Data Science Training

November 2017

Data Visualization

Xavier Bresson

Data Science and AI Research Centre
NTU, Singapore

http://data-science-optum17.tk

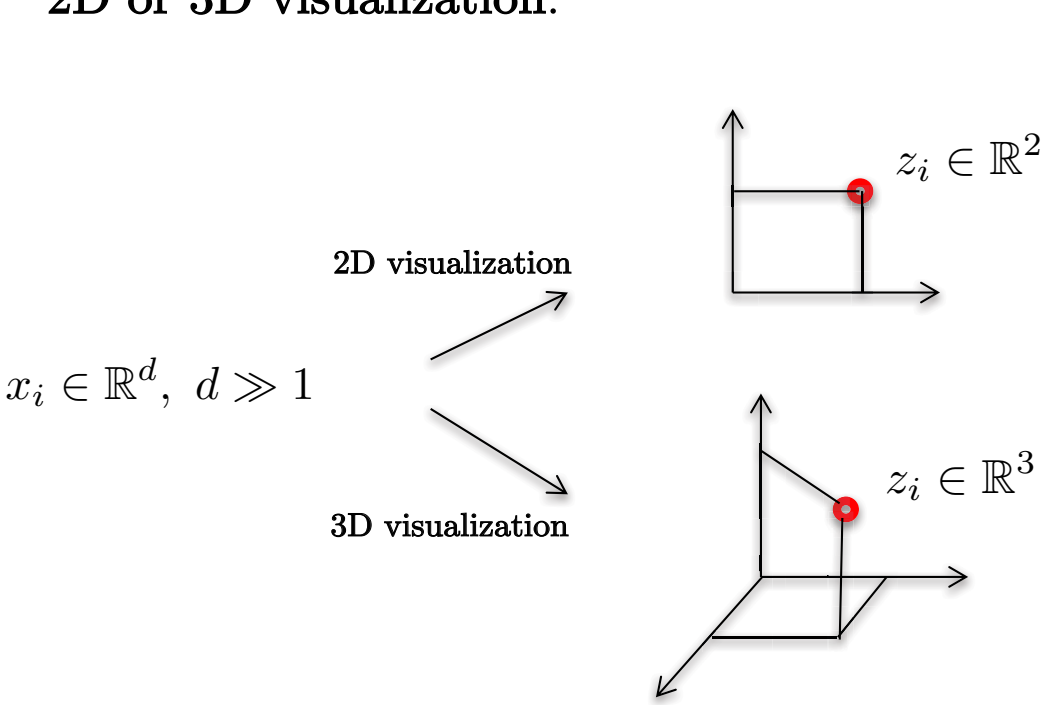
Outline

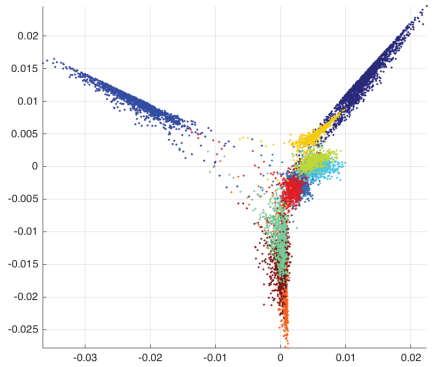
- **Visualization Problem**
- Kernel PCA
- Locally-Linear Embedding (LLE)
- Laplacian Eigenmaps
- T-SNE
- Conclusion

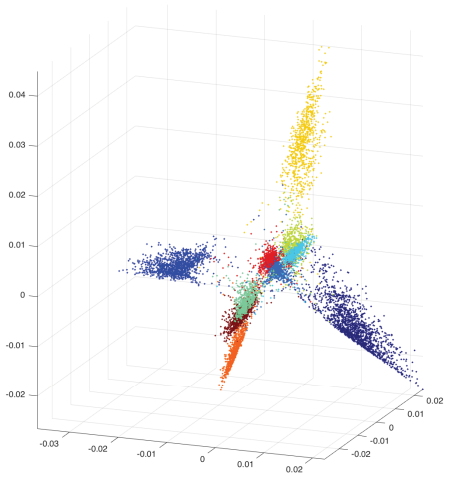
Visualization Problem

- Data **visualization** is the **same** problem as
  (1) Data representation
  (2) Feature extraction

- **Data representation** looks for the best **"dictionary" *D*** where the data x can be represented, and the **projected data *z* on *D* are used as coordinates for 2D or 3D visualization**:





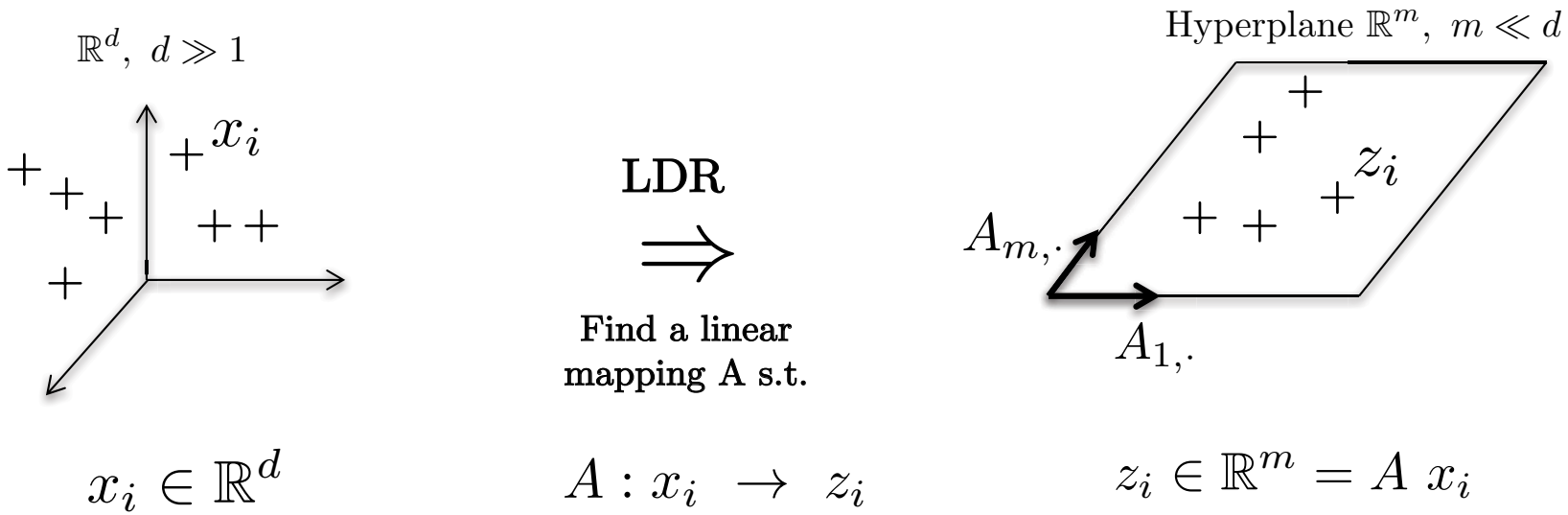


Visualization Techniques

- Visualization techniques are also **dimensionality reduction techniques** because they aim at mapping data into a much lower-dimensional space, 2D or 3D Euclidean spaces.

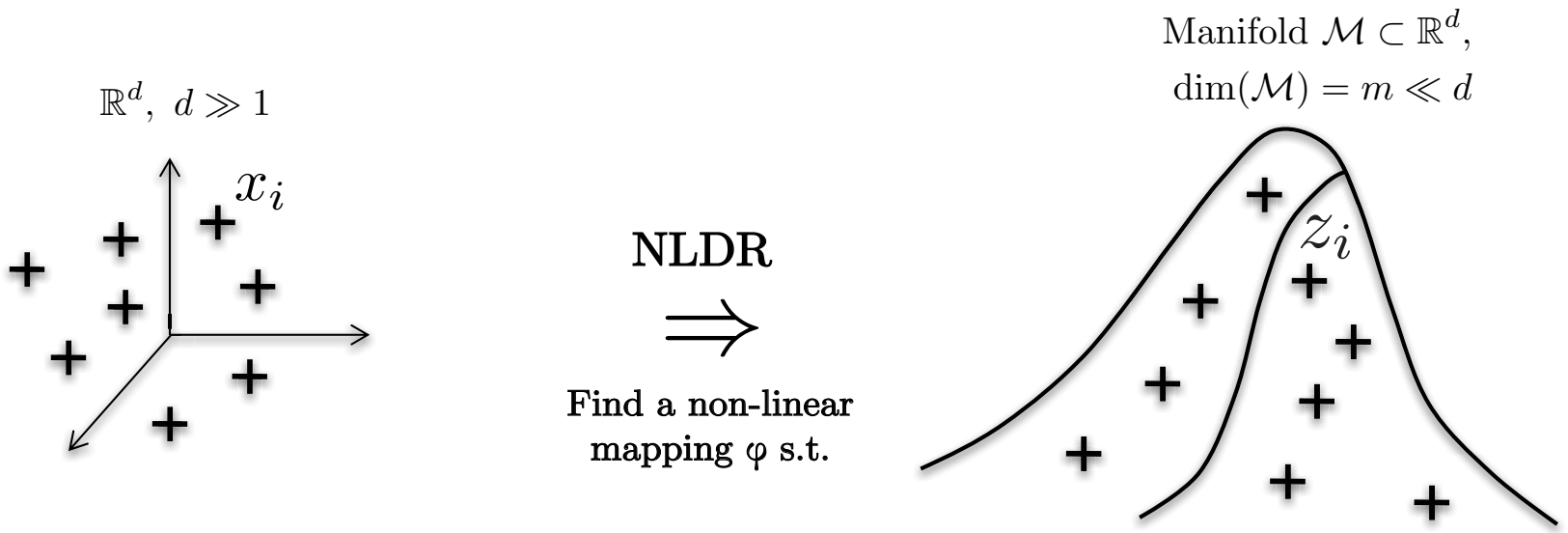
- **Linear dimensionality reduction (LDR) techniques.**

  Assumption: Data that can be represented on a **low-dimensional hyperplane**.



Non-Linear Dimensionality Reduction

- *Assumption:* Data that can be represented on **low-dimensional curved spaces, i.e. manifolds**:



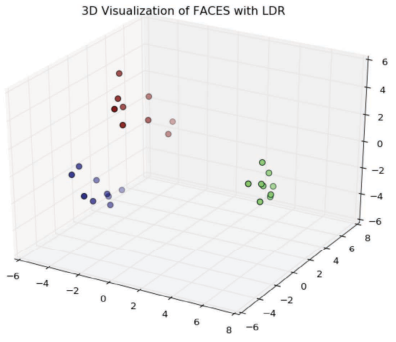
$$x_i \in \mathbb{R}^d \qquad \varphi : x_i \rightarrow z_i \qquad z_i \in \mathbb{R}^m = \varphi(x_i)$$

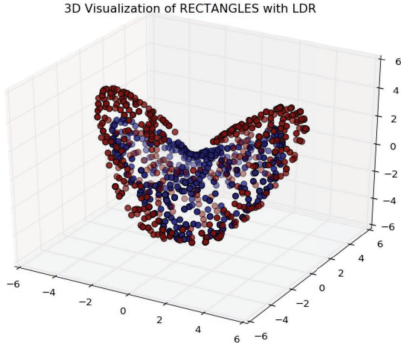
- Non-linear dimensionality reduction (NLDR) techniques are also called **manifold learning** techniques.

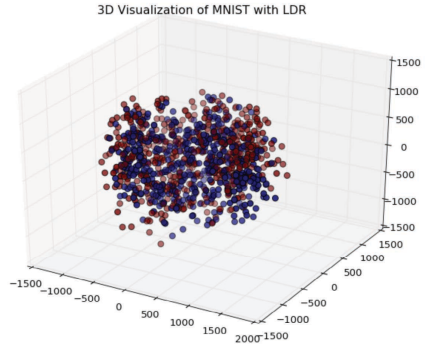
Demo: LDR vs. NLDR

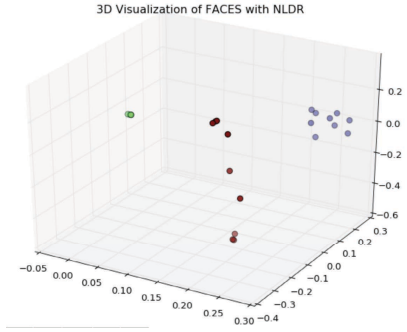
- Run **code01.ipynb**

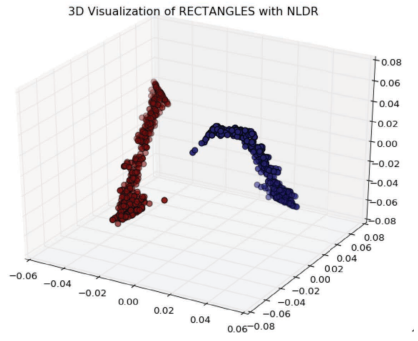
```
# Run LDR = Standard PCA
nb_pca = 3
[PC,PD,EnPD] = compute_pca(X,nb_pca)
Xldr = PC[:,0]
Yldr = PC[:,1]
Zldr = PC[:,2]

# Run NLDR = Laplacian Eigenmaps
W = construct_knn_graph(X,5,'euclidean')
Xnldr,Ynldr,Znldr = nldr_visualization(W)
```

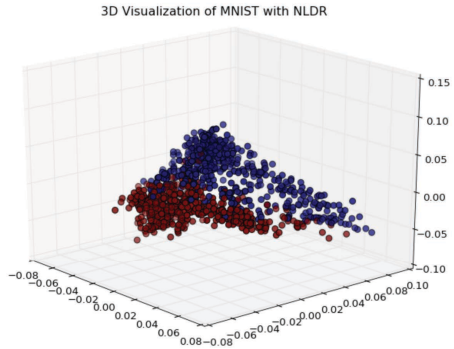












Outline

- Visualization Problem
- **Kernel PCA**
- Locally-Linear Embedding (LLE)
- Laplacian Eigenmaps
- T-SNE
- Conclusion

Kernel PCA [Scholkopf-Smola-Muller'97]

- **Standard PCA:**

$$\text{Gram matrix: } G = XX^T = \begin{bmatrix} \langle x_1, x_1 \rangle & \langle x_1, x_2 \rangle & \cdots & \\ \langle x_2, x_1 \rangle & \langle x_2, x_2 \rangle & & \\ \vdots & & \ddots & \\ & & & \langle x_n, x_n \rangle \end{bmatrix} \overset{EVD}{=} UDU^T \Rightarrow X_{pca} = UD^{1/2}$$

- **Kernel PCA:** Gram matrix in higher-dim space:

$$G = \begin{bmatrix} \langle \phi(x_1), \phi(x_1) \rangle & \langle \phi(x_1), \phi(x_2) \rangle & \cdots & \\ \langle \phi(x_2), \phi(x_1) \rangle & \langle \phi(x_2), \phi(x_2) \rangle & & \\ \vdots & & \ddots & \\ & & & \langle \phi(x_n), \phi(x_n) \rangle \end{bmatrix} \overset{EVD}{=} UDU^T \Rightarrow X_{kpca} = UD^{1/2}$$

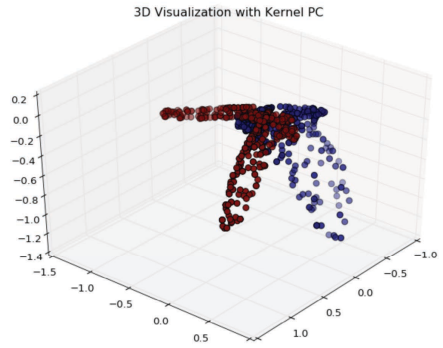
Apply the **kernel trick** to the Gram matrix:

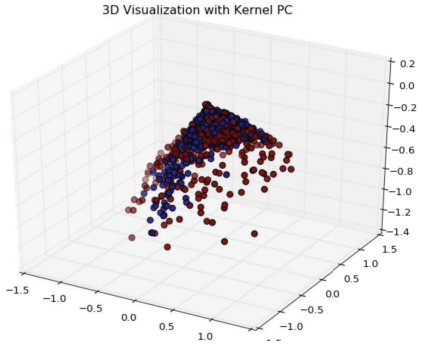
$$K(x, y) = \underbrace{\langle \phi(x), \phi(y) \rangle}_{\text{Never computed}} = e^{-\|x-y\|_2^2/\sigma^2}$$

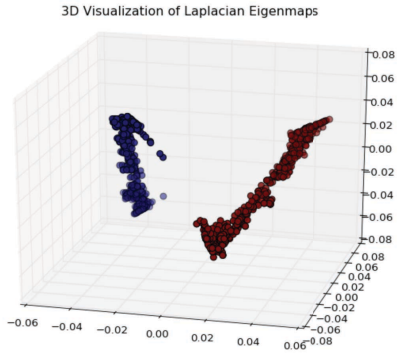
Demo: Kernel PCA

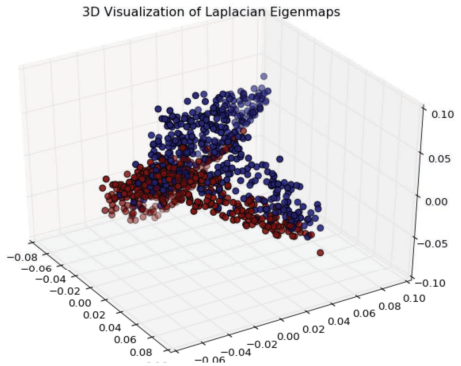
- Run **code02.ipynb**

```
# Run NLDR = Kernel PCA
Ker = construct_kernel(X,'polynomial',[2.,0.,3.])
nb_kernel_pca = 3
lamb, U = scipy.sparse.linalg.eigsh(Ker, k=nb_kernel_pca, which='LM')
D = np.diag(lamb)
kernel_PCs = U.dot( pow(D[:,:nb_kernel_pca], 0.5) )
Xnldr_kernelPCA = kernel_PCs[:,0]
Ynldr_kernelPCA = kernel_PCs[:,1]
Znldr_kernelPCA = kernel_PCs[:,2]
```







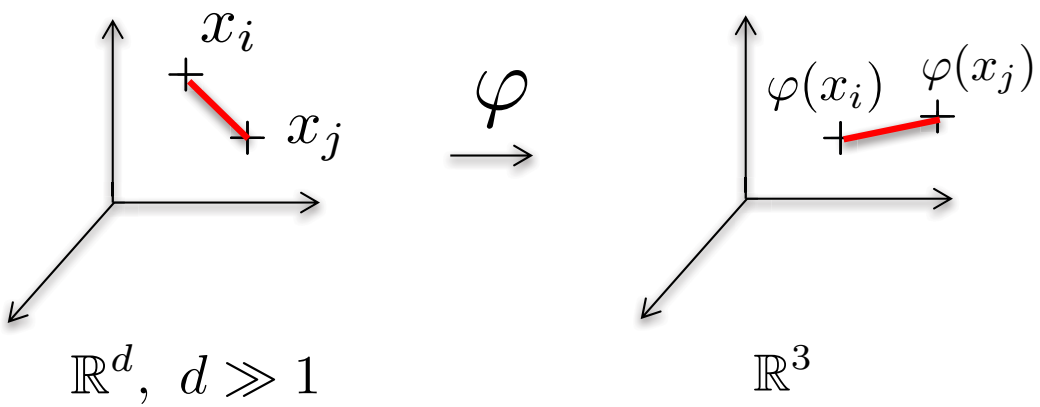


Outline

- Visualization Problem
- Kernel PCA
- **Locally-Linear Embedding (LLE)**
- Laplacian Eigenmaps
- T-SNE
- Conclusion

Locally-Linear Embedding (LLE) [Roweis, Saul'00]

- **Motivation:** Design a **mapping** from high-dimensional space to low-dimensional space such that the **geometric distances between neighbor data are preserved.**



- **Description:** LLE aims at computing a manifold M by locally linear fits, that is each data on M and its neighbors lie on a locally linear patch of M.

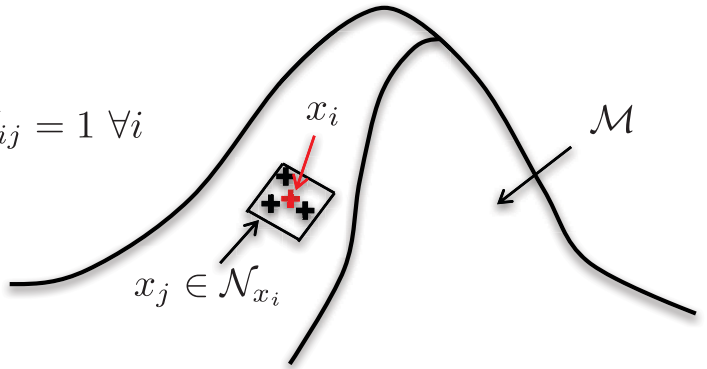
Algorithm

- **Step 1:** For each data x_i, compute the k nearest neighbors.

- **Step 2:** Compute linear patches: find the weights W_{ij} which best linearly reconstruct x_i from its neighbors:

$$\min_W \sum_{i=1}^{n} \left\| x_i - \sum_j W_{ij} x_j \right\|_2^2 \quad \text{s.t.} \quad \sum_j W_{ij} = 1 \ \forall i$$

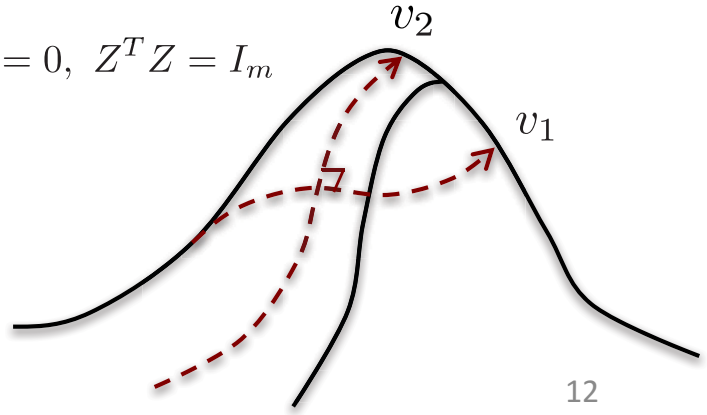
$\Rightarrow$ *Solution:* Ax=b



- **Step 3:** Compute the low-dimensional embedding data z_i, which is best reconstructed by the weights W_{ij}:

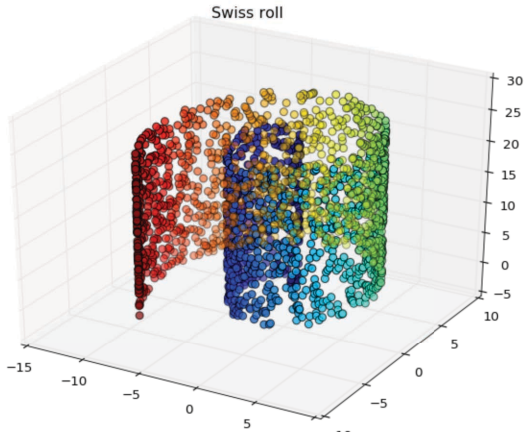
$$\min_{Z=[z_1,\ldots,z_m]} \sum_{i=1}^{n} \left\| z_i - \sum_j W_{ij} z_j \right\|_2^2 \quad \text{s.t.} \quad \sum_i z_i = 0, \ Z^T Z = I_m$$

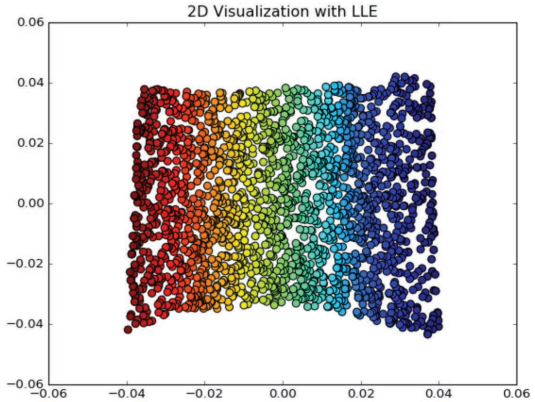
$\Rightarrow$ *Solution:* EVD

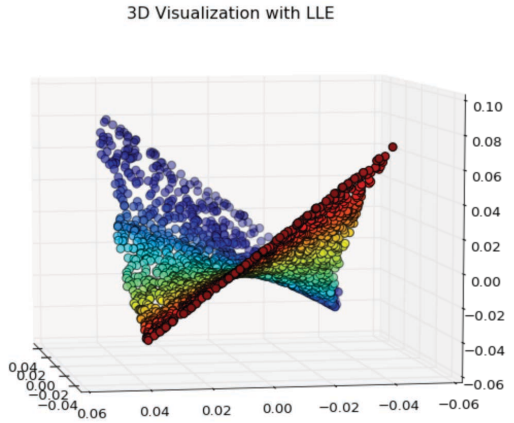


Demo: LLE

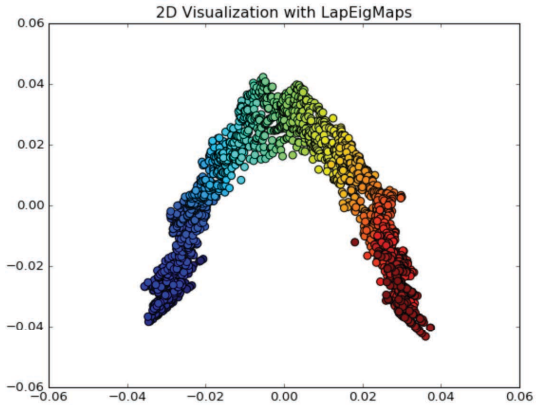
- Run **code03.ipynb**

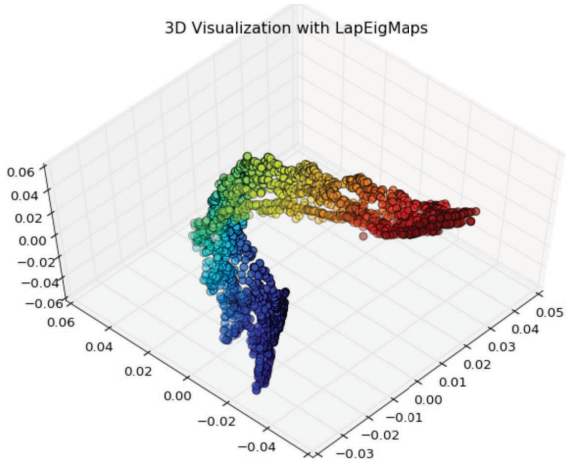







```
# Run NLDR = LLE
X = Xref
X = X - np.mean(X,axis=0) # zero-centered data

# Step 1: Compute k-NN
kNN = 20
WkNN = construct_knn_graph(X,kNN,'euclidean').todense()

# Step 2: Compute locally linear patches
W = np.zeros([n,n])
for i in range(n):
    # Find neighbors of data i
    idx_kNN = np.where(WkNN[i,:]>0.0)[1]
    K = len(idx_kNN)
    if K>kNN:
        K = kNN
        idx_kNN = idx_kNN[:K]
    XkNN = X[idx_kNN,:]
```





Outline

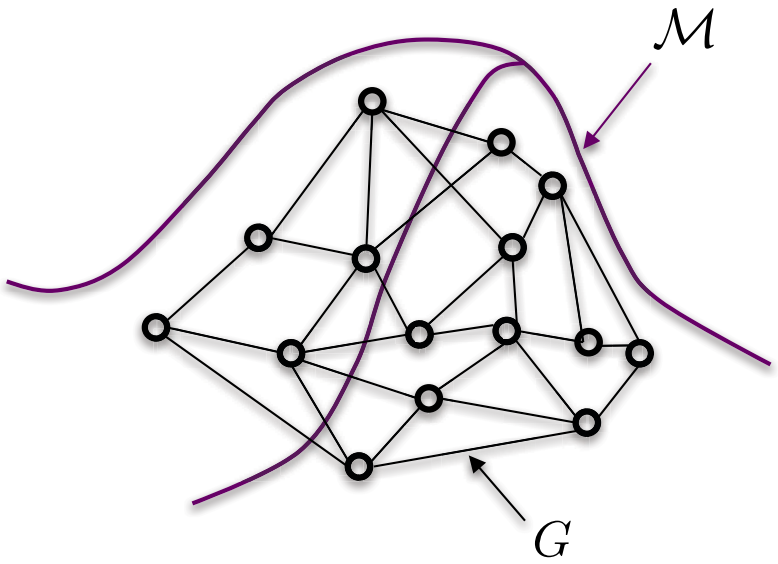
- Visualization Problem
- Kernel PCA
- Locally-Linear Embedding (LLE)
- **Laplacian Eigenmaps**
- T-SNE
- Conclusion

Laplacian Eigenmaps [Belkin, Niyogi'03]

- Very popular visualization technique.

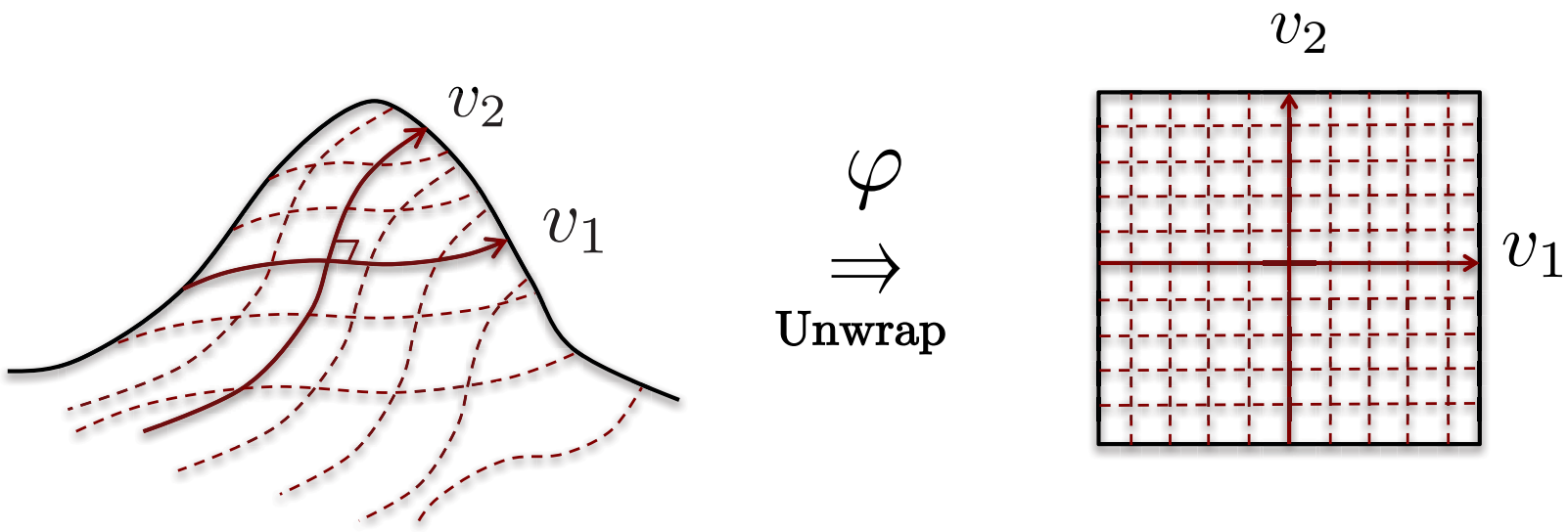
- **Motivation:** Same as LLE but stronger mathematical analysis and understanding.

- **Manifold assumption:** Data are sampled from a manifold M represented by a k - *nearest neighbors graph*.



Differential Geometry

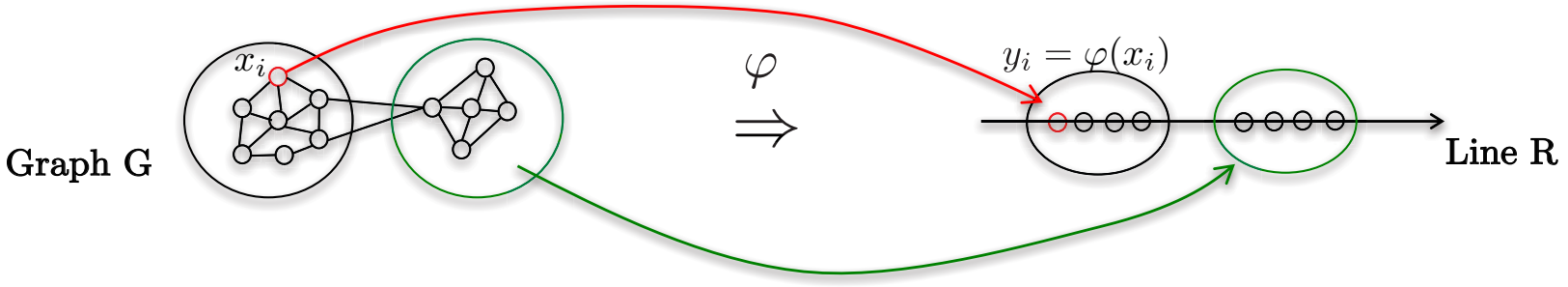
- Eigenfunctions v_k of continuous Laplace-Beltrami Δ_M serves as **embedding coordinates** of M:



- *Note:* **Discretization** of Δ_M = graph Laplacian L

Formalization

- **1D Visualization:** Map a graph $G=(V,E,W)$ to a line such that **neighbor data on G stay as close as possible** on the line.



- **Note:** We look for the mapping φ but we **never** compute it explicitely.

  We look for the coordinates y_i of x_i on the low-dimensional manifold M such that $y_i=\varphi(x_i)$, that is:

$$\min_y \sum_{ij} W_{ij}(y_i - y_j)^2 \quad (1)$$

- **Interpretation:** As $W_{ij}=1$ if x_i close to x_j, then $\min_y \Sigma\, W_{ij}(y_i-y_j)^2$ implies that y_i to be close to y_j.

Generalization to 2D and 3D

- **K-D Visualization:** Generalizing (1) to K dimensions is straightforward:

$$\min_{Y} \sum_{k} Y_{\cdot,k}^{T} L Y_{\cdot,k} = \mathrm{tr}(Y^{T} L Y) \quad \text{s.t.} \quad Y^{T} Y = I_{K}$$

Graph Laplacian

- **Spectral Solution:** Top K eigenvectors of graph Laplacian L.

$$L \overset{EVD}{=} U \Lambda U^{T} \;\rightarrow\; Y = U_{K}$$

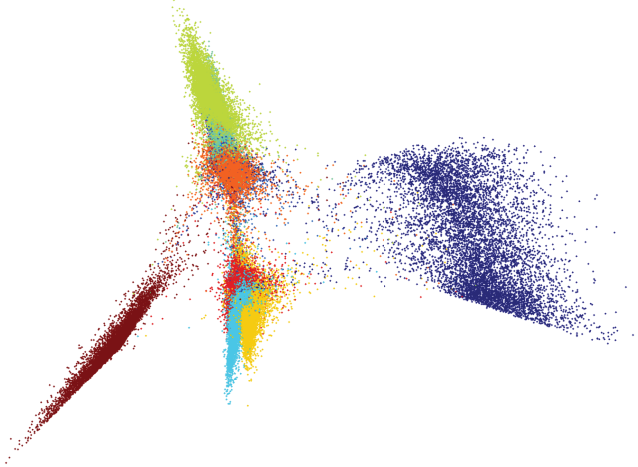
- **Advantages:**
  - *(1) Global solutions (independent of initialization)*
  - *(2) Fast algorithms (for small n)*

Demo: Laplacian Eigenmaps

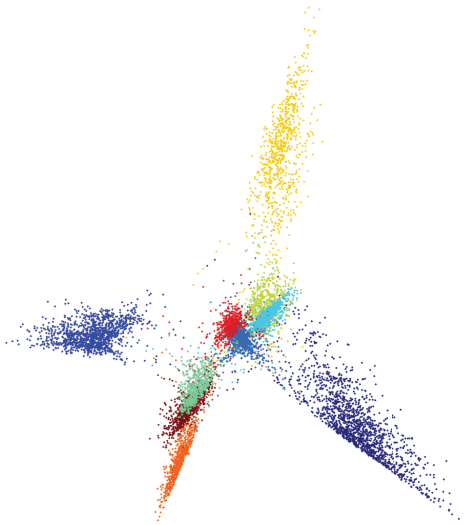
- Run **code04.ipynb**

```
mat = scipy.io.loadmat('datasets/MNIST.mat')
W = mat['W']
n = W.shape[0]
Cgt = mat['C'].squeeze()
print(n)

# Run NLDR = Laplacian Eigenmaps
Xnldr_lapeigmap,Ynldr_lapeigmap,Znldr_lapeigmap = nldr_visualization(W)
```

MNIST
PCA



MNIST
Lap Eigenmpas



USPS
Lap Eigenmpas

Outline

- Visualization Problem
- Kernel PCA
- Locally-Linear Embedding (LLE)
- Laplacian Eigenmaps
- **T-SNE**
- Conclusion

T-SNE [van der Maaten, Hinton'08]

Q: What visualization technique is the most used?

- t-Distribution Stochastic Neighbor Embedding (t-SNE) is the **most popular visualization** technique.

- **Model description:** t-SNE learns the mapping/embedding φ function, such that $y_i=\varphi(x_i)$, by **minimizing the Kullback-Leibler distance** between the distribution of high-dim data and the distribution of the computed low-dim data:

$$\begin{cases} p_{ij} = \dfrac{e^{-\|x_i - x_j\|_2^2/\sigma_i^2}}{\sum_k e^{-\|x_i - x_k\|_2^2/\sigma_i^2}} \\ \qquad\qquad \sigma_i = k^{th} \text{ nearest neighbor distance from } x_i \\ q_{ij}(y) = \dfrac{(1+\|y_i - y_j\|_2^2)^{-1}}{\sum_k (1+\|y_i - y_k\|_2^2)^{-1}} \end{cases}$$

Embedding coordinates of high-dim data

Optimizing Kullback-Leibler

- **Problem:**

$$\min_y \ \mathrm{KL}(P, Q(y)) = \sum_{ij} p_{ij} \log \frac{p_{ij}}{q_{ij}(y)}$$

$$\Downarrow \quad \textbf{Gradient descent technique}$$

$$y_i^{m+1} = y_i^m - 4\tau \sum_j (p_{ij} - q_{ij})(1 + \|x_i - x_j\|_2)^{-1}(y_i^m - y_j)$$

- **Advantages:**

  (1) Local distance preservation (as LapEig, LLE): minimizing KL forces q_{ij} to be close to p_{ij}, the distribution of high-dim data.

  (2) t-SNE does not assume the existence of a manifold: More **flexibility** to visualize more complex hidden structures.

- **Limitations:**

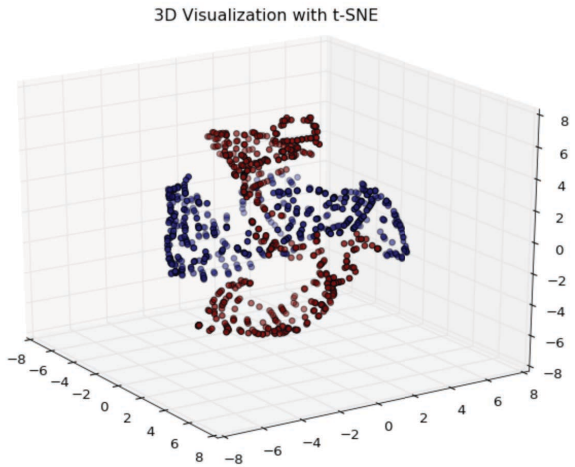
  (1) Non-convex energy $\Rightarrow$ Existence of **bad** local solutions, problem of initialization (PCA is used as initialization).

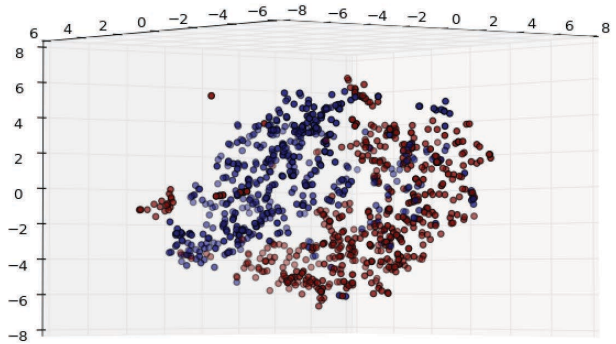
  (2) Slow optimization (gradient descent).

Demo: T-SNE

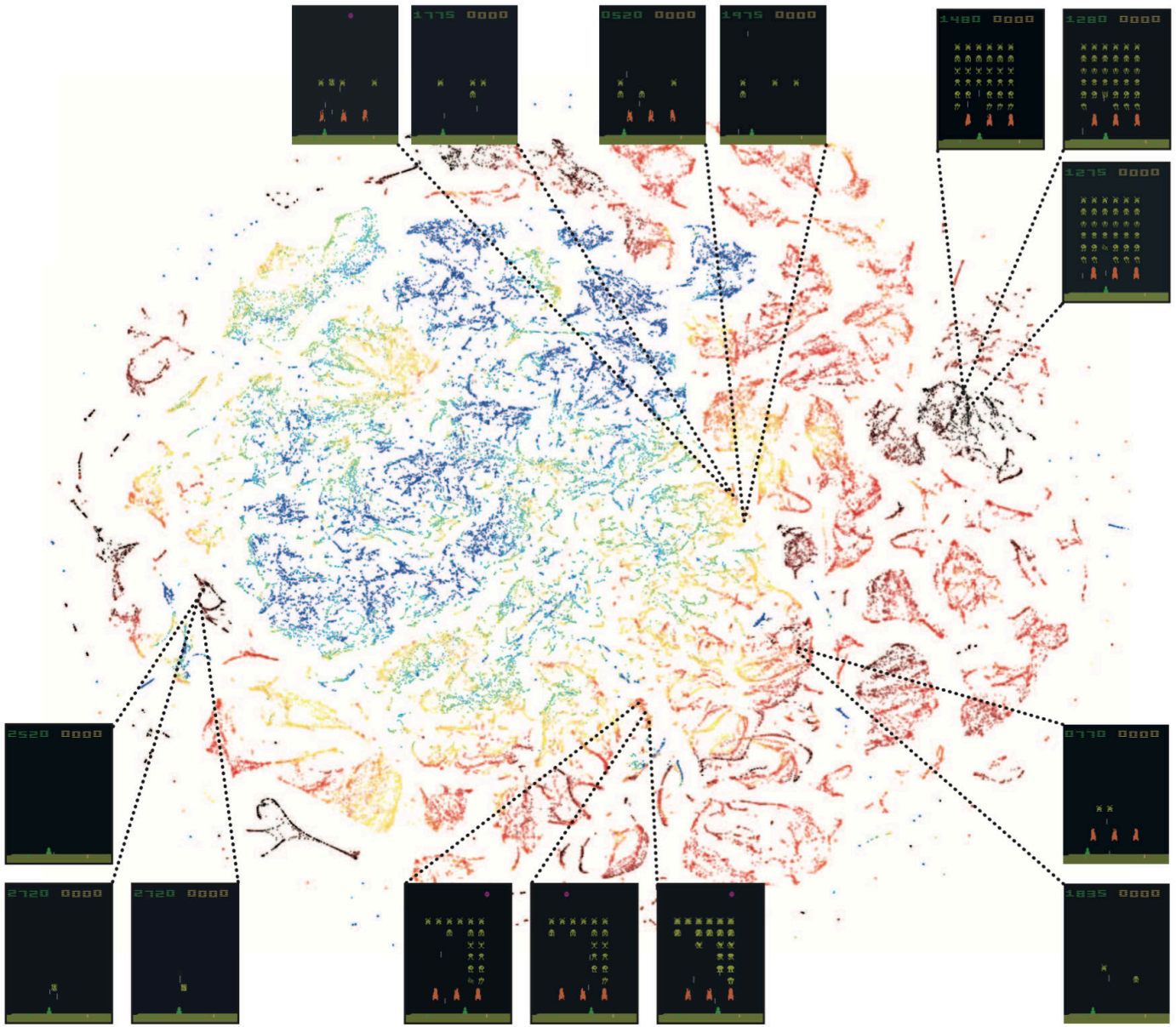
- Run **code05.ipynb**

```
# Run t-SNE
model = TSNE(n_components=3, n_iter=2000, learning_rate=1000)
np.set_printoptions(suppress=True)
XtSNE = model.fit_transform(X) # Xtsne = n x d
Xtsne = XtSNE[:,0]
Ytsne = XtSNE[:,1]
Ztsne = XtSNE[:,2]
```





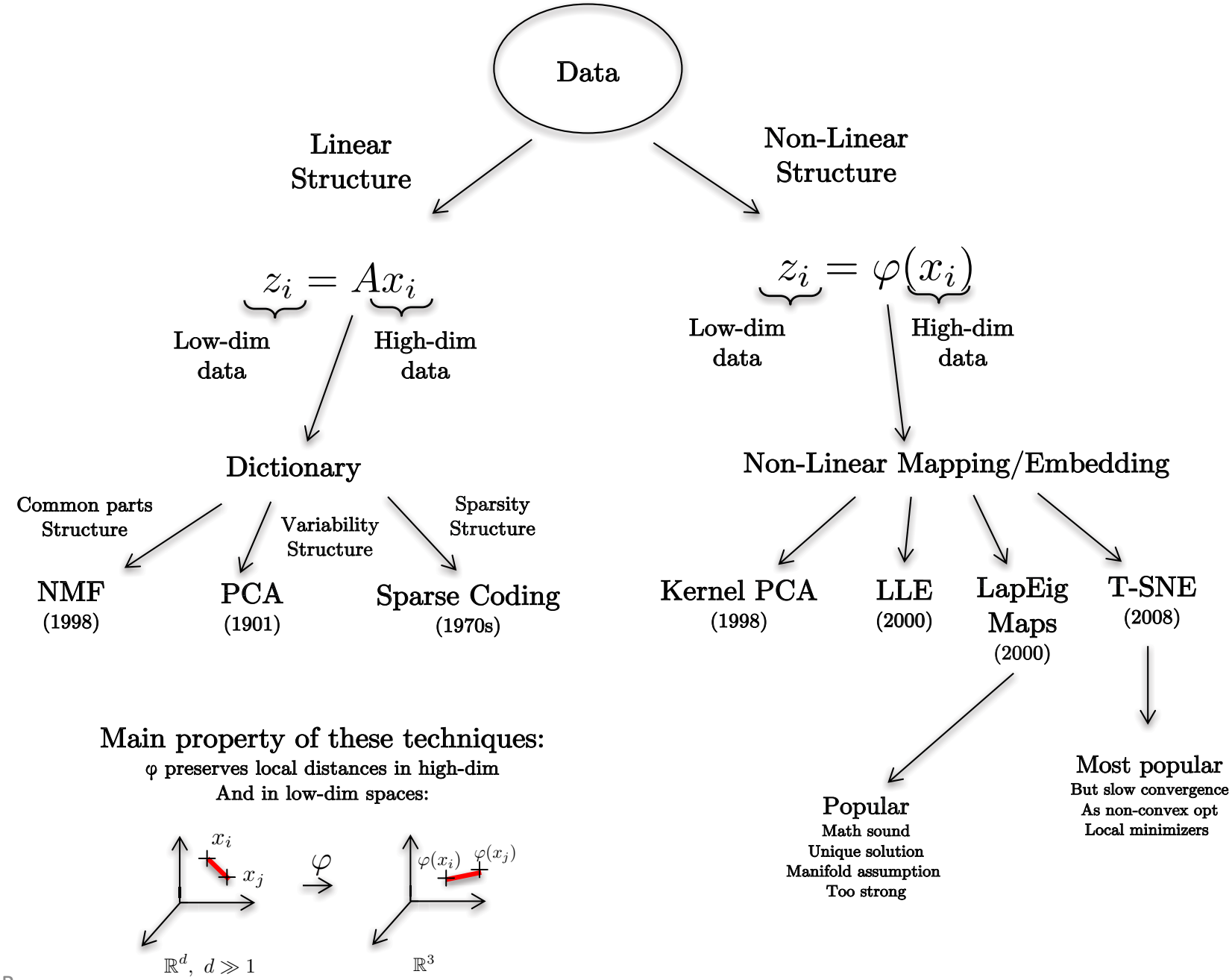
Visualizing Video Games [Mnih-et.al'14]



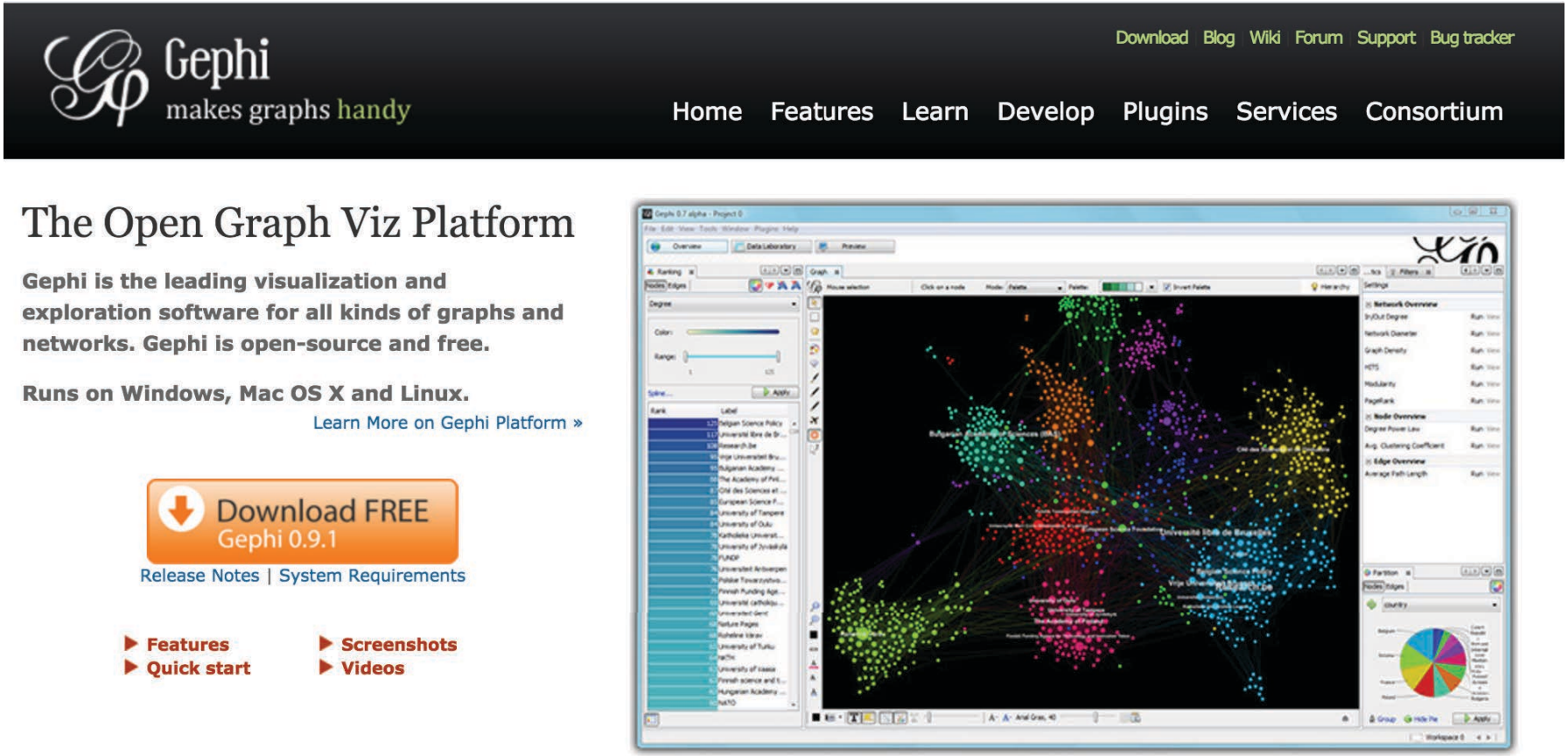
Outline

- Visualization Problem
- Kernel PCA
- Locally-Linear Embedding (LLE)
- Laplacian Eigenmaps
- T-SNE
- **Conclusion**

Summary


Data
Linear Structure
Non-Linear Structure
$z_i = Ax_i$
Low-dim data
High-dim data
$z_i = \varphi(x_i)$
Low-dim data
High-dim data
Dictionary
Non-Linear Mapping/Embedding
Common parts Structure
Variability Structure
Sparsity Structure
NMF (1998)
PCA (1901)
Sparse Coding (1970s)
Kernel PCA (1998)
LLE (2000)
LapEig Maps (2000)
T-SNE (2008)
Popular
Math sound
Unique solution
Manifold assumption
Too strong
Most popular
But slow convergence
As non-convex opt
Local minimizers
Main property of these techniques:
φ preserves local distances in high-dim
And in low-dim spaces:
x_i
x_j
φ
$\varphi(x_i)$
$\varphi(x_j)$
$\mathbb{R}^d,\ d \gg 1$
$\mathbb{R}^3$

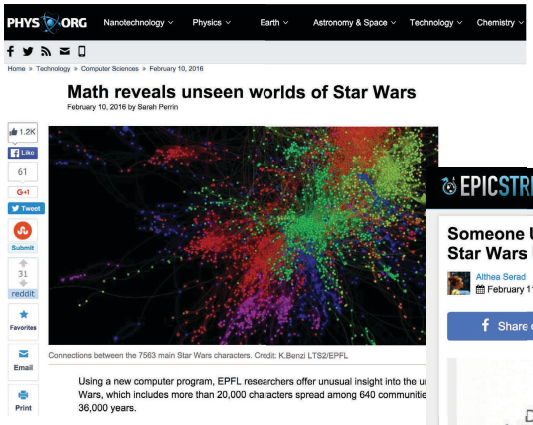
Gephi

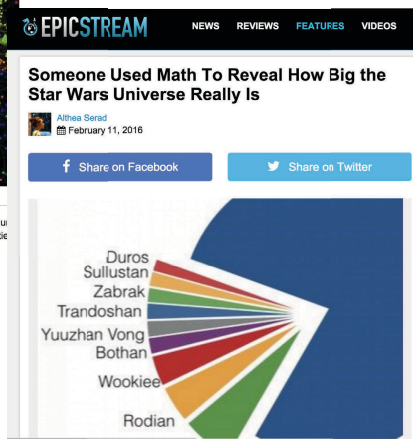


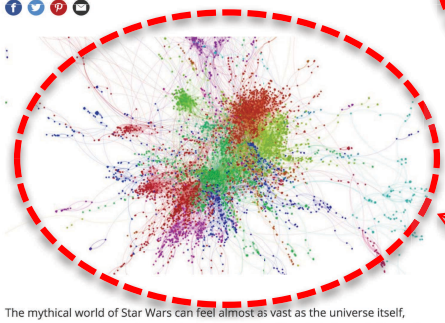
- Awesome visualization tool!
- Run **code06.ipynb** to convert graphs to Gephi format.

Mining Star Wars Universe

100+ international articles *including popular* ***Dailymail*** *(1.66M Twitter followers),* ***Gizmodo*** *(1.48M),* ***Engadget*** *(1.57M),* ***MUY Interesante*** *(7.1M).*







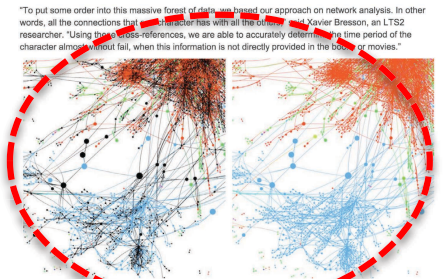
Here's the Star Wars universe like you've never seen it

By Parker Wilhelm a month ago World of tech

As fascinating as it is non-canonical

The mythical world of Star Wars can feel almost as vast as the universe itself, with characters and events spanning not just seven films, but also a plethora of books, comics and video games.

Now, thanks to data collected from the Ecole Polytechnique Fédérale de Lausanne (EPFL), the extended Star Wars universe is easier to take in for nerd and newcomer alike.



"To put some order into this massive forest of data, we based our approach on network analysis. In other words, all the connections that each character has with all the others," said Xavier Bresson, an LTS2 researcher. "Using these cross-references, we are able to accurately determine the time period of the character almost without fail, when this information is not directly provided in the books or movies."

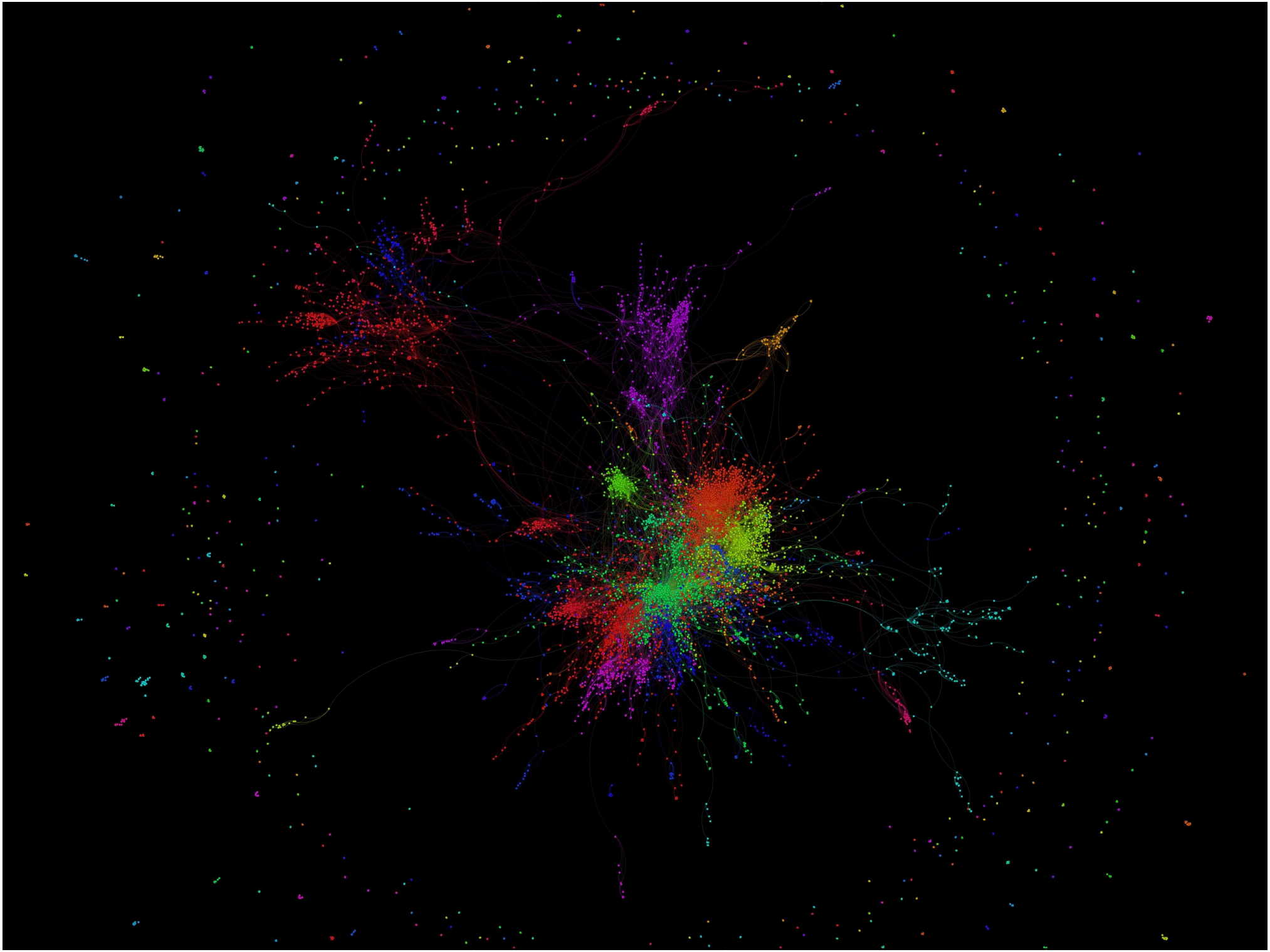
Part of the Star Wars character graph colored by era. Black nodes represent missing values. Red nodes: the Rise of the Empire Era (episodes 1,2,3). Blue nodes: the Rebellion era (episodes 4,5,6). Green ones: both eras.Figure 2: Result of the label propagation algorithm. Black nodes have been replaced by the best compromise using their neighbors.

Mapping out connections

"The researchers want to use this study to demonstrate the program's ability to extract and analyze digital data. "The program maps out connections in the mass of unorganized data available on the net," said Benzi.

Given a huge amount of information, the algorithms developed by the LTS2 researchers offer a service that cannot be matched by human beings. In addition to extracting data according to extremely precise criteria, the algorithms can also create links among data points, sort them, quantify them, interpret them and find missing information. All this in very little time. The results are then presented in the form of interactive charts

With Gephi!



Questions?