# Some uses of Graphs in Machine Learning

Mike PEREIRA

Machine Learning Group - MINES ParisTech

April 26th, 2017

# Outline





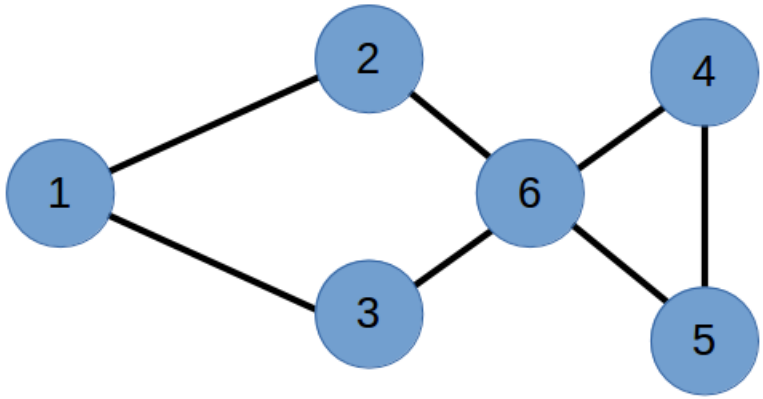
MINES ParisTech

# Graph : a mathematical definition



A graph $\mathcal{G}$ is a triplet $(\mathcal{V}, \mathcal{E}, \mathcal{W})$ where

- $\mathcal{V}$ is the set of vertices of the graph. It is supposed finite with cardinal $N$. The vertices are numbered : $\mathcal{V} = \{1, 2, ..., N\}$.
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges of the graph, defined as couples of vertices $(i, j) \in \mathcal{V} \times \mathcal{V}$. The vertices $i$ and $j$ are then called **neighbors**, and are denoted $i \sim j$.
- $\mathcal{W}$ is the weights of the graph : it is an application that associate to each edge $(i, j) \in \mathcal{E}$ a weight $w_{ij} \in \mathbb{R}$ .



# Similarity graph



## Idea

Given a data set containing $N$ examples $\boldsymbol{X}_1, ..., \boldsymbol{X}_N$, build a graph with $N$ vertices for which the **more similar** $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ are, the **stronger** the edge between $i$ and $j$ will be (if it exists).

Similarity Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ is build in two steps :

1. Define a similarity function $s$ that associates to each pair $(i, j)$ a real value $s_{ij}$ measuring the similarity between $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$
2. For each pair $(i, j)$ decide whether or not $i \sim j$ based on $s_{ij}$. If defined, weight of edge $(i, j)$ is $s_{ij}$.

# Similarity Graph : Similarity Function



Functions include :

- The default choice for the similarity function is the Gaussian function :

$$s_{ij} = \exp\left(-\frac{||\boldsymbol{X}_i - \boldsymbol{X}_j||^2}{2\sigma^2}\right)$$

  Where examples $\boldsymbol{X}$ are considered as vectors of $\mathbb{R}^d$, with $d$ the number of features.
  Choice of $\sigma$
  - Rule of thumb : $\sigma \approx 0.1\text{std(data)}$
  - Metric Learning is possible...
- Cosine similarity function :

$$s_{ij} = \frac{\boldsymbol{X}_j^T \boldsymbol{X}_j}{||\boldsymbol{X}_i|| ||\boldsymbol{X}_j||} = \cos(\theta)$$

- Typical kernels

# Similarity graph : Edge Construction I



3 main ways to decide whether 2 vertices of the graph will be neighbors (given the similarity function).

- $\varepsilon$-**neighborhood graphs** : $i \sim j$ if $|s_{ij}| \geq \varepsilon$.
- $k$-**NN (nearest neighbors) graphs** : Each vertex $i$ is neighbor to the $k$ vertices that have the largest values of similarity $s_{i\bullet}$. 2 options :
  - option AND : $i \sim j$ if $j$ is one of the $k$ vertices most similar to $j$ AND vice versa ("mutual $k$-NN graph").
  - option OR : ignore "direction".
- **Fully connected graph** : All pairs $(i, j)$ are neighbors.

# Similarity graph : Edge Construction II



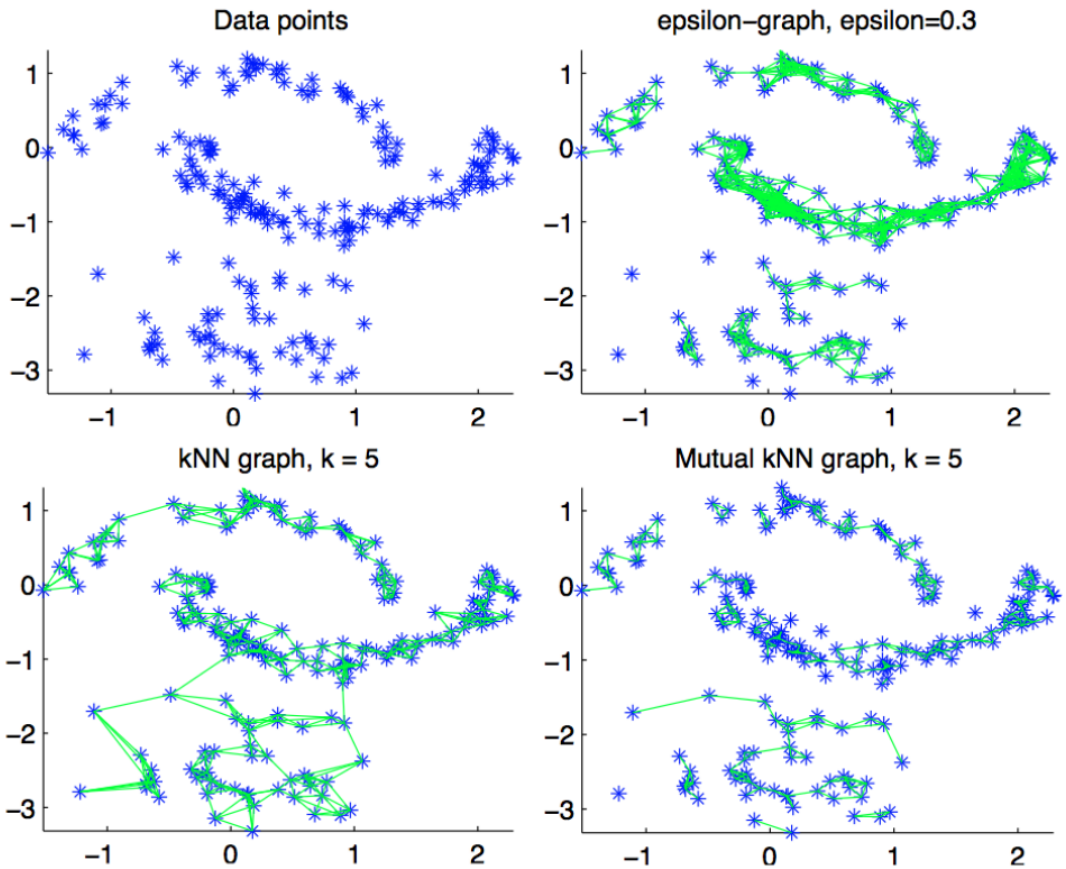



Figure: Comparison of the 3 methods of edge construction on the same data set.

http://researchers.lille.inria.fr/ valko/projects/courses/graphsML/20162017/mlgraphs2.pdf

# Graph matrices : Degree and Weight



$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ a graph with $N$ vertices, whose weights $\mathcal{W}$ are defined by the similarity function $s$.

## Definition

- The weight matrix $\boldsymbol{W}$ is the $N \times N$ matrix defined by :

$$W_{ij} = \begin{cases} s_{ij} & \text{if } i \sim j \\ 0 & \text{otherwise} \end{cases}$$

- The degree matrix $\boldsymbol{D}$ is the $N \times N$ diagonal matrix defined by :

$$D_{ii} = \sum_{j=1}^{N} s_{ij}$$

# Graph matrices : Laplacian



## Definition

The Laplacian matrix $\boldsymbol{L}$ is the $N \times N$ matrix defined by :

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$$

The normalized Laplacian matrix $\boldsymbol{\mathcal{L}}$ is the $N \times N$ matrix defined by :

$$\boldsymbol{\mathcal{L}} = \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$$

## Proposition

A **graph function** on $\mathcal{G}$ is a vector of $\boldsymbol{f} \in \mathbb{R}^N$ assigning to each vertex a value.

The Laplacian verifies for any graph function $\boldsymbol{f}$ :

$$\boldsymbol{f}^T\boldsymbol{L}\boldsymbol{f} = \frac{1}{2}\sum_{i,j=1}^{N} s_{ij}(f_i - f_j)^2$$

# Eigendecomposition



- A vector $\boldsymbol{v}$ is an **eigenvector** of matrix $\boldsymbol{M}$ if there exists $\lambda \in \mathbb{R}$ (or $\mathbb{C}$), called **eigenvalue**, s.t. :

$$\boldsymbol{M}\boldsymbol{v} = \lambda \boldsymbol{v}$$

- A $N \times N$ matrix $\boldsymbol{M}$ is **diagonalizable** over $\mathbb{R}$ (resp. $\mathbb{C}$) if there exists $\lambda_1 \leq ... \leq \lambda_N \in \mathbb{R}$ (resp. $\mathbb{C}$) and $\boldsymbol{Q}$ an invertible matrix of $\mathbb{R}$ (resp. $\mathbb{C}$) s.t. :

$$\boldsymbol{M} = \boldsymbol{Q} \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_N \end{pmatrix} \boldsymbol{Q}^{-1}$$

  The $i$-th column of $\boldsymbol{Q}$ is an eigenvector of $\boldsymbol{M}$ with eigenvalue $\lambda_i$.

- If $\boldsymbol{M}$ is a symmetric real matrix, $\boldsymbol{M}$ is diagonalizable over $\mathbb{R}$ and can be decomposed :

$$\boldsymbol{M} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T, \quad \boldsymbol{Q}^{-1} = \boldsymbol{Q}^T, \boldsymbol{\Lambda} \text{ Diagonal matrix}$$

# Spectral properties of the Laplacian I



## From now on...

Consider that the similarity function is symmetric, i.e., $\forall$ pairs of vertices $(i,j)$, $s_{ij} = s_{ji}$.
The weight matrix, and the Laplacian, are therefore **real symmetric matrices**.

Denote $\lambda_1 \leq ... \leq \lambda_N$ the eigenvalues of $\boldsymbol{L}$ and $\boldsymbol{L} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$ its eigendecomposition.

## Proposition

All eigenvalues of $\boldsymbol{L}$ are real and its smallest eigenvalue is 0 with eigenvector $\mathbf{1}_N$.

# Outline





MINES ParisTech

# Connected components I



The **connected components** of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ are subgraphs $\mathcal{G}' = (\mathcal{V}' \subset \mathcal{V}, \mathcal{E} \subset \mathcal{E})$ for which :

- Any two vertices $(i, j) \in \mathcal{V}'$ can be "joined" by a path
- For any vertex $i \in \mathcal{V}'$, all its neighbors are in $\mathcal{V}'$

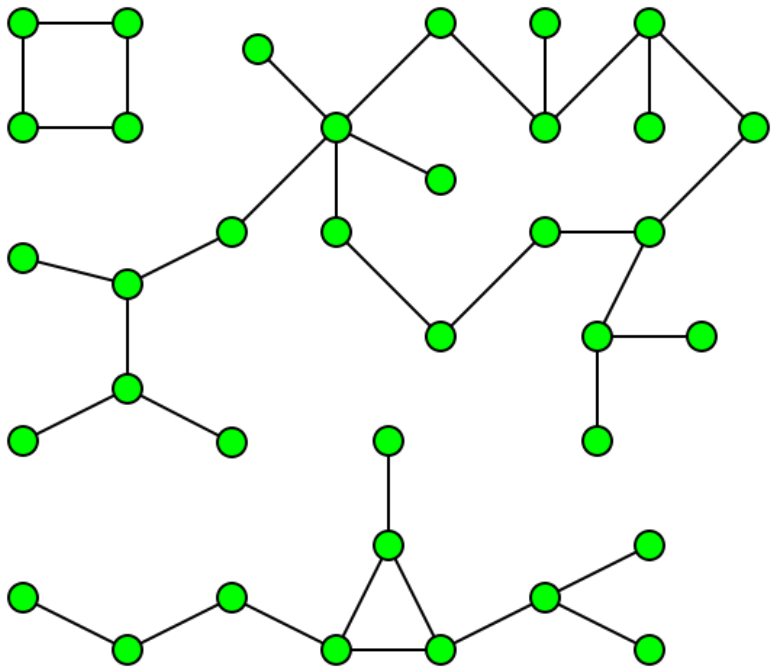


Figure: A graph with 3 connected components

# Connected components II



## Proposition

The Laplacian matrix $\boldsymbol{L}$ of a graph with $K$ connected components is block-diagonal with $K$ blocks :

$$\boldsymbol{L} = \begin{pmatrix} \boldsymbol{L}_1 & & \\ & \ddots & \\ & & \boldsymbol{L}_K \end{pmatrix}$$

Where $\boldsymbol{L}_k$ corresponds to the Laplacian of the $k$-th connected component.

It is easy to show that for a block-diagonal matrix

- Eigenvectors are given by considering eigenvectors of one of its blocks and "completing" it with zeros.
- Eigenvalues are the set of all the eigenvalues of each one of its block.

# Intuition about the algorithm



The **multiplicity** of an eigenvalue $\lambda$ of matrix $\boldsymbol{L}$ is the number of linearly independent vectors that are eigenvectors of $\boldsymbol{L}$ with eigenvalue $\lambda$.

## Proposition

The multiplicity of eigenvalue 0 is equal to the number $K$ of connected components of the graph.
The corresponding eigenvectors $\boldsymbol{x}_1, ..., \boldsymbol{x}_K$ are constructed as follows :

$$[\boldsymbol{x}_k]_i = \begin{cases} 1 \text{ if } i \in k\text{-th connected component} \\ 0 \text{ otherwise} \end{cases}$$

## Idea

Consider clusters as the connected components of a graph...!

# Spectral clustering Algorithm I



## Algorithm

Require : A set $\mathcal{V} = \{\boldsymbol{s}_1, ..., \boldsymbol{s}_N\} \in \mathbb{R}^d$ of data points. A number of clusters $k$.

1. Form the Weight matrix $\boldsymbol{W}$ defined by $W_{ij} = \exp(-||s_i - s_j||^2/2\sigma^2)$ if $i \neq j$, 0 otherwise.
2. Form the diagonal Degree matrix $\boldsymbol{D}$ defined by $D_{ii} = \sum_j W_{ij}$.
3. Form the normalized Laplacian matrix defined by $\boldsymbol{\mathcal{L}} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$.
4. Compute $\boldsymbol{x}_1, ..., \boldsymbol{x}_k$ the eigenvectors associated with the $k$ smallest eigenvalues of $\boldsymbol{\mathcal{L}}$ and stack them into the $N \times k$ matrix $\boldsymbol{X} = (\boldsymbol{x}_1|...|\boldsymbol{x}_k)$.

# Spectral clustering Algorithm II



## Algorithm

5. Form the $N \times k$ matrix $\boldsymbol{Y}$ from $\boldsymbol{X}$ by normalizing each row of $\boldsymbol{X}$ : $Y_{ij} = X_{ij} / \sqrt{\sum_l X_{lj}^2}$
6. Treating each row of $\boldsymbol{Y}$ as a point in $\mathbb{R}^k$, cluster them into $k$ clusters using K-means.
7. Assign data point $s_i$ to cluster $j$ iif row $i$ of $\boldsymbol{Y}$ was assigned to cluster $j$ at the previous step.

## Note

**The number of clusters can be chosen by considering the number of eigenvalues that are close to 0!**

# Spectral clustering Algorithm III



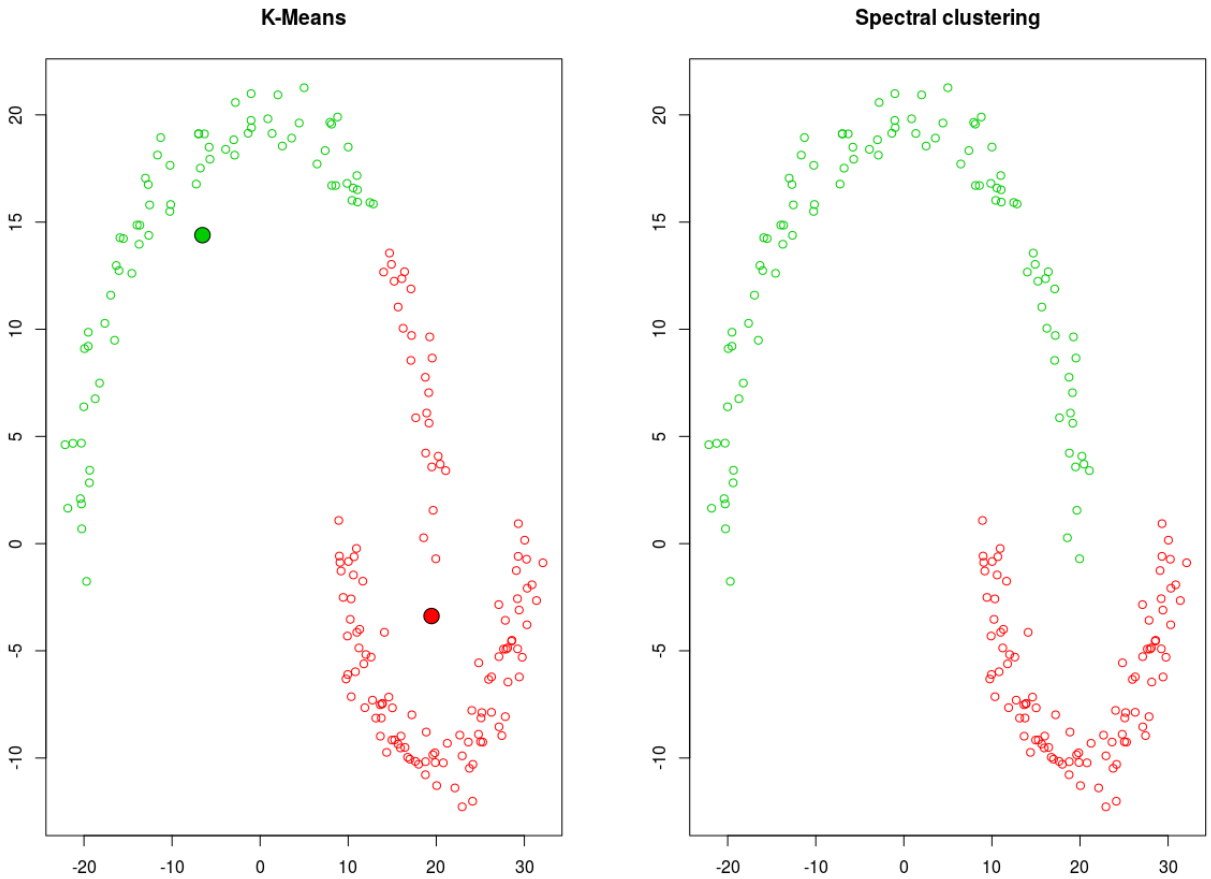



Figure: Comparison between K-Means and Spectral Clustering

# Spectral clustering Algorithm IV



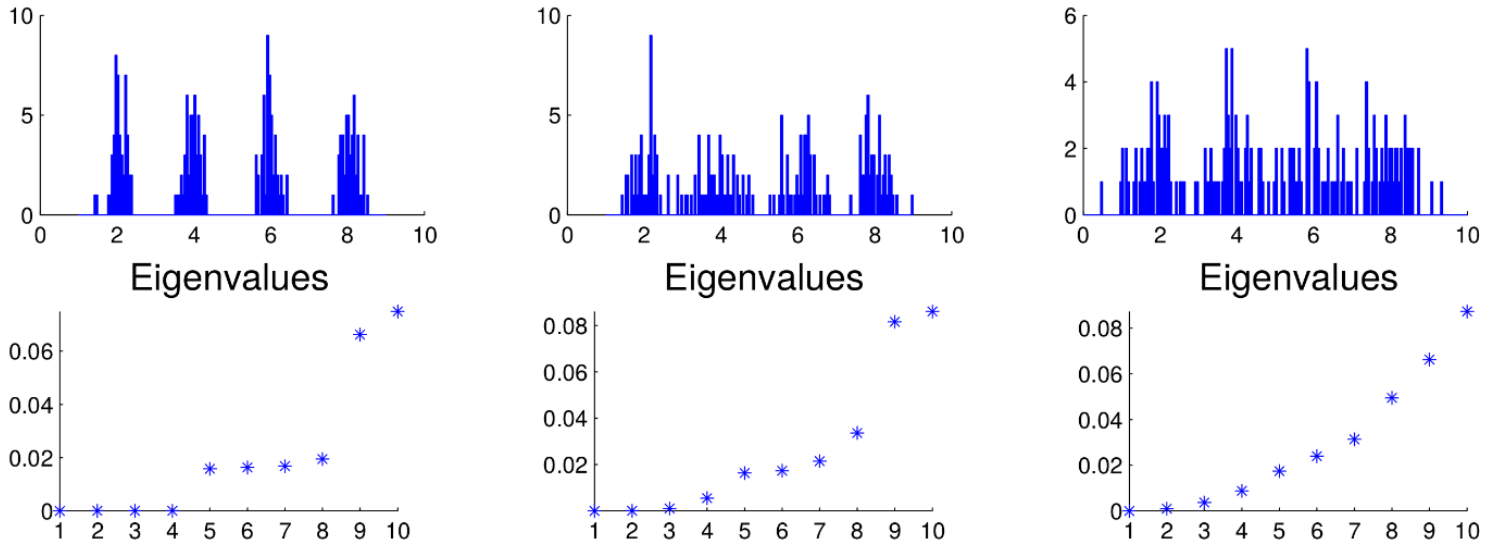



Figure: Different datasets and the eigenvalues of their corresponding Laplacians

# Outline





MINES ParisTech

# Semi-supervised learning problem



## Problem

Take a set of data points $\{x_i\}_{i=1}^N$ from $\mathbb{R}^d$. Suppose that only points $1, ..., n_l$ with $n_l \ll N$ are labeled and denote $\{y_i\}_{i=1}^{n_l}$ those labels.

We aim at :

- labeling the rest of the dataset (i.e., find $\{y_i\}_{i=n_l+1}^N$).
- Predict the label of new data points through a function $f : \mathbb{R}^d \mapsto \{\text{labels}\}$.

# Outline





MINES ParisTech

# SSL by Self Training I



## Self training algorithm

**Input** : labeled data $\mathfrak{L} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n_l}$ and unlabeled data $\mathfrak{U} = \{\boldsymbol{x}_i\}_{i=n_l+1}^{N}$

**Repeat** :

1. Train $f$ using $\mathfrak{L} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n_l}$
2. Apply $f$ to (some) $\mathfrak{U}$ and add them to $\mathfrak{L}$

Problems of this method :

- Heavily depends on the internal classifier
- Errors propagate (works well for well separated cluster, reacts badly to outliers)

Nobody uses it anymore apparently...

# SSL by Self Training II



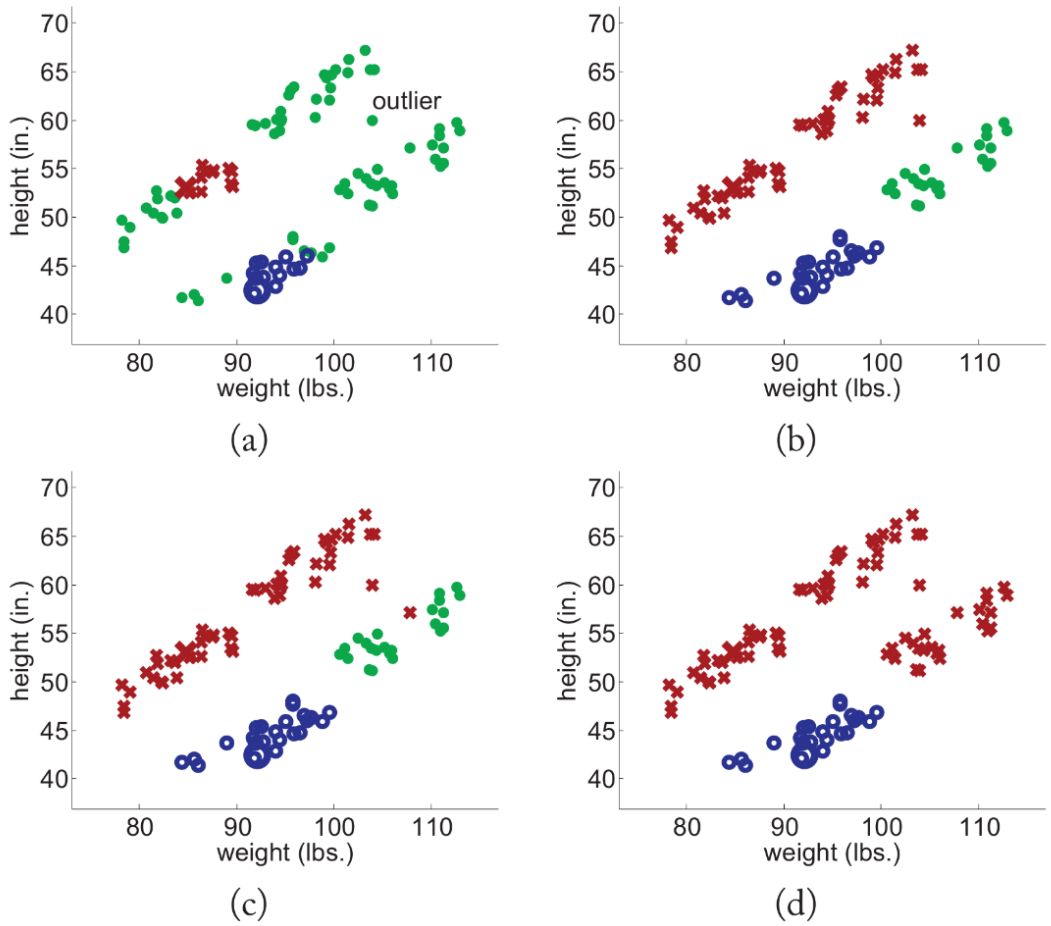



Figure: Behavior of a self training algorithm in presence of an outlier

# Outline





MINES ParisTech

# SSL and SVM



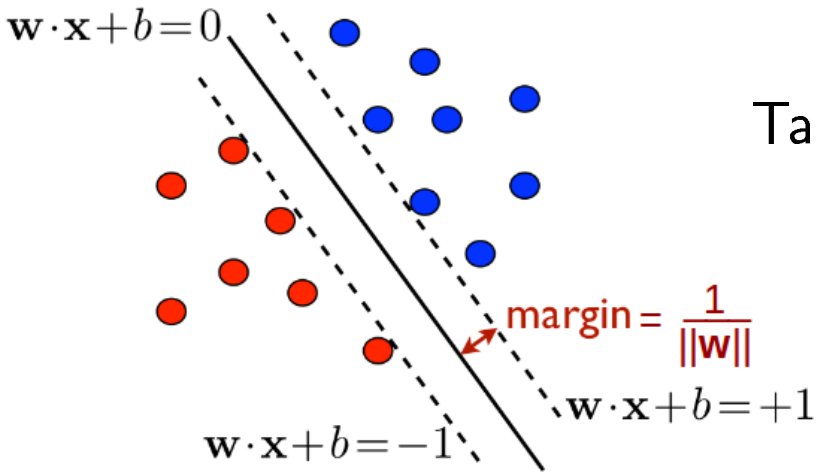
**Question**

How to use Support Vector Machine algorithm with partially labeled data?

## Classical SVM : separable case

**Data** : a labeled dataset $\mathfrak{L} = \{x_i, y_i\}_{i=1}^{n_l}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$.
**Problem** : find a linear classifier $f = w^T x + b$ s.t. $y = \text{sgn}(f)$
**Solution** : consider the hyperplane that maximizes the margin w.r.t the data points.

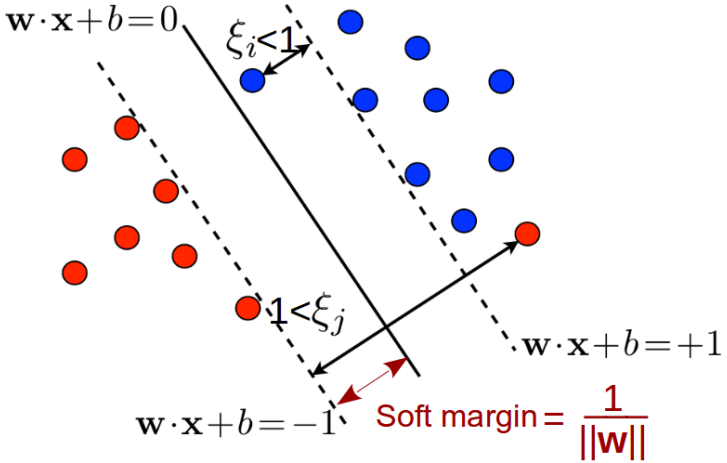
**Separable case**




Take $(w, b)$ as solutions of :

$$\min_{w,b} ||w||^2$$
$$\text{s.t.} \quad y_i(w^T x_i + b) \geq 1, i \in 1 \ldots n_l$$

## Non separable case




Take $(\boldsymbol{w}, b)$ as solutions of :

$$\min_{\boldsymbol{w},b} ||\boldsymbol{w}||^2 + C\sum_{i=1}^{n_l} \xi_i$$
$$\text{s.t. } y_i(\boldsymbol{w}^T x_i + b) \geq 1 - \xi_i \quad i \in 1...n_l$$
$$\xi_i \geq 0$$

With $C$ regularization parameter

This last problem can be written :

$$\min_{\boldsymbol{w},b} ||\boldsymbol{w}||^2 + C\sum_{i=1}^{n_l} \underbrace{h(y_i, f(\boldsymbol{x}_i))}_{\text{"cost"}}$$

Where $f(\boldsymbol{x}_i) := \boldsymbol{w}^T \boldsymbol{x}_i + b$ is the "prediction" at data point $\boldsymbol{x}_i$
And $h(y, f) := \max(1 - y \times f, 0)$ is the hinge loss function.

# SVM with unlabeled data



## Idea

Assign to unlabeled data their predicted label using the relation : $\widehat{y} = \text{sgn}(f(\boldsymbol{x})) = \text{sgn}(\boldsymbol{w}^T \boldsymbol{x} + b)$

## SVM Minimization problem with unlabeled data

The minimization problem becomes :

$$\min_{\boldsymbol{w},b} ||\boldsymbol{w}||^2 + C_1 \sum_{i=1}^{n_l} h(y_i, f(\boldsymbol{x}_i)) + C_2 \sum_{i=n_l+1}^{N} \widehat{h}(\widehat{y}_i, f(\boldsymbol{x}_i))$$

Where $h(y, f) := \max(1 - yf, 0)$ is the **hinge loss** function.
And $h(\widehat{y}, f) := \max(1 - sgn(f)f, 0) = \max(1 - |f|, 0)$ is the **hat loss** function.

# Outline





MINES ParisTech

# Outline





MINES ParisTech

# Label Completion in a dataset I



**Data** : A set of data points $\mathcal{V} = \{\boldsymbol{x}_i\}_{i=1}^{N}$ with labeled data $\mathfrak{L} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n_l}$ and unlabeled data $\mathfrak{U} = \{\boldsymbol{x}_i\}_{i=n_l+1}^{n_l+n_u}$. Labels are in $\{-1, 1\}$.
Denote $\mathcal{G}$ an associated similarity graph.
**Goal** : Find $f : \mathcal{V} \mapsto \mathbb{R}$ such that :

- $\forall l \in \mathfrak{L} : f(l) = y_i$
- $f$ can be used to assign labels to data points in $\mathfrak{U}$
- $f$ is "smooth" w.r.t. $\mathcal{G}$, i.e., vertices close to each other in the graph have similar labels.

**Solution** : Compute that minimizes an energy function :

$$E(f) := \frac{1}{2} \sum_{i,j} s_{ij} (f(i) - f(j))^2$$

With constraint $\forall l \in \mathfrak{L} : f(l) = y_l$

# Label Completion in a dataset II



Denote $\boldsymbol{f}$ the vector defined by $f_i = f(i) : i \in 1, ..., N$. $\boldsymbol{f}$ can be decomposed $\boldsymbol{f} = \begin{pmatrix} \boldsymbol{f}_{\mathfrak{L}} \\ \boldsymbol{f}_{\mathfrak{U}} \end{pmatrix}$

The energy $E(f)$ can be expressed using the Laplacian $\boldsymbol{L}$ of $\mathcal{G}$ :

$$E(f) = \boldsymbol{f}^T \boldsymbol{L} \boldsymbol{f} = \begin{pmatrix} \boldsymbol{f}_{\mathfrak{L}} \\ \boldsymbol{f}_{\mathfrak{U}} \end{pmatrix}^T \begin{pmatrix} \boldsymbol{L}_{\mathfrak{LL}} & \boldsymbol{L}_{\mathfrak{LU}} \\ \boldsymbol{L}_{\mathfrak{LU}}^T & \boldsymbol{L}_{\mathfrak{UU}} \end{pmatrix} \begin{pmatrix} \boldsymbol{f}_{\mathfrak{L}} \\ \boldsymbol{f}_{\mathfrak{U}} \end{pmatrix}$$

## Minimization Problem for label completion

The labeling function $\boldsymbol{f}^*$ is therefore a solution of the minimization problem :

$$\min_{\substack{\boldsymbol{f}_{\mathfrak{U}} \in \mathbb{R}^{nu} \\ \boldsymbol{f}_{\mathfrak{L}} = \boldsymbol{y}_{\mathfrak{L}}}} \begin{pmatrix} \boldsymbol{f}_{\mathfrak{L}} \\ \boldsymbol{f}_{\mathfrak{U}} \end{pmatrix}^T \boldsymbol{L} \begin{pmatrix} \boldsymbol{f}_{\mathfrak{L}} \\ \boldsymbol{f}_{\mathfrak{U}} \end{pmatrix}$$

# Label Completion in a dataset III



Any solution of this problem is **harmonic** over $\mathfrak{U}$, i.e.,

$$[\boldsymbol{L}\boldsymbol{f}^*]_{\mathfrak{U}} = \mathbf{0}_{\mathfrak{U}}$$

Or equivalently,

$$\forall u \in \mathfrak{U} : f^*(u) = \frac{1}{d_u} \sum_{i \sim u} s_{ui} f^*(i), \quad d_u = \sum_{i \sim u} s_{ui}$$

This property ensures that :

- The extrema of $\boldsymbol{f}^*$ on a connected component of $\mathcal{G}$ are reached on the labeled vertices $\Rightarrow \boldsymbol{f}^*$ is constant or
$$\forall u \in \mathfrak{U} : \min(\boldsymbol{f}^*_{\mathfrak{L}}) < f^*(u) < \max(\boldsymbol{f}^*_{\mathfrak{L}})$$
- $\boldsymbol{f}^*$ is unique

The solution $\boldsymbol{f}^*$ is explicitly given by the formulas :

$$\boldsymbol{f}^*_{\mathfrak{L}} = \boldsymbol{y}_{\mathfrak{L}}$$

$$\boxed{\boldsymbol{f}^*_{\mathfrak{U}} = -(\boldsymbol{L}_{\mathfrak{U}\mathfrak{U}})^{-1} \boldsymbol{L}_{\mathfrak{U}\mathfrak{L}} \boldsymbol{f}^*_{\mathfrak{L}}}$$

# Link with random walk I



Consider a random walk on $\mathcal{G}$ : we go from vertex $i$ to vertex $j$ with probability :

$$\mathbb{P}(i \to j) = \frac{s_{ij}}{\sum_{k \sim i} s_{ik}}$$

Take $u \in \mathfrak{U}$. To assign a label to $u$

- Start a random walk from $u$.
- Stop the random walk as soon as it hits a labeled node.
- Assign to $u$ the label of this node ($\in \{-1, 1\}$).

$f_u^*$ can be expressed as :

$$\boxed{f_u^* = \mathbb{P}(\text{assigned label} = 1) - \mathbb{P}(\text{assigned label} = -1)}$$

The larger $|f_u^*|$ the greater the confidence of $\text{sgn}(f_u^*)$ being the label of $u$.

# Link with random walk II



*Proof...*

The matrix $\boldsymbol{P}$ with coefficients $P_{ij} = \mathbb{P}(i \to j)$ (called **transition matrix**) can be expressed using the degree and the weight matrices of the graph: $\boldsymbol{P} = \boldsymbol{D}^{-1}\boldsymbol{W}$

And we have the relation :$\boldsymbol{L} := \boldsymbol{D} - \boldsymbol{W} = \boldsymbol{D}(\boldsymbol{I} - \boldsymbol{P})$

Which yields a new expression for $\boldsymbol{f}^*_{\mathfrak{U}}$ :

$$\boldsymbol{f}^*_{\mathfrak{U}} = (\boldsymbol{I}_{\mathfrak{U}\mathfrak{U}} - \boldsymbol{P}_{\mathfrak{U}\mathfrak{U}})^{-1}\boldsymbol{P}_{\mathfrak{U}\mathfrak{L}}\boldsymbol{f}^*_{\mathfrak{L}} = \sum_{n\in\mathbb{N}}(\boldsymbol{P}_{\mathfrak{U}\mathfrak{U}})^n\boldsymbol{P}_{\mathfrak{U}\mathfrak{L}}\boldsymbol{f}^*_{\mathfrak{L}}$$

Therefore, for $u \in \mathfrak{U}$

$$f^*_u = \sum_{n\in\mathbb{N}}\sum_{v\in\mathfrak{U}}[(\boldsymbol{P}_{\mathfrak{U}\mathfrak{U}})^n]_{uv}[\boldsymbol{P}_{\mathfrak{U}\mathfrak{L}}\boldsymbol{f}^*_{\mathfrak{L}}]_v$$

Where $[(\boldsymbol{P}_{\mathfrak{U}\mathfrak{U}})^n]_{uv}$ = proba to hit $v$ from $u$ through only vertices $\mathfrak{U}$

And $[\boldsymbol{P}_{\mathfrak{U}\mathfrak{L}}\boldsymbol{f}^*_{\mathfrak{L}}]_v$ = from $v$, proba to hit a vertex labeled $+1$ − proba to hit a vertex labeled $-1$

# Regularized Harmonic function I



**Question**

What if we want to control the confidence?

**Solution** : Allow the random walk to stop before it hits a labeled vertex $\Rightarrow$ Add an artificial vertex to the the graph s.t.

- It is linked to all the other vertices of the graph with a constant weight $\gamma_g$
- It has a label 0

$\Rightarrow$ The proba of transition of the RW will all be divided by a factor $(1 + \gamma_g)$ !

# Regularized Harmonic function II



The new Laplacian $\boldsymbol{L}_g$ can be expressed :

$$\boldsymbol{L}_g = \begin{pmatrix} \boldsymbol{L} + \gamma_g \boldsymbol{I} & -\gamma_g \mathbf{1}_N \\ -\gamma_g \mathbf{1}_N^T & N\gamma_g \end{pmatrix}$$

And the harmonic solution on this graph will simply be :

$$\boldsymbol{f}_{\mathfrak{U}}^* = (\boldsymbol{L}_{\mathfrak{U}\mathfrak{U}} + \gamma_g \boldsymbol{I}_{\mathfrak{U}\mathfrak{U}})^{-1} \boldsymbol{W}_{\mathfrak{U}\mathfrak{L}} \boldsymbol{f}_{\mathfrak{L}}^*$$

Simply add $\gamma_g$ to the diagonal of $\boldsymbol{L}$ in the previous setting!

# Soft Harmonic function



What if we don't trust the labels in the dataset?
$\Rightarrow$ Add an additional cost to the minimization :

## Minimization Problem with soft constraints

The solution $\boldsymbol{f}^*$ of the labeling problem is given by :

$$\boldsymbol{f}^* = \arg\min_{\boldsymbol{f}\in\mathbb{R}^N} \boldsymbol{f}^T \boldsymbol{Q} \boldsymbol{f} + (\boldsymbol{f}-\boldsymbol{y})^T \boldsymbol{C} (\boldsymbol{f}-\boldsymbol{y})$$

Where : $\boldsymbol{Q} = L$ or $\boldsymbol{Q} = \boldsymbol{L} + \gamma_g \boldsymbol{I}$

$\boldsymbol{y}$ is defined by $y_i = \begin{cases} \text{true label} & \text{for labeled points} \\ 0 & \text{otherwise} \end{cases}$

$\boldsymbol{C}$ is diagonal with $C_{ii} = \begin{cases} c_l & \text{for labeled points} \\ c_u & \text{otherwise} \end{cases}$

The solution of this problem is given explicitly by the formula :

$$\boldsymbol{f}^* = (\boldsymbol{C}^{-1}\boldsymbol{Q} + \boldsymbol{I})^{-1}\boldsymbol{y}$$

# Outline





MINES ParisTech

# Out of sample extension



**Problem**

Suppose that a new (unlabeled) data point $x_{N+1}$ arrives. How to predict its label?

2 options :

1. Add it to the dataset and start over... Not optimal
2. Train the algorithm on the dataset so that it can predict labels for new data points... Better!

**Solution**: Manifold Regularization

$\Rightarrow$ Allow $f$ to be defined everywhere : $f : \mathcal{X} \subset \mathbb{R}^d \mapsto \mathbb{R}$

$\Rightarrow$ Allow $f(\boldsymbol{x}^i) \neq y_i$ at labeled examples (to deal with noise).

# Manifold Regularization I



Consider a kernel $\mathcal{K}$ (Mercer).We will look for solutions that can be written :

$$f_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}_i) : \boldsymbol{\alpha} \in \mathbb{R}^N$$

## Minimization Problem in manifold regularization

The coefficients $\boldsymbol{\alpha}^*$ of the labeling function $\boldsymbol{f}^*$ are :

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha} \in \mathbb{R}^N} \sum_{l \in \mathfrak{L}} V(f_{\boldsymbol{\alpha}}(\boldsymbol{x}_l), y_l) + \lambda_1 ||f_{\boldsymbol{\alpha}}||^2_{\mathcal{K}} + \lambda_2 \boldsymbol{f}^T_{\boldsymbol{\alpha}} \boldsymbol{L} \boldsymbol{f}_{\boldsymbol{\alpha}}$$

Where $V(f(\boldsymbol{x}), y)$ is a function associating a cost to the prediction of label $y$ of $\boldsymbol{x}$ by the value $f(\boldsymbol{x})$

- $V(f(\boldsymbol{x}), y) = (y - f(\boldsymbol{x}))^2 \Rightarrow$ "Laplacian Regularized Least Squares"
- $V(f(\boldsymbol{x}), y) = max(0, 1 - yf(\boldsymbol{x})) \Rightarrow$ "Laplacian SVM"

# Manifold Regularization II



$$\boldsymbol{\alpha}^* = \arg \min_{\boldsymbol{\alpha} \in \mathbb{R}^N} \sum_{l \in \mathfrak{L}} V(f_{\boldsymbol{\alpha}}(\boldsymbol{x}_l), y_l) + \lambda_1 ||f_{\boldsymbol{\alpha}}||_{\mathcal{K}}^2 + \lambda_2 \boldsymbol{f}_{\boldsymbol{\alpha}}^T \boldsymbol{L} \boldsymbol{f}_{\boldsymbol{\alpha}}$$

- Denote $\boldsymbol{K} := [\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j \in 1 \dots N}$. $||f_{\boldsymbol{\alpha}}||_{\mathcal{K}}^2 := \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha}$ characterizes the complexity of $f_{\boldsymbol{\alpha}}$.
- $\lambda_1$ controls the "overall" complexity of the solution
- $\lambda_2$ controls the smoothness of the solution w.r.t the intrinsic structure of the data points

# Manifold Regularization III



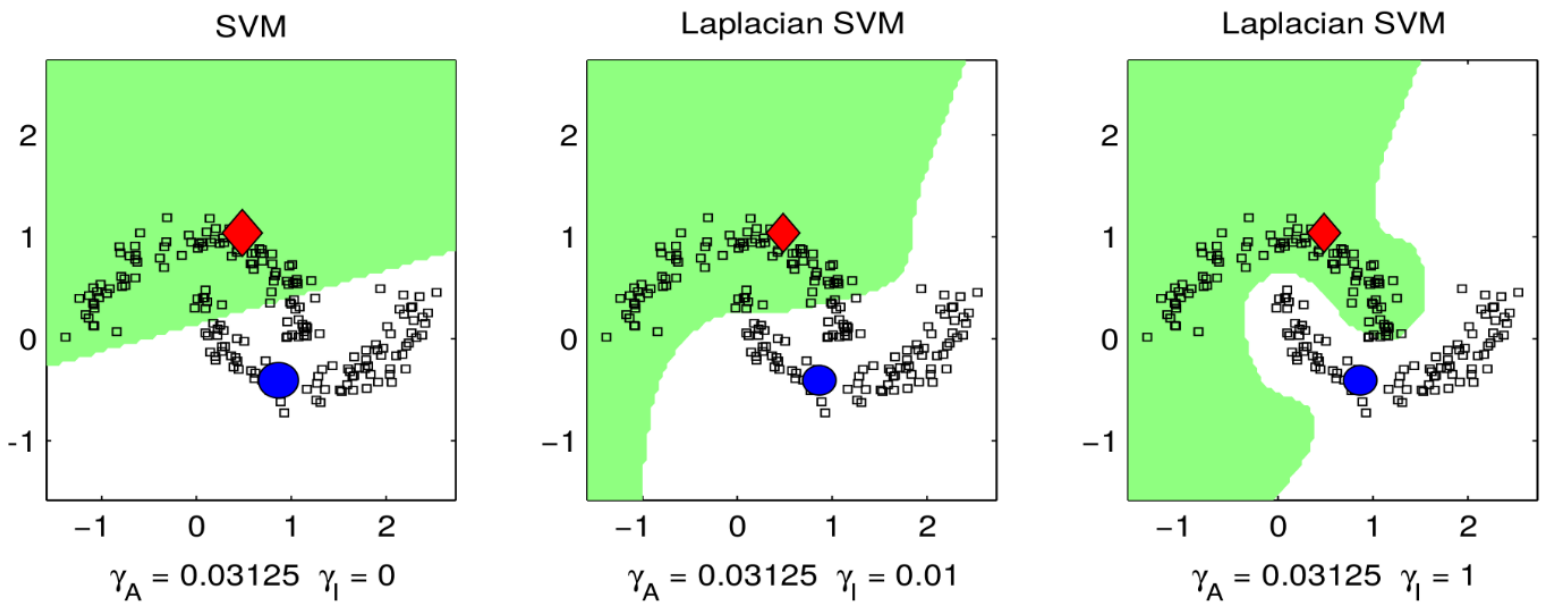



Figure: Results of SVMs on a dataset