INF5300 – Basics of graph-based semi-supervised learning (SSL)

- SSL and often-made assumptions
- Graph representation
- Label propagation and measure of label non-smoothness
- Example: User-guided image segmentation

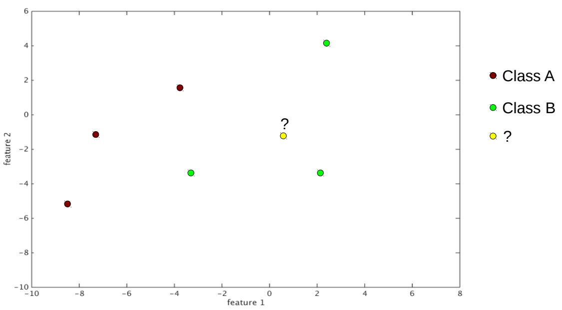
Curriculum: Book “Semi-Supervised Learning”, O. Chapelle et. al. 2006;
Sections 1.1 and 1.2 in the introductory chapter, 11.1, 11.2 (not algorithm 11.3 and its explanatory text), 11.3.1 (first page only), 11.3.2 (we skip the “alternative criterion”), 11.3.3, 11.3.4, 11.5.

See web page for links to pdfs. Note that we also link to some supplementary reading material.

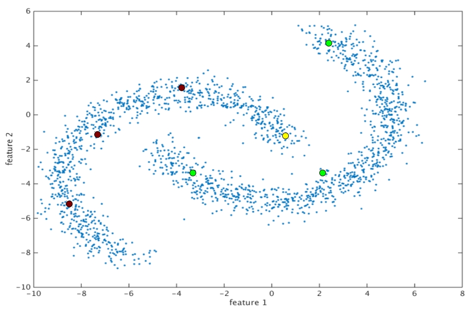
SSL – Motivation

- Typically limited amount of labeled data available
- Often a lot of unlabeled data
- **Goal:** Use both labeled and unlabeled data to build better classifiers than possible when using labeled data alone

Example: SSL vs. supervised



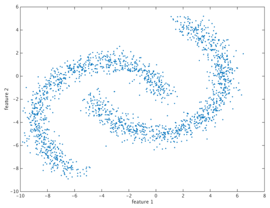
Example: SSL vs. supervised cont.



By labeling it red, you are likely assuming there is *class-relevant* information in the *distribution* of the unlabeled data.

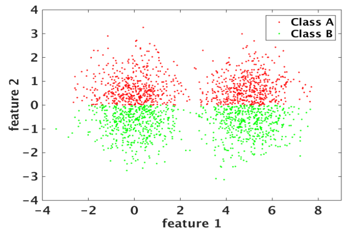
Assumptions

- Smoothness assumption
  - If two points in a high-density region are close, then so should be the corresponding outputs
- Cluster assumption
  - If points are in the same cluster, they are likely to be of the same class
- Low-density separation assumption
  - Decision boundary should lie in a low-density region
- Manifold assumption
  - The (high-dimensional) data lie (roughly) on a low-dimensional manifold



Assumptions cont.

Note: The assumptions do not always hold!



Transductive vs. inductive learning

- Transductive learning
  - Perform predictions only for the *given* unlabeled data
  - For new samples; relabel *all* data
- Inductive learning
  - Construct a prediction function defined on the entire feature space
- Transductive learning is our focus here

A sidenote: Self-learning

- Probably the very first attempt at SSL
- Inductive learning
- Wrapper based; based on given classifier
- Can be quite powerful
- Linked to EM-type of SSL
- Can deteriorate with increased number of unlabeled data; typically caused by model misspecification

1. Use labeled samples to build supervised classifier
2. Classify unlabeled data
3. Set our most confident predictions to "labeled"
4. Repeat until convergence or reached max number of iterations

Graph representation

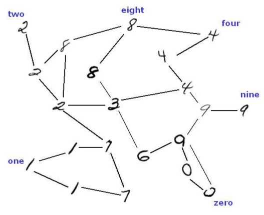


Fig. from Zhu et al., 2003.

- Nodes are labeled and unlabeled data points
- Edges reflect “similarity”
- We will assume: Similar points have similar labels
- Labels “propagate” through the graph

Why graph-based SSL?

- Very versatile
  - Some datasets are naturally represented by graphs
  - .. others easily converted into them
  - “Similarity” in nodes can cover broadly
  - One framework to cover them all
- Scalable to large datasets (easily parallelizable)
- Often easily implemented, and often do well
- Why not; creating good graphs is not always easy!, and they are more or less intrinsically transductive.

Our graphs and notation

- We assume two classes (extension to 3+ later)
  - Each node $y_i \in \{-1, 1\}$
- Edges encoded in similarity matrix **W**

Symmetric

  - E.g. $w_{i,j} = e^{-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{\sigma^2}}$, where $\mathbf{x}_i$ are features of sample *i*

Note: "E.g."!

  - More on this later

Graph smoothness

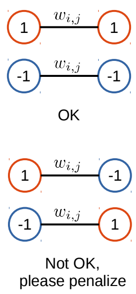
- Let **y** be a given set of labels
- Measure non-smoothness
- A useful, concrete example of a such a measure:

$$\frac{1}{2}\sum_{i,j} w_{i,j}(y_i - y_j)^2$$

½ for later convenience

One can also think of this as an "energy" term (which we will later minimize)

If two nodes have unequal labels, penalize by how "similar" they are (given by the weight, w_{ij})



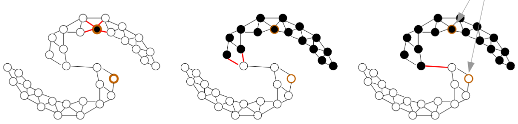
Example | Non-smoothness measure

- $E(\mathbf{y}) = \frac{1}{2}\sum_{i,j} w_{i,j}(y_i - y_j)^2$

Without, all nodes either black or white gives E(**y**)=0!

- Let $w_{i,j}$ be either 0 or 1

Labeled data / fixed nodes



E(**y**)=4*4=16 E(**y**)=2*4=8 E(**y**)=1*4=4

(Figs courtesy of Z. Ghahramani.)

- "Minimum cut" gives here an optimum

.. and in matrix notation

$$\begin{aligned} E(\mathbf{y}) &= \frac{1}{2}\sum_{i,j} w_{i,j}(y_i - y_j)^2 \\ &= \frac{1}{2}\left(2\sum_{i=1}^{n} y_i^2 \sum_{j=1}^{n} \mathbf{W}_{ij} - 2\sum_{i,j=1}^{n} \mathbf{W}_{ij} y_i y_j\right) \\ &= \mathbf{y}^{\top}(\mathbf{D} - \mathbf{W})\,\mathbf{y} \\ &= \mathbf{y}^T \mathbf{L} \mathbf{y} \end{aligned}$$

Known as the graph's Laplacian matrix

D is a diagonal matrix containing each node's outbound weights summed:

$$D = \begin{bmatrix} \sum w_{1:} & & 0 \\ & \cdots & \\ 0 & & \sum w_{n:} \end{bmatrix}$$

Relaxing labels

- Turning our crisp labels (-1 and 1) into real-valued ones makes optimizing E(**y**) much simpler and faster
- So, let us use "soft" labels: $y_i \in \Re$
  - We get the final labels later by thresholding (e.g. by sign(y_i))
- Manipulating and optimizing E(**y**) = $\mathbf{y}^T\mathbf{L}\mathbf{y}$ can now be performed as any other real-valued quadratic function

Finding a solution I/II

- We want to minimize E(**y**) = $\mathbf{y}^T\mathbf{L}\mathbf{y}$ s.t. $\mathbf{y}_l$ fixed
- For simplicity, re-order the samples such that the labeled ones come first:

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_l \\ \mathbf{y}_u \end{bmatrix}$$

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_{ll} & \mathbf{L}_{lu} \\ \mathbf{L}_{ul} & \mathbf{L}_{uu} \end{bmatrix}$$

Note that $\mathbf{L}_{lu} = \mathbf{L}_{ul}^T$

Finding a solution II/II

- We can now write E(**y**) like:

$$\mathbf{y}^T\mathbf{L}\mathbf{y} = \mathbf{y}_l^T\mathbf{L}_{ll}\mathbf{y}_l + \mathbf{y}_u^T\mathbf{L}_{lu}^T\mathbf{y}_l + \mathbf{y}_l^T\mathbf{L}_{lu}\mathbf{y}_u + \mathbf{y}_u^T\mathbf{L}_{uu}\mathbf{y}_u$$

- Its derivative w.r.t. the unlabeled data:

$$\frac{\partial E}{\partial \mathbf{y}_u} = \ldots = 2\mathbf{L}_{lu}^T\mathbf{y}_l + 2\mathbf{L}_{uu}\mathbf{y}_u$$

- Setting $\frac{\partial E}{\partial \mathbf{y}_u} = 0$ gives:

$$\boxed{\mathbf{y}_u = -\mathbf{L}_{uu}^{-1}\mathbf{L}_{lu}^T\mathbf{y}_l}$$

Note that $\mathbf{L}_{lu} = -\mathbf{W}_{lu}$

Solving iteratively

- Why? Handle large data sets (matrix inversion slow, popular solution easily parallelizable)
- Jocobi iterations (at least inspired by such):

$$\text{iterate} \begin{cases} \mathbf{y}^{(k+1)} = \mathbf{D}^{(-1)}\mathbf{W}\mathbf{y}^{(k)} \\ \mathbf{y}_l^{(k+1)} = \mathbf{y}_l \end{cases}$$

Each node's label is just the weighted average of its neighbors' current values!

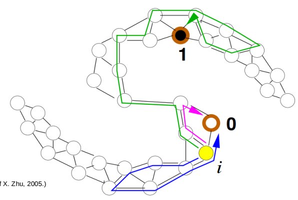
- One can of course also solve using other numeric schemes:
  - Gradient descent (repeat moving slightly along the gradient)
  - Gauss-Seidel
  - Conjugate gradient
- Jacob iterations are simple, and very much suited for parallelization

Interpretation as random walk

$$P(i|j) = \frac{w_{i,j}}{\sum_j w_{i,j}}$$

$$y_i = P(\text{reach label } 1|\text{starting at node } i)$$

The two classes are here encoded as y_i=0 and y_i=1, respectively



(Fig. courtesy of X. Zhu, 2005.)

There is also an interpretation as node voltage on an electric network

Multiple classes

- Replace **y** with (soft) indicator matrix **Y**
  - **Y** is an *n* x *#classes* matrix; each row corresponds to a class and has 1 for samples of that class, 0 otherwise
  - Example: **Y** version of our two-class **y**;

$$\left[\frac{1}{2}(\mathbf{y}+1) \quad -\frac{1}{2}(\mathbf{y}-1)\right]$$

- Process each column independently/simultaneously
  - Final class label for sample *i* is $\max_k \hat{y}_{i,k}$

Class priors

- We will focus on *class mass normalization*
- After propagating the labels (finding $\mathbf{y}_u$), the average "mass" of class k is

$$m_k = \frac{1}{n}\sum_i y_{i,k}$$

The average of column *k* of a "soft" indicator matrix **Y**

- Let p_k be our chosen prior for class k
- We now scale each column of $\hat{\mathbf{Y}}$ by p_k/m_k such that we have $\hat{m}_i = p_k$ before applying the "max-column" classification rule from the previous slide

More on the weight matrix

See also the "graph from images" slide (that we will get to); can have multiple components in the weight term.

- k-NN
  - Might yield asymmetric graphs, and irregular graphs
  - Naturally sparse graphs
- ε-NN
  - Not scale invariant, and might get fully disconnected components
- Gaussian kernel
  - Result heavily dependent on parameters ( ) (as for all the others)
  - Too small σ → 1-NN
  - Too large σ → ignoring unlabeled data (ignoring "structure" in data)

$$w_{i,j} = e^{-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{\sigma^2}}$$

$$w_{i,j} = e^{-(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{A} (\mathbf{x}_i - \mathbf{x}_j)}$$

Eigendecomposition of the Laplacian

Relationship between Eigenvalues of the Laplacian and Smoothness

Eigenvector of L

Eigenvalue of L

$$Lg = \lambda g$$

$$g^T L g = \lambda \boxed{g^T g}$$

= 1, as we can say ||**g**||=1 without loss of generality

$$g^T L g = \lambda$$

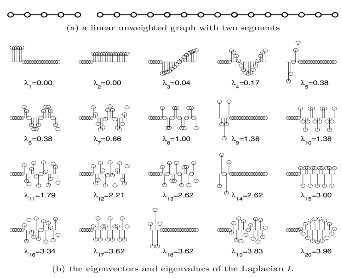
Measure of Non-Smoothness (previous slide)

If an eigenvector is used to classify nodes, then the corresponding eigenvalue gives the measure of non-smoothness

(Fig. courtesy of A. Subramanya & P.P. Talukdar, 2012.)

Eigendecomposition of the Laplacian cont.

- Example of a graph and an eigendecomposition of its Laplacian matrix →
- Note how these eigenvectors can be used for grouping / segmenting the nodes / samples (cf. image segmentation)
  - Very much related to "minimum cut"-based algorithms



(Fig. courtesy of X. Zhu, 2006.)

Ease trust in labeled data

- Noise in labeled data?
- Might reduce "overfitting"
- New criterion function (replacing E(**y**)):

A popular variant termed Modified Absorption (MAD), replaces the last term by one that penalizes distances from a per-sample "prior", instead of distances to zero

Look: Unconstrained!

$$C(\hat{Y}) = \|\hat{Y}_l - Y_l\|^2 + \mu\hat{Y}^\top L\hat{Y} + \mu\epsilon\|\hat{Y}\|^2$$

Keep original labeling

Smoothness

Extra: Regularization / numerical stability

$\min_{\hat{\mathbf{Y}}} C$

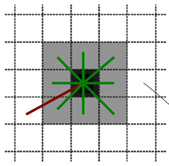
$$\boxed{\hat{Y} = (\mathbf{S} + \mu L + \mu\epsilon\mathbf{I})^{-1}\,\mathbf{S}Y}$$

Let **S** be a diagonal indicator matrix for labeled data: $\|\hat{\mathbf{Y}}_l - \mathbf{Y}_l\|^2 = \|\mathbf{S}\hat{\mathbf{Y}} - \mathbf{S}\mathbf{Y}\|^2$

Example of graph from single image

.. when the goal is image segmentation

- Pixels are nodes
- Edges (weights):
  - 1) Encode both the influence of spatial distance between a pair of pixels (e.g. in extremity; set weights to zero for pixels not bordering each other)
  - 2) .. and a measure of similarity of pixels (intensity, color, or texture)



Green: Nonzero-weighted edge. All others (including the red example red edge) no edge (having zero edge weight).

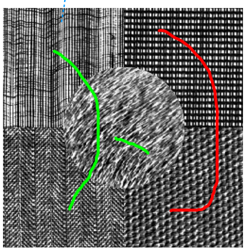
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Possibly multi-featured pixels

Example: User-guided segmentation

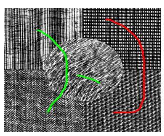
- Nodes / pixels
  - Local-texture description
    - Absolute values of discrete Fourier transform coefficients
- Edges
  - Non-zero for 8-neighbors
    - Gaussian kernel, single σ^2
    - Here, no data-adaptivity
- Initial labels
  - Provided by user
    - Green and red classes (y_i=-1 or y_i=1)

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

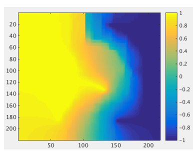


Example: User-guided segmentation cont.

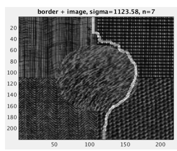
Input



Labels propagated



Thresholded
(and border extracted)



Summary

- SSL and often-made assumptions .. which do not always hold!
- Graph representation | Nodes and edges (datapoints and their similarity)
- Non-smoothness penalty term [$E(\mathbf{y})=\mathbf{y}^T\mathbf{L}\mathbf{y}$] | .. and finding its minimum (label propagation)
- Building the graph / finding edge-weights | k-NN, ε-NN, Gaussian kernel etc. Parameters!
- Eigendecomposition of the Laplacian matrix | Connectivity and partitioning
- Ease trust in labeled data | Replace $E(\mathbf{y})$ with other quadratic criterion
- Image → graph | Cf. the example of user-guided image segmentation