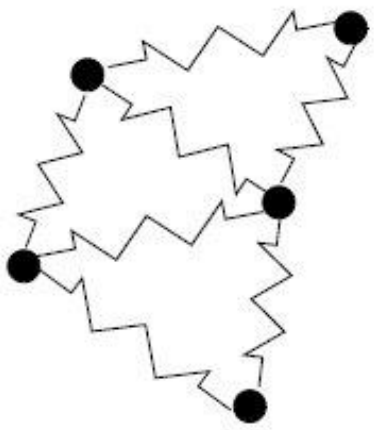
- iterative procedure to find a good configuration of image points
  - 1) Initialization
    → Begin with some (arbitrary) initial configuration
  - 2) Alter the image points and try to find a configuration of points that minimizes the following sum-of-squares error function:

$$E = \sum_{i<j}^{N} \left(D_{ij} - d_{ij}\right)^2$$

Force-Directed Algorithm

Spring-like system

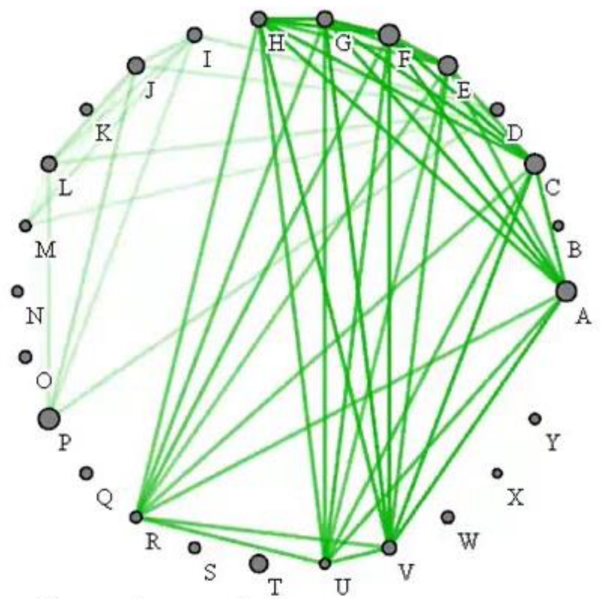
- insert springs within each node
- the length of the spring encodes the desired node distance
- start at an initial configuration
- iteratively move nodes until an energy minimum is reached



Force-Directed Algorithm

Spring-like system

- insert springs within each node
- the length of the spring encodes the desired node distance
- start at an initial configuration
- iteratively move nodes until an energy minimum is reached

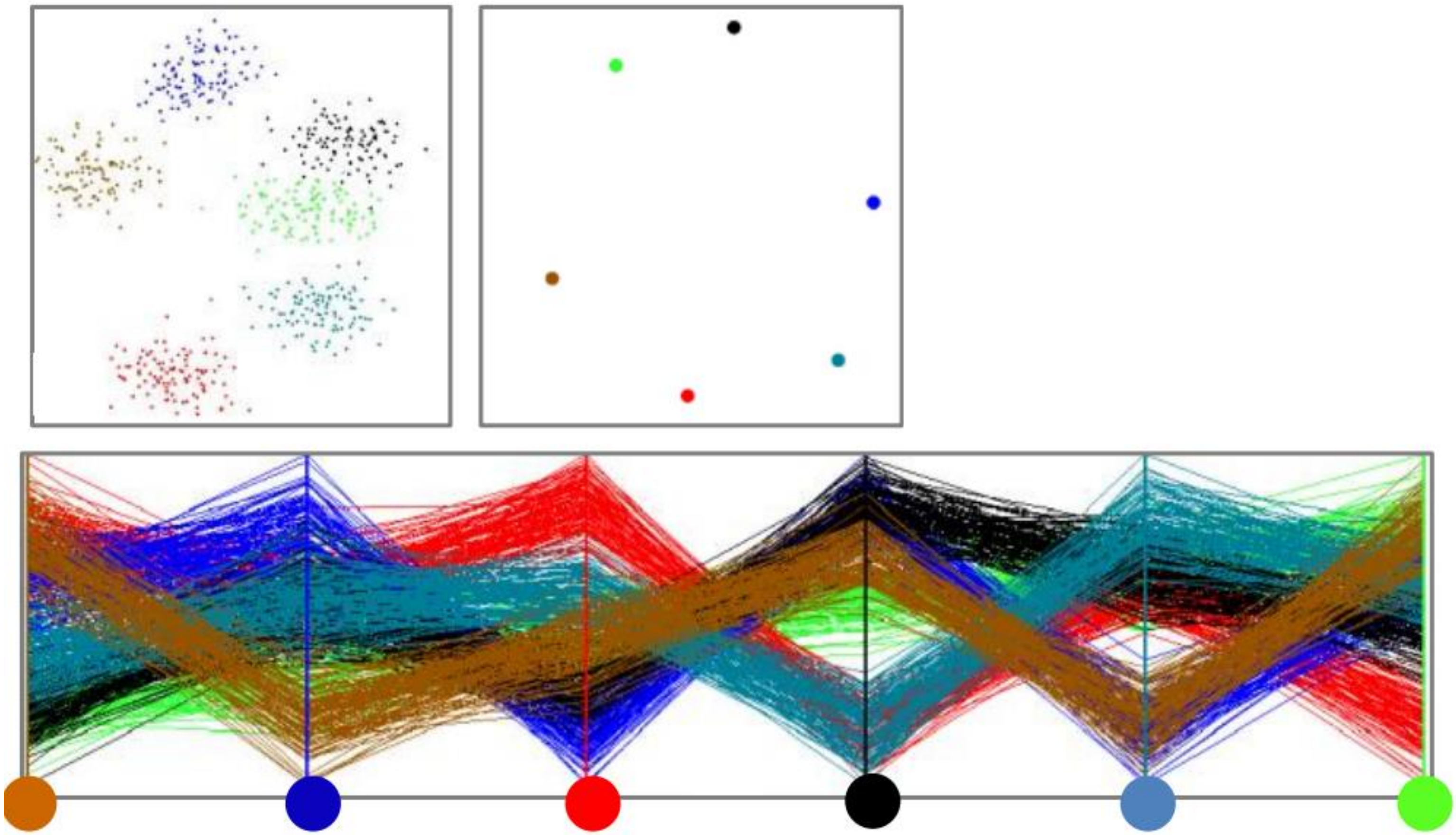


Vertex layout by correlations.

USES OF MDS

Data layout

Attribute layout

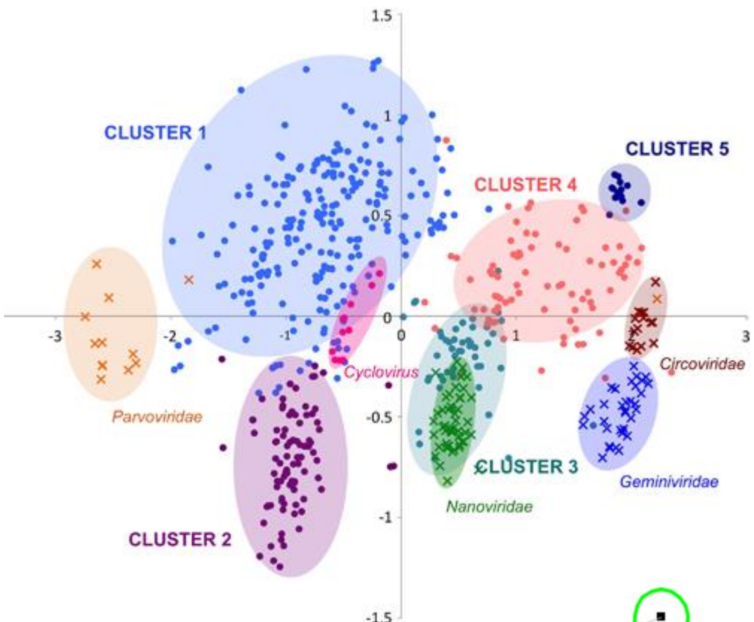


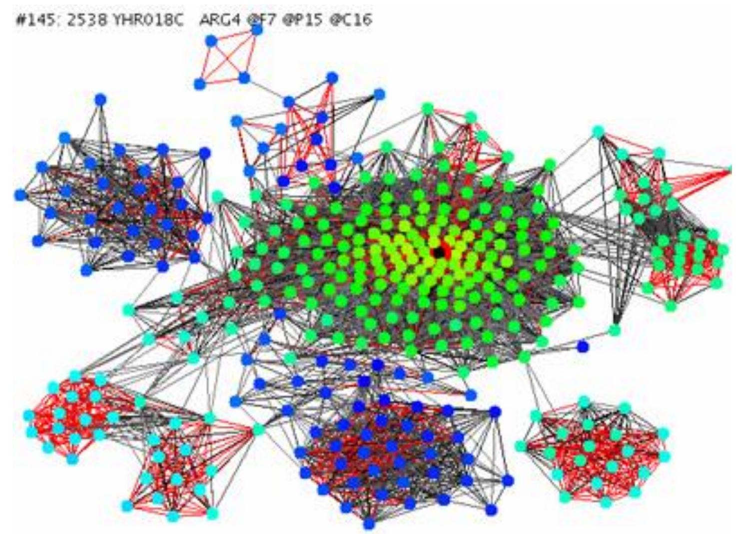
Uses of MDS

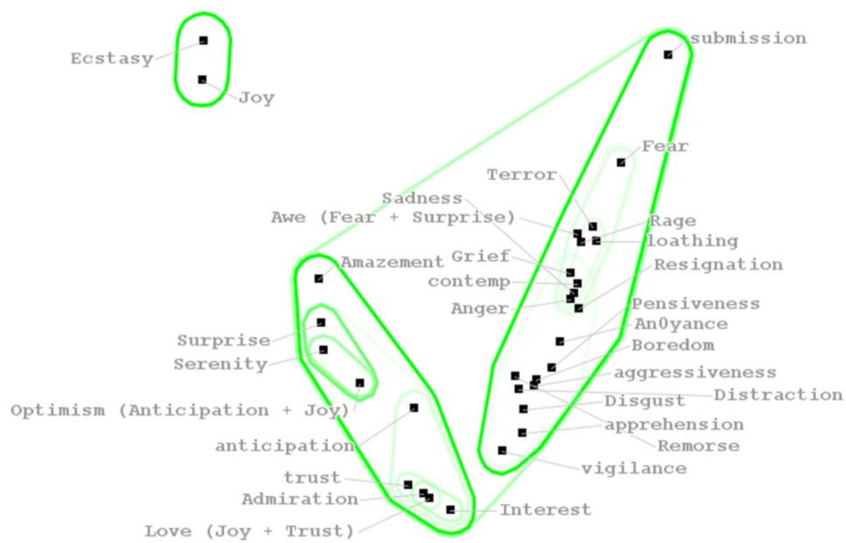
Distance (similarity) metric

- Euclidian distance (best for data)
- Cosine distance (best for data)
- |1-correlation| distance (best for attributes)
- use 1-correlation to move correlated attribute points closer
- use | | if you do not care about positive or negative correlations

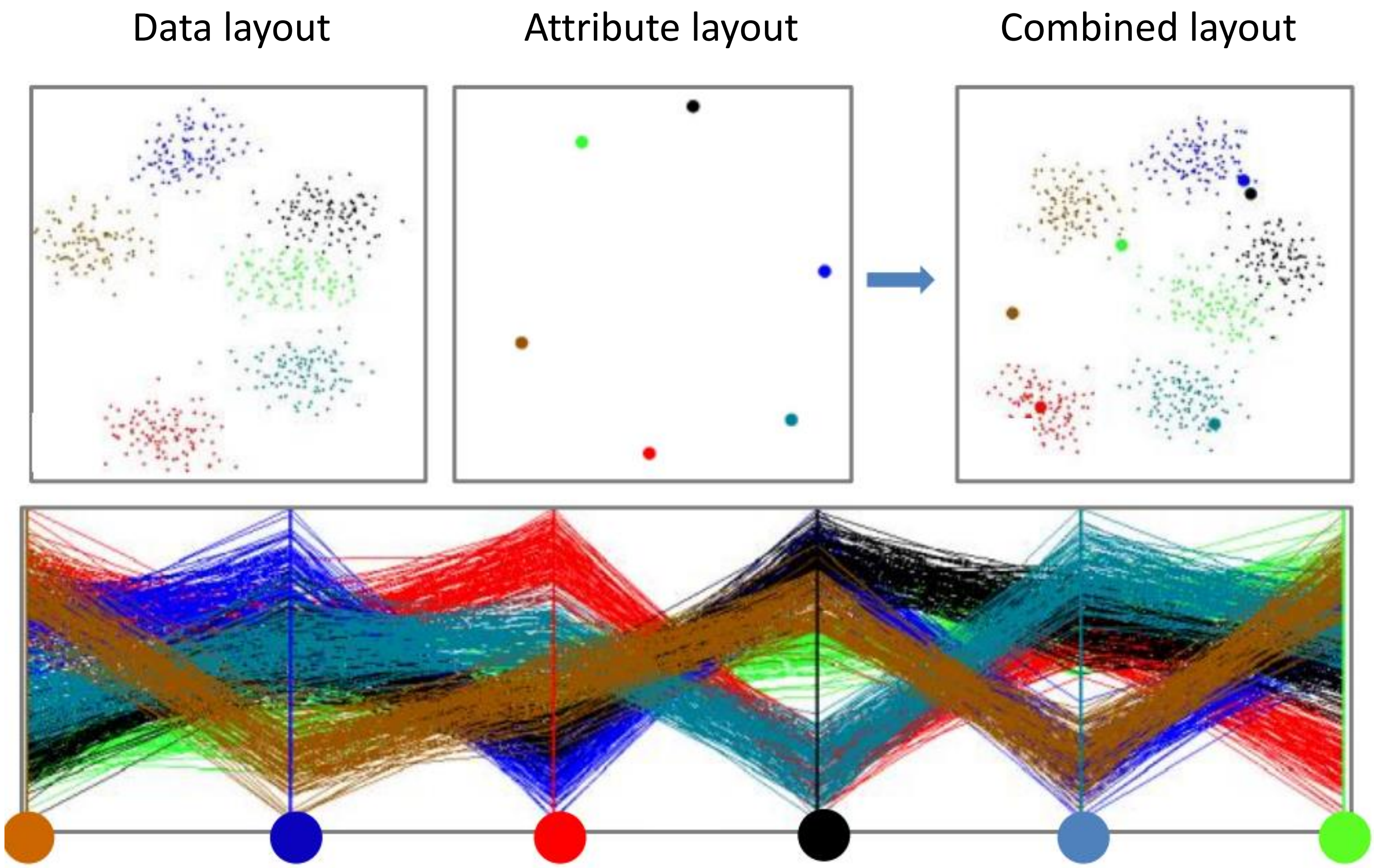
MDS Examples



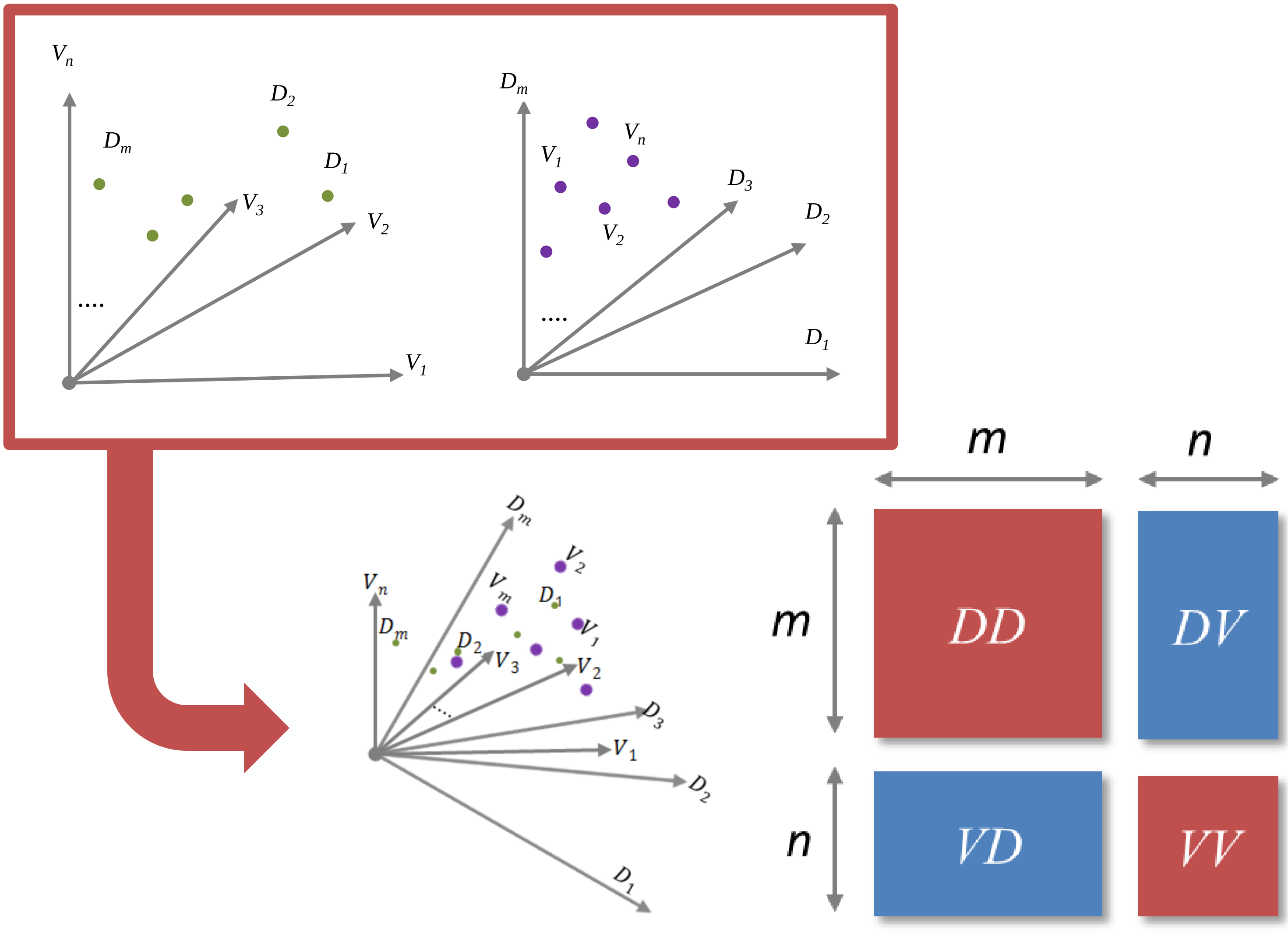




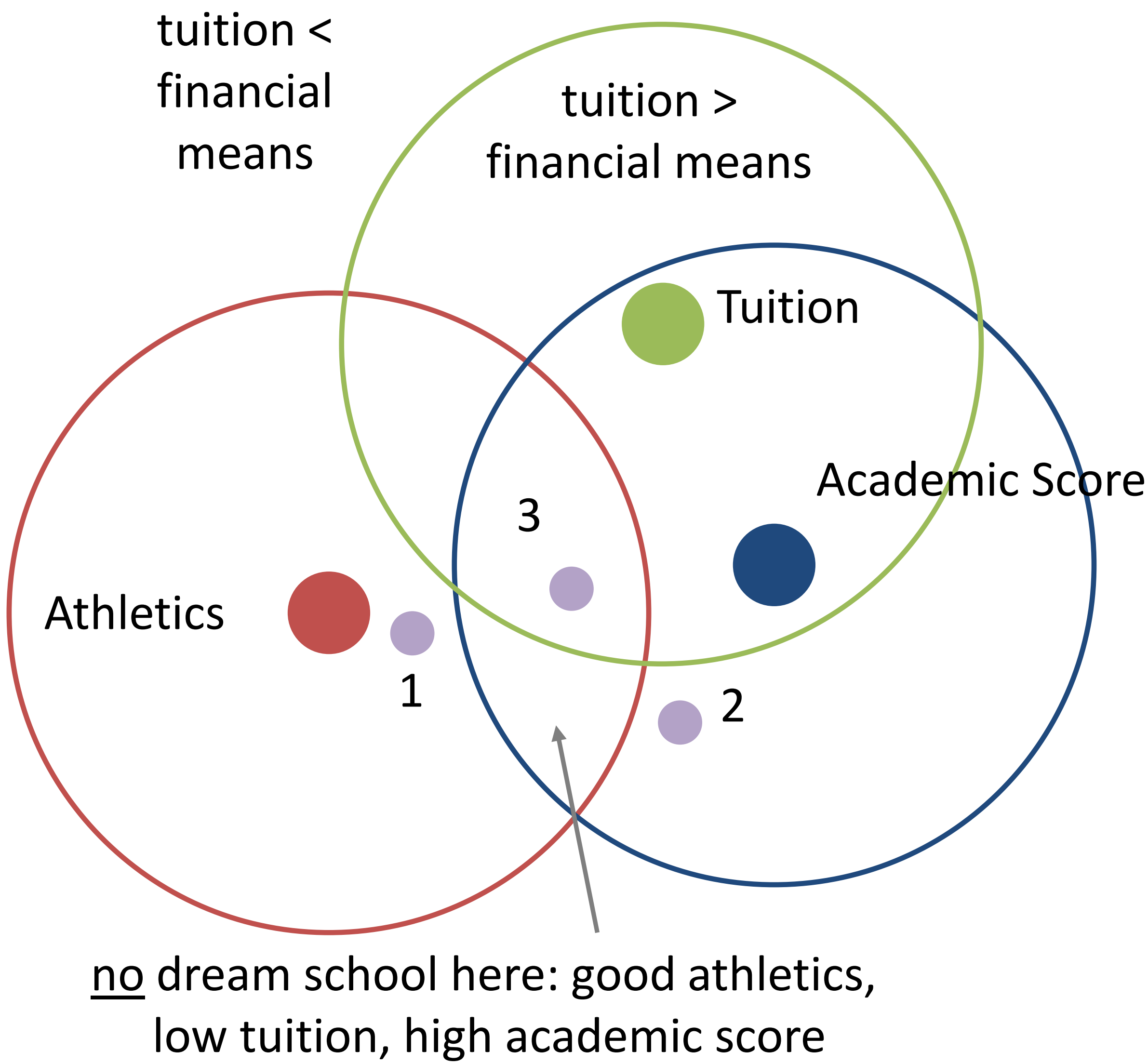
Combine Data and Attribute Layouts



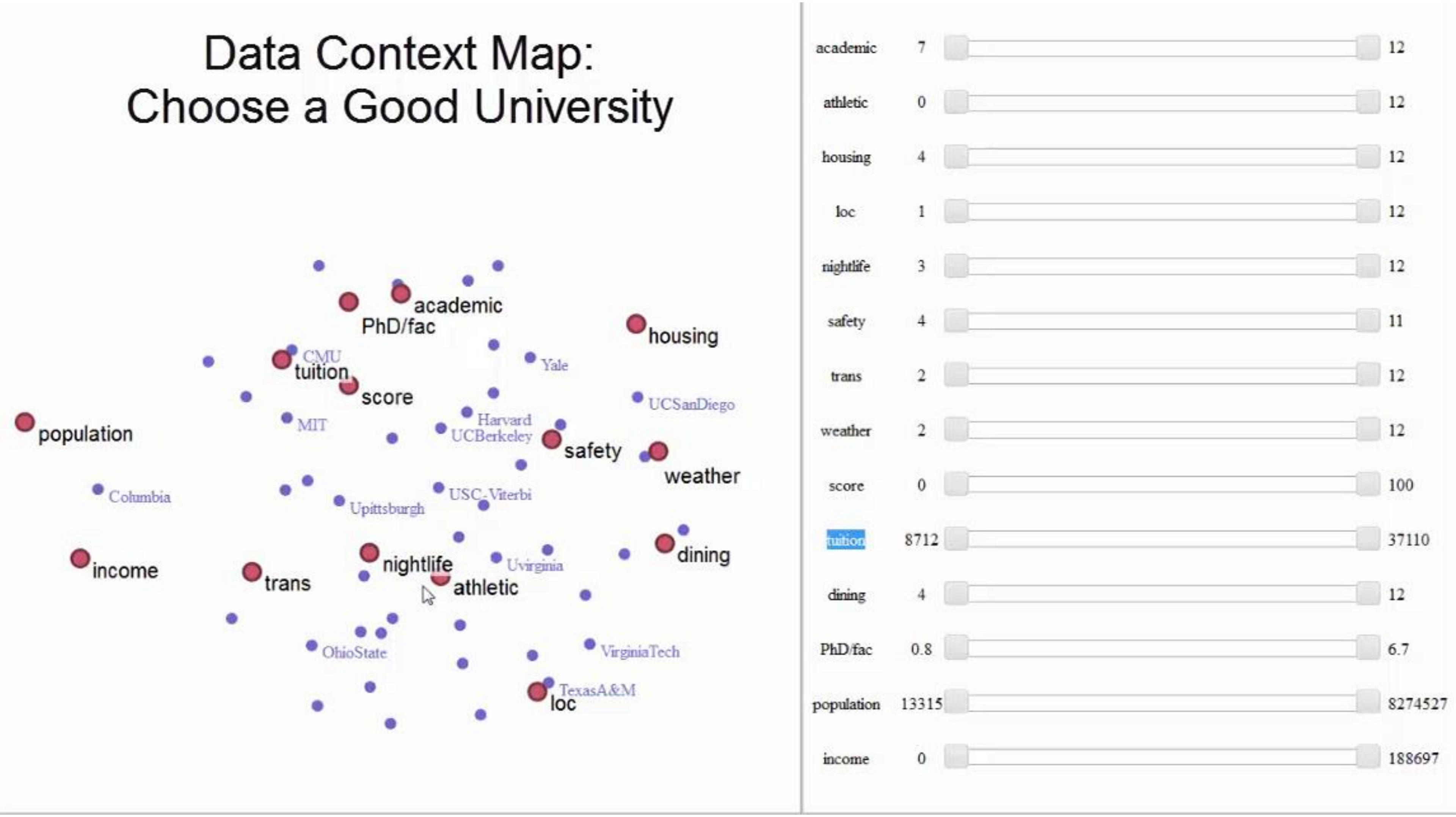
Achieved by Joint Matrix Optimization



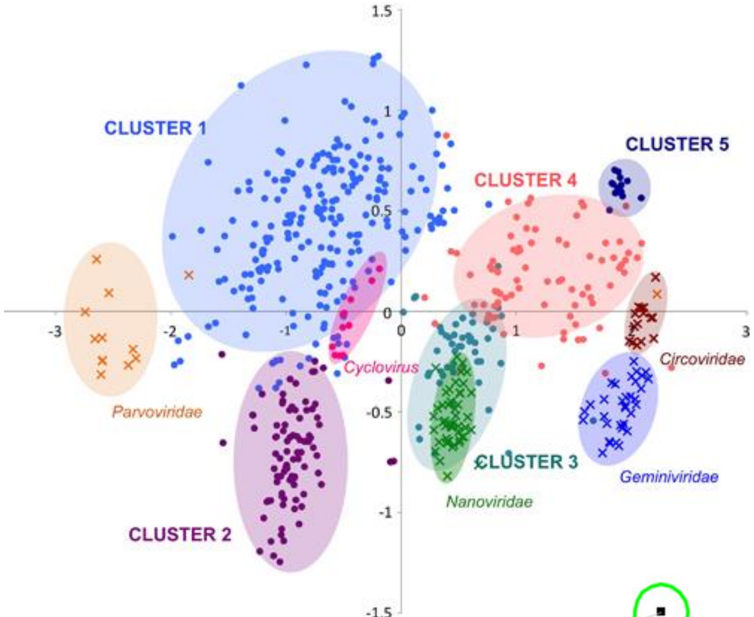
Example College Selection

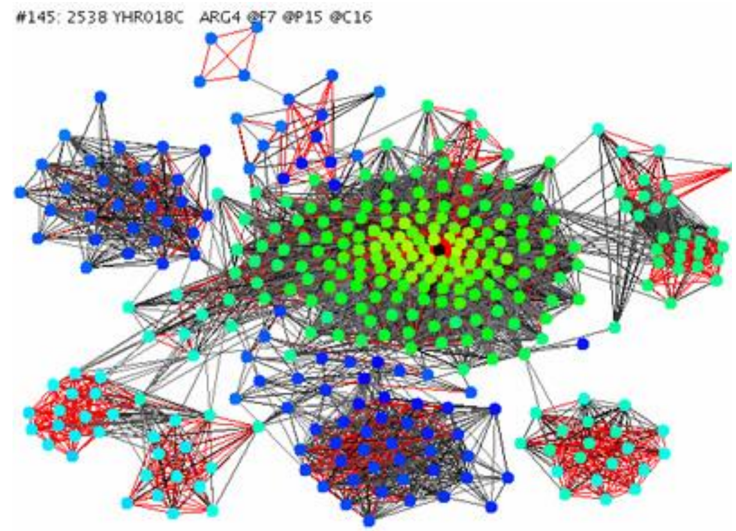


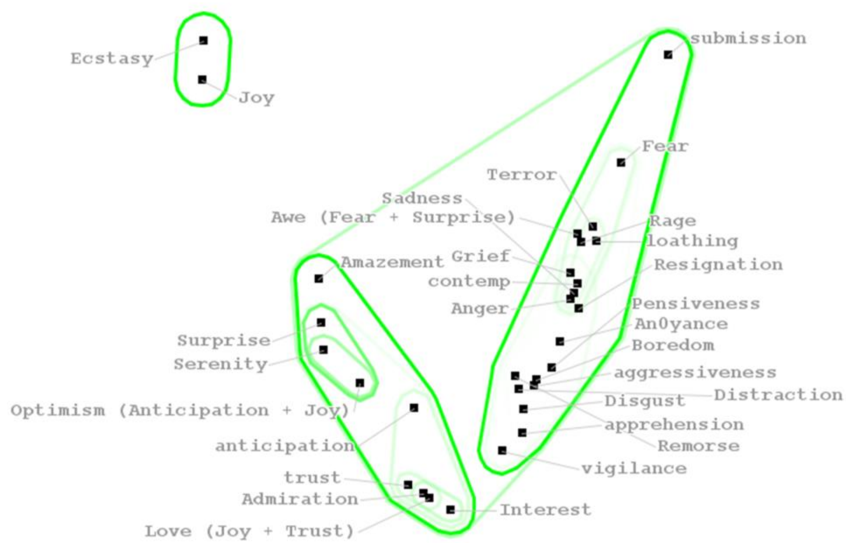
The Data Context Map



MDS Examples

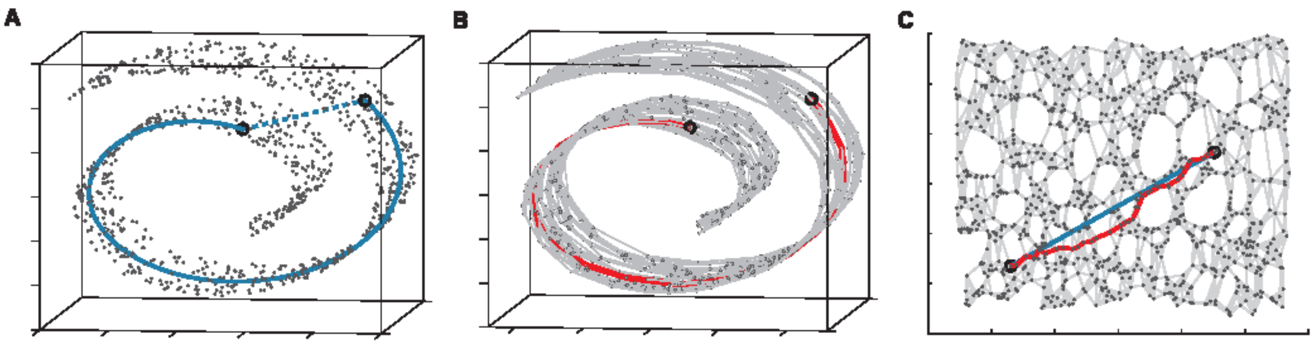






Manifold Learning: Isomap

by: J. Tenenbaum, V. de Silva, J. Langford, Science, 2000



Tries to unwrap a high-dimensional surface (A) → manifold

- noisy points could be averaged first and projected onto the manifold

Algorithm

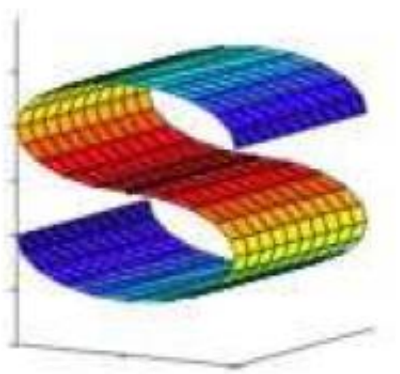
- construct neighborhood graph G → (B)
- for each pair of points in G compute the shortest path distances by adding small Euclidian hops (Floyd's, Dijkstra's algorithm) → geodesic distances
- fill similarity matrix with these geodesic distances
- embed (layout) in low-D (2D) with MDS → (C)

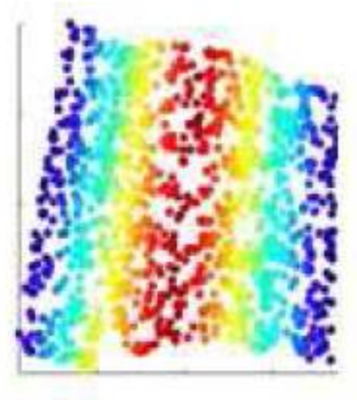
Manifold Learning: Locally Linear Embedding (LLE)

by: S. Roweis, L. Saul, Science, 2000

Based on simple geometric intuitions.

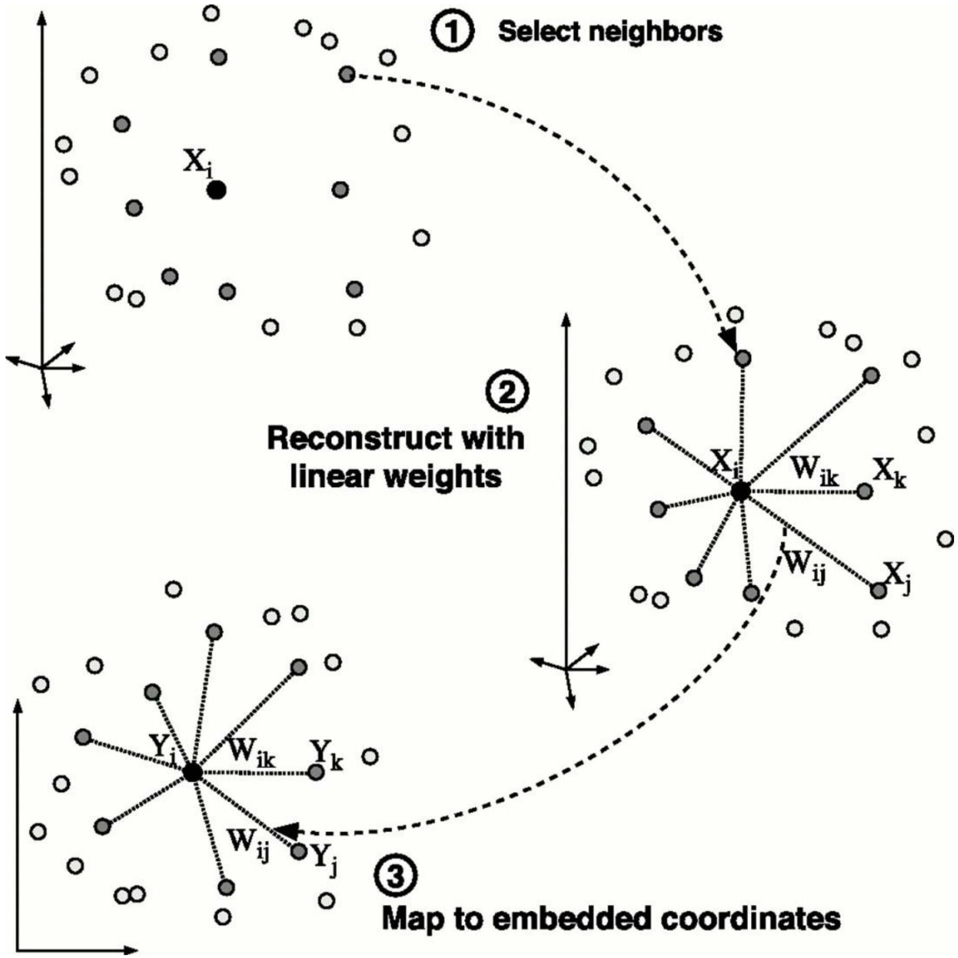
- suppose the data consist of N real-valued vectors X_i, each of dimensionality D
- each data point and its neighbors are expected to lie on or close to a locally linear patch of the manifold





High dimensional Manifold Low dimensional Manifold

LLE Overview



LLE Details

Steps:

- assign K neighbors to each data point $\vec{X}_i$
- compute the weights W_{ij} that best linearly reconstruct the data point from its K neighbors, solving the constrained least-squares problem

$$\varepsilon(W) = \sum_i \left| \vec{X}_i - \sum_j W_{ij} \vec{X}_j \right|^2 \qquad \sum_j \mathbf{W}_{ij} = 1$$

- compute the low-dimensional embedding vectors $\vec{Y}_i$ best reconstructed by W_{ij}

$$\Phi(Y) = \sum_i \left| \vec{Y}_i - \sum_j W_{ij} \vec{Y}_j \right|^2$$

Self-Organizing Maps (SOM)

Introduced by Teuvo Kohonen

- unsupervised learning and clustering algorithm
- has advantages compared to hierarchical clustering
- often realized as an artificial neural network

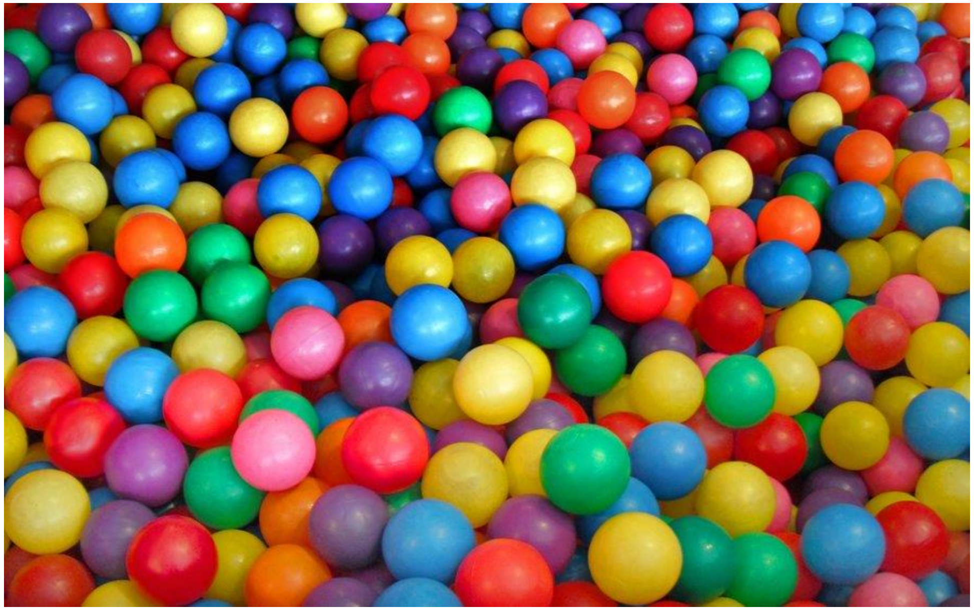
SOMs group the data

- perform a nonlinear projection from N-dimensional input space onto two-dimensional visualization space
- provide a useful topological arrangement of information objects in order to display clusters of similar objects in information space

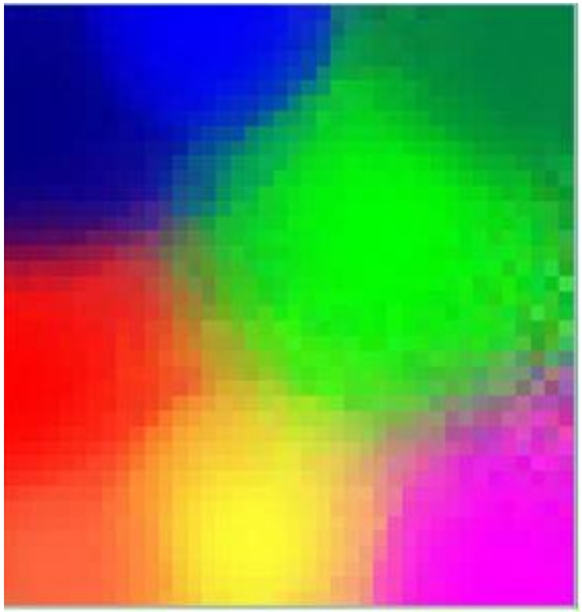
SOM Example

Map a dataset of 3D color vectors into a 2D plane

- assume you have an image with 5 colors
- want to see how many there are of each
- compute an SOM of the color vectors



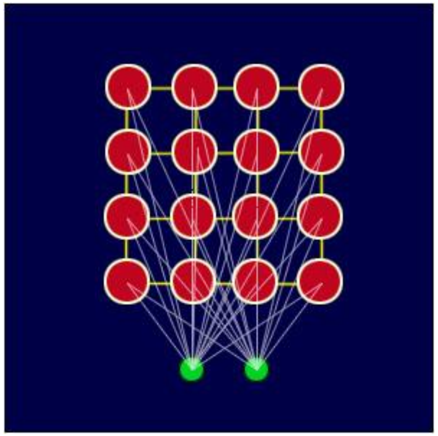
SOM



SOM Algorithm

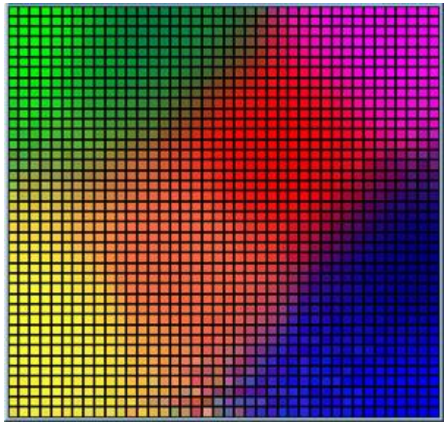
Create array and connect all elements to the N input vector dimensions

- shown here: 2D vector with 4×4 elements
- initialize weights



For each input vector chosen at random

- find node with weights most like the input vector
- call that node the Best Matching Unit (BMU)
- find nodes within neighborhood radius *r* of BMU
  - initially *r* is chosen as the radius of the lattice
  - diminishes at each time step
- adjust the weights of the neighboring nodes to make them more like the input vector
  - the closer a node is to the BMU, the more its weights get altered

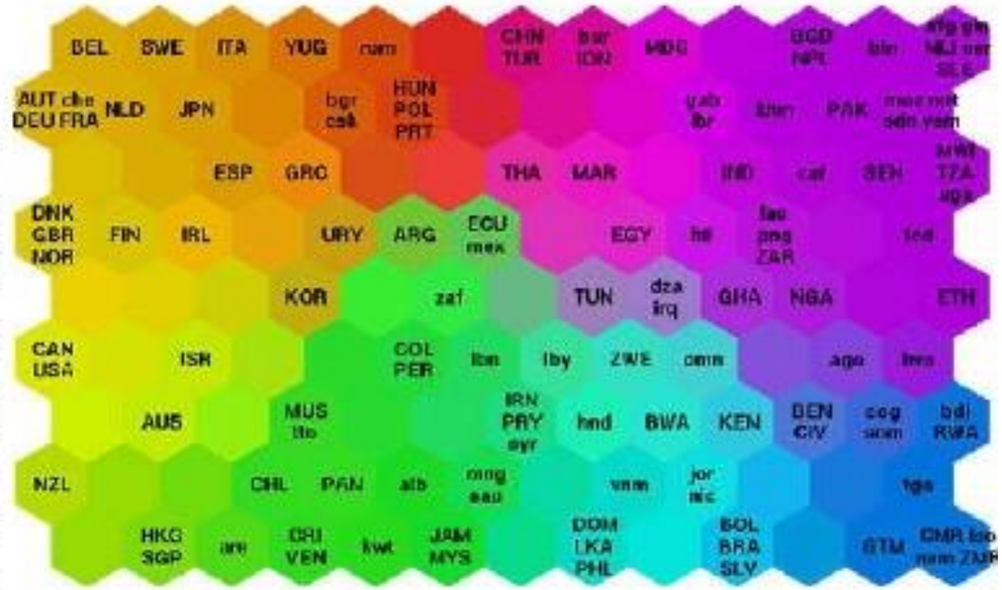


SOM Example: Poverty Map

SOM – Result Example

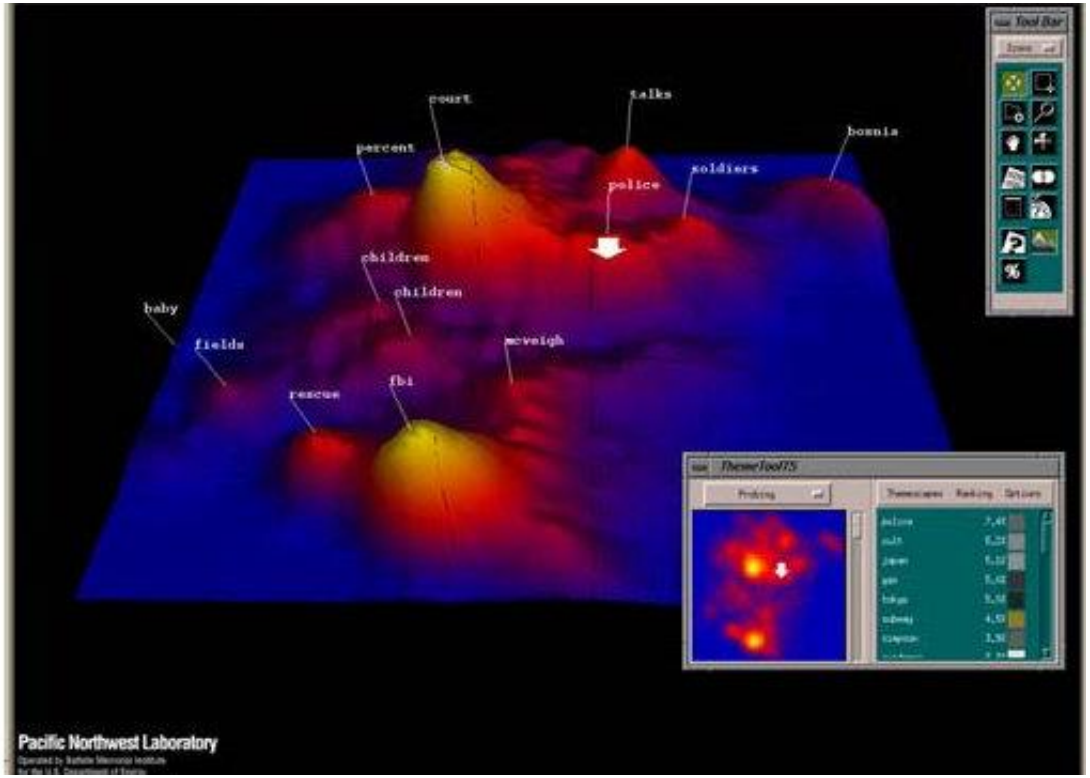
World Poverty Map

A SOM has been used to classify statistical data describing various quality-of-life factors such as state of health, nutrition, educational services etc. . **Countries with similar quality-of-life factors end up clustered together**. The countries with better quality-of-life are situated toward the upper left and the most poverty stricken countries are toward the lower right.



‘Poverty map’ based on 39 indicators from World Bank statistics (1992)

SOM Example: ThemeScape



Height represents density or number of documents in the region

Invented at Pacific Northwest National Lab (PNNL)

SOM / MDS Example: VxInsight (Sandia)

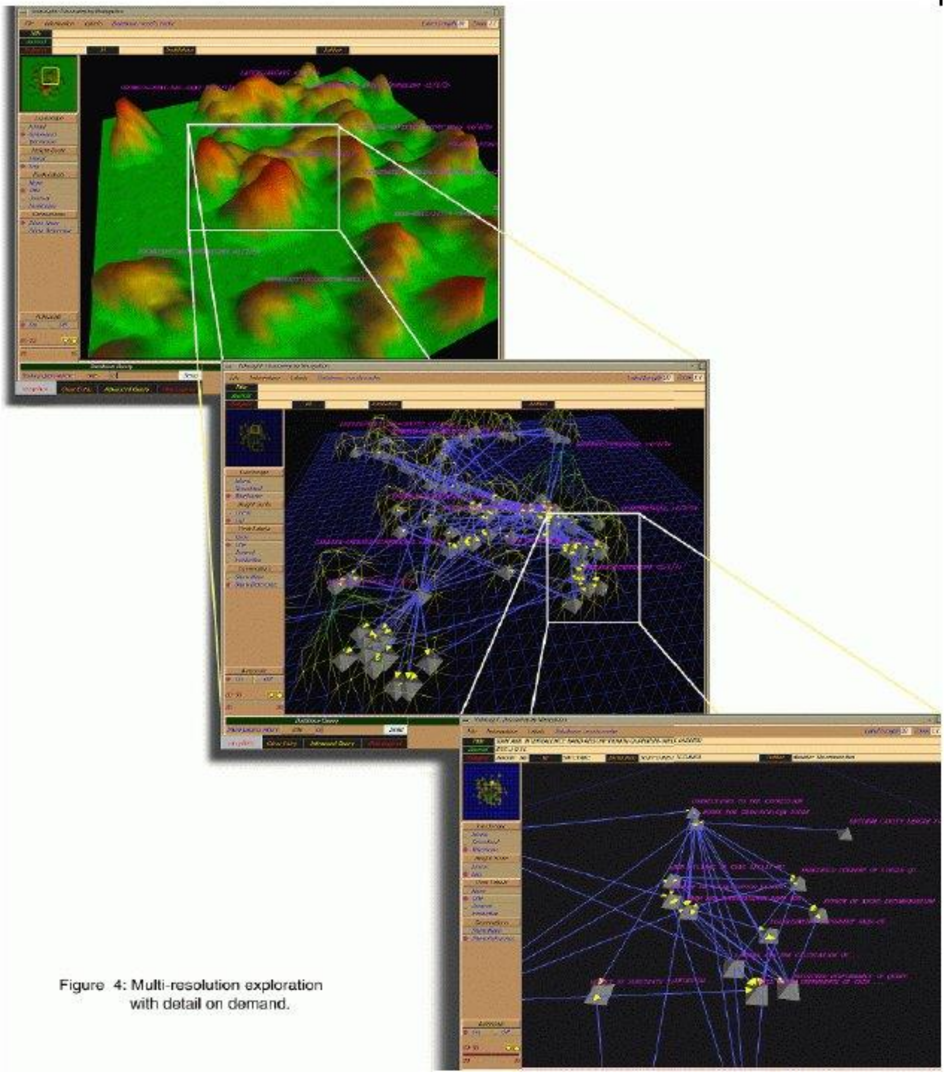


Figure 4: Multi-resolution exploration with detail on demand.

Practical Aspects

Coupling D3 with Python

See [this excellent page]for more detail

- uses MongoDB as a NoSQL database (non-relational SQL)

Step 1: Build a python server, say app.py

- use Flask as the web framework

```
from flask import Flask
from flask import render_template

app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")

if __name__ == "__main__":
    app.run(host='0.0.0.0',port=5000,debug=True)
```

Example 1:
Make an index.html file containing

```
<h1>Hello World!</h1>
```

Run the below from a terminal window

```
$ python app.py
```

Open a browser and go to http://localhost:5000/, you will see the message Hello World!.

Coupling D3 with Python

Step 2: Add all your processing code to app.py

- in this case it mainly involves storing data into the database

```
@app.route("/")
def index():
    return render_template("index.html")

@app.route("/donorschoose/projects")
def donorschoose_projects():
    connection = MongoClient(MONGODB_HOST, MONGODB_PORT)
    collection = connection[DBS_NAME][COLLECTION_NAME]
    projects = collection.find(projection=FIELDS)
    json_projects = []
    for project in projects:
        json_projects.append(project)
    json_projects = json.dumps(json_projects, default=json_util.default)
    connection.close()
    return json_projects

if __name__ == "__main__":
    app.run(host='0.0.0.0',port=5000,debug=True)
```

Example 2:
Start the server by running python app.py
Go to (in this example) http://localhost:5000/donorschoose/projects
You will see all the projects data printed in the browser.

Coupling D3 with Python

Step 3: Build the charts

- create a JavaScript file, say, charts.js
- gets the data from the python URL and other provided JSON files
- calls function, here makeGraphs(), to do the d3 rendering

```
queue()
    .defer(d3.json, "/donorschoose/projects")
    .defer(d3.json, "static/geojson/us-states.json")
    .await(makeGraphs);
```

wait for data read

```
function makeGraphs(error, projectsJson, statesJson) {
    ...
};
```

- check the webpage for more detail on how to build the charts

Coupling D3 with Python

Step 3: Build the charts

- ....
- call the renderAll() function for rendering all the charts

```
dc.renderAll();
```

- within index.html need to reference all the charts we defined in charts.js
- for example, if you want to show the US map chart, you will have to add the following line below to the index.html file.

```
<div id="us-chart"></div>
```

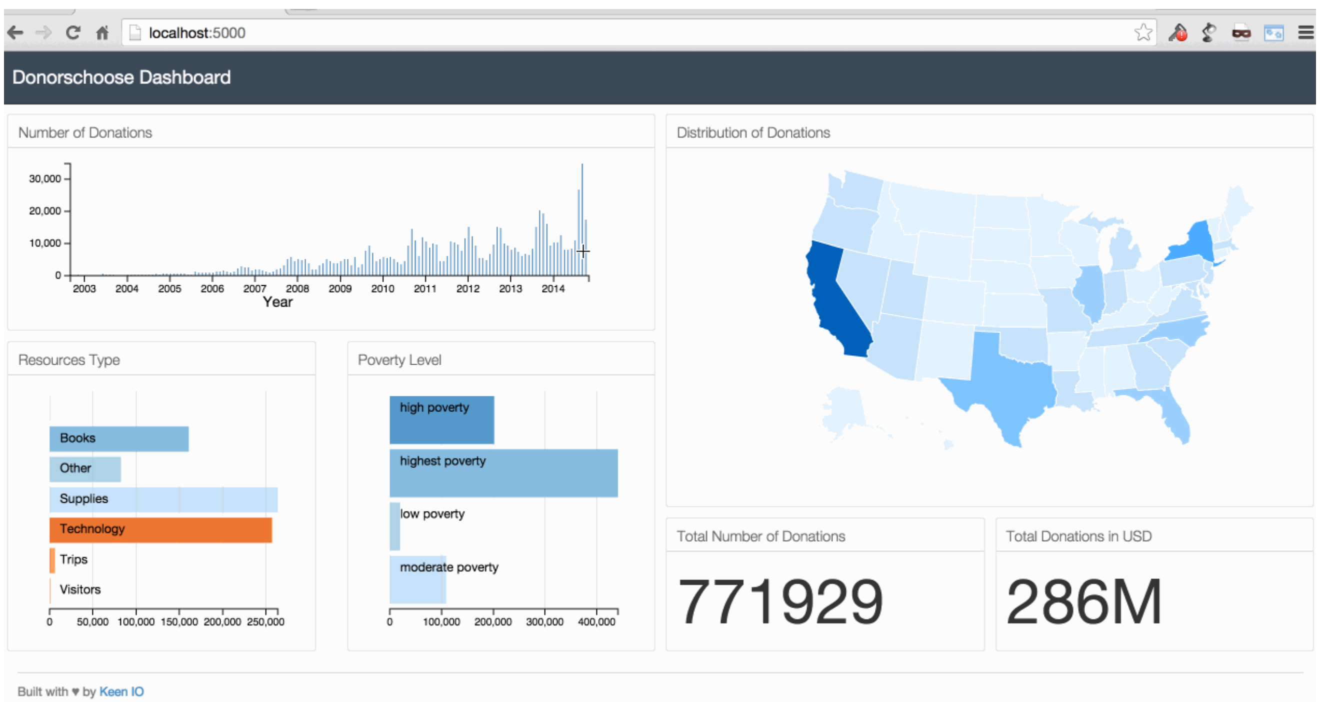
Coupling D3 with Python

Start app.py web server

Add query to the index.html file

Call http://localhost:5000/ in the browser to see the dashboard

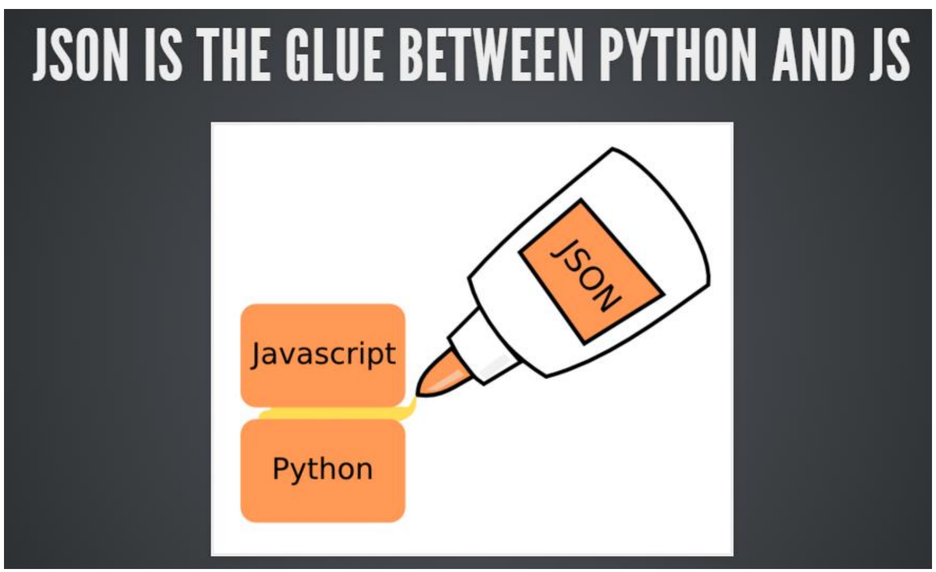
```
$ python app.py
```

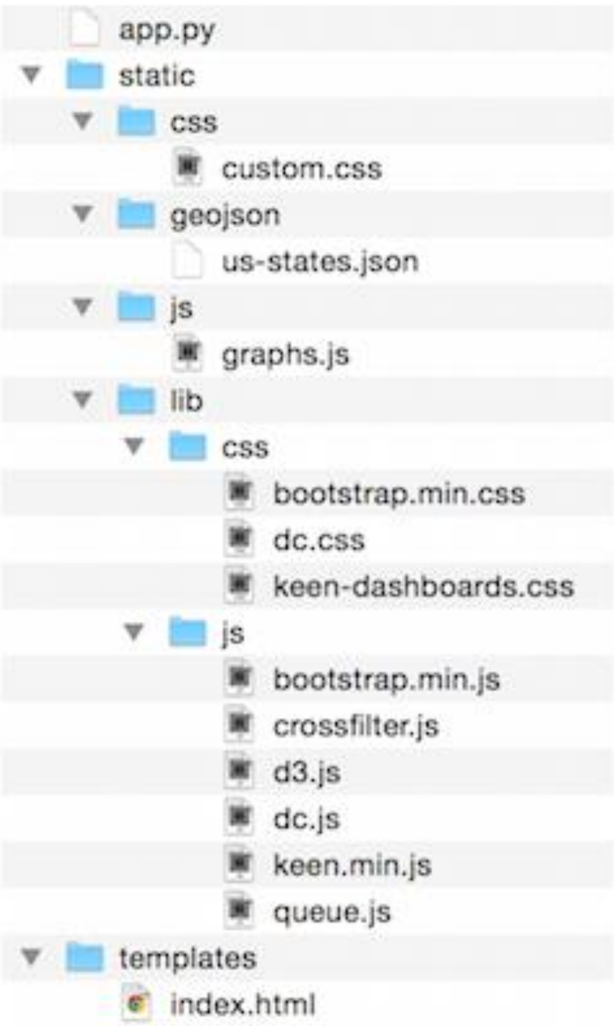


Folder Structure

All files are available in a dedicated [github repository]()

One more thing:





Some Useful pages

http://adilmoujahid.com/posts/2015/01/interactive-data-visualization-d3-dc-python-mongodb/

- csv data gets stored in MongoDB (4th most popular database)

http://kyrandale.com/static/talks/pydata-to-the-web/index.html#/

There are other pages … use google