CS6208 : Advanced Topics in Artificial Intelligence
Graph Machine Learning

Lecture 3: Graph Analysis Techniques without Feature Learning

Semester 2 2022/23

Xavier Bresson

https://twitter.com/xbresson

Department of Computer Science
National University of Singapore (NUS)

Outline

- Graph clustering
- Classification with graphs
- Recommendation with graphs
- Dimensionality reduction with graphs

No feature learning !

Outline

- Graph clustering
- Classification with graphs
- Recommendation with graphs
- Dimensionality reduction with graphs

No feature learning !

Outline

- Graph clustering
  - Definition
  - Linear K-Means
  - Kernel K-Means
  - Balanced Cuts
  - NCut, PCut
  - Louvain Algorithm

No feature learning !

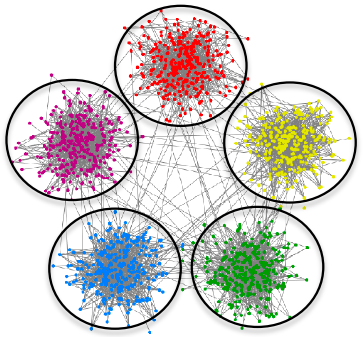
Outline

- Graph clustering
  - Definition
  - Linear K-Means
  - Kernel K-Means
  - Balanced Cuts
  - NCut, PCut
  - Louvain Algorithm

No feature learning !

Graph clustering

- Graph clustering consists in partitioning a graph into meaningful clusters.
  - Essential for large-scale graph computation s.a. Google PageRank w/ billions of nodes.
  - Exploration tool to find patterns in data.
- We consider the task of unsupervised graph clustering.
  - Unsupervised means that no data label is used (no prior information).
- What is the most popular unsupervised data clustering algorithm?



Source: Abbe, JMLR'17

Outline

- Graph clustering
  - Definition
  - Linear K-Means
  - Kernel K-Means
  - Balanced Cuts
  - NCut, PCut
  - Louvain Algorithm

No feature learning !

K-means[1]

- Most popular clustering algorithm

- Three types of k-means techniques
  (1) Standard/linear k-means
  (2) Kernel/non-linear k-means – Expectation-Maximization (EM) approach
  (3) Kernel/non-linear k-means – Spectral approach

[1] MacQueen, Some Methods for classification and Analysis of Multivariate Observations, 1967

Standard/linear k-means[1,2]

- Description : Given n data x_i in R^d, k-means technique partitions the data into K clusters $S_1,\ldots,S_K$ that minimize the least-squares objective:

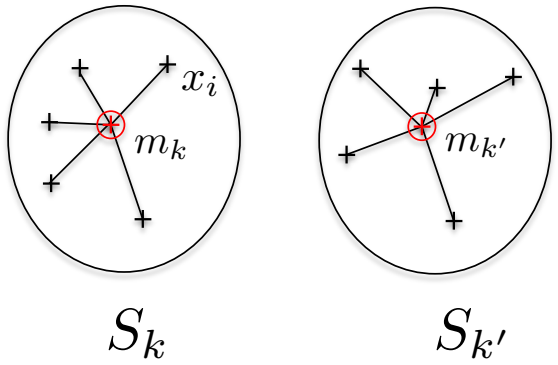
$$E(M,S) = \sum_{k=1}^{K} \sum_{x_i \in S_k} \overbrace{\|x_i - m_k\|_2^2}^{\text{Distance between } x_i \text{ and its mean } m_k}$$

m_k: k^{th} mean

S_k: k^{th} cluster

Means: $M = \{m_1, ..., m_K\}$

Clusters: $S = \{S_1, ..., S_K\}$



[1] Steinhaus, Sur la division des corps matériels en parties, 1957
[2] MacQueen, Some Methods for classification and Analysis of Multivariate Observations, 1967

Algorithm[1,2]

- Expectation-Maximization (EM) approach

  Initialization: Random initial means

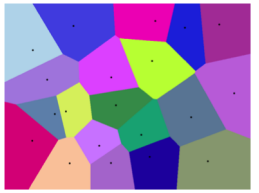
  Iterate until convergence: l=0,1,...

  (1) Cluster update (expectation step) :

$$S_k^{l+1} = \{x_i : \|x_i - m_k^l\|_2^2 \leq \|x_i - m_{k'}^l\|_2^2, \ \forall k' \neq k\}$$

  (2) Mean update (maximization step) :

$$m_k^{l+1} = \frac{\sum_{x_i \in S_k^{l+1}} x_i}{|S_k^{l+1}|}$$



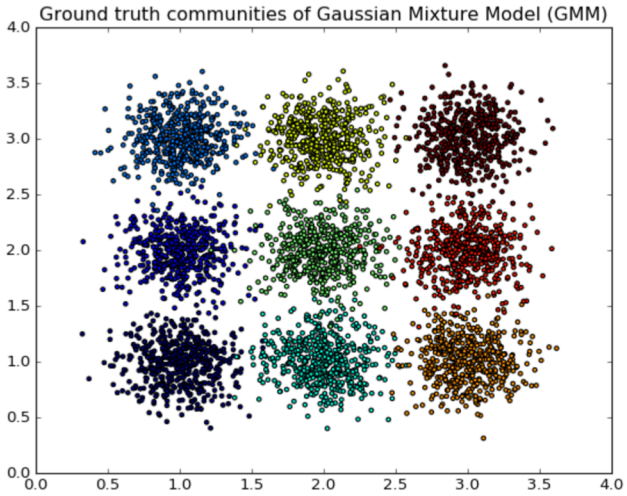
Voronoi cells

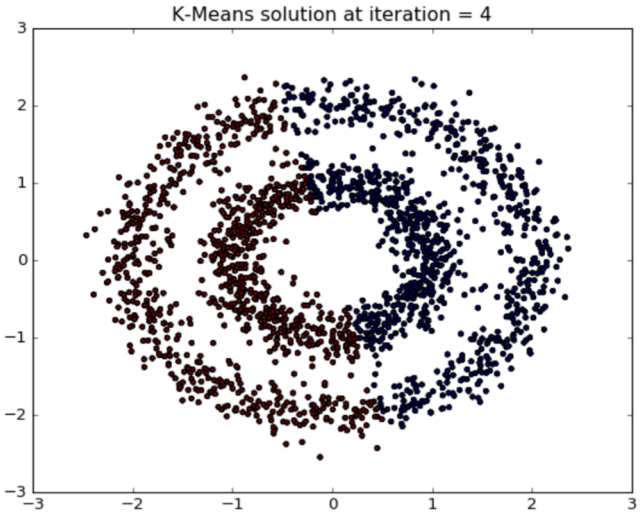
[1] Lloyd, Least square quantization in PCM, 1957
[2] Forgy, Cluster analysis of multivariate data: efficiency versus interpretability of classifications, 1965

Lab 1: Standard k-means

- Run 01_graph_clustering/code01.ipynb

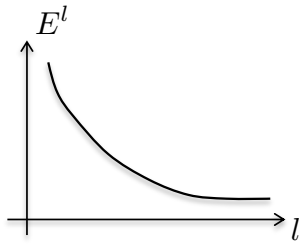
```
In [4]: plt.figure(1)
        size_vertex_plot = 10
        plt.scatter(X[:,0], X[:,1], s=size_vertex_plot*np.ones(n), c=Cgt)
        plt.title('Ground truth communities of Gaussian Mixture Model (GMM)')
        plt.show()
```





Properties of EM Algorithm

- Advantages

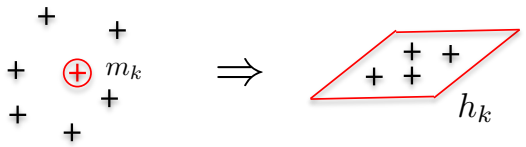


(1) Monotonic : $E^{l+1} \leq E^l$ for all iterations

(2) Convergence is guaranteed

(3) Speed/complexity: O(n d K #iter)

(4) Easy to implement



(5) Several extensions : K-Medians, K-hyperplans, other distances

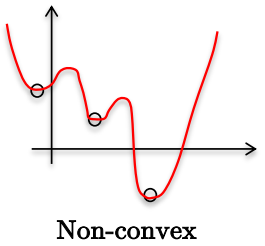
(6) K-means has relationships with other popular algorithms (PCA, NMF)

- Limitations

(1) Non-convex energy (NP-hard)

⇒ Existence of local minimizers, saddle points

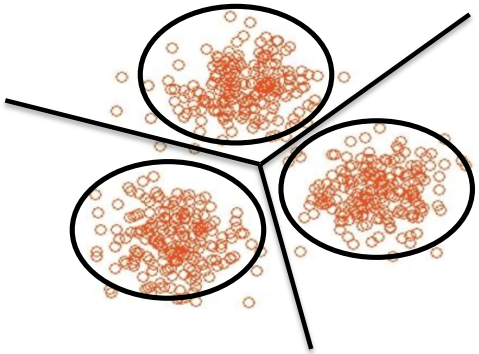
⇒ Good initialization is critical. Standard trick is to restart several times and pick the solution with the lowest energy value.



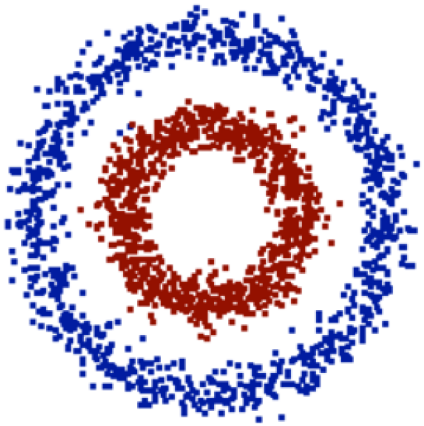
(2) Data is supposed to be linearly separable.

Main limitation

- Assumption : Standard k-means suppose data follow a Gaussian Mixture Model (GMM), meaning that clusters are linearly separable and spherical.



- Consequence: Standard k-means does not work for non-linear separable data.



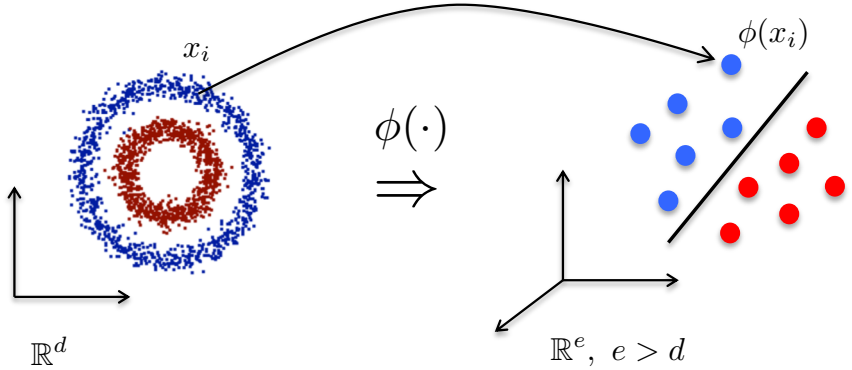
Outline

- Graph clustering
  - Definition
  - Linear K-Means
  - Kernel K-Means
  - Balanced Cuts
  - NCut, PCut
  - Louvain Algorithm

No feature learning !

Kernel k-means[1]

- Kernel trick : Map data to a higher-dimensional space where data can be linearly separated.



- New loss : Weighted kernel k-means energy

$$E(M, S) = \sum_{k=1}^{K} \sum_{x_i \in S_k} \theta_i \|\phi(x_i) - m_k\|_2^2$$

Weight contribution of data x_i

$x_i \rightarrow \phi(x_i)$

[1] Mika, Scholkopf, Smola and Muller, Scholz, Ratsch, Kernel PCA and de-noising in feature spaces, 1998

Cluster Update

- **Value of k^{th} cluster S_k :**

$$S_k^{l+1} = \{x_i : \underbrace{\|\phi(x_i) - m_k^l\|_2^2}_{d(x_i,m_k)} \leq \underbrace{\|\phi(x_i) - m_{k'}^l\|_2^2}_{d(x_i,m_{k'})},\ \forall k' \neq k\}$$

with $d(x_i, m_k) = \|\phi(x_i) - m_k\|_2^2 = \langle \phi(x_i) - m_k, \phi(x_i) - m_k \rangle$

$$= \langle \phi(x_i), \phi(x_i) \rangle - 2\langle \phi(x_i), m_k \rangle + \langle m_k, m_k \rangle$$

$K(x_i, x_i) = K_{ii}$ $\quad (I)_{ik} \quad (II)_{kk}$

Kernel matrix

Linear algebra : $(I)_{ik} = (K\Theta F\Lambda)_{ik}$

$$(II)_{kk} = (\Lambda F^T \Theta K \Theta F \Lambda)_{kk}$$

$$\Theta = \text{diag}(\theta_1, ..., \theta_n)$$

$$\Lambda = \text{diag}(1/\sum_{i\in S_1} \theta_i, ..., 1/\sum_{i \in S_K} \theta_i)$$

$$F_{ik} = \begin{cases} 1 \text{ if } x_i \in S_k \\ 0 \text{ otherwise} \end{cases}$$

$K(x, y) = \langle \phi(x), \phi(y) \rangle$ **Kernel matrix**

$$F = \begin{matrix} S_1 \\ \\ \\ S_2 \\ \\ S_3 \end{matrix} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{K \text{ columns}} \; n \text{ rows}$$

F_k = **Indicator function of set S_k**

Kernel k-means algorithm

- Initialization : Random initial means
- Iterate until convergence: l=0,1,...

(1) Cluster update/expectation step

Compute all distances : $D_{ik}^{l} = d(x_i, m_k^l)$

$$D = \mathrm{diag}(K) - 2K\Theta F\Lambda + \mathrm{diag}(\Lambda F^T \Theta K \Theta F \Lambda)$$

Update clusters : $F_{ik}^{l+1} = \begin{cases} 1 \text{ if } D_{ik}^l = \mathrm{argmin}_{k'} D_{ik'}^l \\ 0 \text{ otherwise} \end{cases}$

$\downarrow$ $\mathbf{F_k}$ is an implicit representation of cluster $\mathbf{S_k}$

$$S_k^{l+1} = \{x_i : F_{ik}^{l+1} = 1\}$$

$$F = \begin{array}{c} \\ S_1 \\ \\ \\ S_2 \\ \\ S_3 \end{array} \overset{K}{\overbrace{\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}} \; n$$

= Indicator function of set $\mathbf{S_k}$

(2) Mean update/maximization step

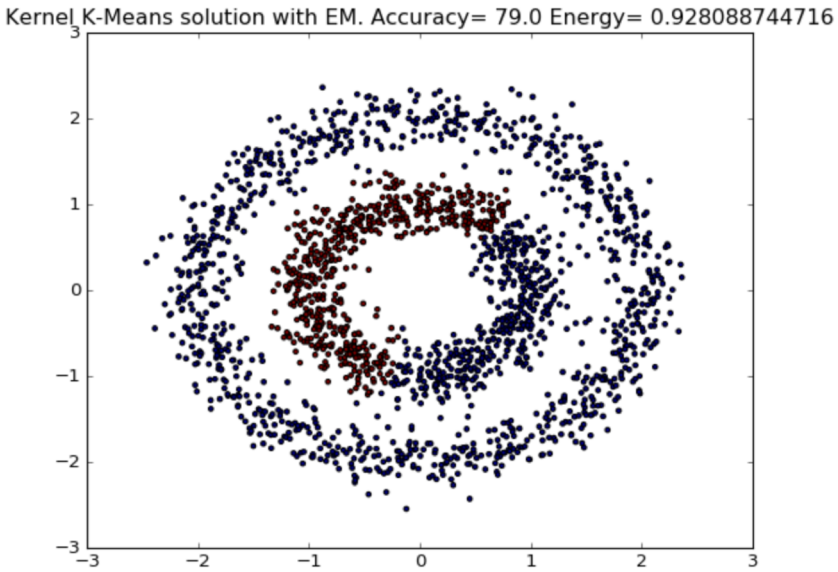
No update needed here! It is implicitly done in (1).

Lab 2 : Kernel k-means

- Run 01_graph_clustering/code02.ipynb

```
In [7]: # Run Kernel/Non-Linear K-Means with EM approach

        # Compute linear Kernel for standard K-Means
        Ker = construct_kernel(X,'kNN_gaussian',100)

        # Kernel K-Means with EM approach
        C_kmeans, En_kmeans = compute_kernel_kmeans_EM(nc,Ker,Theta,10)
```

$$K_{ij} = e^{-\|x_i - x_j\|_2^2/\sigma^2}$$



Advantage of kernel trick

- Do we need to compute the kernel mapping ϕ ?
  - No, we never use explicitly the non-linear function ϕ !
    - The exact expression is actually irrelevant, only the kernel matrix $\boldsymbol{K}$ is important.
  - Why is this useful?

$$K(x_i, x_j) = (a\langle x_i, x_j\rangle + b)^c = \langle \phi(x_i), \phi(x_j)\rangle$$

$\langle x_i, x_j\rangle$ Low-dim products, e.g. x_i in R^{100}

$\langle \phi(x_i), \phi(x_j)\rangle$ High-dim products, e.g. $\phi(x_i)$ in $R^{1,000}$

→ Time/memory consuming !

- Popular kernels
  - Gaussian kernels : $K_{ij} = e^{-\|x_i - x_j\|_2^2/\sigma^2}$
  - Polynomial kernels : $K_{ij} = (a\langle x_i, x_j\rangle + b)^c$

Properties of kernel k-means algorithm

- Advantages

  (1) Same advantages of linear k-means

  (2) Data can be non-linearly separable

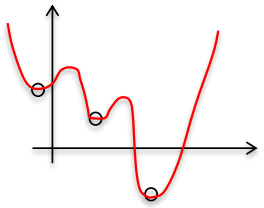
  (3) Computations are linear : Intel/AMD includes optimised linear algebra libraries (MKL, LAPACK, BLAS) for CPU and CUDA for GPUs.

- Limitations

  (1) Non-convex energy

  ⇒ Existence of local minimizers and saddle points.

  ⇒ Good initialization is critical. Standard trick is to restart several times and pick the solution with the lowest energy value.



Non-convex

Spectral approach of kernel k-means[1]

- Let us start again from the weighted kernel k-means objective :

$$\min_{M,S} \; E(M,S) = \sum_{k=1}^{K} \sum_{x_i \in S_k} \theta_i \|\phi(x_i) - m_k\|_2^2$$

- Let us rewrite it as a trace optimization problem under some constraints :

after some linear algebra...

$$\min_{Y} \; -\mathrm{tr}(Y^T \Theta^{1/2} K \Theta^{1/2} Y) \;\; \text{s.t.} \;\; Y^T Y = I_K \text{ and } Y \in S_{Ind}$$

(weighted) indicator of clusters

$$Y_{ik} = \begin{cases} (\frac{\theta_i}{\sum_{j \in S_k} \theta_j})^{1/2} & \text{if } i \in S_k \\ 0 & \text{otherwise} \end{cases}$$

$$\max_{Y} \; \mathrm{tr}(Y^T \Theta^{1/2} K \Theta^{1/2} Y) \;\; \text{s.t.} \;\; Y^T Y = I_K \text{ and } Y \in S_{Ind}$$

[1] Dhillon, Guan, Kulis, Kernel k-means: spectral clustering and normalized cuts, 2004

Spectral relaxation

- Maximizing the objective is a NP-hard problem (no polynomial-time algorithm)

$$\max_{Y} \ \mathrm{tr}(Y^T \Theta^{1/2} K \Theta^{1/2} Y) \ \text{ s.t. } \ Y^T Y = I_K \text{ and } Y \in S_{Ind}$$

This binary constraint makes the problem NP-hard

- Spectral relaxation : We drop the indicator constraint Y in S_{ind}

$$\max_{Y} \ \mathrm{tr}(Y^T A Y) \ \text{ s.t. } \ Y^T Y = I_K \ \text{ with } A = \Theta^{1/2} K \Theta^{1/2}$$

- Solution given by the spectral theorem[1] : Top K eigenvectors of matrix A given by EVD.

[1] Hawkins, Cauchy and the spectral theory of matrices, 1975

Eigenvalue decomposition (EVD)

- Suppose A is symmetric and positive semi-definite (PSD) (all kernel matrices are symmetric and PSD by construction), then EVD gives :

$$Ay_k = \lambda_k y_k \qquad \text{with} \qquad \lambda_{\max} = \lambda_1 \geq \lambda_2 \geq ... \geq \lambda_K$$

$$\langle y_k, y_{k'} \rangle = \begin{cases} 1 \text{ if } k = k' \\ 0 \text{ otherwise} \end{cases}$$

$$Y^T Y = I_K$$

- Top K eigenvalues/eigenvectors : K largest λ_k

$$\text{tr}(Y^T AY) = \sum_{k=1}^{K} y_k^T Ay_k = \sum_{k=1}^{K} y_k^T \lambda_k y_k = \sum_{k=1}^{K} \lambda_k \langle y_k, y_k \rangle = \underbrace{\sum_{k=1}^{K} \lambda_k}_{\text{K largest values}}$$

$$\Updownarrow$$

$$\max_Y \ \text{tr}(Y^T AY) \ \text{ s.t. } \ Y^T Y = I_K$$

Spectral kernel k-means algorithm

- Three steps

  (1) Compute : $A = \Theta^{1/2} K \Theta^{1/2}$

  (2) Solve EVD for the top K eigenvectors : $Ay_k = \lambda_k y_k$

  (3) Binarization step : Use the solution Y as embedding coordinates of X and apply standard k-means on Y.

- Notes
  - Remember original K-Means problem is NP-hard ⇒ we drop the indicator constraint Y in S_{ind} ⇒ we get an approximate solution.
  - Spectral techniques (EVD) are vastly used in data analysis because theory/tool is well understood. However, they do not scale well to big data as EVD complexity is $O(n^3)$ (stochastic EVD can reduce to O(n) but it provides an approximation).

Outline

- Graph clustering
  - Definition
  - Linear K-Means
  - Kernel K-Means
  - Balanced Cuts
  - NCut, PCut
  - Louvain Algorithm

No feature learning !

Graph clustering with balanced cuts

- Motivations

  (1) Kernel techniques are sound, but they do not scale well to large datasets because K is full, memory/speed requirement is $O(n^2)/O(n^3)$ for EVD.

  Ex: n=1M $\Rightarrow$ n^2=1Tera, n^3=1Yotta

  What alternative? Sparse matrix, which means graph!

  (2) Natural graphs : Partitioning graphs is a fundamental problem for

  (i) Identifying connected groups (users on social networks),

  (ii) Balanced graph partitioning is critical for efficient distributed big graph computing.

  What technique? A class called balanced cut algorithms

- Balanced graph cuts play a central role in

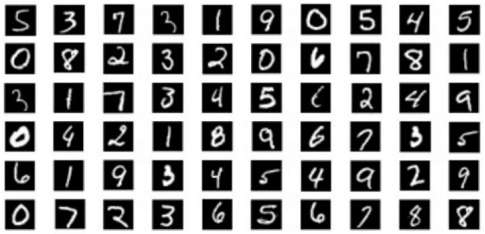
  (1) Graph theory (define classes of networks and their properties).

  (2) Applications (state-of-the-art for unsupervised clustering e.g. Metis[1]/Graclus[2]).

[1] Karypis, Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, 1998
[2] Dhillon, Guan, Kulis, Weighted Graph Cuts without Eigenvectors: A Multilevel Approach, 2007

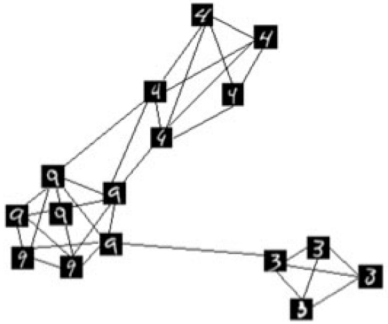
Balanced cut for data clustering

- Graph representation : Given a set of data $V=\{x_1,...,x_n\}$ in R^d, construct a k-NN graph $G=(V,E,W)$



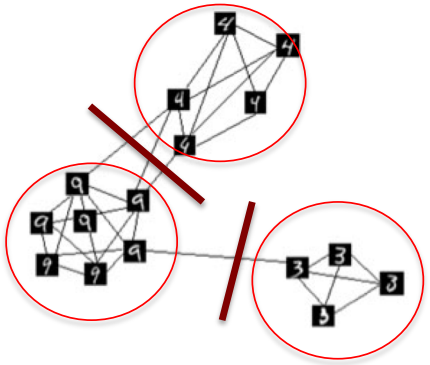
$$V = \{x_1, ..., x_n\} \in \mathbb{R}^d$$

k-NN graph construction



$$G = (V, E, W)$$

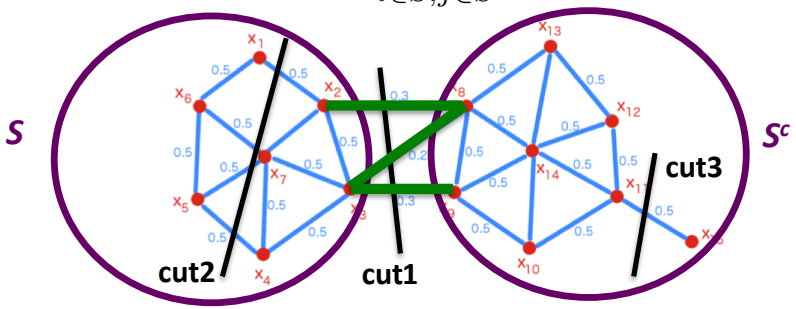
- Data clustering : Observe that data close on graphs are similar.
  - Finding clusters of data can be done by cutting the graph at some specific locations.



Min Cut[1]

- Cut operator : Given a graph G, a cut partitions G into two sets S and S^c with value

$$Cut(S, S^c) = \sum_{i \in S, j \in S^c} W_{ij}$$



Value of cut1: $Cut(S,S^c) = 0.3 + 0.2 + 0.3 = 0.8$
Value of cut2: $Cut(S,S^c) = 0.5 + 0.5 + 0.5 + 0.5 = 2.0$
Value of cut3: $Cut(S,S^c) = 0.5$

- Min cut is biased : It favors small sets of isolated data.
  - Solution : Find clusters of similar sizes/volumes while minimizing the cut operator

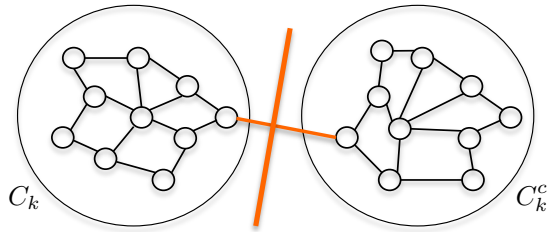
$$\min\ Cut \text{ and } \max\ Vol \quad \Leftrightarrow \quad \min \frac{Cut}{Vol}$$ Balanced cuts

[1] Wu, Leahy, An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation, 1993

Most popular balanced cuts

- Cheeger Cut[1] (most popular in graph theory) :

$$\min_S \frac{Cut(S, S^c)}{\min(Vol(S), Vol(S^c))}$$



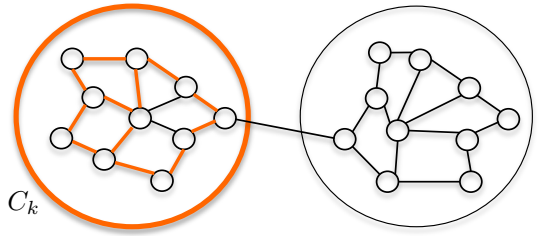
Partitioning by min edge cuts.

- Normalized Cut[2] (most popular in applications) :

$$\min_S \frac{Cut(S, S^c)}{Vol(S)} + \frac{Cut(S, S^c)}{Vol(S^c)}$$

- Normalized Association[3] :

$$\min_S \frac{Assoc(S, S)}{Vol(S)} + \frac{Assoc(S^c, S^c)}{Vol(S^c)}$$



Partitioning by max vertex matching.

with $Cut(S, S^c) = \sum_{i \in S, j \in S^c} W_{ij}$ #connections between S and S^c

$Vol(S) = \sum_{i \in S} d_i$, with $d_i = \sum_{j \in V} W_{ij}$

$Assoc(S, S) = \sum_{i \in S, j \in S} W_{ij}$ #connections between all vertices in S

[1] Cheeger, Pinching theorems for a certain class of Riemannian manifolds, 1969
[2] Shi, Malik, Normalized cuts and image segmentation, 2000
[3] Karypis, Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, 1998

Spectral relaxation of balanced cuts

- Major issue : Solving balanced cut problems is directly intractable as NP-hard combinatorial problems ⇒ We need to find good approximation (as close to original solution as possible).
  - Good approximate techniques are based on spectral relaxation.

- Normalized Association :

$$\min_{S_k} \sum_{k=1}^{K} \frac{Assoc(S_k, S_k)}{Vol(S_k)} \quad (1)$$

- Reformulation : Rewrite combinatorial problem as a continuous optimization problem

$$(1) \quad \Leftrightarrow \quad \max_{F} \sum_{k=1}^{K} \frac{F_k^T W F_k}{F_k^T D F} \quad (2)$$

$$\Updownarrow \quad \text{with the change of variable:} \quad Y_k = \frac{D^{1/2} F_k}{\|D^{1/2} F_k\|_2}$$

$$(2) \quad \Leftrightarrow \quad \max_{Y} \ \mathrm{tr}(Y^T A Y) \ \text{ s.t. } \ Y^T Y = I_K, \ Y \in S_{Ind} \ \text{ and } \ A = D^{-1/2} W D^{-1/2}$$

$$\downarrow$$

$$Y_{ik} = \begin{cases} \left(\frac{D_{ii}}{Vol(S_k)}\right)^{1/2} & \text{if } i \in S_k \\ 0 & \text{otherwise} \end{cases}$$

Spectral relaxation of balanced cuts

- Binary constraint Y in S_{ind} makes the problem NP-hard – let us drop it :

$$\max_{Y} \ \mathrm{tr}(Y^T A Y) \ \text{ s.t. } \ Y^T Y = I_K$$

- Solution : Top K eigenvectors of A given by EVD.
  - Have we seen this before? :)

Relationship between kernel k-means and balanced cuts[1]

- **Kernel K-Means:**

$$\max_{Y} \ \mathrm{tr}(Y^T A Y) \ \text{ s.t. } \ Y^T Y = I_K \ \text{ with } A = \Theta^{1/2} K \Theta^{1/2}, \ Y \in S_{Ind} \quad (1)$$

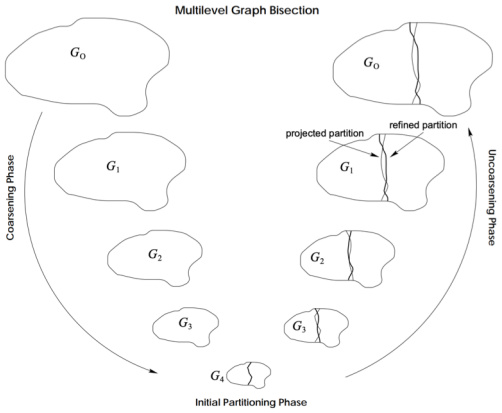
- **Balanced Cuts:**

$$\max_{Y} \ \mathrm{tr}(Y^T A Y) \ \text{ s.t. } \ Y^T Y = I_K \ \text{ with } \ A = D^{-1/2} W D^{-1/2}, \ Y \in S_{Ind} \quad (2)$$

- **Equivalence:**

$$(1) \quad \Leftrightarrow \quad (2) \quad \text{for} \quad \Theta = D^{-1}, \ K = W$$

- **Consequence : Balanced cut problems can also be solved by (hierarchical) EM approach !**
  - **Metis[1]/Graclus[2] do not require EVD ⇒ Scale up to large datasets (one of the best graph partitioning techniques)**



[1] Dhillon, Guan, Kulis, Kernel k-means: spectral clustering and normalized cuts, 2004
[2] Karypis, Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, 1998
[3] Dhillon, Guan, Kulis, Weighted Graph Cuts without Eigenvectors: A Multilevel Approach, 2007

Lab 3 : Metis/Graclus

- Run 01_graph_clustering/code03.ipynb

In [26]:
```
# Run Graclus
_,part_vert = pymetis.part_graph(nc, adjacency=W_louvain)
C_graclus = np.array(part_vert,dtype='int32')
acc = compute_purity(C_graclus,Cgt,nc)
print('Accuracy graclus=',acc)
```

```
Accuracy graclus= 36.73139158576052
```

In [27]:
```
# Run PCut
Cpcut,acc = compute_pcut(W,Cgt,nc,5,50)
print('Accuracy PCut=',acc)
```

```
Accuracy PCut= 45.98705501618123
```

- Either EM or Spectral approaches still compute approximate solutions.
  - Can we improve the quality of clustering/partitioning solutions?

Outline

- Graph clustering
  - Definition
  - Linear K-Means
  - Kernel K-Means
  - Balanced Cuts
  - NCut, PCut
  - Louvain Algorithm

No feature learning !

NCut Algorithm[1]

- Motivation : The binary constraint Y in S_{ind} or Y in {0,1} must be dropped to compute a solution of the NP-hard problem. Spectral solutions do not naturally satisfy this constraint.
  - Can we enforce this constraint during the optimization?

- NCut : The most popular spectral graph clustering algorithm !

  It has two steps

  (1) Compute spectral solutions (as before).

  (2) Find best solution that satisfies the binary constraint (new step).

[1] Yu, Shi, Multiclass spectral clustering, 2003

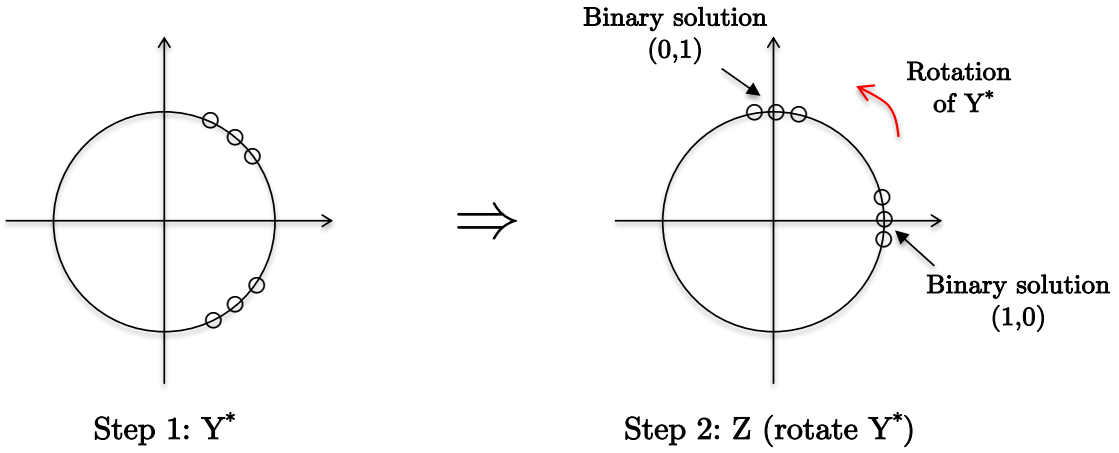
Analysis

- **Step 1:** $Y^\star = \underset{Y}{\text{argmax}}\ \text{tr}(Y^T A Y)\ \text{ s.t. }\ Y^T Y = I_K\ \text{ with }\ A = D^{-1/2} W D^{-1/2}$

  ⇒ Solved by EVD

- **Step 2:** $\underset{Z,R}{\min}\ \|Z - Y^\star R\|_F^2\ \text{ s.t. }\ R^T R = I_K,\ \text{ and }\ Z \in \{0,1\}$

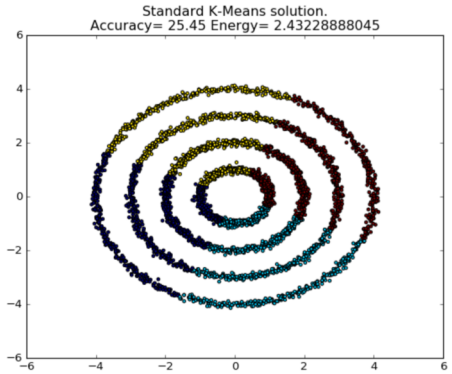
  ⇒ Solved by SVD (EVD for non-square and non-PSD matrix)

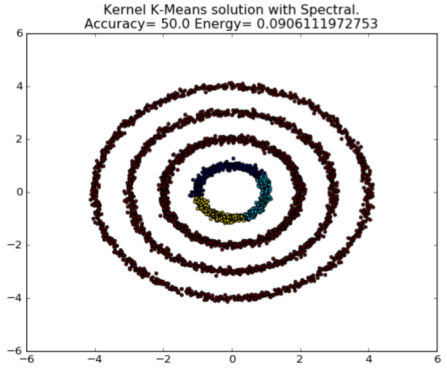


Lab 4 : NCut

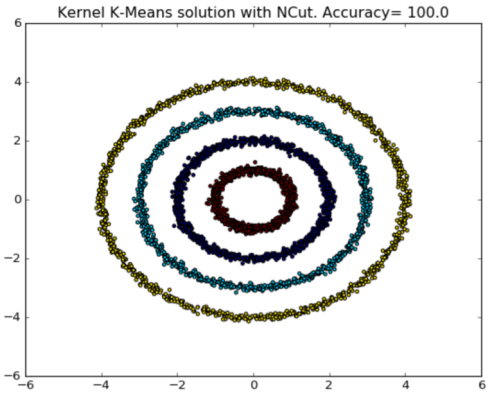
- Run 01_graph_clustering/code04.ipynb

```
In [3]: # Run Standard/Linear K-Means
Theta = np.ones(n) # Equal weight for each data
# Compute linear Kernel for standard K-Means
Ker = construct_kernel(X,'linear')
# Standard K-Means
C_kmeans, En_kmeans = compute_kernel_kmeans_EM(nc,Ker,Theta,10)
# Plot
plt.figure(201)
size_vertex_plot = 10
plt.scatter(X[:,0], X[:,1], s=size_vertex_plot*np.ones(n), c=C_kmeans)
plt.title('Standard K-Means solution. Accuracy= ' + str(compute_purity(C_kmeans,Cgt,nc)) +
          ' Energy= ' + str(En_kmeans))
plt.show()
```





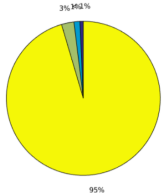

```
In [5]: W = construct_knn_graph(X,50,'euclidean_zelnik_perona')
        Cncut,acc = compute_ncut(W,Cgt,nc)
```



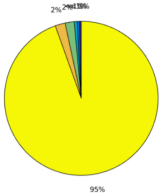
Limitation of NCut

- NCut on real-world networks WEBK4 and CITESEER

	Partition $\mathcal{P}$ of WEBKB4 found by by the **Ncut** algo.
$e^{-H(\mathcal{P})}$	.7946
$\mathbf{Pcut}(\mathcal{P})$	.8697
$\mathbf{Ncut}(\mathcal{P})$	**.5004**



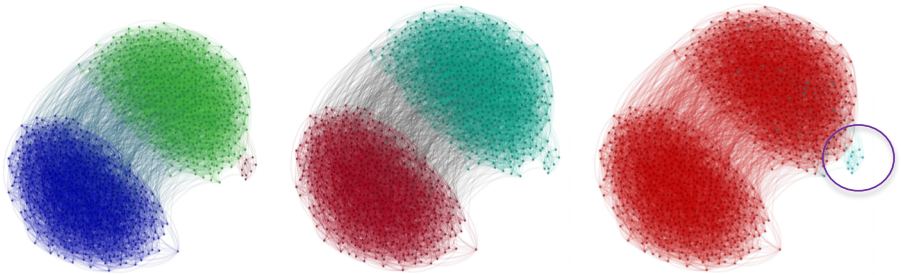
	Partition $\mathcal{P}$ of CITESEER found by by the **Ncut** algo.
$e^{-H(\mathcal{P})}$	.7494
$\mathbf{Pcut}(\mathcal{P})$	.8309
$\mathbf{Ncut}(\mathcal{P})$	**.5217**



- NCut is biased toward small clusters.

PCut[1] -- SOTA for balanced cut

- Balanced Cuts are biased towards cluster outliers :



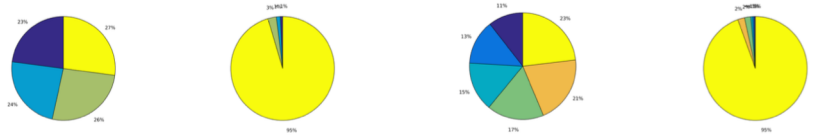
(a) A_n is in blue, B_n in green and C in blue.

(b) $\mathcal{P}_n^{0,\text{good}} = (A_n, B_n \cup C)$

(c) $\mathcal{P}_n^{0,\text{bad}} = (A_n \cup B_n, C)$

- Results :

	Partition $\mathcal{P}$ of WEBKB4 found by by the **Pcut** algo.	Partition $\mathcal{P}$ of WEBKB4 found by by the **Ncut** algo.	Partition $\mathcal{P}$ of CITESEER found by by the **Pcut** algo.	Partition $\mathcal{P}$ of CITESEER found by by the **Ncut** algo.
$e^{-H(\mathcal{P})}$	.2506	.7946	.1722	.7494
Pcut$(\mathcal{P})$	**.5335**	.8697	**.4312**	.8309
Ncut$(\mathcal{P})$	.5257	**.5004**	.5972	**.5217**

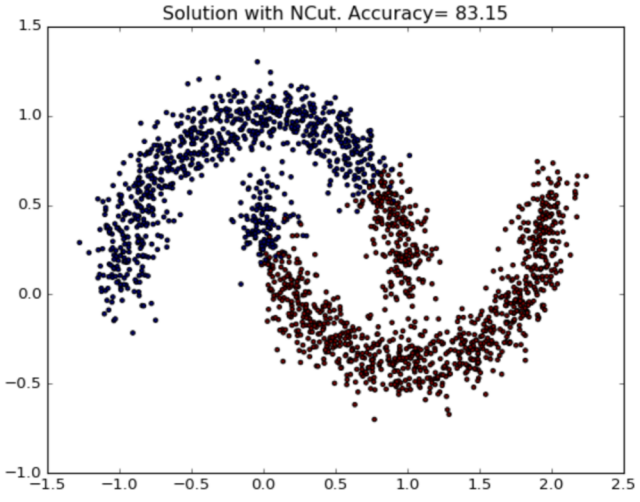


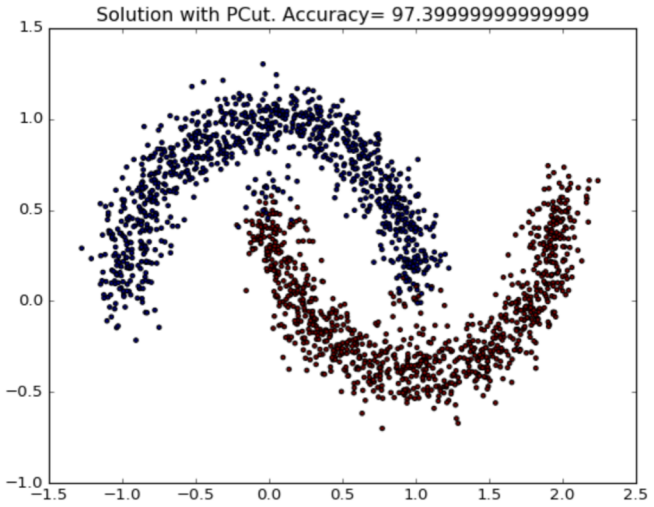
[1] Laurent, von Brecht, Bresson, The product cut, 2016

Lab 5 : PCut

- Run graph_clustering/code05.ipynb

```
In [5]: W = construct_knn_graph(X,10,'euclidean')
        Cncut,acc = compute_ncut(W,Cgt,nc)
        Cpcut,acc = compute_pcut(W,Cgt,nc,1,200)
```





Outline

- Graph clustering
  - Definition
  - Linear K-Means
  - Kernel K-Means
  - Balanced Cuts
  - NCut, PCut
  - Louvain Algorithm

No feature learning !

Louvain Algorithm[1]

- Previous techniques assume to know the number K of clusters.
- If K is unknown, we have two approaches :

  (1) Define a quality measure of clustering (domain expertise), use previous techniques with different K values and pick the K with the best measure value.

  (2) K is a variable of the clustering problem : Louvain Algorithm.
- Louvain Technique : Very popular in social science
  - It is greedy algorithm that optimizes the modularity objective :

$$\max_f \; Q(f) = \sum_{ij} (W_{ij} - \frac{\gamma}{2m} d_i d_j) \delta(f_i, f_j) \quad \text{with} \quad 2m = \sum_{ij} W_{ij} = Vol(V)$$

$$\delta(f_i, f_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

$$\Updownarrow$$

$$\min_{S_k} \sum_{k=1}^{K} Cut(S_k, S_k^c) - \gamma Vol(S_k).Vol(S_k^c) \quad \Leftrightarrow \quad \min_{S_k} \sum_{k=1}^{K} \frac{Cut(S_k, S_k^c)}{Vol(S_k) Vol(S_k^c)}$$

Same property as balanced cut, but extra parameter γ to select.

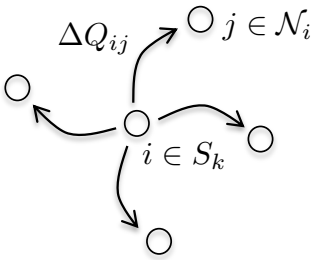
[1] Blondel, Guillaume, Lambiotte, Lefebvre, Fast unfolding of communities in large networks, 2008

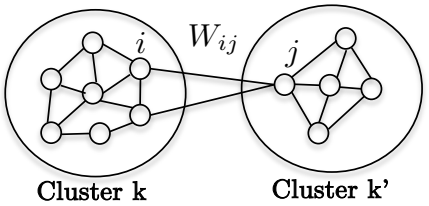
Louvain greedy algorithm

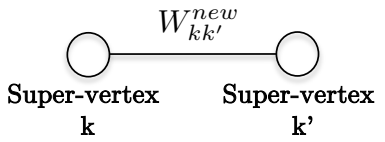
- Step 1 : Energy minimization step
  - Find communities by maximizing locally the modularity.
  - Each node is first associated to its own community then for each node i, we assign i to the community of its neighbor that best increases the modularity. The process is repeated until no changes occur.
- Step 2 : Graph coarsening step
  - Compute a new graph by merging the communities to a super-vertex.
  - From Step 1, a new graph is constructed by forming a new adjacency matrix.
- Properties

  (1) (Relatively) fast algorithm

  (2) No theoretical guarantee on the solution







$$W_{kk'}^{new} = \sum_{i \in S_k, j \in S_{k'}} W_{ij}$$

Lab 6 : Louvain algorithm

- Run graph_clustering/code06.ipynb

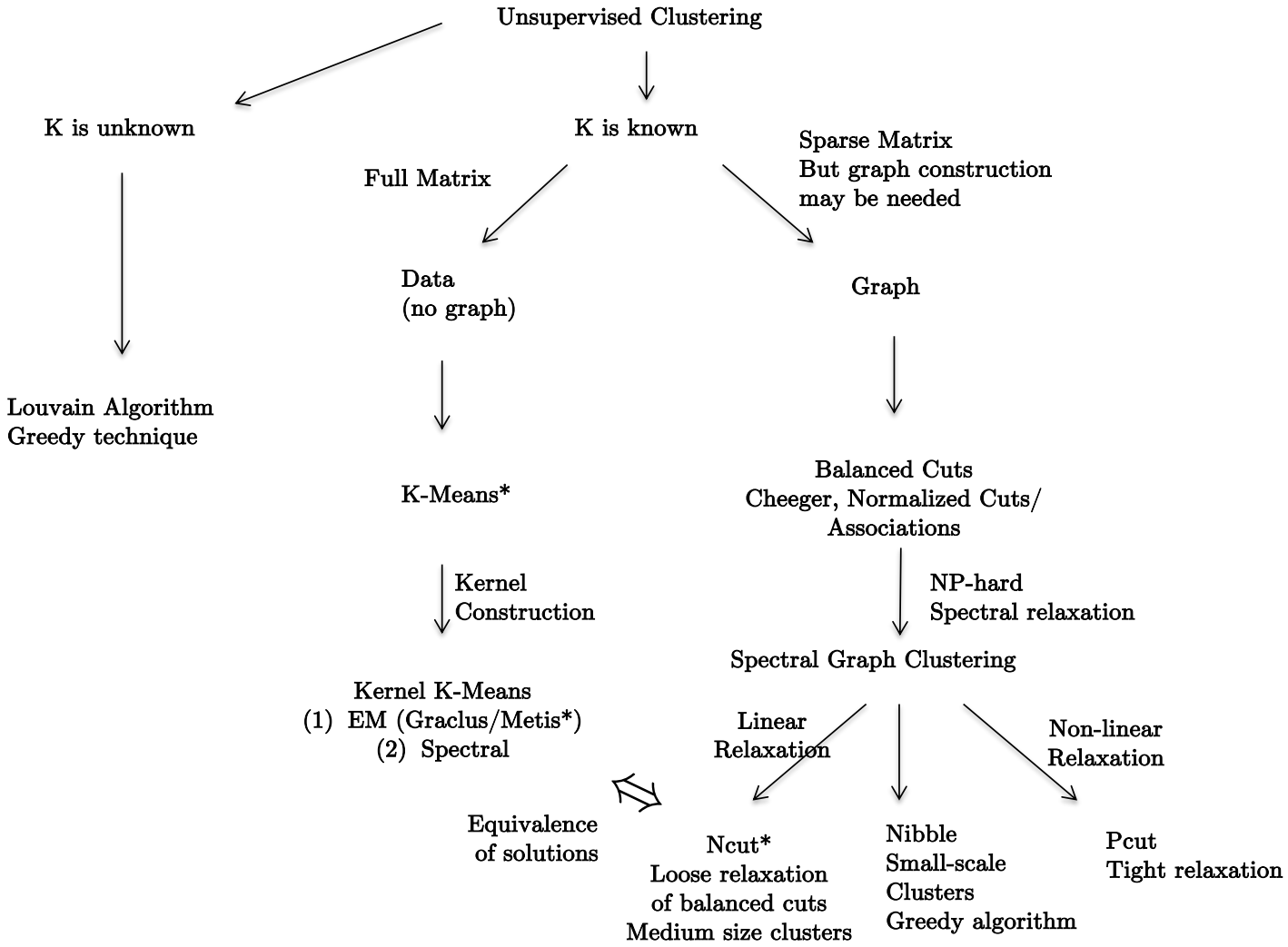
```
# Louvain partition, Blondel-et.al, Fast unfolding of communities in large networks, 2008
partition = community.best_partition(Wnx)
nc_louvain = len(np.unique( [partition[nodes] for nodes in partition.keys()] ))
n = len(Wnx.nodes())
print('nb_data:', n , ', nb_clusters=', nc_louvain)

# Extract clusters
Clouv = np.zeros([n])
clusters = []
k = 0
for com in set(partition.values()):
    list_nodes = [nodes for nodes in partition.keys() if partition[nodes] == com]
    Clouv[list_nodes] = k
    k += 1
    clusters.append(list_nodes)

# Accuracy
acc = compute_purity(Clouv,Cgt,nc_louvain)
print('accuracy_louvain=',acc,' with nb_clusters=',nc_louvain)
```

```
Name:
Type: Graph
Number of nodes: 9298
Number of edges: 77679
Average degree:  16.7088
nb_data: 9298 , nb_clusters= 15
accuracy_louvain= 95.30006453000645  with nb_clusters= 15
```

```
# Run NCut
Cncut,acc = compute_ncut(W,Cgt,nc_louvain)
print(acc)
```

```
89.8902989890299
```

Data and graph clustering algorithms



Outline

- Graph clustering
- Classification with graphs
- Recommendation with graphs
- Dimensionality reduction with graphs

No feature learning !

Outline

- Classification with graphs
  - Problem formulation
  - Linear SVM
  - Soft-margin SVM
  - Kernel techniques
  - Non-linear/kernel SVM
  - Graph SVM
  - Conclusion

No feature learning !

Outline

- Classification with graphs
  - Problem formulation
  - Linear SVM
  - Soft-margin SVM
  - Kernel techniques
  - Non-linear/kernel SVM
  - Graph SVM
  - Conclusion

No feature learning !

Learning techniques

- Four classes of learning algorithms
  - Supervised learning : Algorithms that use labeled data (i.e. human annotation).
  - Unsupervised learning : Algorithms that do not use any label information (data labeling is time/money consuming).
  - Self-supervised learning : Algorithms that do not use any label information but learn data representation that can be transferred to downstream tasks.
  - Semi-supervised learning : Algorithms that use both labeled and unlabeled data.
- Graph clustering belongs to unsupervised learning.
- This section covers supervised learning (SVM classification) and semi-supervised learning (SVM with graph structure).

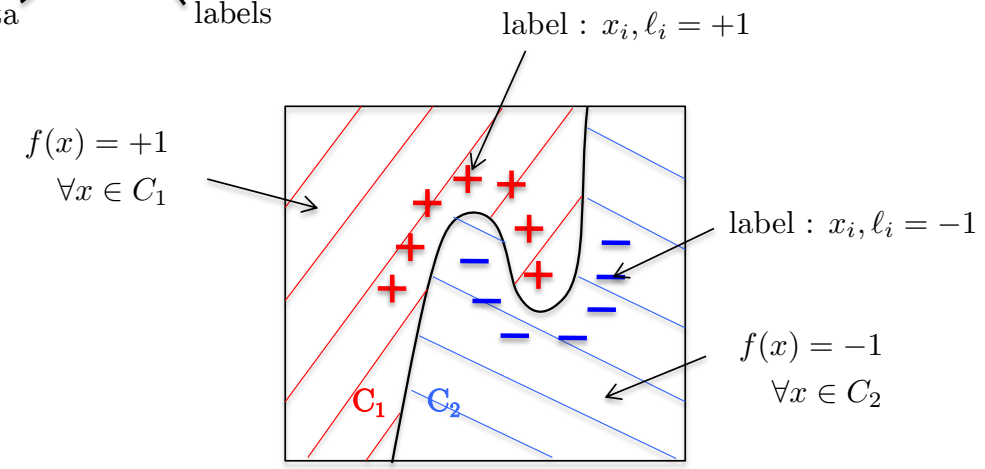
Support Vector Machine (SVM)

- SVM was the most popular classification technique before deep learning.
- SVM loss, a.k.a. hinge loss, is still used in theory of deep learning (for geometric interpretation).
- Problem formulation : Given a set of labeled data that belong to two classes, construct a classification function that outputs the class of any new data.

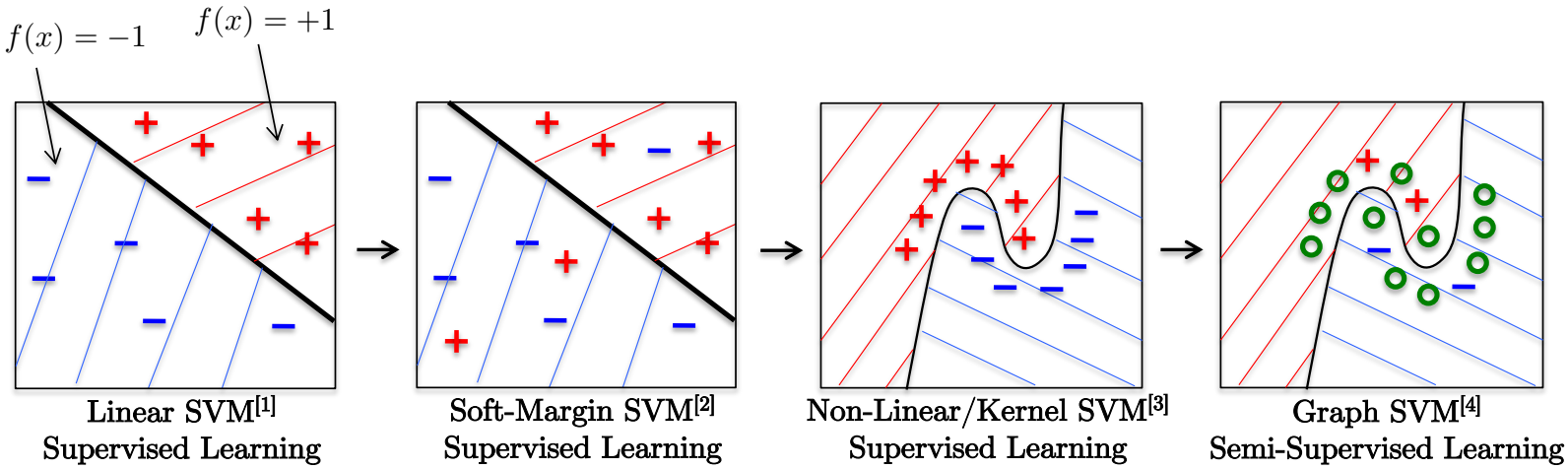
☺

$$V = \{x_i, \ell_i\}_{i=1}^{n},\ x_i \in \mathbb{R}^d,\ \ell_i \in \{-1, +1\}$$

data ↗ ↖ labels



Development of SVM techniques



[1] Vapnik, Chervonenkis, On a perceptron class, 1964
[2] Cortes, Vapnik, Support-vector networks, 1995
[3] Boser, Guyon, Vapnik, A training algorithm for optimal margin classifiers, 1992
[4] Belkin, Niyogi, Sindhwani,Manifold regularization: A geometric framework for learning from labeled and unlabeled examples, 2006

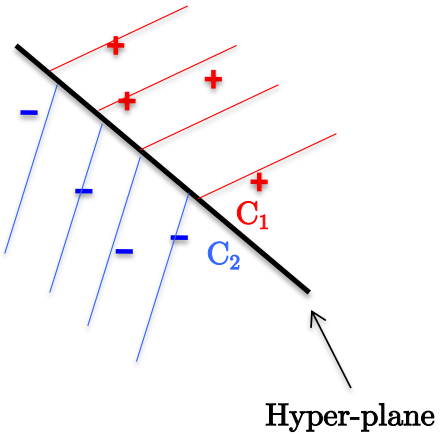
Outline

- Classification with graphs
  - Problem formulation
  - Linear SVM
  - Soft-margin SVM
  - Kernel techniques
  - Non-linear/kernel SVM
  - Graph SVM
  - Conclusion

No feature learning !

Standard/linear SVM[1]

- Assumption : Training (and test) data are linearly separable, i.e. data can be perfectly separated with a hyperplane :

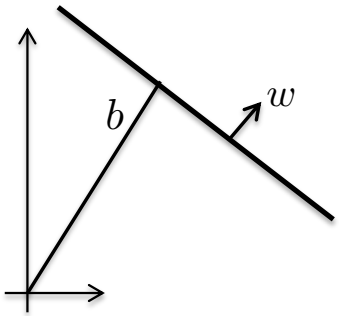


- Linear SVM : Given a training dataset V=$\{x_i,l_i\}$, design a classification function that assigns a new data x to the class that is best "consistent" with V :

$$f : x \in \mathbb{R}^d \rightarrow \{-1, +1\}$$

- Assuming we have linear data points, determine the hyper-plane that best separates the two classes.

- Any hyper-plane is parameterized by two variables (w,b), where w is the normal vector of hyperplane, and b is the offset value :



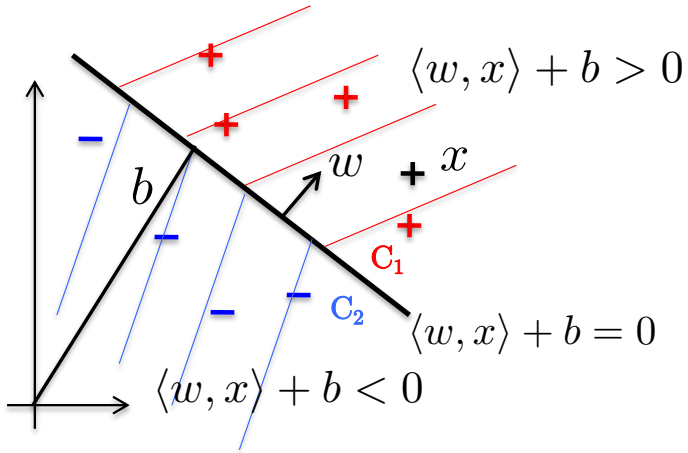
Hyper-plane equation: $\langle w, x \rangle + b = 0$

[1] Vapnik, Chervonenkis, On a perceptron class, 1964

Linear SVM classifier

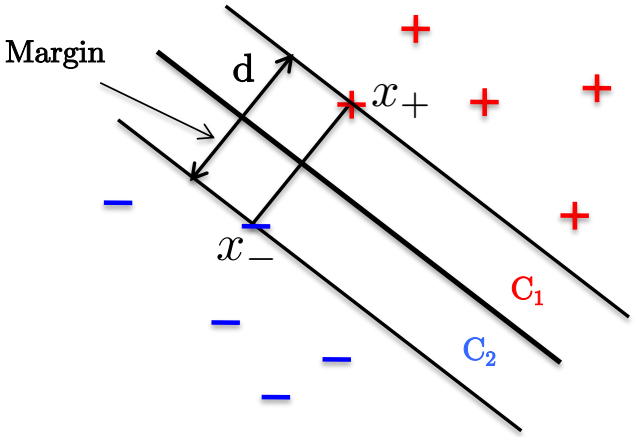
- **Classification function :**

$$f(x) = \text{sign}(\langle w, x \rangle + b) = \begin{cases} +1 & \text{if } x \in C_1 \\ -1 & \text{if } x \in C_2 \end{cases}$$



Parameter estimation

- Define the best hyper-plane that maximizes the margin d between the 2 classes :



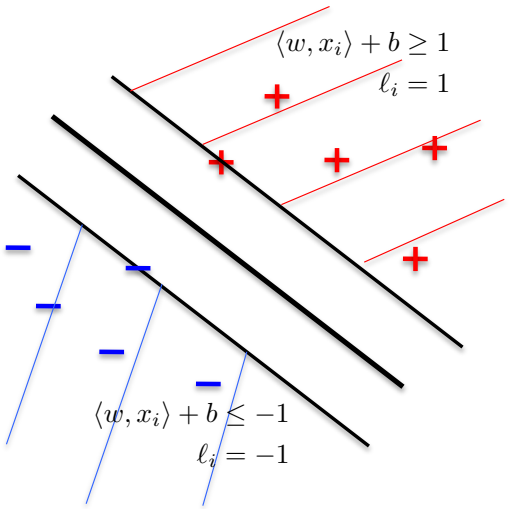
- What is the relationship between w and d ?

$$\begin{array}{l} \langle w, x_+ \rangle + b = +1 \\ \langle w, x_- \rangle + b = -1 \\ \hline \langle w, x_+ - x_- \rangle = 2 \end{array}$$

$$\vec{d} = x_+ - x_- = \alpha w \quad \rightarrow \quad \alpha = \frac{2}{\|w\|_2^2} \quad \rightarrow \quad d = \frac{2}{\|w\|_2}$$

$\Rightarrow$ **Maximize the margin d :** $\max\ d \Leftrightarrow \max\ \frac{2}{\|w\|_2} \Leftrightarrow \min\ \|w\|_2^2$ $\Leftrightarrow$ **Minimize the weight w**

Primal optimization problem



- Variable w is called the primal variable.
- Optimization problem w.r.t. w is a constrained opt problem :

$$f_i = \langle w, x_i \rangle + b = \left\{ \begin{array}{ll} \geq +1 & \text{if } x \in C_1 \\ \leq -1 & \text{if } x \in C_2 \end{array} \right.$$

$$\ell_i = \left\{ \begin{array}{ll} +1 & \text{if } x \in C_1 \\ -1 & \text{if } x \in C_2 \end{array} \right.$$

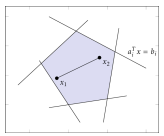
$$\rightarrow \quad \ell_i . f_i \geq 1 \ \forall i \in V$$

- Overall, a SVM classifier f is given by the solution of the quadratic programming (QP) problem :

$$\min_{w,b} \ \|w\|_2^2 \ \text{ s.t. } \ \ell_i . f_i \geq 1 \ \forall i \in V \qquad \rightarrow \quad f(x) = \langle w, x \rangle + b$$

Quadratic function

Convex set (polytope)



SVM classifier

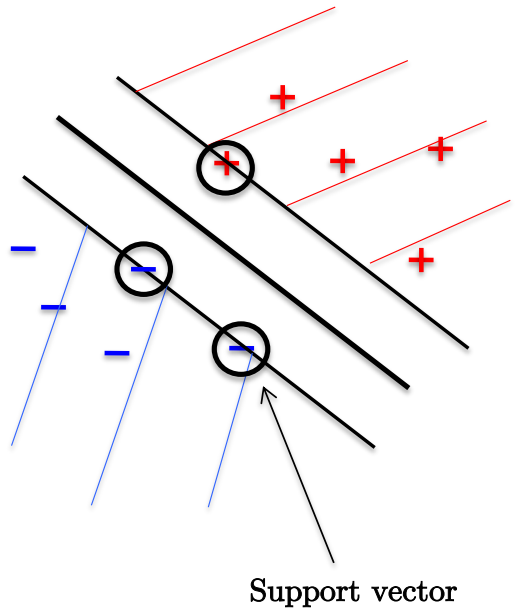
Support vectors

- Support vectors are the data that are exactly localized on the margin hyperplanes.
- Offset value b is defined by :

$$\ell_i.(\langle w, x_i^{SP}\rangle + b) - 1 = 0, \ \forall x_i^{SP} \rightarrow \quad b_i = \ell_i - \langle w, x_i^{SP}\rangle$$

$$\downarrow$$

$$b = \mathbb{E}(\{b_i\})$$



Dual optimization problem

- Primal problem (w/ primal variable w) :

$$\min_{w,b} \|w\|_2^2 \quad \text{s.t.} \quad \ell_i.f_i \geq 1 \ \forall i \in V$$

- Dual problem (w/ primal variable α) : ⇕

After computations..

$$\min_{\alpha \geq 0} \frac{1}{2}\alpha^T Q \alpha - \langle \alpha, 1 \rangle \quad \text{s.t.} \quad \langle \alpha, 1 \rangle = 0$$

QP problem

With:

$$Q = LKL$$

$$L = \text{diag}(\ell_1, ..., \ell_n)$$

$$K_{ij} = \langle x_i, x_j \rangle$$

Linear kernel

- The dual problem naturally uses kernels.
  - Scalar products $\langle x_i, x_j \rangle$ defines the linear kernel.
  - It is easy to generalize to non-linear kernels with the kernel trick (later discussed).

Optimization algorithm

- Classification function :

$$f(x) = \text{sign}(\langle w^\star, x \rangle + b^\star)$$
$$= \text{sign}(\alpha^{\star T} LK(x) + b^\star)$$

- Optimization algorithm :
  - Solution α* is given by an iterative scheme :

Initialization : $\alpha^{l=0} = y^{l=0} = 0 \qquad \tau = \frac{1}{\|Q\|},\ \sigma = \frac{1}{\|L\|}$

Iterate until convergence : l=0,1,2,...

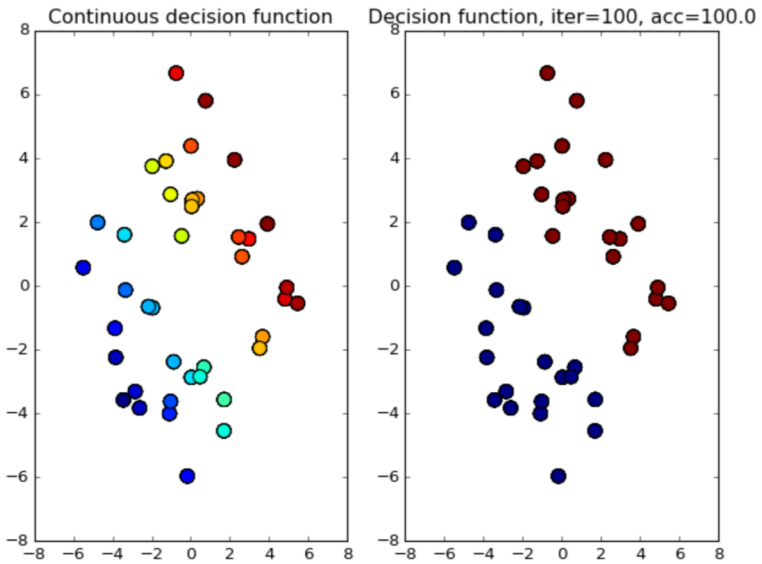
$$\alpha^{l+1} = P_{\cdot \geq 0}[(\tau Q + I_n)^{-1}(\alpha^k + \tau - \tau L y^k)]$$
$$y^{l+1} = y^k + \sigma L \alpha^{l+1}$$

$$\rightarrow \quad \alpha^\star = \alpha^\infty$$

Lab 1: Standard/linear SVM

- Run 02_classification/code01.ipynb

In [4]:
```
# Run Linear SVM

# Compute linear kernel, J, Q
Ker = Xtrain.dot(Xtrain.T)
L = np.diag(l_train)
l = l_train
Q = L.dot(Ker.dot(L))
```



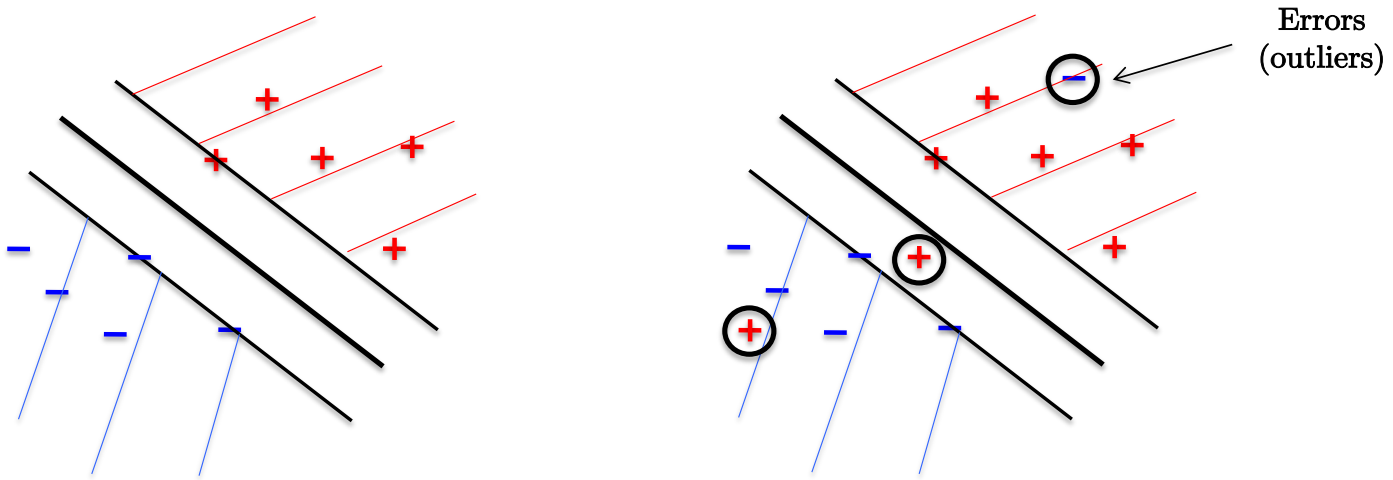
Outline

- Classification with graphs
  - Problem formulation
  - Linear SVM
  - Soft-margin SVM
  - Kernel techniques
  - Non-linear/kernel SVM
  - Graph SVM
  - Conclusion

No feature learning !

Soft-margin SVM[1]

- Assumption : Linear SVM supposes that data is linearly separable, i.e. there exists an hyper-plane perfectly separating the two classes. But real data has outliers.
- How to design SVM in the presence of outliers?
  - Soft-margin SVM : Find an hyper-plane that best separates the data (by maximizing the margin) while allowing as few outliers as possible.

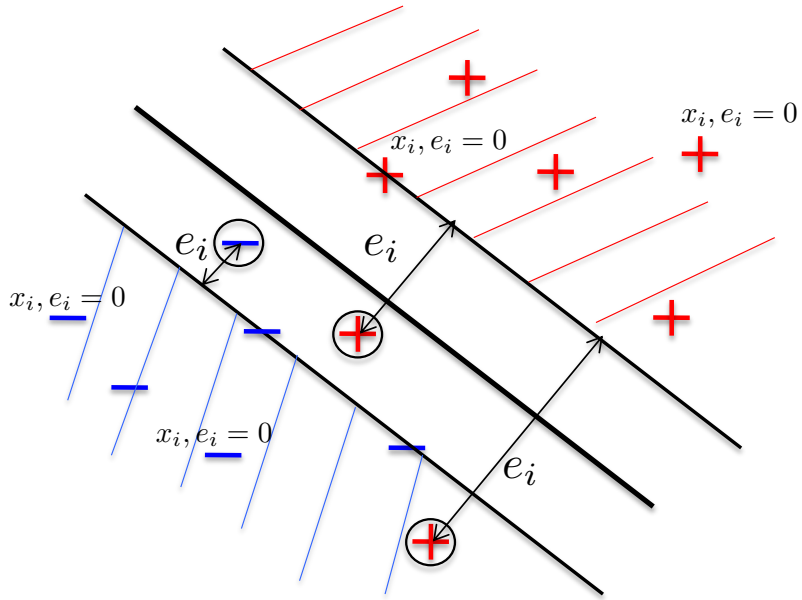


[1] Cortes, Vapnik, Support-vector networks, 1995

Modeling errors with slack variables

- Slack variables e_i measures the error for each data x_i to be an outlier.
- New optimization problem :

$$\min_{w,b} \underbrace{\|w\|_2^2 + \gamma \sum_{i=1}^{n} e_i}_{\text{Trade-off between large margin and small errors}} \quad \text{s.t.} \quad \ell_i.f_i \geq 1 - e_i, \;\; e_i \geq 0 \;\forall i \in V$$



Dual optimization problem

- **Primal problem :**

$$\min_{w,b} \ \|w\|_2^2 + \gamma \sum_{i=1}^{n} e_i \ \text{ s.t. } \ \ell_i . f_i \geq 1 - e_i, \ e_i \geq 0 \ \forall i \in V$$

- **Dual problem :** $\Updownarrow$

After computations..

$$\min_{0 \leq \alpha \leq \gamma} \frac{1}{2} \alpha^T Q \alpha - \langle \alpha, 1 \rangle \ \text{ s.t. } \ \langle \alpha, 1 \rangle = 0$$

with : $Q = LKL$

$L = \text{diag}(\ell_1, ..., \ell_n)$

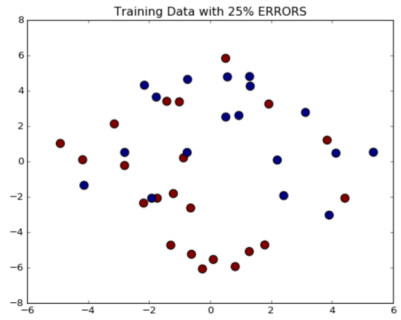
$K_{ij} = \langle x_i, x_j \rangle$

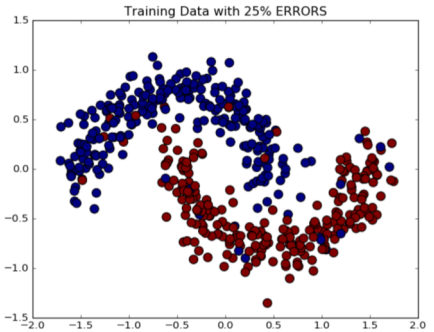
This (trivial) modification

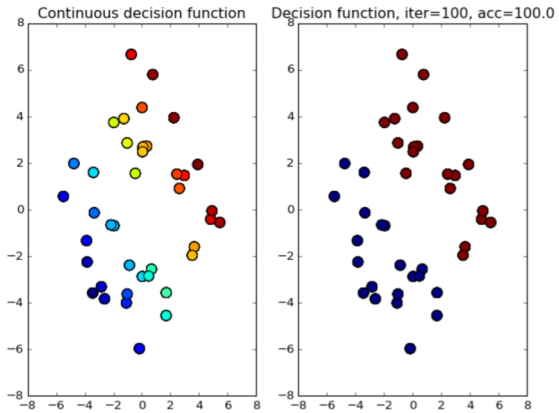
Lab 2: Soft-margin SVM

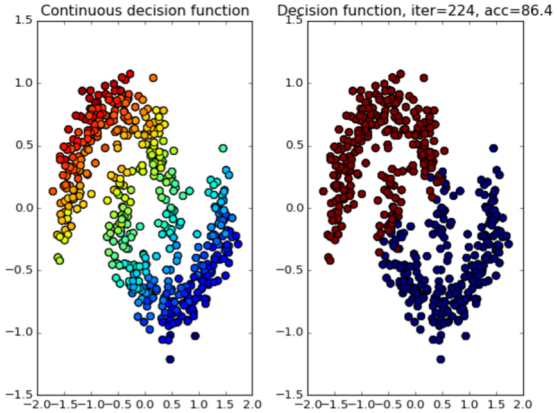
- Run 02_classification/code02.ipynb

```
In [5]: # Run soft SVM
        _,_,_,_ = compute_SVM(Xtrain,Cgt_train,l_train,'soft_linear',[0.1],Xtest,Cgt_test,[3,100])

        Run Linear SVM
        Construct Linear Kernel
```









Hinge loss

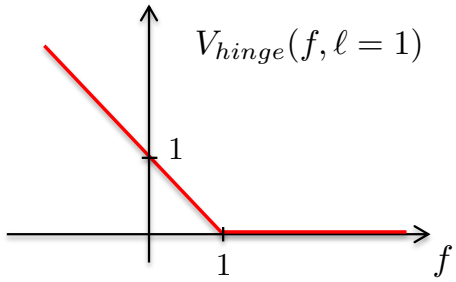
- **Primal problem :**

$$\min_{w,b} \ \|w\|_2^2 + \gamma \sum_{i=1}^{n} e_i \ \text{ s.t. } \ \ell_i.f_i \geq 1 - e_i, \ e_i \geq 0 \ \forall i \in V$$

$$\Updownarrow$$

$$\min_{w,b} \ \|w\|_2^2 + \gamma \sum_{i=1}^{n} V_{hinge}(f_i, \ell_i)$$

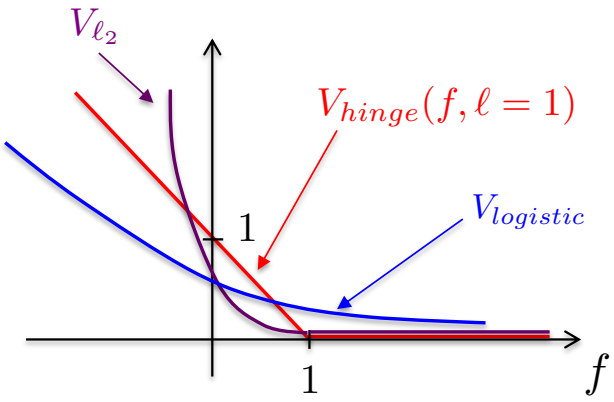
$$V_{hinge}(f_i, \ell_i) = \max(0, 1 - f_i.\ell_i)$$



Loss functions

- **Quadratic/L2 loss :** $V_{\ell_2}(f_i, \ell_i) = \begin{cases} (1 - f_i.\ell_i)^2 & \text{if } f_i.\ell_i \leq 1 \\ 0 & \text{otherwise} \end{cases}$
- **Elastic net loss :** $V_{ElasticNet}(f_i, \ell_i) = \begin{cases} (1 - f_i.\ell_i)^2 + \beta|1 - f_i.\ell_i| & \text{if } f_i.\ell_i \leq 1 \\ 0 & \text{otherwise} \end{cases}$
- **Huber loss :** $V_{Huber}(f_i, \ell_i) = \begin{cases} \frac{1}{2} - f_i.\ell_i & \text{if } f_i.\ell_i \leq 0 \\ \frac{1}{2}(1 - f_i.\ell_i)^2 & \text{if } 0 < f_i.\ell_i \leq 1 \\ 0 & \text{otherwise} \end{cases}$
- **Logistic loss :** $V_{Logistic}(f_i, \ell_i) = e^{1 - f_i.\ell_i}$
- **Hinge loss :** $V_{hinge}(f_i, \ell_i) = \max(0, 1 - f_i.\ell_i)$

  Popular loss function because of good properties/results[1]



[1] Rosasco, De Vito, Caponnetto, Are loss functions all the same? 2004

Outline

- Classification with graphs
  - Problem formulation
  - Linear SVM
  - Soft-margin SVM
  - Kernel techniques
  - Non-linear/kernel SVM
  - Graph SVM
  - Conclusion

No feature learning !

Kernel technique

- Most popular feature learning technique (until deep learning)
- Reproducing Kernel Hilbert Space[1] (RKHS) : A space associated to bounded, symmetric, positive semidefinite (PSD) operators called kernels K that can reproduce any smooth function f.
- Representer Theorem[2] : Any continuous and smooth function in a RKHS can be represented as a linear combination of the kernel function K evaluated at the data points :

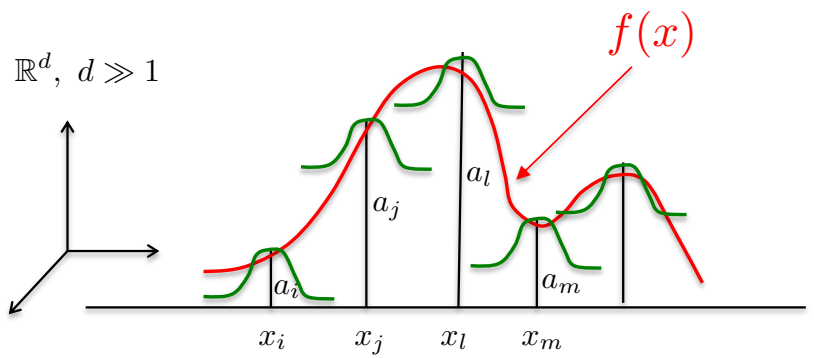
$$f(x) = \sum_{i=1}^{n} a_i K(x, x_i) + b$$

[1] Aronszajn, Theory of reproducing kernels, 1950
[2] Scholkopf, Herbrich, Smola, A generalized representer theorem, 2001

Representer theorem[1]

- The Representer Theorem is a tool to interpolate function in high-dimensional space :

$$f(x) = \sum_{i=1}^{n} a_i K(x, x_i) + b$$



with $K(x, x_i) = e^{-\|x - x_i\|_2^2 / \sigma^2}$

- Standard kernels
  - Linear kernel : $K(x, y) = \langle x, y \rangle$
  - Gaussian kernel : $K(x, y) = e^{-\|x - y\|_2^2 / \sigma^2}$
  - Polynomial kernel : $K(x, y) = (a\langle x, y \rangle + b)^c$

[1] Scholkopf, Herbrich, Smola, A generalized representer theorem, 2001

Feature map and kernel trick[1]

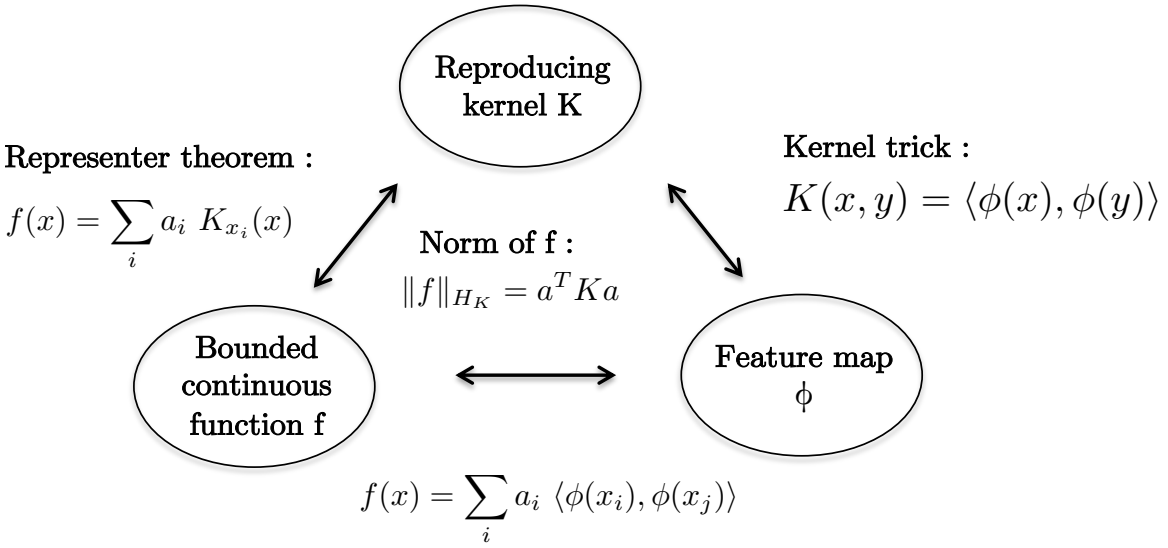
- Definition : Any feature map ϕ defines a reproducing kernel K (sym and PSD matrix)

$$\langle \phi(x), \phi(y) \rangle \overset{\text{def}}{=} K(x, y)$$

and inversely

$$K'(x, y) \overset{\text{def}}{=} \langle \phi'(x), \phi'(y) \rangle$$

- Overall



[1] Hofmann, Scholkopf, Smola, Kernel Methods in Machine Learning, 2008

Outline

- Classification with graphs
  - Problem formulation
  - Linear SVM
  - Soft-margin SVM
  - Kernel techniques
  - Non-linear/kernel SVM
  - Graph SVM
  - Conclusion

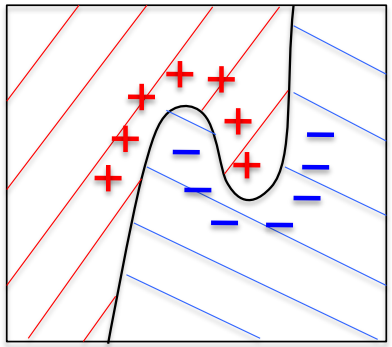
No feature learning !

Non-linear/kernel SVM[1]

- Assumption : Linear/soft SVM assume data is linearly separable (up to a few outliers).
- But for several real-world data, the hyper-plane assumption is not satisfied, i.e. data is not linearly separable.
  - A better separation is then a non-linear hyperplane, i.e. a hyper-surface with curvature.
- Kernel trick : Project data into a higher-dimensional space with feature map ϕ where the data is linearly separable.
- Decision function in high-dim :

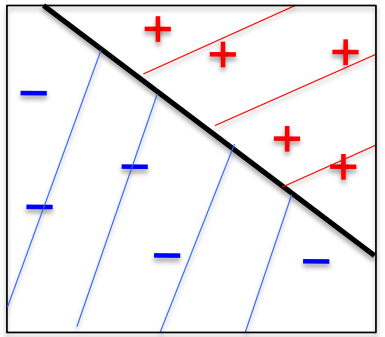
$$\left.\begin{array}{rcl} f(x) = \langle w, x\rangle + b & \xrightarrow{\phi} & f(x) = \langle w, \phi(x)\rangle + b \\ w = \sum_i \alpha_i \ell_i x_i & \xrightarrow{\phi} & w = \sum_i \alpha_i \ell_i \phi(x_i) \end{array}\right\}$$

$$\longrightarrow \quad f(x) = \sum_i \alpha_i \ell_i \langle \phi(x), \phi(x_i)\rangle + b = \sum_i \alpha_i \ell_i \underbrace{K(x_i, x)}_{\text{Kernel}} + b$$



Non-linear separator

ϕ



Linear separator

[1] Boser, Guyon, Vapnik, A training algorithm for optimal margin classifiers, 1992

Optimization

- **Dual problem :**

$$\min_{0 \leq \alpha \leq \gamma} \frac{1}{2} \alpha^T Q \alpha - \langle \alpha, 1 \rangle \quad \text{s.t.} \quad \langle \alpha, 1 \rangle = 0$$

$$\text{with} \quad Q = LKL$$

$$L = \text{diag}(\ell_1, ..., \ell_n)$$

$$K_{ij} = \langle x_i, x_j \rangle \quad \xrightarrow{\phi} \quad K_{ij} = \langle \phi(x_i), \phi(x_j) \rangle$$

Same optimization algorithm as before !

- **Reminder : We never compute explicitly ϕ and the products $\langle \phi(x_i), \phi(x_j) \rangle$ but only the Kernel matrix :**

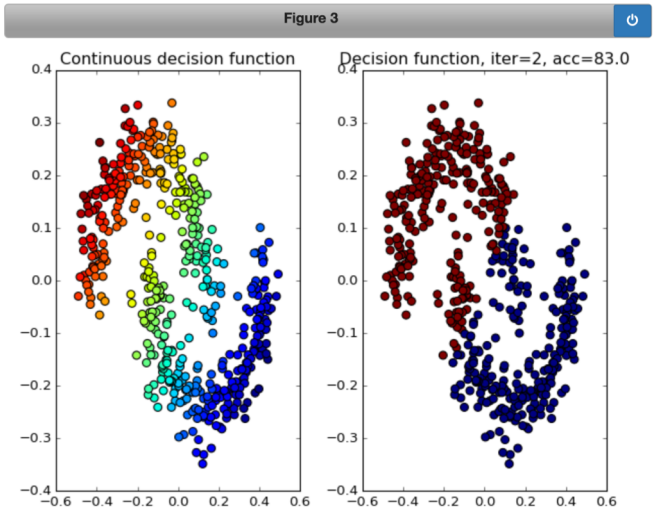
$$K(x, y) = \begin{cases} (a\langle x, y \rangle + b)^c \\ e^{-\|x-y\|_2^2 / \sigma^2} \end{cases}$$

Lab 3: Kernel/non-linear SVM

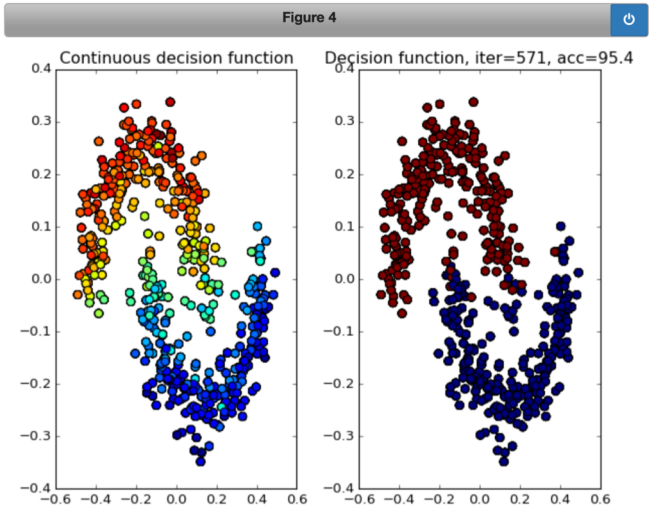
- Run classification/code03.ipynb

```
In [6]: # Run Kernel SVM
        _,_,_,_ = compute_SVM(Xtrain,Cgt_train,l_train,'gaussian_kernel',[1.0,0.5],Xtest,Cgt_test,[4,50])

Run Kernel SVM
Construct Gaussian Kernel
```

```
Run Linear SVM
Construct Linear Kernel
```




```
Run Kernel SVM
Construct Gaussian Kernel
```



Supervised learning

- Generalization

$$\min_w \ \|w\|_2^2 + \gamma \sum_{i=1}^{n} V_{hinge}(f_i, \ell_i)$$

$$\downarrow$$

$$\min_{f \in H_K} \ \|f\|_{H_K}^2 + \gamma \sum_{i=1}^{n} V_{loss}(f_i, \ell_i)$$

Regularity of f

Error for inaccurate predictions

Trade-off

$$\|f\|_{H_K}^2 = \|w\|_2^2 \quad \text{for} \quad f(x) = \langle w, x \rangle$$

Representer theorem : $$f(x) = \sum_{i=1}^{n} a_i K(x, x_i)$$

Norm of in RKHS : $$\|f\|_{H_K}^2 = \langle f, f \rangle_{H_K} = \sum_{ij} f_i f_j K_{ij} = f^T K f$$

Outline

- Classification with graphs
  - Problem formulation
  - Linear SVM
  - Soft-margin SVM
  - Kernel techniques
  - Non-linear/kernel SVM
  - Graph SVM
  - Conclusion

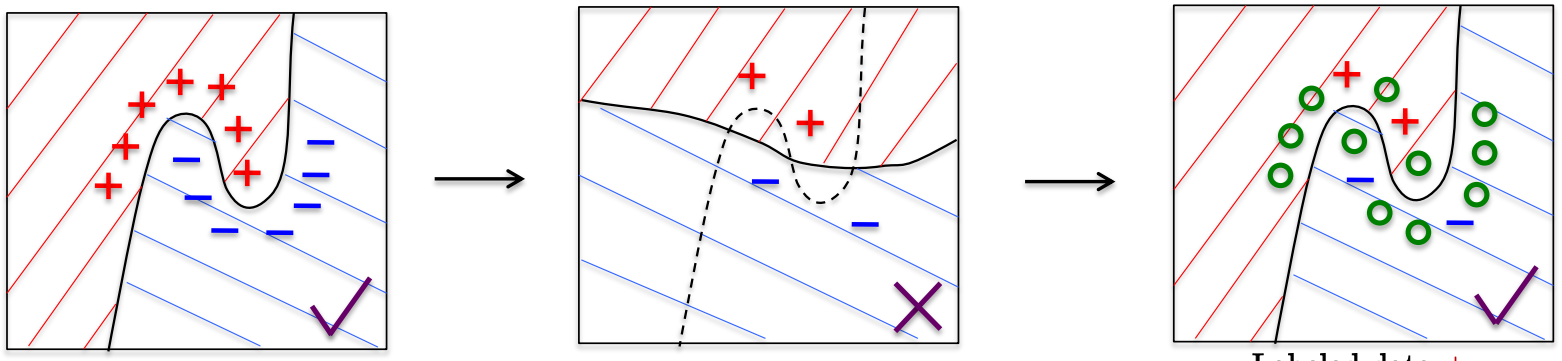
No feature learning !

Semi-supervised learning

- It is time/money consuming to label data.
- However it is cheap to collect unlabeled data.
  - Unlabeled data also comes with information s.a. geometric structure.
- How to leverage simultaneously labeled and unlabeled data to design better classification function?
  - Semi-supervised learning is the framework to combine labeled and unlabeled data.
  - It is quite useful in the case of few labels :
    - That is, n $\ll$ m where n / m are resp. the number of labelled / unlabelled data.
    - Extreme case is one label per class.

Semi-supervised learning with Graph SVM

- Unlabeled data hold information, particularly geometric statistical structure.
  - Graph SVM[1] leverages the geometric manifold assumption.



Labeled data +,-
Supervised Learning

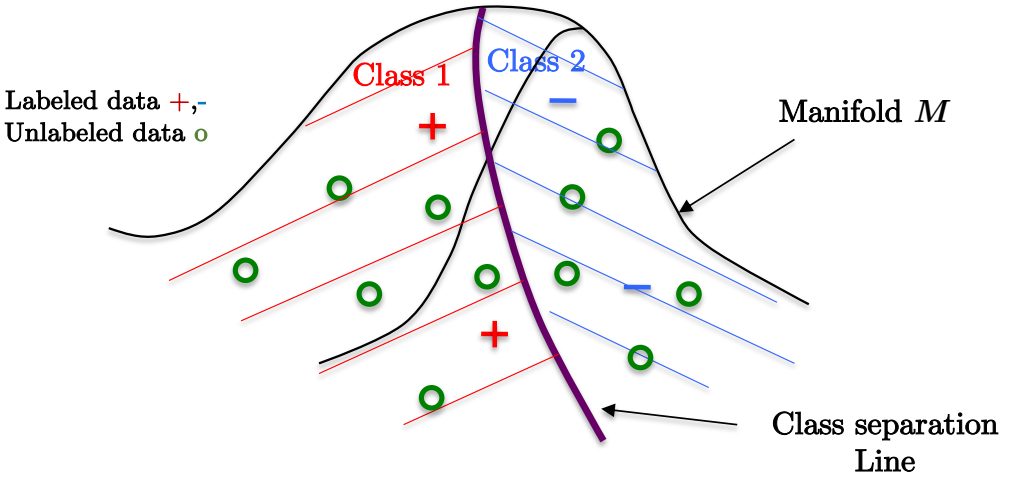
Labeled data +,-
Supervised Learning

Labeled data +,-
Unlabeled data o
Semi-Supervised Learning

[1] Belkin, Niyogi, Sindhwani,Manifold regularization: A geometric framework for learning from labeled and unlabeled examples, 2006

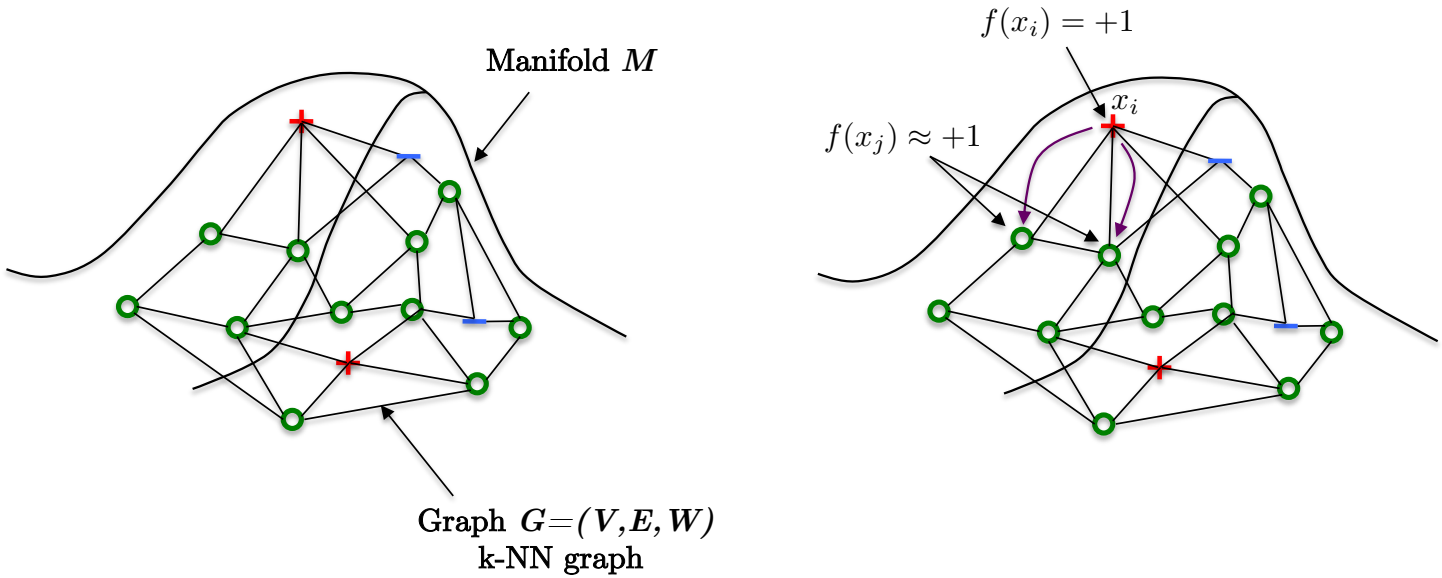
Manifold assumption

- Observe that geometry of data is independent of labels.
- Labeled and unlabeled data are assumed to lie on a manifold, where tasks s.a. classification, clustering etc are carried out.



Manifold assumption

- How to introduce the manifold geometry in the SVM objective loss ?
  - First, the manifold M is approximated with a neighborhood graph, here a k-NN graph.
  - Second, we will add a regularization term that forces the classification function f to be smooth on the manifold approximated by the graph.



Optimization algorithm

- Optimization problem :

$$\min_{f \in H_K} \|f\|_{H_K}^2 + \gamma_l \sum_{i=1}^{n} V_{loss}(f_i, \ell_i) + \gamma_G \underbrace{\int_{\mathcal{M}} |\nabla f|^2}$$

Dirichlet energy:
(1) It forces f to be smooth on M
(2) Derivative is $\Delta f = 0$ (heat diffusion)

- Discretization of Dirichlet on graphs :

$$\int_{\mathcal{M}} |\nabla f|^2 \approx \sum_{ij} W_{ij} |f(x_i) - f(x_j)|^2 = f^T \mathcal{L} f$$

Laplacian operator

- This additional loss implicitly enforces the classification function f to lie on the manifold parametrized by the Laplacian eigenvectors[1].

[1] Belkin, Niyogi, Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering, 2001

Optimization algorithm

- **Semi-supervised/Laplacian SVM[1] :**

$$\min_{f \in H_K} \|f\|^2_{H_K} + \gamma_l \sum_{i=1}^{n} V_{hinge}(f_i, \ell_i) + \gamma_G f^T \mathcal{L} f$$

$$\Downarrow$$

$$f(x) = \text{sign}\Big(\sum_{i=1}^{n} a_i^\star K(x, x_i) \Big)$$

$$a^\star = (I + \gamma_G \mathcal{L} K)^{-1} H L \alpha^\star$$

$$\alpha^\star = \arg \min_{0 \leq \alpha \leq \gamma_l} \frac{1}{2} \alpha^T Q \alpha - \langle \alpha, 1 \rangle \quad \text{s.t.} \quad \langle \alpha, 1 \rangle = 0$$

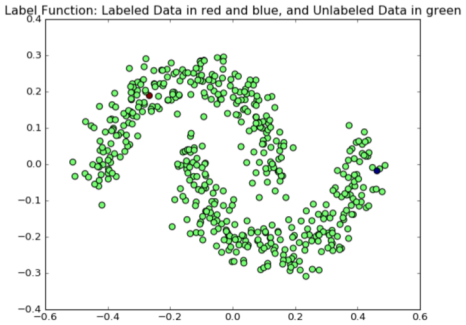
$$\text{with} \quad Q = LHK(I + \gamma_G \mathcal{L} K)^{-1} HL$$

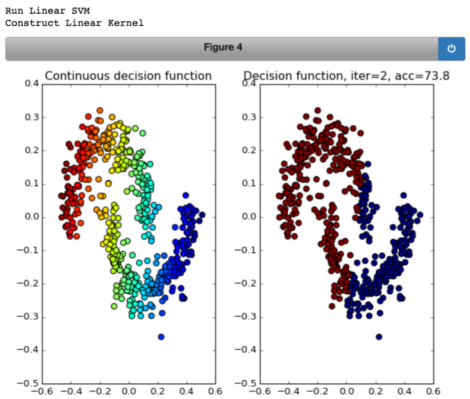
[1] Belkin, Niyogi, Sindhwani,Manifold regularization: A geometric framework for learning from labeled and unlabeled examples, 2006

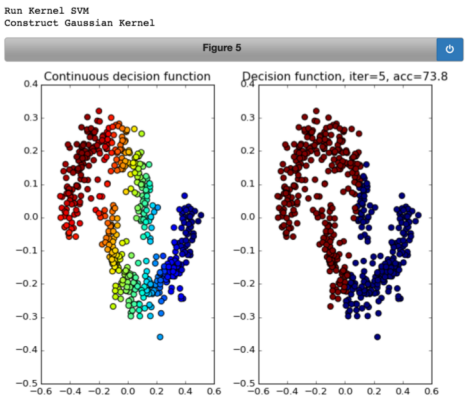
Lab 4: Graph SVM

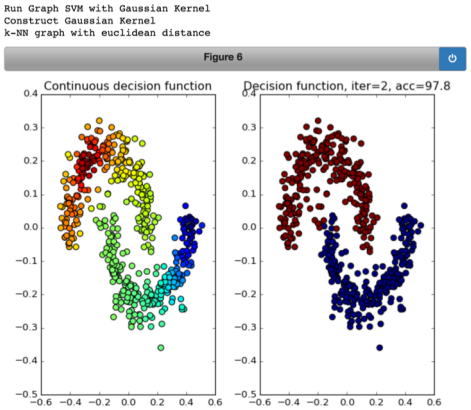
- Run classification/code04.ipynb

```
# Run Graph SVM
_,_,_,_ = compute_SVM(Xtrain,Cgt_train,l_train,'gaussian_kernel',[1.0,0.114],Xtest,Cgt_test,[6,50],
                      'euclidean',[10,24.0])
```







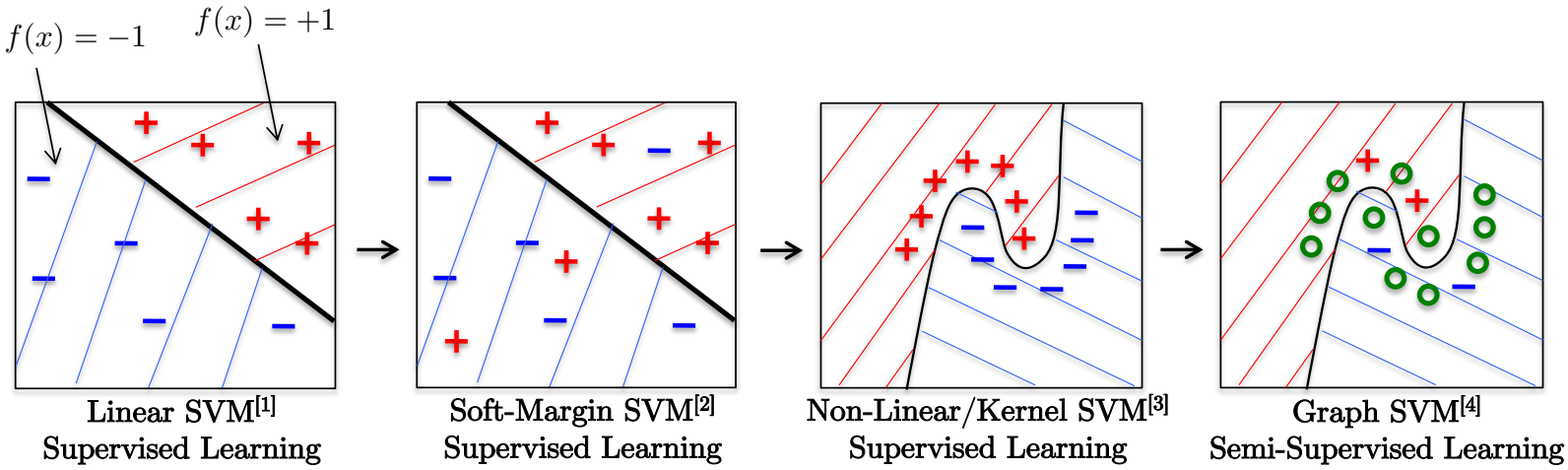


Outline

- Classification with graphs
  - Problem formulation
  - Linear SVM
  - Soft-margin SVM
  - Kernel techniques
  - Non-linear/kernel SVM
  - Graph SVM
  - Conclusion

No feature learning !

Development of SVM techniques



[1] Vapnik, Chervonenkis, On a perceptron class, 1964
[2] Cortes, Vapnik, Support-vector networks, 1995
[3] Boser, Guyon, Vapnik, A training algorithm for optimal margin classifiers, 1992
[4] Belkin, Niyogi, Sindhwani,Manifold regularization: A geometric framework for learning from labeled and unlabeled examples, 2006

Summary

- General supervised and semi-supervised optimization techniques :

$$\min_{f \in H_K} \|f\|^2_{H_K} + \gamma_l \sum_{i=1}^{n} V_{loss}(f_i, \ell_i) + \gamma_G R_{graph}(f)$$

Regularity of f

Error for inaccurate predictions

Graph regularization for unlabeled data

$$V_{loss} = \begin{cases} \text{Hinge} \\ L_2 \\ L_1 \\ \text{Huber} \\ \text{Logistic} \end{cases} \qquad R_{graph}(\cdot) = \begin{cases} \text{Dirichlet: } \|\nabla_G \cdot\|_2^2 \\ \text{Total Variation: } \|\nabla_G \cdot\|_1 \\ \text{Wavelets: } \|D_{wavelets} \cdot\|_2^2 \end{cases}$$

Outline

- Graph clustering
- Classification with graphs
- **Recommendation with graphs**
- Dimensionality reduction with graphs

No feature learning !

Outline

- Recommendation with graphs
  - Recommender systems
  - Google PageRank
  - Collaborative recommendation
  - Content recommendation
  - Hybrid systems
  - Conclusion

No feature learning !

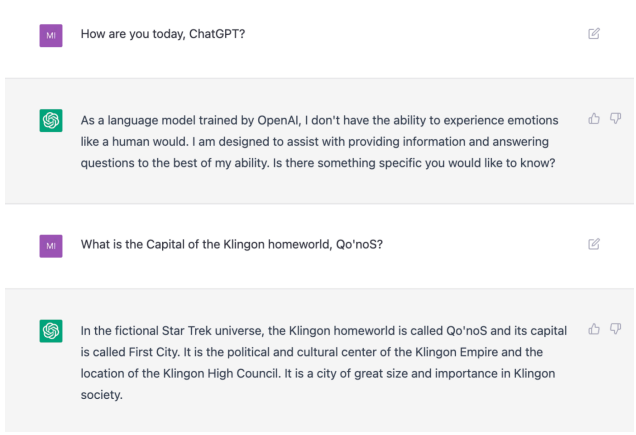
Outline

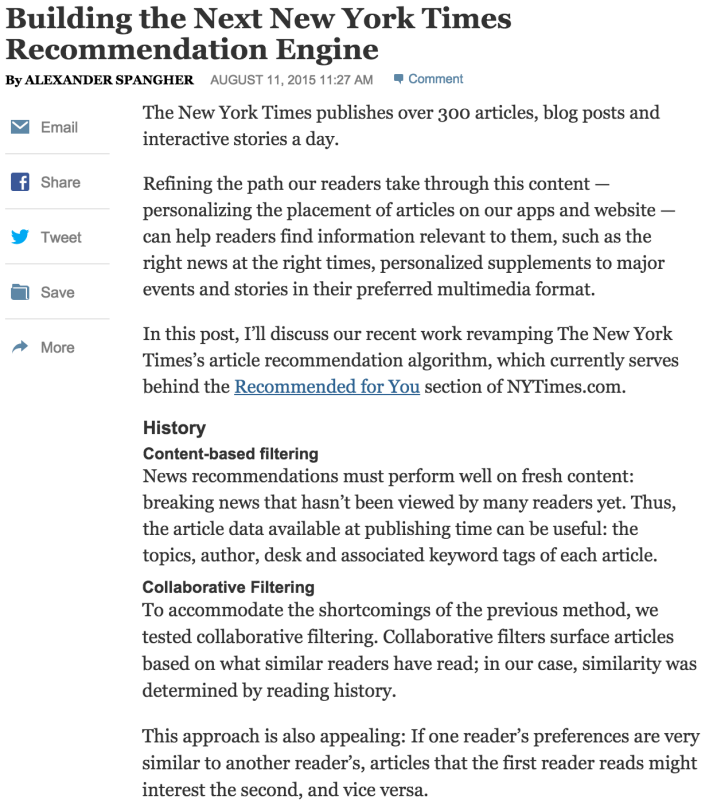
- Recommendation with graphs
  - Recommender systems
  - Google PageRank
  - Collaborative recommendation
  - Content recommendation
  - Hybrid systems
  - Conclusion

No feature learning !

Recommender systems

- Recommendation is a central part of any intelligent system.
- Examples
  - Google search engine recommends webpages on internet
  - Recommending movies on Netflix
  - Friends on Facebook
  - Products on Amazon
  - Jobs on LinkedIn
  - Articles on NY Times website
  - ChatGPT 😊
  - Etc





Building the Next New York Times Recommendation Engine

By ALEXANDER SPANGHER AUGUST 11, 2015 11:27 AM Comment

Email Share Tweet Save More

The New York Times publishes over 300 articles, blog posts and interactive stories a day.

Refining the path our readers take through this content — personalizing the placement of articles on our apps and website — can help readers find information relevant to them, such as the right news at the right times, personalized supplements to major events and stories in their preferred multimedia format.

In this post, I'll discuss our recent work revamping The New York Times's article recommendation algorithm, which currently serves behind the Recommended for You section of NYTimes.com.

History

Content-based filtering

News recommendations must perform well on fresh content: breaking news that hasn't been viewed by many readers yet. Thus, the article data available at publishing time can be useful: the topics, author, desk and associated keyword tags of each article.

Collaborative Filtering

To accommodate the shortcomings of the previous method, we tested collaborative filtering. Collaborative filters surface articles based on what similar readers have read; in our case, similarity was determined by reading history.

This approach is also appealing: If one reader's preferences are very similar to another reader's, articles that the first reader reads might interest the second, and vice versa.

Outline

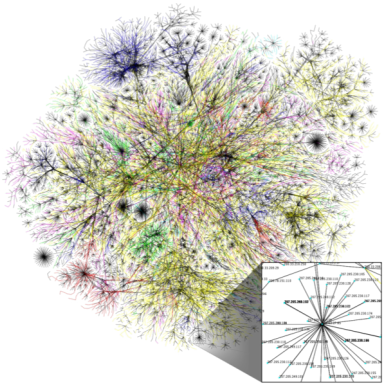
- Recommendation with graphs
  - Recommender systems
  - Google PageRank
  - Collaborative recommendation
  - Content recommendation
  - Hybrid systems
  - Conclusion

No feature learning !

Google PageRank[1]

- PageRank is an algorithm that ranks internet's webpages.
  - It was at the core of Google Search Engine until deep learning.
  - It revolutionized recommendation as ranking was done manually by humans previously.
    - 1998 : The size of internet was 2.4M of webpages.
    - 2023 : The number of webpages is 30B !

Source: Google



Source: Wikipedia

[1] Page, Brin, Motwani, Winograd, The PageRank citation ranking: Bringing order to the web, 1999

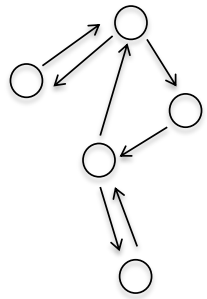
PageRank

- PageRank is a graph-based recommender algorithm that is mathematically well defined and memory/speed efficient.
- PageRank sorts the nodes of a directed graph G using the stationary state of G, which encodes the importance of the nodes in terms of their connectivity influence (nodes with in-coming edges).
- The stationarity state and other modes of vibration of a network are computed with the EVD matrix factorization of the adjacency matrix A :

$$Ax_l = \lambda_l x_l$$

Eigenvectors/
Mode of vibration

Eigenvalues/
Frequency of vibration



- Perron-Frobenius theorem[1,2] defines the stationary state of A.

[1] Perron, Zur Theorie der Matrices, 1907
[2] Frobenius, Ueber Matrizen aus nicht negativen Elementen, 1912

Perron-Frobenius theorem[1,2]

- Given a graph G=(V,E,A) defined by a stochastic and irreducible adjacency matrix A (later defined), the Perron-Frobenius theorem states that the largest left eigenvector (with eigenvalue 1) is the stationary state solution, a.k.a. PageRank :

$$x_{\max}^T A = x_{\max}^T \lambda_{\max}^T = x_{\max}^T$$

Max left eigenvector — Eigenvalue=1

- Consequence : x_{PageRank} is the solution of the following (left) eigenproblem

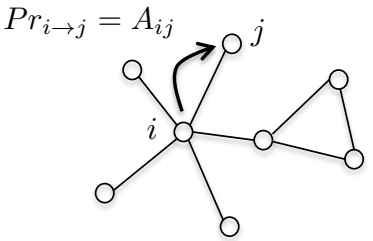
$$x^T A = x^T$$

[1] Perron, Zur Theorie der Matrices, 1907
[2] Frobenius, Ueber Matrizen aus nicht negativen Elementen, 1912

Stochastic matrix

- A matrix is stochastic when the rows are normalized as probability density functions.

$$\mathrm{A} = \begin{bmatrix} \\ \\ \end{bmatrix} \sum_j A_{ij} = 1 \quad \Leftrightarrow \quad A1 = 1$$



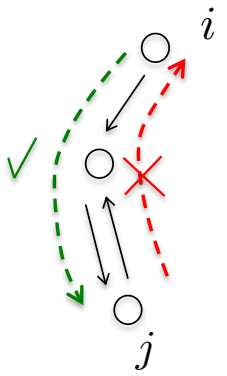
- Interpretation : $\mathrm{A_{ij}}$ = Probability to move from vertex i to vertex j.

- Make any matrix A a stationary matrix :

$$A \leftarrow D^{-1}A \quad \text{for} \quad D_{ii}^{-1} = \begin{cases} (\sum_j A_{ij})^{-1} & \text{if } i\text{th row} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Irreducible matrix

- A matrix is irreducible when it represents a strongly connected graph.
  - A graph is strongly connected when there exists for any pair of nodes (i,j)
    - A directed path from i to j,
    - A directed path from j to i.



- Make any matrix A an irreducible matrix :

$$A_{\text{si}} \leftarrow \alpha D^{-1} A + (1-\alpha)\frac{I_N}{N}$$

Matrix of 1s

Number of nodes

Stochastic and irreducible matrix
= Full matrix

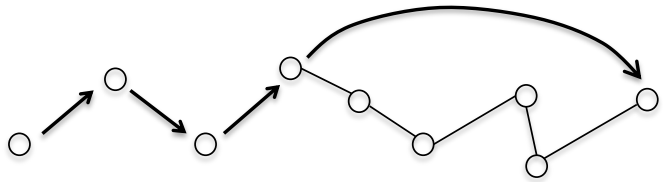
Original matrix
= Sparse matrix

Original choice : α=0.85

Modeling internet users

- Term (1-α)I_N/N is equivalent to a random surfer/user who can go to any webpage.
- Terms αD^{-1}A + (1-α)I_N/N models a user who follows the internet structure α% of the time and for (1-α)% of the time, the user clicks to a webpage that has no connection to the previous page.

$$A_{\text{si}} \leftarrow \alpha D^{-1} A + (1-\alpha)\frac{I_N}{N}$$



Spectral algorithm

- Direct solution : Solve the following EVD problem

$$x^T A_{\text{si}} = x^T \quad \text{Left eigenproblem}$$
$$\Updownarrow$$
$$(x^T A_{\text{si}})^T = (x^T)^T$$
$$\Updownarrow$$
$$A_{\text{si}}^T x = x \quad \text{Right eigenproblem}$$

- Limitations : Computing eigensolution is
  - Slow O(N^2)
  - Memory consuming O(N^2)
  - Not parallelizable (eigenvectors are solutions of global linear system)

    ⇒ EVD cannot scale to large networks like internet (w/ billions of nodes).

Power method[1,2]

- Solution of $A_{si}^{T}x=x$ can be found with a fixed-point iterative scheme :

$$x^{k+1} = A_{\text{si}}^{T} x^{k}$$

- At convergence :

$$x^{\infty} = A_{\text{si}}^{T} x^{\infty} \quad \rightarrow \quad x^{\infty} = x_{\text{PageRank}}$$

- Algorithm[2]
  - Initialization : $x^{k=0} = \frac{1_N}{N}$
  - Iterate until convergence : $x^{k+1} = \alpha D^{-1} A^{T} x^{k} + (1-\alpha)\frac{1_N}{N}$

[1] Mises, Pollaczek-Geiringer, Praktische Verfahren der Gleichungsauflösung, 1929
[2] Page, Brin, Motwani, Winograd, The PageRank citation ranking: Bringing order to the web, 1999

Algorithm properties

- Memory complexity
  - EVD : $A_{\text{si}}^T x = x$
  - Power Method : $x^{k+1} = \alpha D^{-1} A^T x^k + (1-\alpha)\frac{1_N}{N}$

Full matrix O(N²)

Sparse matrix O(E)

- Speed complexity
  - EVD : O(N³)
  - Power Method :
    - O(E.K), K is the number of iterations to converge to a precision ε as follows

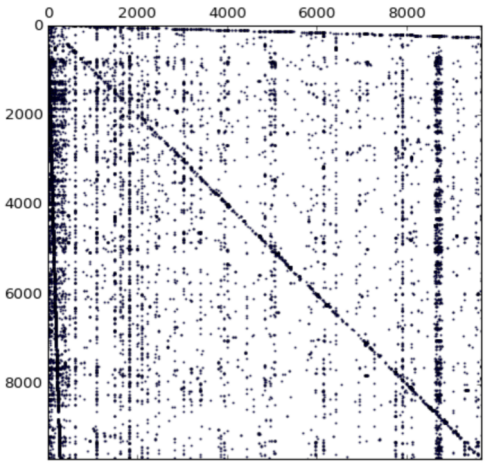
$$\|x^{k+1} - x^k\|_1 \leq \varepsilon \quad \text{for} \quad K = \frac{\log_{10}\varepsilon}{\log_{10}\alpha} = \begin{cases} 85 & \text{for } \alpha = 0.85 \\ 1833 & \text{for } \alpha = 0.99 \end{cases}, \text{ and } \varepsilon = 10^{-6}$$

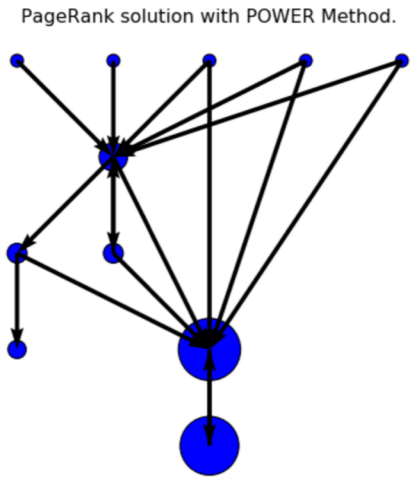
    - Highly parallelizable (matrix-vector multiplications)
    - Note: Power method is the best technique to compute the largest or smallest eigenvector.

Lab 1: PageRank

- Run 03_recom/code01.ipynb

```
# Power Method

# Initialization
x = e
diffx = 1e10
k = 0

# Iterative scheme
start = time.time()
alpha = 0.85
while (k<1000) & (diffx>1e-6):

    # Update iteration
    k += 1

    # Update x
    xold = x
    x = alpha* P.dot(x) + e.dot( alpha* a.T.dot(x) + (1.0-alpha) )

    # Stopping condition
    diffx = np.linalg.norm(x-xold,1)
```





Outline

- Recommendation with graphs
  - Recommender systems
  - Google PageRank
  - Collaborative recommendation
  - Content recommendation
  - Hybrid systems
  - Conclusion

No feature learning !

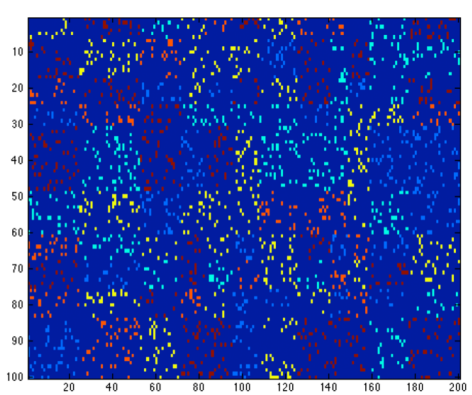
Collaborative recommendation

- Two types of recommender system that predicts the rating of an item s.a. a movie or a product :
  - Collaborative recommendation/filtering
  - Content recommendation/filtering
- Collaborative Filtering
  - A famous example is the 2009 1M$ Netflix Prize[1]
  - Netflix prize was a competition to design the best algorithm that can predict user ratings based on a sparse number of previously rated movies.
  - Dataset : 480,189 users, 17,770 movies, 100,480,507 ratings ⇒ Available rating is 0.011% only

[1] https://en.wikipedia.org/wiki/Netflix_Prize

Formalization

- Problem : Given a few ratings/observations M_{ij} of movie j and user i, find a matrix X that best fits the ratings.

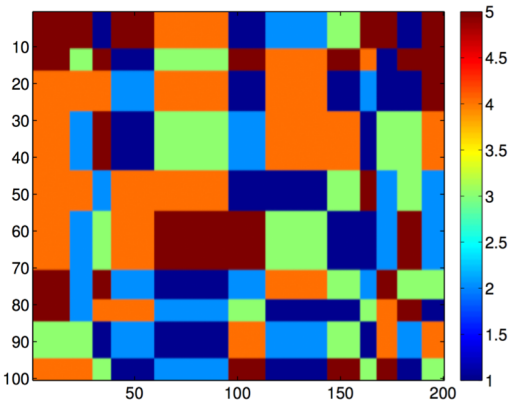


M

Recommendation

=

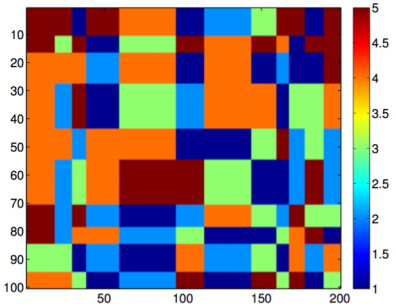
Matrix completion



X

Low-rank recommendation

- We suppose that a rating matrix is low-rank.
- A low-rank matrix is a matrix with several columns and rows being linearly dependent.
  - The rank of a matrix is the number of independent rows and columns.
- Low-rank assumption : This hypothesis has been valid for several real-world datasets.
  - For example, Netflix :
    - There exist communities of users who rate movies the same way.
    - There are groups of movies that receive the same ratings.
  - Same assumptions for Amazon (users, products), LinkedIn (users, jobs), Facebook (users, ads), etc.

$X =$ 

$\#$ independent rows $= 13$
$\#$ independent cols $= 15$
$\Rightarrow \text{rank}(X) = \max(13,15) = 15$

Modeling low-rank recommendation

$$\min_X \underbrace{\text{rank}(X)}_{\text{Combinatorial NP-hard problem}} \quad \text{s.t.} \quad \begin{cases} X_{ij} = M_{ij} & \forall ij \in \Omega_{obs} \\ X_{ij} = M_{ij} + n_{ij} & \forall ij \in \Omega_{obs} \end{cases}$$

Noiseless case
(Observations are clean)

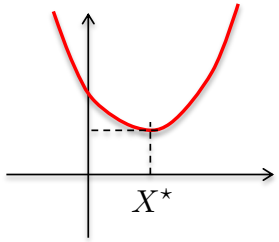
Noisy case
(Observations may be corrupted)

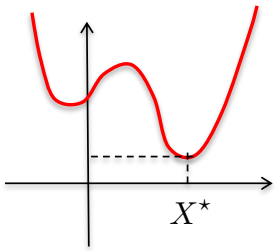
Relaxation needed

Convex Relaxation

Non-Convex Relaxation

Nuclear norm optimization[1]





Matrix factorization[2,3]

[1] Candes, Recht, Exact matrix completion via convex optimization, 2009
[2] Luo, Zhou, Xia, Zhu, An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems, 2014
[3] Sarwar, Karypis, Konstan, Riedl, Application of dimensionality reduction in recommender system-a case study, 2000

Unconstrained low-rank optimization

- From constrained to unconstrained problem[1] :

$$\min_X \ \text{rank}(X) \quad \text{s.t.} \quad X_{ij} = M_{ij} + n_{ij} \ \forall ij \in \Omega_{obs}$$

$$\Updownarrow$$

$$\min_X \ \underbrace{\text{rank}(X)}_{\text{promotes low-rank}} + \underbrace{\frac{\lambda}{2}\|I_{obs} \circ (X - M)\|_F^2}_{\text{promotes data fidelity and robustness}} \quad \text{with} \quad (I_{obs})_{ij} = \begin{cases} 1 & \text{if } ij \in \Omega_{obs} \\ 0 & \text{otherwise} \end{cases}$$

Combinatorial NP-hard problem

[1] Candes, Recht, Exact matrix completion via convex optimization, 2009

Convex optimization[1]

- A powerful and well-studied tool in mathematics and machine learning (active research field)
- NP-hard unconstrained combinatorial problem :

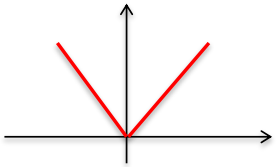
$$\min_X \ \text{rank}(X) + \frac{\lambda}{2}\|I_{obs} \circ (X - M)\|_F^2 \qquad \text{with} \quad (I_{obs})_{ij} = \begin{cases} 1 & \text{if } ij \in \Omega_{obs} \\ 0 & \text{otherwise} \end{cases}$$

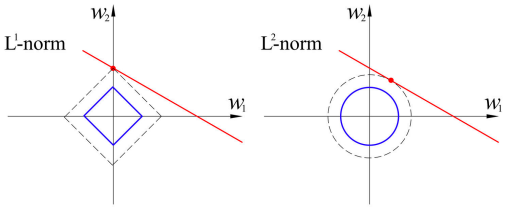
- Continuous, non-smooth and convex relaxation[2] :

$$\min_X \underbrace{\|X\|_\star}_{\text{Nuclear norm}} + \frac{\lambda}{2}\|I_{obs} \circ (X - M)\|_F^2$$

$$\sum_{k=1}^{p=\min(m,n)} |\sigma_k(X)| \qquad \text{Singular values with SVD :} \quad X = U\Sigma V^T$$

$$\Sigma = \text{diag}(\sigma_1, ..., \sigma_p)$$



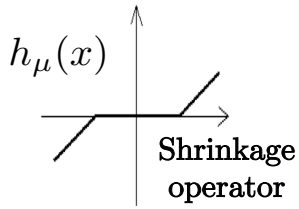


[1] Boyd, Boyd, Vandenberghe, Convex optimization, 2004
[2] Candes, Recht, Exact matrix completion via convex optimization, 2009

Primal-dual optimization[1]

- Algorithm
  - Initialization : $X^{k=0} = M \quad Y^{k=0} = 0$
  - Iterate until convergence between primal variable X and dual variable Y :

$$\begin{cases} Y^{k+1} = Y^k - \sigma U h_{1/\sigma}(\Sigma) V^T & \text{with} \quad \underset{\text{SVD}}{U\Sigma V^T} = Y^k + \sigma^k X^k \\ X^{k+1} = \dfrac{X^k - \tau^k Y^{k+1} + \tau^k \lambda M}{1 + \tau^k \lambda I_{obs}} \end{cases}$$



  - Convergence rate (optimal) :

$$F(x^k) - F(x^\star) = O(\frac{1}{k^2})$$

[1] Liu, Vandenberghe, Interior-point method for nuclear norm approximation with application to system identification, 2010

Algorithmic properties

- Advantages
  - Well-posed optimization algorithm
  - Unique and stable solution (independent of the initialization)

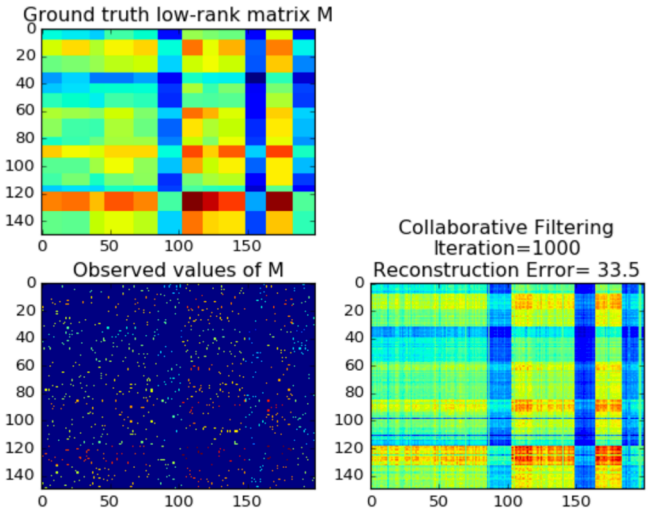
- Limitations
  - Complexity is dominated by SVD, i.e. $O(N^3)$.
  - Memory requirement is $O(N^2)$.

    $\Rightarrow$ Convex algorithms cannot scale up to large-scale datasets.

Lab 2: Collaborative filtering

- Run 03_recom/code02.ipynb

```
# Collaborative Filtering / Low-Rank Approximation by Nuclear Norm

# Indentify zero columns and zero rows in the data matrix X
idx_zero_cols = np.where(np.sum(Otraining,axis=0)<1e-9)[0]
idx_zero_rows = np.where(np.sum(Otraining,axis=1)<1e-9)[0]
nb_zero_cols = len(idx_zero_cols)
nb_zero_rows = len(idx_zero_rows)

# Regularization parameter
OM = O*M
normOM = np.linalg.norm(OM,2)
lambdaNuc = normOM/4.
lambdaDF = 1e3
```



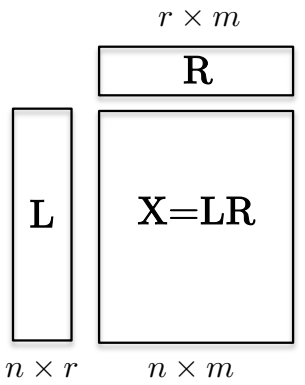
Matrix factorization

- NP-hard unconstrained combinatorial problem :

$$\min_X \ \text{rank}(X) + \frac{\lambda}{2}\|I_{obs} \circ (X - M)\|_F^2 \qquad \text{with} \qquad (I_{obs})_{ij} = \begin{cases} 1 & \text{if } ij \in \Omega_{obs} \\ 0 & \text{otherwise} \end{cases}$$

- Relaxation required
  - Continuous, non-smooth and convex relaxation with nuclear norm[1]
  - Continuous, smooth and non-convex relaxation with matrix factorization techniques like SVD[2] or NMF[3] (non-negative matrix factorization)

$$\min_{L,R} \ \frac{1}{2}\|L\|_F^2 + \frac{1}{2}\|R\|_F^2 + \frac{\lambda}{2}\|I_{obs} \circ (LR - M)\|_F^2$$



$r \ll n, m$

[1] Candes, Recht, Exact matrix completion via convex optimization, 2009
[2] Luo, Zhou, Xia, Zhu, An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems, 2014
[3] Sarwar, Karypis, Konstan, Riedl, Application of dimensionality reduction in recommender system-a case study, 2000

Algorithmic properties

- Advantages
  - Optimization problem is smooth and quadratic : Fast opt with conjugate gradient, Newton, etc.
  - The objective is differentiable : Stochastic gradient descent technique can be applied for large-scale recommender systems.
  - Monotonicity property : Several classes of factorization algorithm s.a. NMF are guaranteed to be monotonic, $E^{k+1} \leq E^k$, $\forall k$

- Limitations
  - Non-convex : Local minimizers (good initialization is critical)
  - New hyper-parameter : The rank value r needs to be selected.

Outline

- Recommendation with graphs
  - Recommender systems
  - Google PageRank
  - Collaborative recommendation
  - Content recommendation
  - Hybrid systems
  - Conclusion

No feature learning !

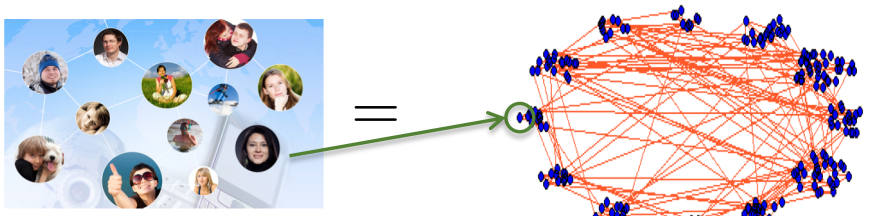
Content recommendation[1]

- Collaborative recommendation focuses on low-rank approximation of ratings.
- Content recommendation uses similarities between users and between products to predict the rating.
- Task formulation
  - Given a few ratings M_{ij} of product j and user i, a set of user features and product features, find a matrix X that best fits the ratings and satisfies the similarities between users and products.
- User features and product features:
  - User features/attributes : Gender, age, job, hobbies, etc
  - Product features/attributes : Domain, price, production date, size, etc

[1] Pazzani, Billsus, Content-based recommendation systems, 2007

Similarity encoding

- Similarities between users and between products are encoded with graphs.

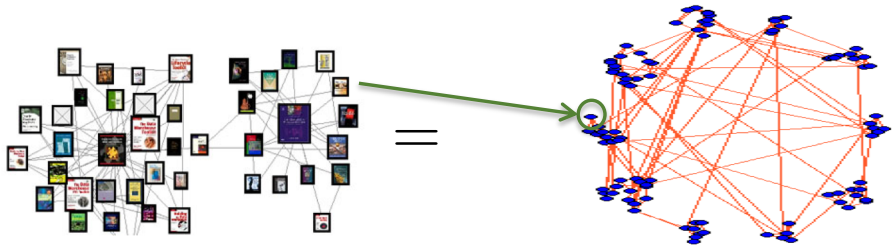


Network of users

Rows/users graph :

$G_r = (V_r, E_r, W_r)$

Adjacency matrix of rows



Network of products

Cols/products graph :

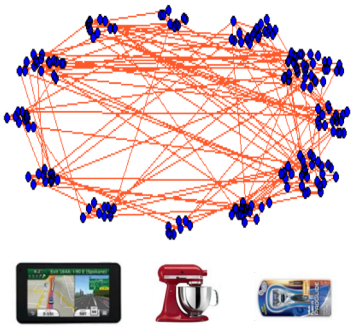
$G_c = (V_c, E_c, W_c)$

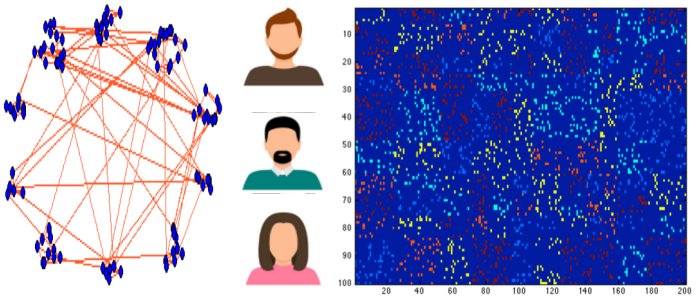
Adjacency matrix of columns

Graph-based content filtering[1]

Cols/products graph

$G_c = (V_c, E_c, W_c)$





Rows/users graph

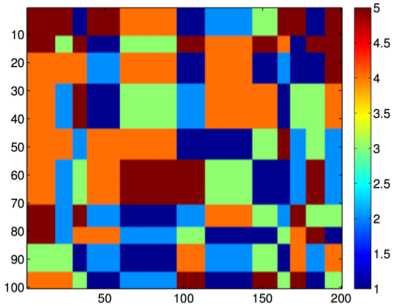
$G_r = (V_r, E_r, W_r)$

M

Recommendation

=

Matrix completion



X

[1] Huang, Chung, Ong, Chen, A graph-based recommender system for digital library, 2002

Task modeling

- How to fill up M with the graph of users and the graph of products ?
  - Simple idea : Diffuse/smooth the ratings on the networks of users and products.
- Optimization formulation :

$$\min_X \ \|X\|^{\text{diff}}_{\text{G}_r} + \|X\|^{\text{diff}}_{\text{G}_c} + \frac{\lambda}{2}\|I_{\text{obs}} \circ (X - M)\|^2_F$$

- How to design the diffusion loss on graphs ?

Graph diffusion loss

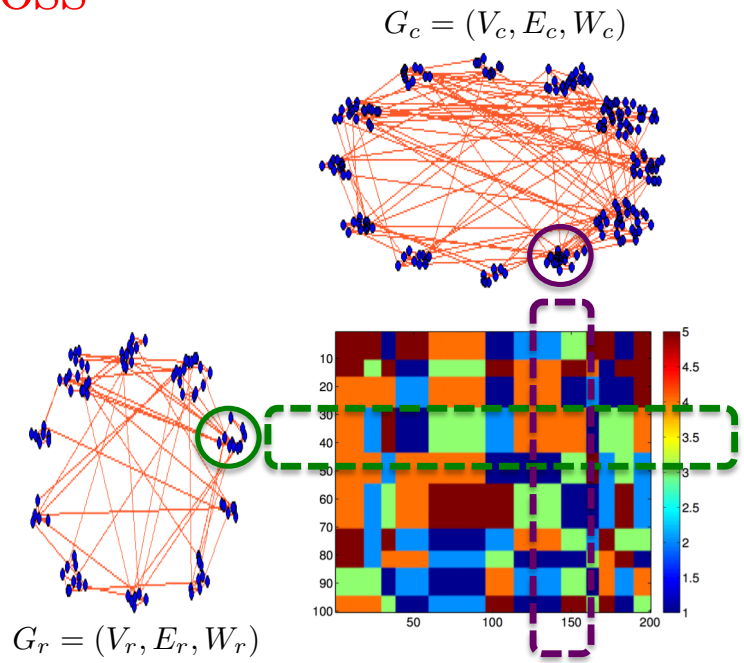
- Observation : When user i is close to user i' on the graph G_r=G_{users}, it means that the two users i,i' are similar, and so there is a high chance they will rate the products almost the same way.

  ⇒ Row_i and $row_{i'}$ on X are expected to be close.

- The diffusion loss is designed to force the ratings to be smooth on graph.
  - The most popular choice of graph smoothness is called the Dirichlet norm[1] :

$$\|X\|_G^{\text{diff}} = \|X\|_G^{\text{Dir}} = \text{tr}(X^T L X)$$

where L is the graph Laplacian



[1] Chung, Spectral graph theory, 1997

Algorithm

- Optimization problem :

$$\min_X \ \|X\|_{\mathrm{G}_r}^{\mathrm{Dir}} + \|X\|_{\mathrm{G}_c}^{\mathrm{Dir}} + \frac{\lambda}{2}\|I_{\mathrm{obs}} \circ (X - M)\|_F^2$$

$$\downarrow$$

$$\min_X \ \mathrm{tr}(X^T L_r X) + \mathrm{tr}(X L_c X^T) + \frac{\lambda}{2}\|I_{\mathrm{obs}} \circ (X - M)\|_F^2$$

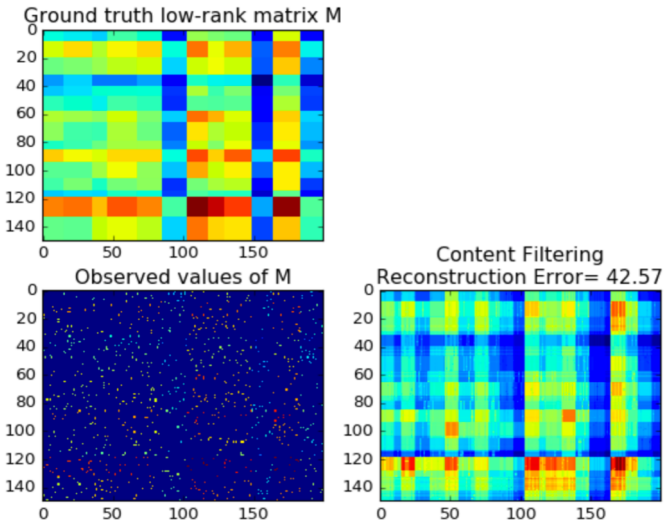
- The problem is smooth and quadratic.
  - It can be reduced to solve a linear system of equations (Ax=b) with e.g. conjugate gradient technique or stochastic gradient descent for large-scale dataset.

$$(I_m \otimes L_r + L_c \otimes I_n + \lambda I_{mn})X = \lambda M \qquad \rightarrow \qquad Ax = b$$

Lab 3: Content filtering

- Run 03_recom/code03.ipynb

```
# Content Filtering / Graph Regularization by Dirichlet Energy

#Compute Graph Laplacians
Lr = graph_laplacian(Wrow)
Lc = graph_laplacian(Wcol)
I = scipy.sparse.identity(m, dtype=Lr.dtype)
Lr = scipy.sparse.kron( I, Lr )
Lr = scipy.sparse.csr_matrix(Lr)
I = scipy.sparse.identity(n, dtype=Lc.dtype)
Lc = scipy.sparse.kron( Lc, I )
Lc = scipy.sparse.csr_matrix(Lc)
```



Outline

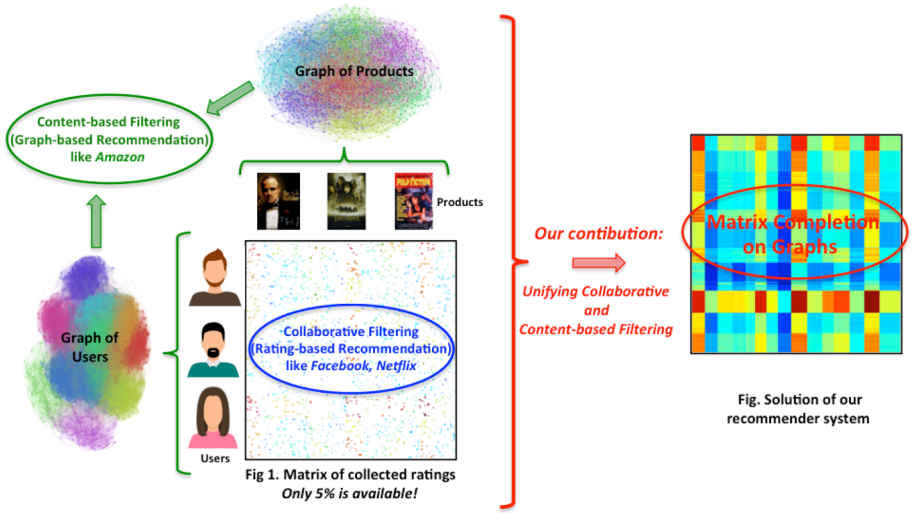
- Recommendation with graphs
  - Recommender systems
  - Google PageRank
  - Collaborative recommendation
  - Content recommendation
  - Hybrid systems
  - Conclusion

No feature learning !

Hybrid systems

- Can we combine collaborative and content recommendation? Yes[1,2] ☺
- Task formulation : Given a few ratings M_{ij} of product j and user i, user features and product features, design a recommender system that best
  - Captures the low-rank property of ratings (collaborative filtering)
  - Diffuses the ratings on the graphs of users and products (content filtering)

$$\min_X \ \|X\|_\star + \frac{\lambda_G}{2}\mathrm{tr}(X^T L_r X) + \frac{\lambda_G}{2}\mathrm{tr}(X L_c X^T) + \frac{\lambda}{2}\|I_{obs} \circ (X - M)\|_F^2$$

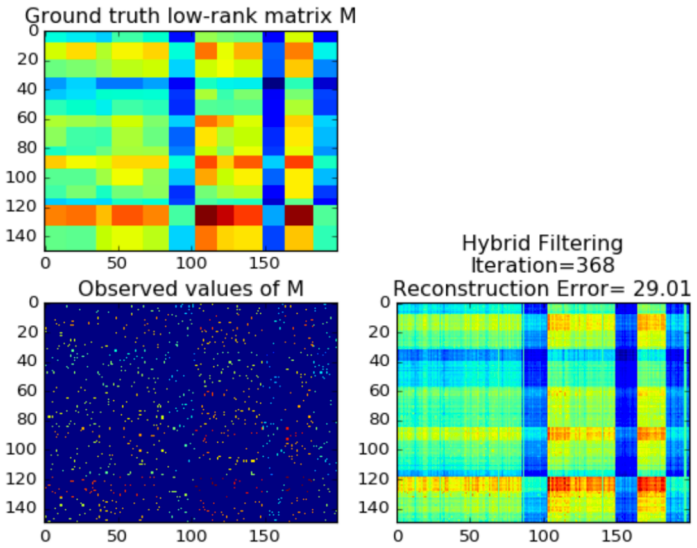


[1] Ma, Zhou, Liu, Lyu, King, Recommender systems with social regularization, 2011
[2] Kalofolias, Bresson, Bronstein, Vandergheynst, Matrix completion on graphs, 2014

Lab 4: Hybrid filtering

- Run 03_recom/code04.ipynb

```
# Hybrid system: Matrix Completion on graphs

#Compute Graph Laplacians
Lr = graph_laplacian(Wrow)
Lc = graph_laplacian(Wcol)
I = scipy.sparse.identity(m, dtype=Lr.dtype)
Lr = scipy.sparse.kron( I, Lr )
Lr = scipy.sparse.csr_matrix(Lr)
I = scipy.sparse.identity(n, dtype=Lc.dtype)
Lc = scipy.sparse.kron( Lc, I )
Lc = scipy.sparse.csr_matrix(Lc)
```



Outline

- Recommendation with graphs
  - Recommender systems
  - Google PageRank
  - Collaborative recommendation
  - Content recommendation
  - Hybrid systems
  - Conclusion

No feature learning !

Recommender systems

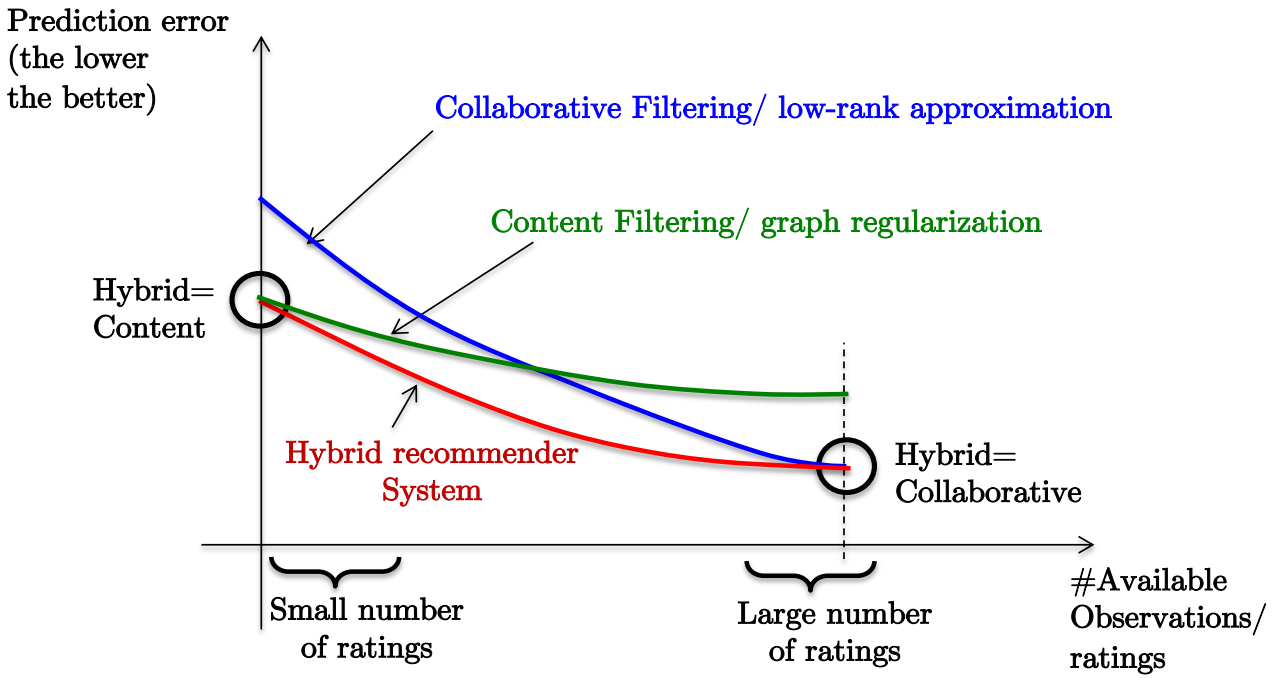


Fig. Accuracy w.r.t. # training data

Conclusion

- If not enough ratings then focus on collecting data features.
- When enough ratings then give less importance to features.

Outline

- Graph clustering
- Classification with graphs
- Recommendation with graphs
- **Dimensionality reduction with graphs**

No feature learning !

Outline

- Dimensionality reduction with graphs
  - Dimensionality reduction
  - Linear techniques
    - Standard PCA
    - Robust PCA
    - Graph PCA
  - Non-linear techniques
    - Kernel PCA
    - Locally-linear embedding
    - Laplacian eigenmaps
    - T-SNE
  - Conclusion

No feature learning !

Outline

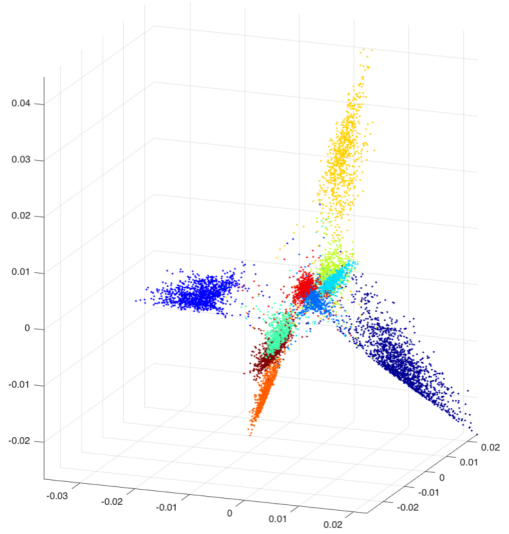
- Dimensionality reduction with graphs
  - Dimensionality reduction
  - Linear techniques
    - Standard PCA
    - Robust PCA
    - Graph PCA
  - Non-linear techniques
    - Kernel PCA
    - Locally-linear embedding
    - Laplacian eigenmaps
    - T-SNE
  - Conclusion

No feature learning !

Dimensionality reduction

- Dimensionality reduction techniques aims at projecting high-dimensional data into low-dimensional spaces, by compressing the original information and discarding "details/noise".
- Linear dimensionality reduction : Low-dimensional spaces are Euclidean (flat) spaces.
- Non-linear dimensionality reduction : Low-dimensional spaces are then manifolds (curved surfaces).

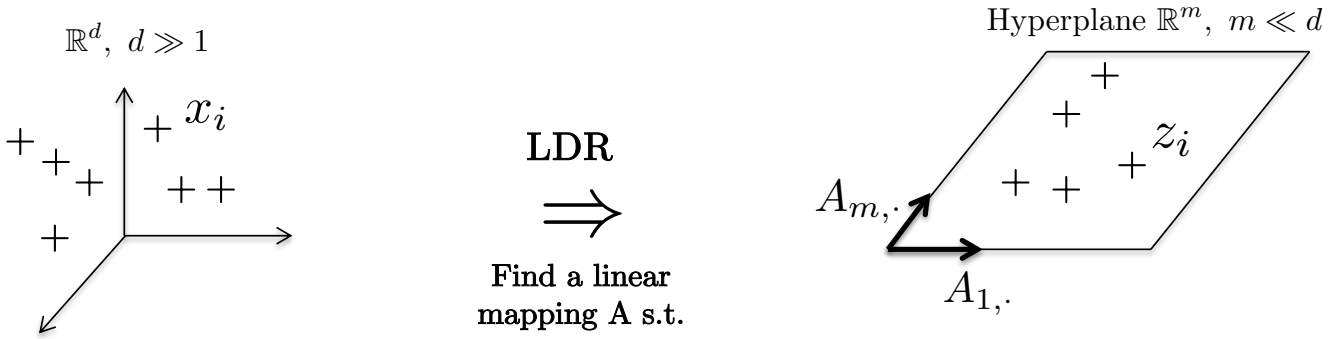
$$x \in \mathbb{R}^d,\ d \gg 1 \quad \rightarrow \quad z = \varphi(x) \in \mathbb{R}^k,\ k \ll d$$



MNIST : $x \in R^{684} \Rightarrow z \in R^3$

Linear dimensionality reduction

- **Assumption : Data lie on low-dimensional Euclidean spaces.**



$x_i \in \mathbb{R}^d$

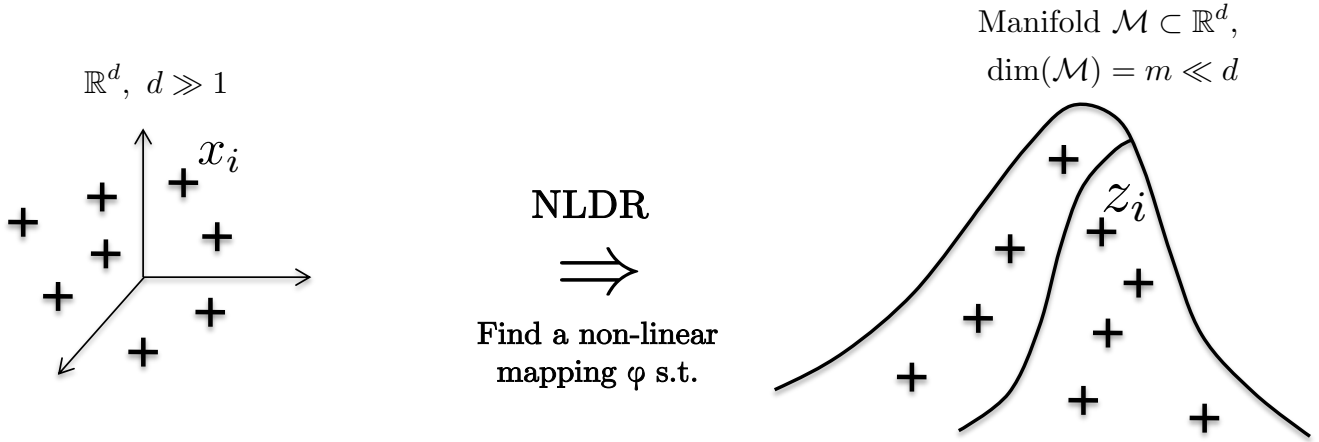
$z_i \in \mathbb{R}^m = A\ x_i$

$$\varphi : x_i \to z_i = \varphi(x_i) = Ax_i$$

[1] xxx

Non-linear dimensionality reduction

- **Assumption : Data lie on low-dimensional curved spaces, i.e. manifolds.**
  - **These techniques are called manifold learning.**



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

Outline

- Dimensionality reduction with graphs
  - Dimensionality reduction
  - Linear techniques
    - Standard PCA
    - Robust PCA
    - Graph PCA
  - Non-linear techniques
    - Kernel PCA
    - Locally-linear embedding
    - Laplacian eigenmaps
    - T-SNE
  - Conclusion

No feature learning !

Linear techniques

- Task modeling

$$z = \varphi(x) = Ax$$

Low-dim data — Dictionary of patterns (or basis functions) — High-dim data

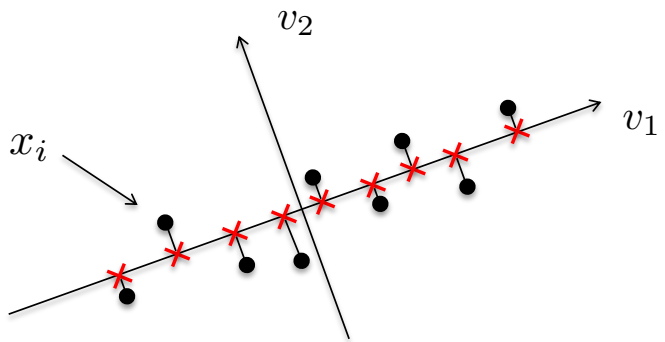
$$z = Ax = \begin{bmatrix} \langle A_{1,.}, x \rangle \\ \vdots \\ \langle A_{K,.}, x \rangle \end{bmatrix} \quad \text{with} \quad z_i = \langle A_{i,.}, x \rangle$$

i^{th} element — i^{th} filter

- How to compute A ?
  - Several techniques : PCA, ICA, NMF, Sparse Coding, etc.

Principal component analysis[1]

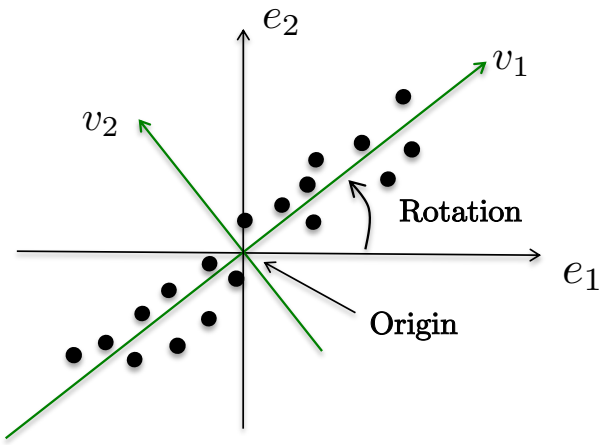
- PCA was introduced by Pearson in 1901 for physics.
- PCA is the most popular technique for linear representation.
- Task formulation : Given a set of data points, PCA aims at projecting data onto an orthogonal basis that best captures the data variance.
  - The first basis function or principal direction v_1 captures the largest possible variance of data.
  - The second basis function or principal direction v_2 captures the largest possible variance of data while satisfying the orthogonality constraint $\langle v_1, v_2 \rangle = 0$.
  - Etc.



[1] Pearson, On lines and planes of closest fit to systems of points in space, 1901

Task formalization

- PCA defines an orthogonal transformation (rotation) that maps the data to a new coordinate system (v_1,v_2,...,v_K) called principal directions such that the v_k's capture the largest possible data variance.
- Pre-processing
  - PCA requires the data to be centered.
  - PCA is not invariant w.r.t. normalization, and its result changes depending on the normalization function.



Covariance matrix

- The Covariance Matrix C is defined as

$$C = X^T X$$
$$\text{d x d} \quad \text{d x n} \quad \text{n x d}$$

X= n x d data matrix
n = number of data
d = number of dimensions

- Covariance matrix C encodes all data variances along each dimension :

$$C_{\alpha\alpha} = \langle X_{\cdot,\alpha}, X_{\cdot,\alpha} \rangle = \|X_{\cdot,\alpha}\|_2^2 = \sum_{i=1}^{n} |X_{i\alpha}|^2 = \sum_{i=1}^{n} x_{i,\alpha}^2$$

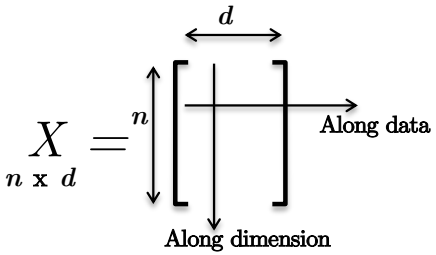
Variance of data along α^{th} dimension

$$C_{\alpha\beta} = \langle X_{\cdot,\alpha}, X_{\cdot,\beta} \rangle = \sum_{i=1}^{n} X_{i\alpha} X_{i\beta} = \sum_{i=1}^{n} x_{i,\alpha} x_{j,\alpha}$$

Covariance of data along α^{th} and β^{th} dimensions

Reminder : x_i are zero-centered along each dimension α :

$$\mathbb{E}(\{x_i\}) = \sum_{i=1}^{n} x_{i,\alpha} = \sum_{i=1}^{n} X_{i\alpha} = 0 \ \forall \alpha$$



X
n x d

PCA is EVD of covariance matrix

- Let us show

$$Cv_1 = \lambda_1 v_1$$

Principal direction (PD) v_1 — Largest variance of data along PD v_1

Vector v_1 = 1st principal direction

= Direction of largest data variance

$$= \underset{\|v\|_2=1}{\operatorname{argmax}} \sum_{i=1}^{n} |\langle x_i, v \rangle|^2$$

Square distance of data projected on v

Matrix notation ⇕

$$= \underset{\|v\|_2=1}{\operatorname{argmax}} \|Xv\|_2^2 = (Xv)^T(Xv) = v^T X^T X v = v^T C v$$

⇓

Spectral solution: Solution v_1 is the largest eigenvector of C

$$Cv_1 = \lambda_1 v_1 \;\rightarrow\; v_1^T C v_1 = v_1^T \lambda_1 v_1 = \lambda_1 \|v_1\|_2^2 = \lambda_1 = \underset{\|v\|_2=1}{\operatorname{argmax}} \sum_{i=1}^{n} |\langle x_i, v \rangle|^2 \quad \square$$

PCA is EVD of covariance matrix

- **Vector v_2 = 2nd principal direction**

  = Direction of second largest data variance

  $$= \underset{\|v\|_2=1}{\arg\max} \quad v^T C v \quad \text{s.t.} \quad \langle v, v_1 \rangle = 0$$

  $$\Downarrow$$

  $$C v_2 = \lambda_2 v_2, \quad \text{with} \quad \lambda_1 \geq \lambda_2$$

- **Vector v_3 = 3rd principal direction ...**

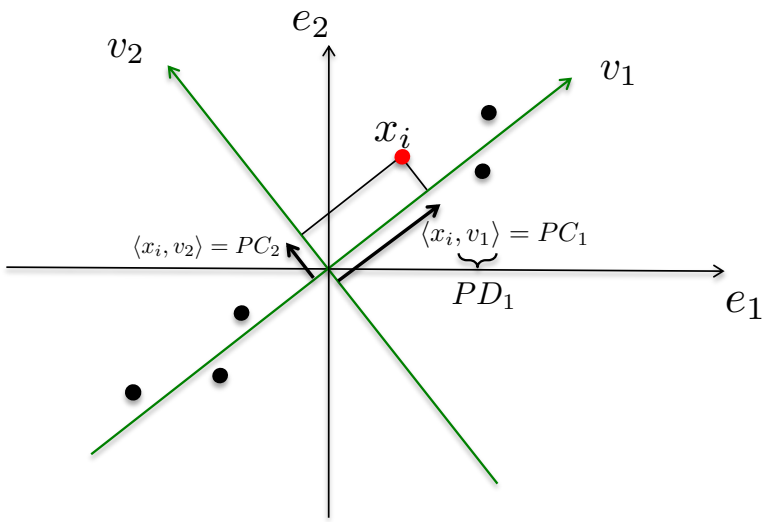
- **Matrix factorization : Full EVD of C**

  $$C = V \Lambda V^T$$

  $$\text{with} \quad V = \Big[v_1, ..., v_d \Big], \; V^T V = I_d, \; \Lambda = \text{diag}(\lambda_1, ..., \lambda_d)$$

Principal components

- PCs are the coordinates of original data projected into the basis of principal directions (PDs).



$$X_{pca} = XV$$

PCA is SVD of data matrix

- **Matrix factorization :** $\underset{\text{n x d}}{X} = \underset{\text{n x n}}{U}\underset{\text{n x d}}{\Sigma}\underset{\text{d x d}}{V^T}$ with $UU^T = I_n,\ VV^T = I_d$

- **Relationship between EVD and SVD :**

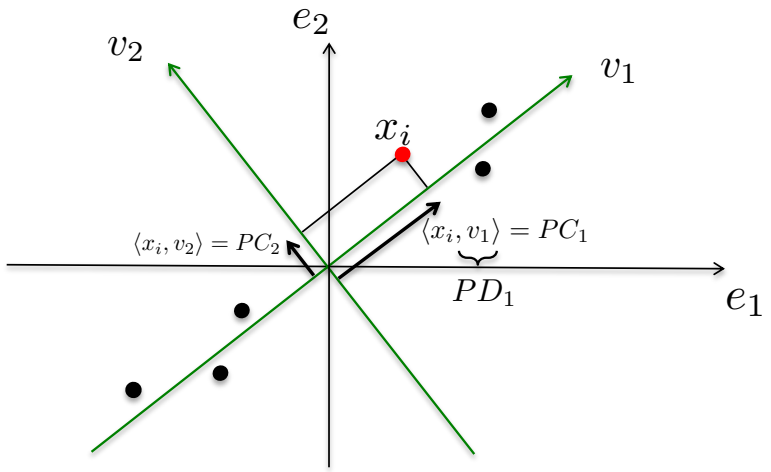
$$C = X^T X = (U\Sigma V^T)^T(U\Sigma V^T) = V\Sigma(U^T U)\Sigma V^T = V_{svd}\Sigma^2 V_{svd}^T = V_{evd}\Lambda V_{evd}^T \quad \rightarrow \quad \begin{cases} V_{svd} = V_{evd} \\ \Sigma^2 = \Lambda \quad \rightarrow \ \lambda_k = \sigma_k^2 \\ X_{pca} = XV_{evd} = U_{svd}\Sigma \end{cases}$$

- **PCA with EVD or SVD ?**
  - **It depends on the value of the size (n x d) of the data matrix X :**
    - **For n<d : use SVD**
    - **For d<n : use EVD**

Dimensionality reduction with PCA

- **Justification : Linear data are concentrated along the first principal directions.**

$$x_i = \langle x_i, v_1 \rangle v_1 + \langle x_i, v_2 \rangle v_2$$
$$\approx \langle x_i, v_1 \rangle v_1$$



The first PDs are enough to provide a good representation of linear data, i.e.

$$\|X - X_K\|_2^2 \text{ small for a small } K$$

Approximation of X with first K PDs: $X_K = U \Sigma_K V_K^T$

Truncated Σ with first K data variances

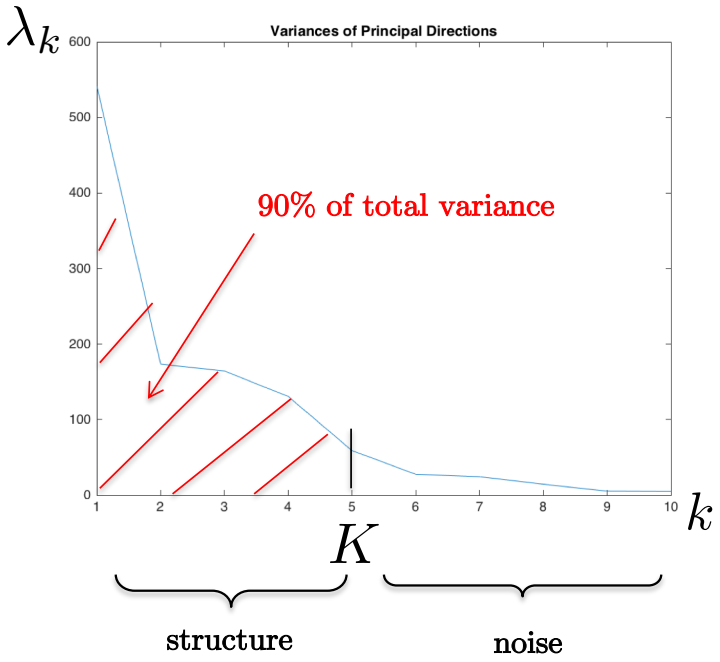
Truncated V with first K PDs

K-dimensionality reduction

- Data variance is captured with the principal directions.
  - If one wants to retain 90% of total data variance then K can be selected as follows :

$$\frac{\sum_{k=1}^{K} \lambda_k}{\sum_{k=1}^{d} \lambda_k} \geq 0.9$$

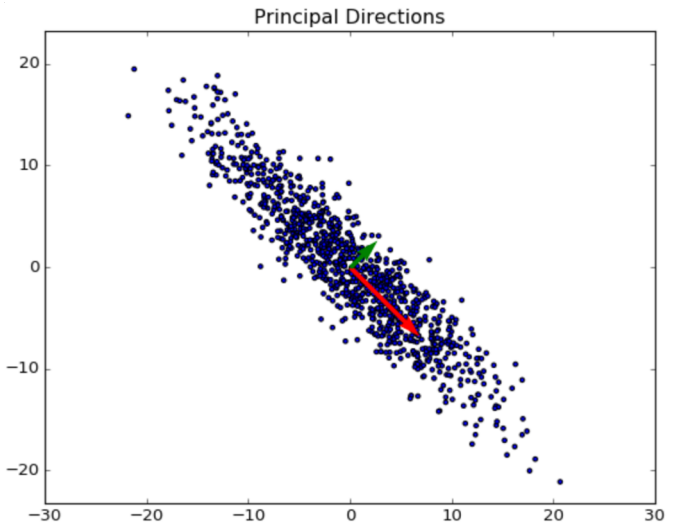
YaleBFaces dataset



Lab 1: Standard/linear PCA

- Run 04_dim_reduc/code01.ipynb

```
# Zero-centered data
Xzc = X - np.mean(X,axis=0)

# Covariance matrix
CovX = (Xzc.T).dot(Xzc)

# Compute largest 5 eigenvectors/eigenvalues
nb_pca = 5
CovX = scipy.sparse.csr_matrix(CovX)
lamb, U = scipy.sparse.linalg.eigsh(CovX, k=nb_pca, which='LM') # U = d x nb_pca
EVec = U[:,::-1] # largest = index 0
EVal = lamb[::-1]

# Principal Components
Xpc = X.dot(EVec)

# Principal Directions
v1 = EVec[:2,0]
v2 = EVec[:2,1]
```



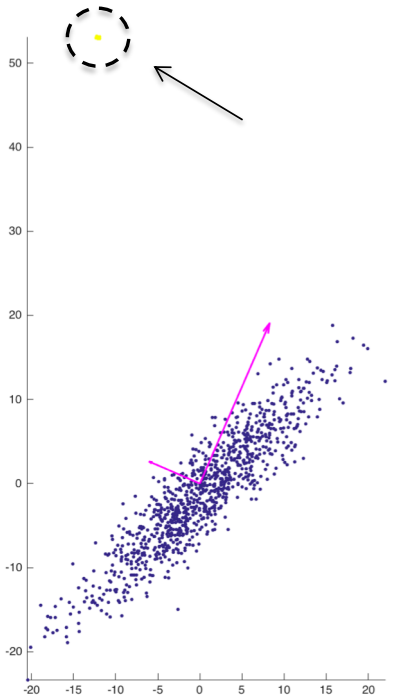
Outline

- Dimensionality reduction with graphs
  - Dimensionality reduction
  - Linear techniques
    - Standard PCA
    - Robust PCA
    - Graph PCA
  - Non-linear techniques
    - Kernel PCA
    - Locally-linear embedding
    - Laplacian eigenmaps
    - T-SNE
  - Conclusion

No feature learning !

Robust PCA[1]

- Standard PCA is sensitive to outliers, i.e. a single outlier may change significantly the (clean) PCA solution.
- Robust PCA is a technique that separates outliers from the clean data where PCA is performed.



[1] Candes, Li, Ma, Wright, Robust principal component analysis, 2011

Task formalization

- **Standard PCA :** $\min_{L} \|X - L\|_F^2 \quad \text{s.t.} \quad \text{rank}(L) = K$

- **Robust PCA :** $\min_{L,S} \underbrace{\text{rank}(L) + \lambda \, \text{card}(S)}_{\text{NP-hard combinatorial problem}} \quad \text{s.t.} \quad X = L + S \quad (1)$

  $\Rightarrow$ Continuous relaxation needed

  Data

  Low-rank matrix
  Standard PCA
  (structure)

  Sparse matrix
  captures outliers
  (no structure)

  $\Downarrow$ Convex relaxation

$$\min_{L,S} \|L\|_\star + \lambda \, \|S\|_1 \quad \text{s.t.} \quad X = L + S \quad (2)$$

Result[1] : Solution of (2) is almost the same solution of (1) !

[1] Candes, Li, Ma, Wright, Robust principal component analysis, 2011

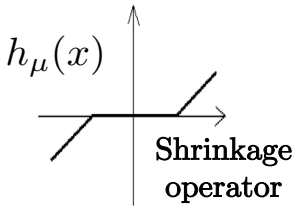
Algorithm

- ADMM technique[1] : Fast, robust and accurate solution

Initialization : $L^{m=0} = X \qquad S^{m=0} = Z^{m=0} = 0$

Iterate until convergence :

$$\begin{cases} L^{m+1} = U h_{1/r}(\Lambda) V^T \text{ with } U\Lambda V^T \overset{svd}{=} X - S^m + Z^m/r \\ S^{m+1} = h_{\lambda/r}\left(X - L^{m+1} + Z^m/r\right) \\ Z^{m+1} = Z^m + r(X - L^{m+1} - S^{m+1}) \end{cases}$$

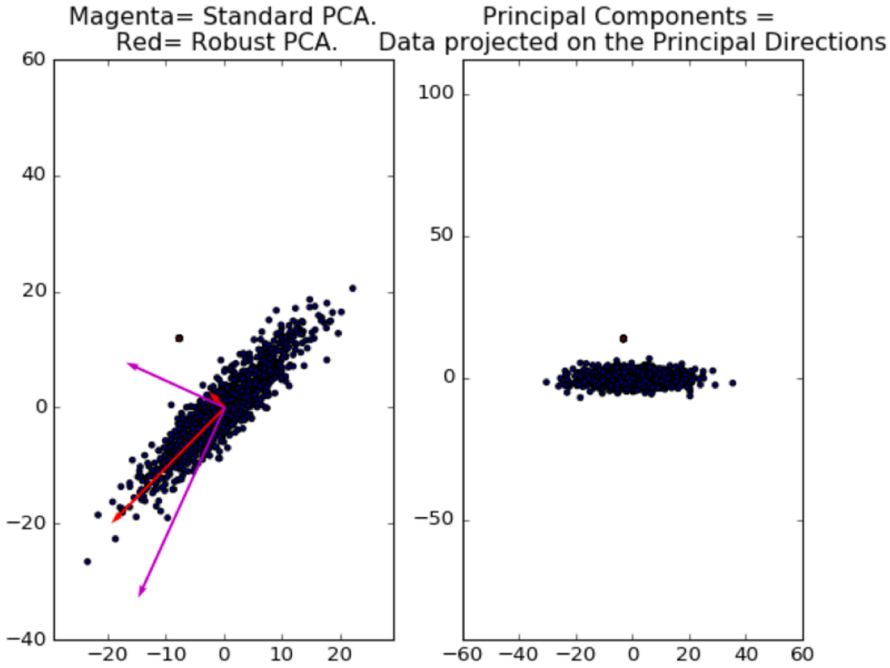


[1] Glowinski, Le Tallec, Augmented Lagrangian and operator-splitting methods in nonlinear mechanics, 1989

Lab 2: Robust PCA

- Run 04_dim_reduc/code02.ipynb

```
# Run Robust PCA
X = Xref - np.mean(Xref,axis=0)
L = X
S = np.zeros(X.shape)
Z = np.zeros(X.shape)
n,m = X.shape
min_nm = np.min([n,m])
r = 1
lambdaN = 1.
lambdaS = 0.1
for i in range(1000):

    # Update L
```



Outline

- Dimensionality reduction with graphs
  - Dimensionality reduction
  - Linear techniques
    - Standard PCA
    - Robust PCA
    - Graph PCA
  - Non-linear techniques
    - Kernel PCA
    - Locally-linear embedding
    - Laplacian eigenmaps
    - T-SNE
  - Conclusion

No feature learning !

Graph PCA[1]

- **Motivation : Enhance PCA with data similarities.**

- **Task formalization :**

$$\min_{L,S} \ \text{rank}(X) + \lambda_s \ \text{card}(S) + \lambda_G \underbrace{\|L\|_{G\_smooth}}_{\text{Force smoothness on graphs}} \ \text{s.t.} \quad X = L + S$$

$$\Downarrow \text{ Continuous convex relaxation}$$

$$\min_{L,S} \ \|X\|_\star + \lambda_s \ \|S\|_1 + \lambda_G \|L\|_{G\_Dir} \ \text{s.t.} \quad X = L + S$$

[1] Shahid, Kalofolias, Bresson, Bronstein, Vandergheynst, Robust principal component analysis on graphs, 2015

Application to video surveillance

- Separate background from moving objects[1]

[1] Shahid, Kalofolias, Bresson, Bronstein, Vandergheynst, Robust principal component analysis on graphs, 2015

Outline

- Dimensionality reduction with graphs
  - Dimensionality reduction
  - Linear techniques
    - Standard PCA
    - Robust PCA
    - Graph PCA
  - Non-linear techniques
    - Kernel PCA
    - Locally-linear embedding
    - Laplacian eigenmaps
    - T-SNE
  - Conclusion

No feature learning !

Kernel PCA[1]

- **Standard PCA :**

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

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

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

Apply the kernel trick to the Gram matrix :

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

[1] Scholkopf, Smola, Muller, Kernel principal component analysis, 1997

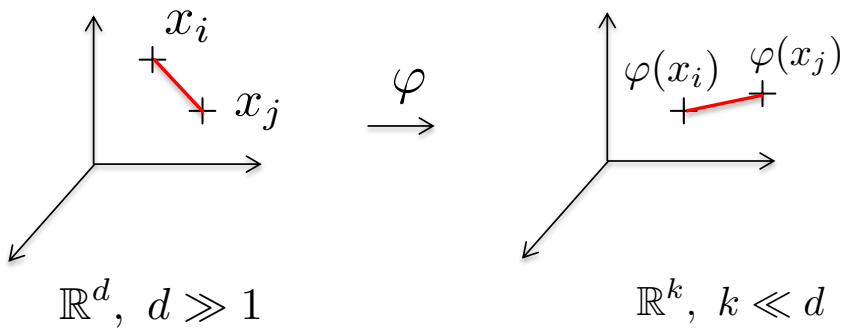
Outline

- Dimensionality reduction with graphs
  - Dimensionality reduction
  - Linear techniques
    - Standard PCA
    - Robust PCA
    - Graph PCA
  - Non-linear techniques
    - Kernel PCA
    - Locally-linear embedding
    - Laplacian eigenmaps
    - T-SNE
  - Conclusion

No feature learning !

Locally-Linear Embedding[1] (LLE)

- Design a mapping from high-dimensional space to low-dimensional space such that the geometric distance between neighborhood data is preserved.



- LLE aims at computing a manifold M by local linear fits, i.e. each data on M and its neighbors lies on a locally linear patch of M.

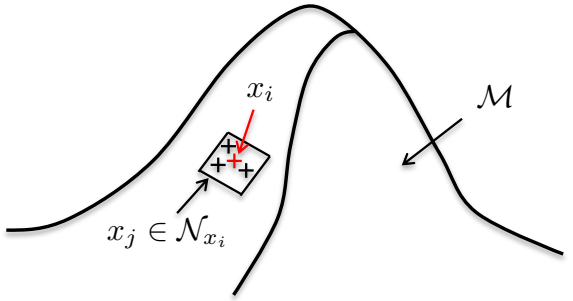
[1] Roweis, Saul, Nonlinear dimensionality reduction by locally linear embedding, 2000

Algorithm

- Step 1 : For each data x_i, compute its k nearest neighbors.
- Step 2 : Compute linear patches.
  - Find the weights W_{ij} which best linearly reconstruct x_i from its neighbors :

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

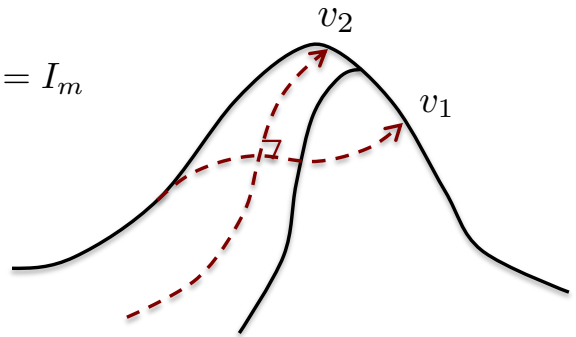
$\Rightarrow$ Solution : Ax=b



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

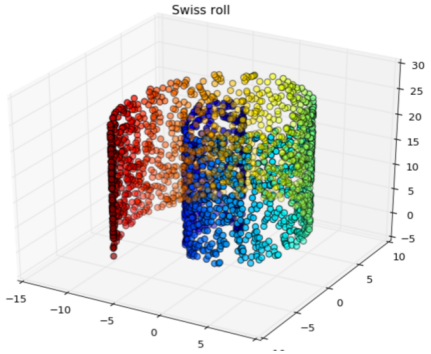
$$\min_{Z=[z_1,\ldots,z_m]} \sum_{i=1}^{n} \left\| z_i - \sum_j W_{ij} z_j \right\|_2^2 \quad \text{s.t.} \quad \sum_i z_i = 0, \ Z^T Z = I_m$$

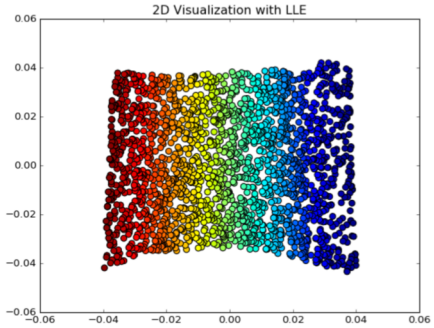
$\Rightarrow$ Solution : EVD

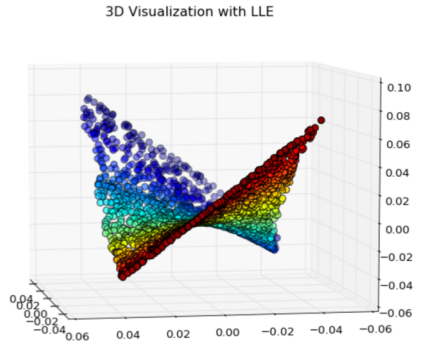


Lab 3: LLE

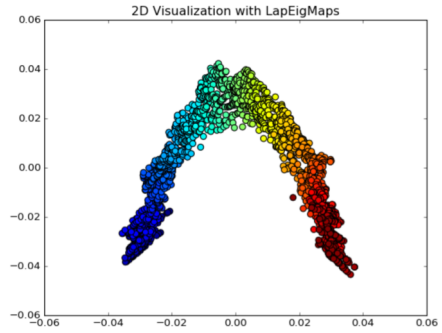
- Run 04_dim_reduc/code03.ipynb

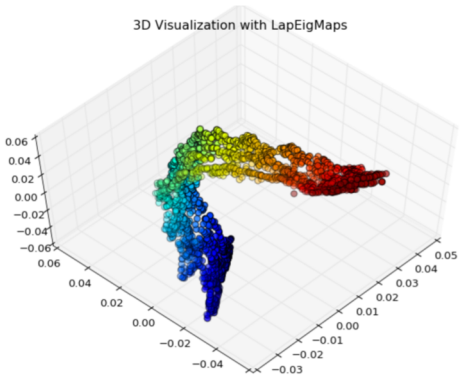







```
# Run NLDR = LLE
X = Xref
X = X - np.mean(X,axis=0) # zero-centered data

# 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,:]
```





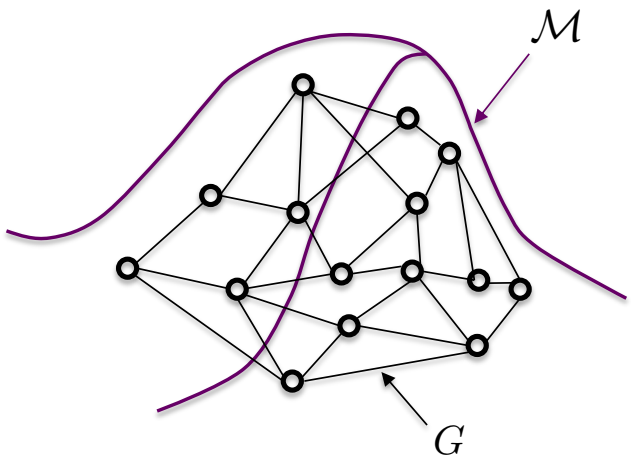
Outline

- Dimensionality reduction with graphs
  - Dimensionality reduction
  - Linear techniques
    - Standard PCA
    - Robust PCA
    - Graph PCA
  - Non-linear techniques
    - Kernel PCA
    - Locally-linear embedding
    - Laplacian eigenmaps
    - T-SNE
  - Conclusion

No feature learning !

Laplacian eigenmaps[1]

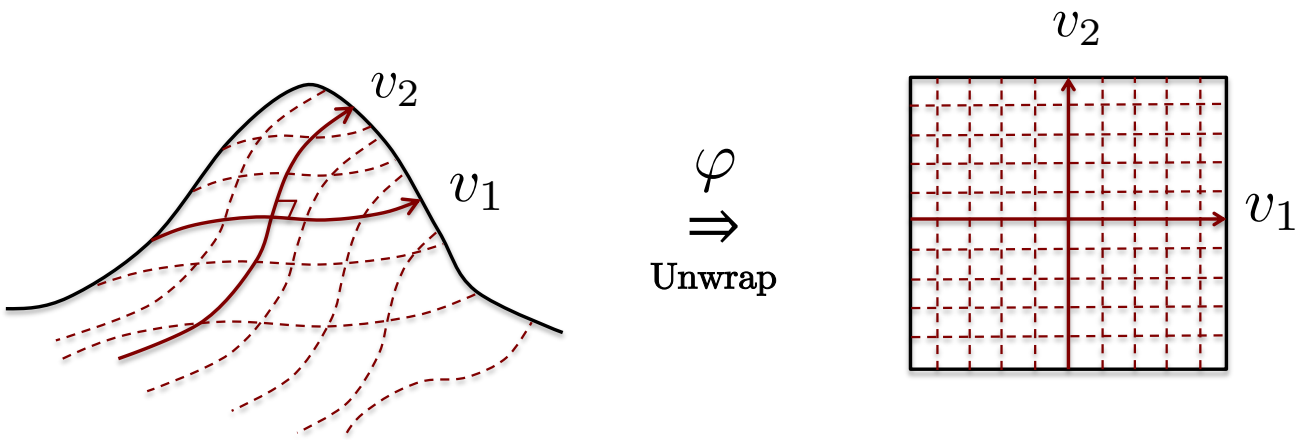
- Manifold assumption : Data is sampled from a manifold M approximated by a k-nearest neighbors graph.



[1] Belkin, Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, 2003

Spectral analysis and differential geometry

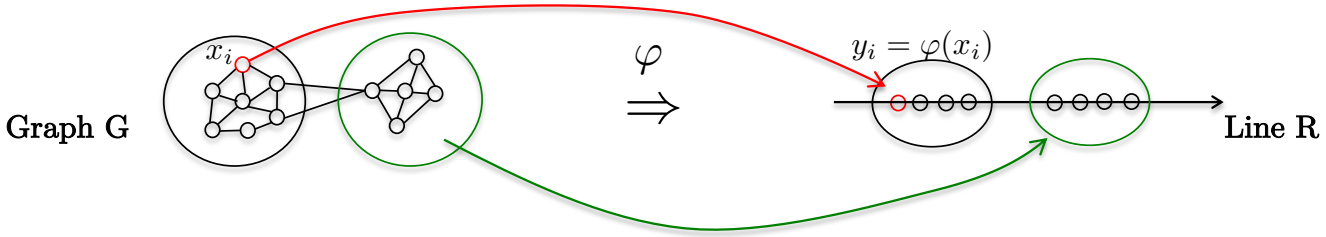
- Eigenfunctions v_k of continuous Laplace-Beltrami Δ_M serves as embedding coordinates of M :
  - Discretization of Δ_M = graph Laplacian $L^{[1]}$



[1] Chung, Spectral graph theory, 1997

Task formalization

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



- 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

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

- Note that the mapping φ is never computed explicitly.
- 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 K dimensions

- K-D dim reduction : Generalizing (1) to K dimensions is straightforward.

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

Graph Laplacian

- Spectral Solution : Solution is given by the largest K eigenvectors of the graph Laplacian L.

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

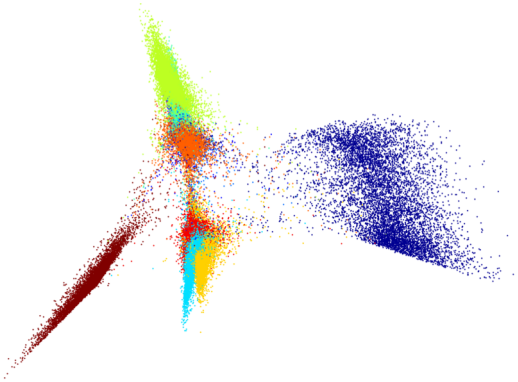
- Properties
  - Global solution (independent of initialization)
  - Complexity is O(N^2.K)

Lab 4: Laplacian eigenmaps

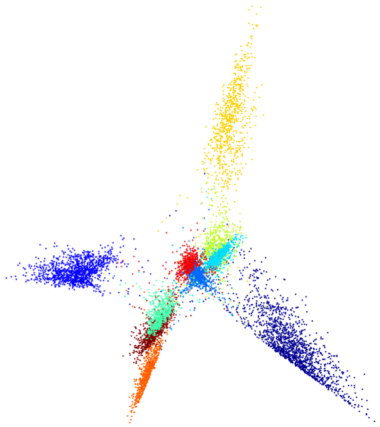
- Run 04_dim_reduc/code04.ipynb

```
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

- Dimensionality reduction with graphs
  - Dimensionality reduction
  - Linear techniques
    - Standard PCA
    - Robust PCA
    - Graph PCA
  - Non-linear techniques
    - Kernel PCA
    - Locally-linear embedding
    - Laplacian eigenmaps
    - T-SNE
  - Conclusion

No feature learning !

T-distributed stochastic neighbor embedding[1]

- t-SNE computes the embedding φ function, $y_i=\varphi(x_i)$, 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

[1] Van der Maaten, Hinton, Visualizing data using t-SNE, 2008

Algorithm

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

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

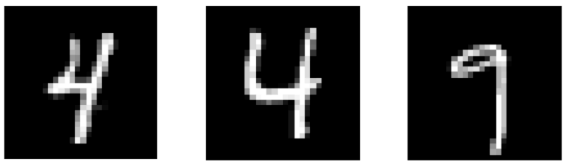
$$y_i^{m+1} = y_i^m - 4\tau \sum_j (p_{ij} - q_{ij})(1 + \|x_i - x_j\|_2)^{-1}(y_i^m - y_j)$$

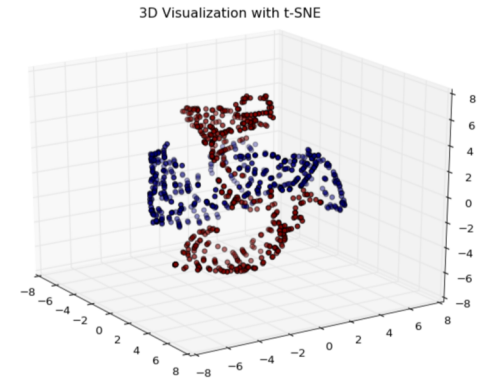
- Advantages
  - Local distance preservation : Minimizing KL forces the low-dim distribution q_{ij} to be close to the high-dim data distribution p_{ij}.
  - t-SNE does not enforce the manifold assumption : Higher flexibility to represent well complex hidden structures.
- Limitations
  - Non-convex energy : Existence of (bad) local solutions, problem of initialization (PCA is used as initialization)
  - Slow optimization (gradient descent technique)

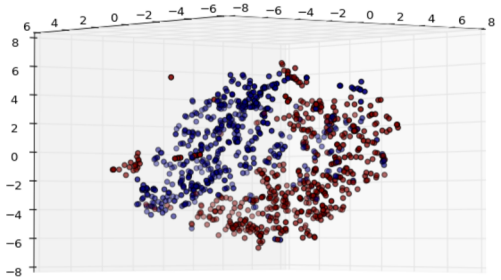
Lab 5: t-SNE

- Run 04_dim_reduc/code05.ipynb

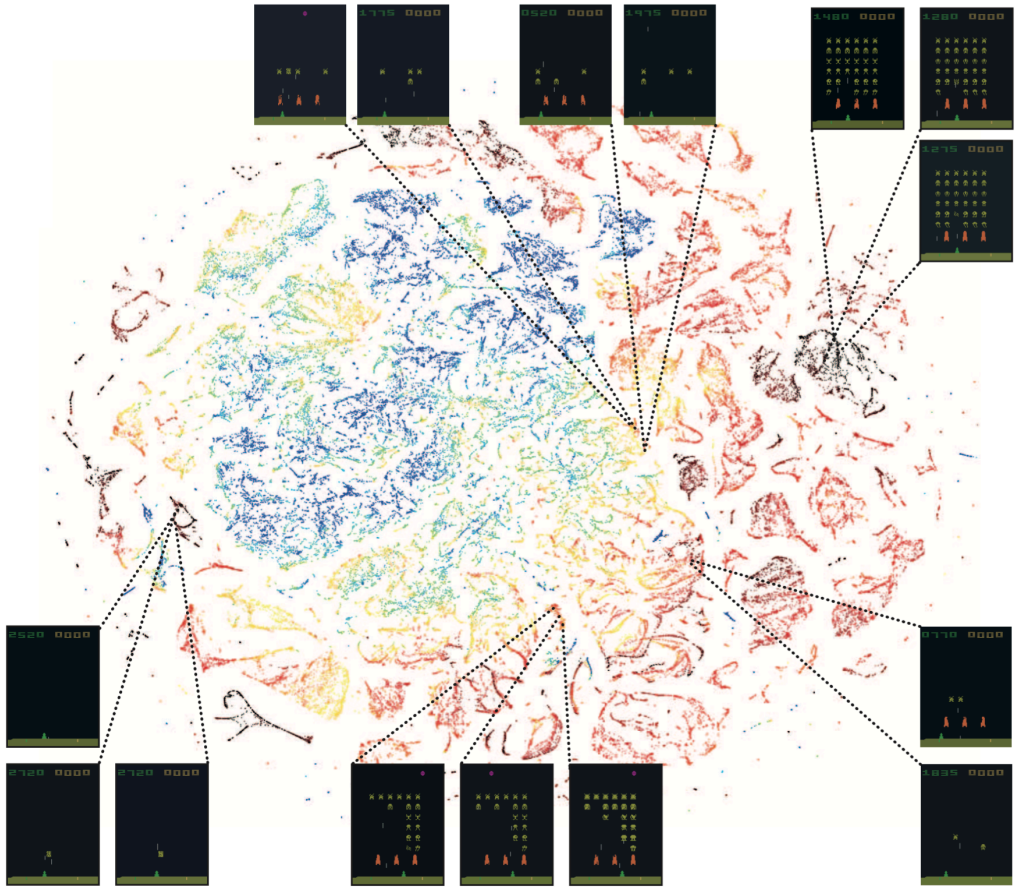
```
# Run t-SNE
model = TSNE(n_components=3, n_iter=2000, learning_rate=1000)
np.set_printoptions(suppress=True)
XtSNE = model.fit_transform(X) # Xtsne = n x d
Xtsne = XtSNE[:,0]
Ytsne = XtSNE[:,1]
Ztsne = XtSNE[:,2]
```







Visualizing video games[1]



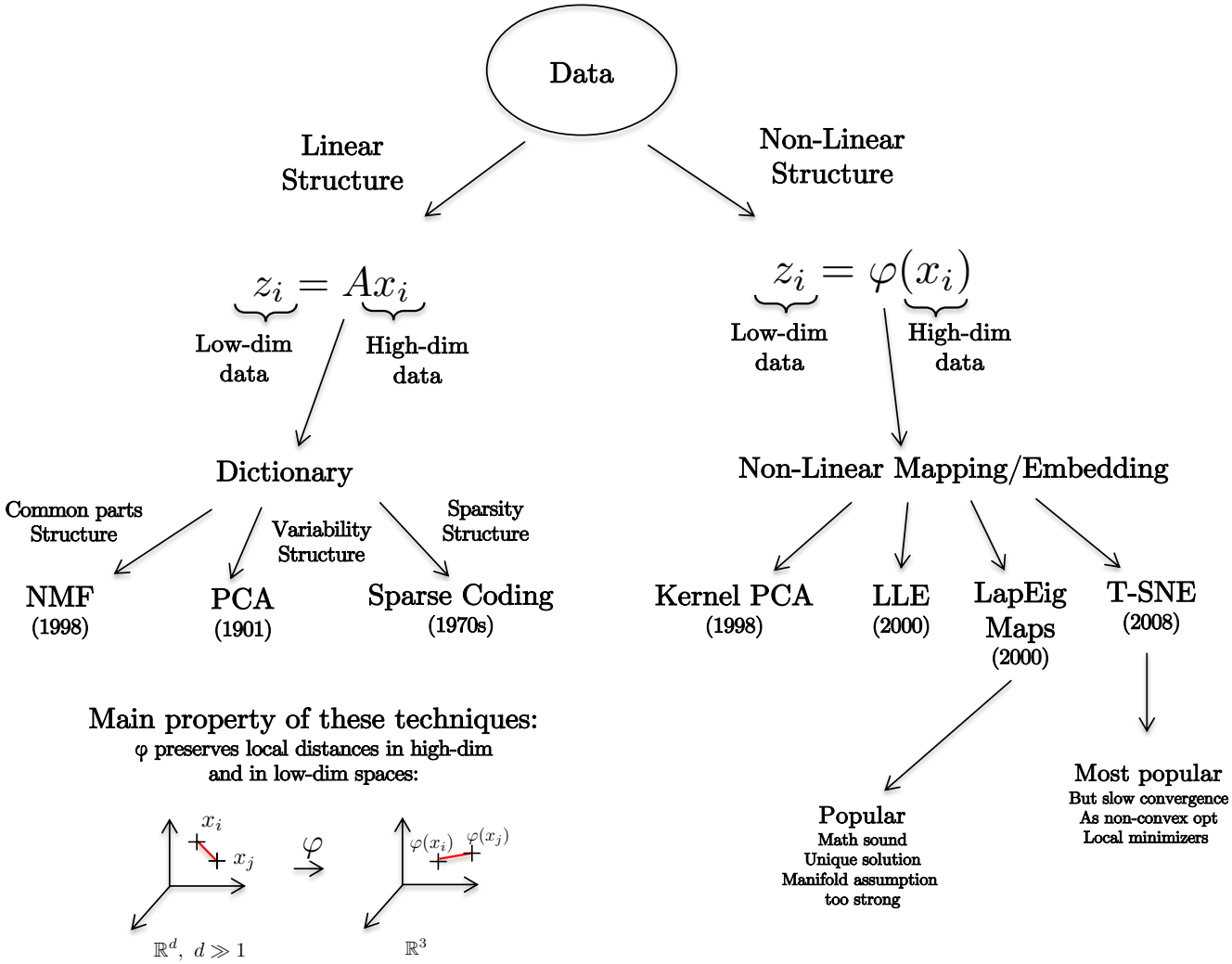
[1] Mnih, Human-level control through deep reinforcement learning, 2015

Outline

- Dimensionality reduction with graphs
  - Dimensionality reduction
  - Linear techniques
    - Standard PCA
    - Robust PCA
    - Graph PCA
  - Non-linear techniques
    - Kernel PCA
    - Locally-linear embedding
    - Laplacian eigenmaps
    - T-SNE
  - Conclusion

No feature learning !

Conclusion


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)
Main property of these techniques:
φ preserves local distances in high-dim and in low-dim spaces:
x_i
x_j
φ
$\varphi(x_i)$
$\varphi(x_j)$
$\mathbb{R}^d,\ d \gg 1$
$\mathbb{R}^3$
Popular
Math sound
Unique solution
Manifold assumption
too strong
Most popular
But slow convergence
As non-convex opt
Local minimizers

Questions?