Semi-Supervised Learning: A Brief Survey

深圳大學
SHENZHEN UNIVERSITY

Muhammed Jamshed Alam Patwary
PhD Research Fellow, Big Data Institute
College of Computer Science and Software Engineering
Shenzhen University

Outline

- Introduction to Semi-Supervised Learning
- Semi-Supervised Learning Algorithms
  - Self-training
  - Co-training
  - Multiview Learning
  - Fuzziness based Semi-Supervised Learning
  - EM with generative mixture models
  - Semi-supervised support vector machine and Entropy regularization
  - Graph-based SSL
- Which semi-supervised learning method should I use?
- Some Challenges for Future Research

Disclaimer:

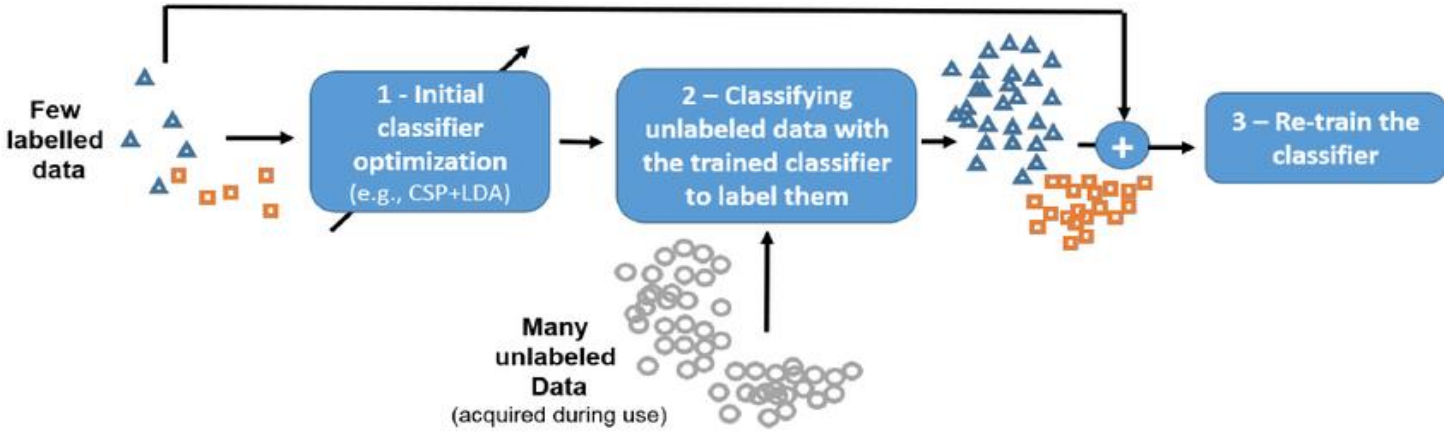
Some of the pictures and slides are taken from Xiaojin Zhu's(University of Wisconsin, Madison, USA) presentation slides.

Introduction to Semi-Supervised Learning

The Traditional View:

- Labeled instances are difficult to get
  - Expensive and time consuming to obtain.
  - They require the effort of experienced human annotator.
- Unlabeled data is cheap

- **Semi-supervised learning** is a class of supervised learning tasks and techniques that also make use of unlabeled data for training
- 1965, Scudder



Introduction to Semi-Supervised Learning

- Why Semi-supervised learning?
- The learning problem
  - Goal: Using both labeled and unlabeled data to build better learners, then using each one alone.

Notation:

- input features x, label y
- learner $f : \mathcal{X} \mapsto \mathcal{Y}$
- labeled data $(X_l, Y_l) = \{(x_{1:l}, y_{1:l})\}$
- unlabeled data $X_u = \{x_{l+1:n}\}$
- usually $l \ll n$

How can X_u help?

Introduction to Semi-Supervised Learning

- The landscape

supervised learning (classification, regression)

$\{(x_{1:n}, y_{1:n})\}$

$\updownarrow$

semi-supervised classification/regression

$\{(x_{1:l}, y_{1:l}), x_{l+1:n}\}$

$\updownarrow$

semi-supervised clustering $\{x_{1:n}$, must-, cannot-links$\}$

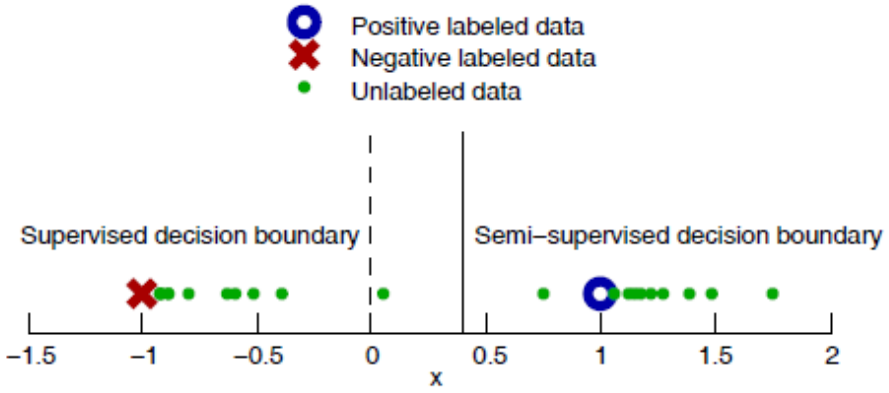
$\updownarrow$

unsupervised learning (clustering) $\{x_{1:n}\}$

transduction (limited to $x_{1:n}$) $\leftrightarrow$ **induction** (unseen data)

Introduction to Semi-Supervised Learning

How can unlabeled data ever help?



- assuming each class is a coherent group (e.g. Gaussian)
- with and without unlabeled data: decision boundary shift

Semi-Supervised Learning Algorithms- Self-training

Self-training:

1. Train f from (X_l, Y_l)
2. Predict on $x \in X_u$
3. Add $(x, f(x))$ to labeled data
4. Repeat

- Variations in Self-training
  - Add a few most confident $(x, f(x))$ to labeled data
  - Add all $(x, f(x))$ to labeled data
  - Add all $(x, f(x))$ to labeled data, weigh each by confidence

Semi-Supervised Learning Algorithms-Self-training

Self-training example: image categorization

1. Train a naïve Bayes classifier on the two initial labeled images

2. Classify unlabeled data, sort by confidence $\log p(y = \text{astronomy}|x)$

Semi-Supervised Learning Algorithms-Self-training

Self-training example: image categorization

3. Add the most confident images and predicted labels to labeled data

4. Re-train the classifier and repeat

Semi-Supervised Learning Algorithms-Self-training

- Advantages of Self-training
  - The simplest semi-supervised learning method.
  - A wrapper method, applies to existing (complex) classifiers.
  - Often used in real tasks like natural language processing.
- Disadvantages of Self-training
  - Early mistakes could reinforce themselves.
    - Heuristic solutions, e.g. "un-label" an instance if its confidence falls below a threshold.
  - Cannot say too much in terms of convergence.
    - But there are special cases when self-training is equivalent to the Expectation-Maximization (EM) algorithm.
    - There are also special cases (e.g., linear functions) when the closed-form solution is known.

Co-training

Co-training

Two views of an item: image and HTML text

Co-training

Feature split

Each instance is represented by two sets of features $x = [x^{(1)}; x^{(2)}]$

- $x^{(1)}$ = image features
- $x^{(2)}$ = web page text
- This is a natural feature split (or multiple views)

Co-training idea:

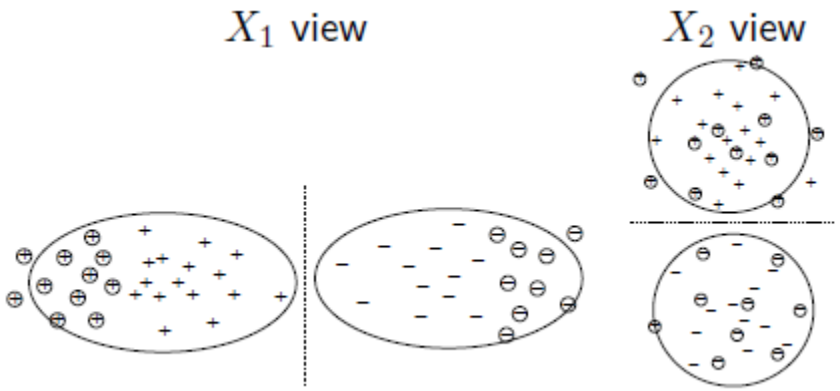
- Train an image classifier and a text classifier
- The two classifiers teach each other

Co-training

Co-training assumptions

Assumptions

- feature split $x = [x^{(1)}; x^{(2)}]$ exists
- $x^{(1)}$ or $x^{(2)}$ alone is sufficient to train a good classifier
- $x^{(1)}$ and $x^{(2)}$ are conditionally independent given the class



Co-training

Co-training algorithm

Co-training algorithm

1. Train two classifiers: $f^{(1)}$ from $(X_l^{(1)}, Y_l)$, $f^{(2)}$ from $(X_l^{(2)}, Y_l)$.
2. Classify X_u with $f^{(1)}$ and $f^{(2)}$ separately.
3. Add $f^{(1)}$'s k-most-confident $(x, f^{(1)}(x))$ to $f^{(2)}$'s labeled data.
4. Add $f^{(2)}$'s k-most-confident $(x, f^{(2)}(x))$ to $f^{(1)}$'s labeled data.
5. Repeat.

Co-training

Pros and cons of co-training

Pros

- Simple wrapper method. Applies to almost all existing classifiers
- Less sensitive to mistakes than self-training

Cons

- Natural feature splits may not exist
- Models using BOTH features should do better

Co-training

Variants of co-training

Co-EM: add all, not just top k

- Each classifier probabilistically label X_u
- Add (x, y) with weight $P(y|x)$

Fake feature split

- create random, artificial feature split
- apply co-training

Multiview: agreement among multiple classifiers

- no feature split
- train multiple classifiers of different types
- classify unlabeled data with all classifiers
- add majority vote label

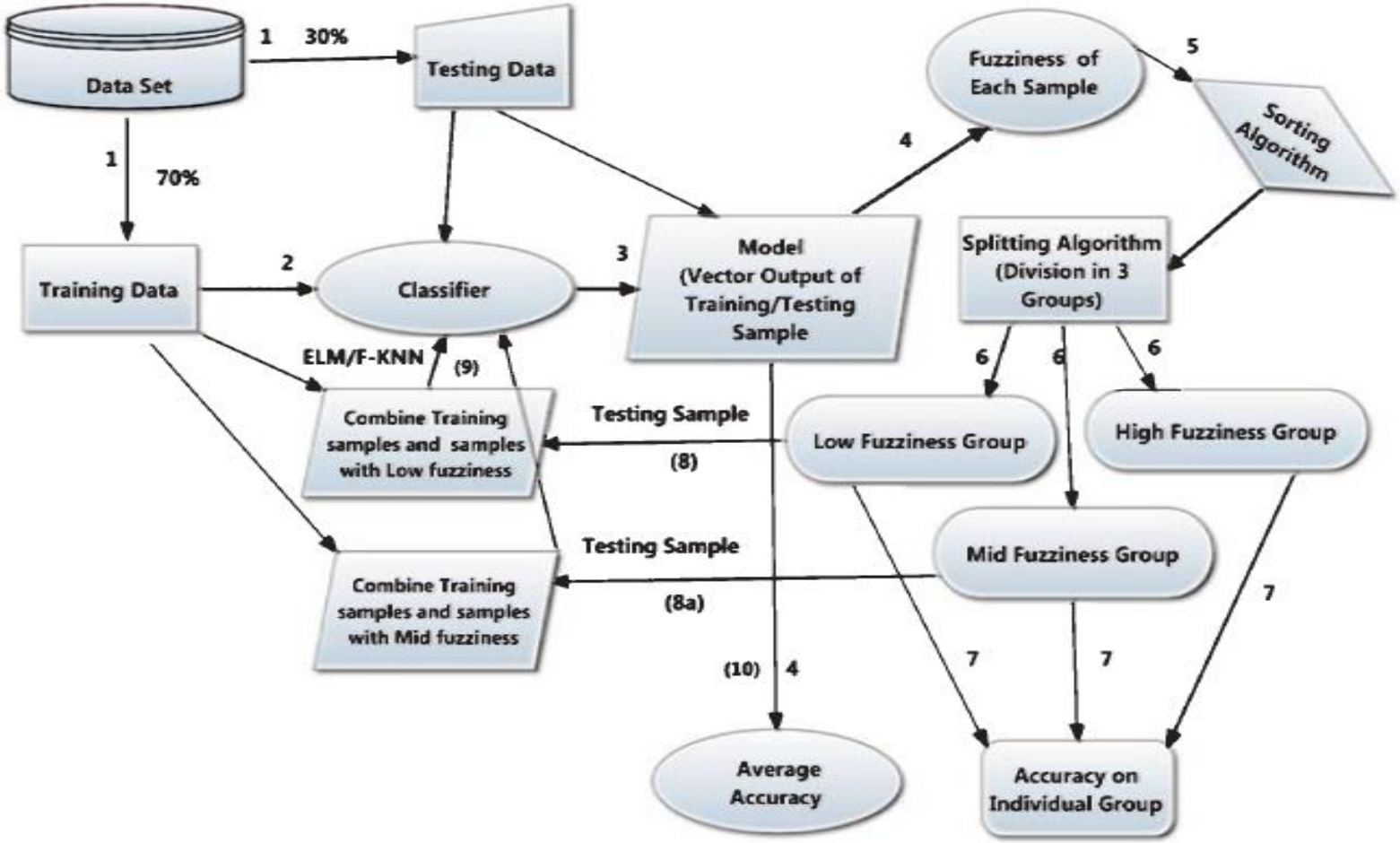
Multiview Learning

A regularized risk minimization framework to encourage multi-learner agreement:

$$\min_f \sum_{v=1}^{M} \left(\sum_{i=1}^{l} c(y_i, f_v(x_i)) + \lambda_1 \|f\|_K^2 \right) + \lambda_2 \sum_{u,v=1}^{M} \sum_{i=l+1}^{n} (f_u(x_i) - f_v(x_i))^2$$

M learners. $c()$ is the loss function, e.g., hinge loss.

Fuzziness based semi-supervised learning



- **EM with generative mixture models**

Fuzzy Cluster

- In hard clustering methods
  - Every data object is assigned to exactly one cluster
- Some applications may need for fuzzy or soft cluster assignment
  - Ex. An e-game could belong to both entertainment and software
- Example: Popularity of cameras is defined as a fuzzy mapping

Camera	Sales (units)
A	50
B	1320
C	860
D	270

$$\text{Pop}(o) = \begin{cases} 1 & \text{if } 1,000 \text{ or more units of } o \text{ are sold} \\ \frac{i}{1000} & \text{if } i \ (i < 1000) \text{ units of } o \text{ are sold} \end{cases}$$

- Then, *A*(0.05), *B*(1), *C*(0.86), *D*(0.27)

Fuzzy (Soft) Clustering

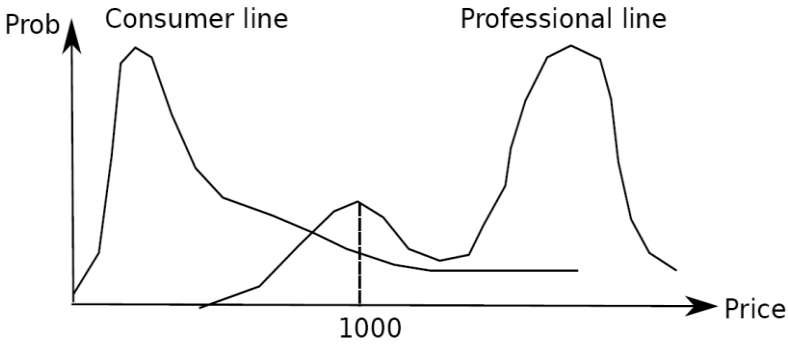
Review-id	Keywords
R_1	digital camera, lens
R_2	digital camera
R_3	lens
R_4	digital camera, lens, computer
R_5	computer, CPU
R_6	computer, computer game

$$M = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ \frac{2}{3} & \frac{1}{3} \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

- Example: Let cluster features be
  - C_1 :"digital camera" and "lens"
  - C_2: "computer"
- Fuzzy clustering
  - k fuzzy clusters C_1, …,C_k ,represented as a partition matrix M = [w_{ij}]
  - P1: for each object o_i and cluster C_j, $0 \le w_{ij} \le 1$ (fuzzy set)
  - P2: for each object o_i, $\sum_{j=1}^{k} w_{ij} = 1$, equal participation in the clustering
  - P3: for each cluster C_j , $0 < \sum_{i=1}^{n} w_{ij} < n$ ensures there is no empty cluster
- Let c_1, …, c_k as the center of the k clusters
- For an object o_i, sum of the squared error (SSE), p is a parameter:
- For a cluster C_j, SSE: $\mathrm{SSE}(C_j) = \sum_{i=1}^{n} w_{ij}^p dist(o_i, c_j)^2$ $\mathrm{SSE}(o_i) = \sum_{j=1}^{k} w_{ij}^p dist(o_i, c_j)^2$
- Measure how well a clustering fits the data: $\mathrm{SSE}(\mathcal{C}) = \sum_{i=1}^{n} \sum_{j=1}^{k} w_{ij}^p dist(o_i, c_j)^2$

Probabilistic Model-Based Clustering

- Cluster analysis is to find hidden categories.
- A hidden category (i.e., *probabilistic cluster)* is a distribution over the data space, which can be mathematically represented using a probability density function (or distribution function).
- Ex. 2 categories for digital cameras sold
  - consumer line vs. professional line
  - density functions f_1, f_2 for C_1, C_2
  - obtained by probabilistic clustering



- A **mixture model** assumes that a set of observed objects is a mixture of instances from multiple probabilistic clusters, and conceptually each observed object is generated independently
- **Our task**: infer a set of *k* probabilistic clusters that is most likely to generate *D* using the above data generation process

Probabilistic Model-Based Clustering

- A set C of k probabilistic clusters $C_1, \ldots, C_k$ with probability density functions $f_1, \ldots, f_k$, respectively, and their probabilities $\omega_1, \ldots, \omega_k$.
- Probability of an object o generated by cluster C_j is $P(o|C_j) = \omega_j f_j(o)$
- Probability of o generated by the set of cluster $\boldsymbol{C}$ is $P(o|\boldsymbol{C}) = \sum_{j=1}^{k} \omega_j f_j(o)$
- Since objects are assumed to be generated independently, for a data set D = {o_1, …, o_n}, we have,

$$P(D|\boldsymbol{C}) = \prod_{i=1}^{n} P(o_i|\boldsymbol{C}) = \prod_{i=1}^{n} \sum_{j=1}^{k} \omega_j f_j(o_i)$$

- Task: Find a set C of k probabilistic clusters s.t. $P(D|\boldsymbol{C})$ is maximized
- However, maximizing $P(D|\boldsymbol{C})$ is often intractable since the probability density function of a cluster can take an arbitrarily complicated form
- To make it computationally feasible (as a compromise), assume the probability density functions being some parameterized distributions

Univariate Gaussian Mixture Model

- O = {o_1, …, o_n} (n observed objects), Θ = {θ_1, …, θ_k} (parameters of the k distributions), and $P_j(o_i|\theta_j)$ is the probability that o_i is generated from the j-th distribution using parameter θ_j, we have

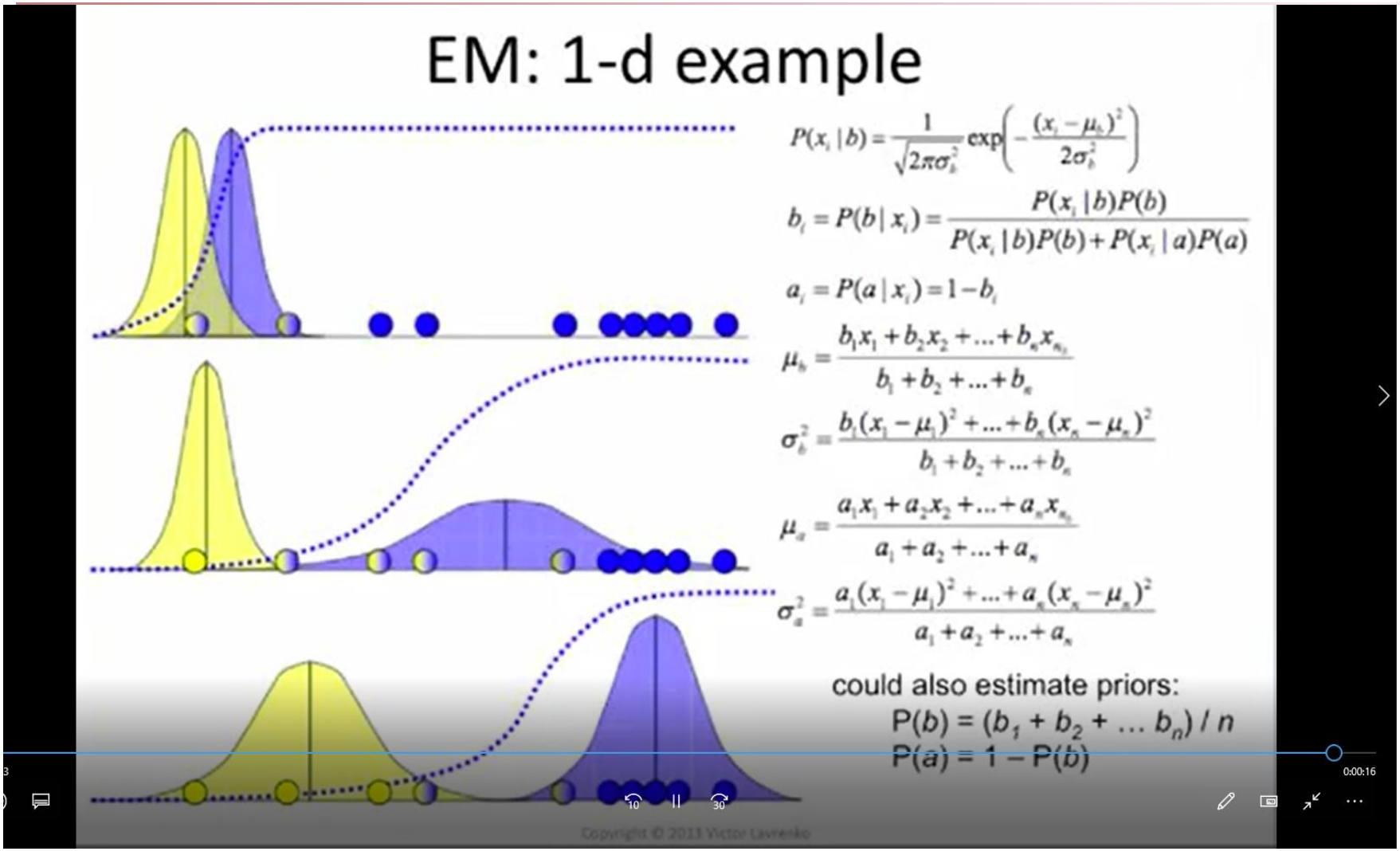
$$P(o_i|\Theta) = \sum_{j=1}^{k} \omega_j P_j(o_i|\Theta_j) \qquad P(\mathbf{O}|\Theta) = \prod_{i=1}^{n} \sum_{j=1}^{k} \omega_j P_j(o_i|\Theta_j)$$

- Univariate Gaussian mixture model
  - Assume the probability density function of each cluster follows a 1-d Gaussian distribution. Suppose that there are k clusters.
  - The probability density function of each cluster are centered at μ_j with standard deviation σ_j, θ_j, = (μ_j, σ_j), we have

$$P(o_i|\Theta_j) = \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(o_i-\mu_j)^2}{2\sigma^2}} \qquad P(o_i|\Theta) = \sum_{j=1}^{k} \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(o_i-\mu_j)^2}{2\sigma^2}}$$

$$P(\mathbf{O}|\Theta) = \prod_{i=1}^{n} \sum_{j=1}^{k} \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(o_i-\mu_j)^2}{2\sigma^2}}$$

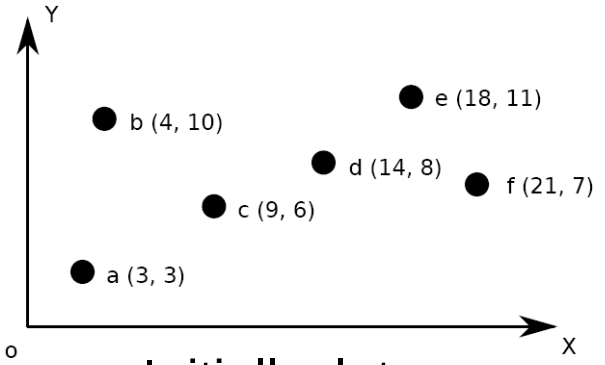
Univariate Gaussian Mixture Model



The EM (Expectation Maximization) Algorithm

- The k-means algorithm has two steps at each iteration:
  - **Expectation Step** (E-step): Given the current cluster centers, each object is assigned to the cluster whose center is closest to the object: An object is *expected to belong to the closest cluster*
  - **Maximization Step** (M-step): Given the cluster assignment, for each cluster, the algorithm *adjusts the center* so that *the sum of distance* from the objects assigned to this cluster and the new center is minimized
- **The (EM) algorithm:** A framework to approach maximum likelihood or maximum a posteriori estimates of parameters in statistical models.
  - **E-step** assigns objects to clusters according to the current fuzzy clustering or parameters of probabilistic clusters
  - **M-step** finds the new clustering or parameters that maximize the sum of squared error (SSE) or the expected likelihood

Fuzzy Clustering Using the EM Algorithm



Iteration	E-step	M-step
1	$M^T = \begin{bmatrix} 1 & 0 & 0.48 & 0.42 & 0.41 & 0.47 \\ 0 & 1 & 0.52 & 0.58 & 0.59 & 0.53 \end{bmatrix}$	$c_1 = (8.47, 5.12)$, $c_2 = (10.42, 8.99)$
2	$M^T = \begin{bmatrix} 0.73 & 0.49 & 0.91 & 0.26 & 0.33 & 0.42 \\ 0.27 & 0.51 & 0.09 & 0.74 & 0.67 & 0.58 \end{bmatrix}$	$c_1 = (8.51, 6.11)$, $c_2 = (14.42, 8.69)$
3	$M^T = \begin{bmatrix} 0.80 & 0.76 & 0.99 & 0.02 & 0.14 & 0.23 \\ 0.20 & 0.24 & 0.01 & 0.98 & 0.86 & 0.77 \end{bmatrix}$	$c_1 = (6.40, 6.24)$, $c_2 = (16.55, 8.64)$

- Initially, let c_1 = a and c_2 = b
- 1st E-step: assign o to c_1,w. wt = $\frac{\frac{1}{dist(o,c_1)^2}}{\frac{1}{dist(o,c_1)^2} + \frac{1}{dist(o,c_2)^2}} = \frac{dist(o,c_2)^2}{dist(o,c_1)^2 + dist(o,c_2)^2}$
  - $w_{c,c_1} = \frac{41}{45+41} = 0.48$
- 1st M-step: recalculate the centroids according to the partition matrix, minimizing the sum of squared error (SSE)

$$c_j = \frac{\sum_{\text{each point } o} w_{o,c_j}^2 o}{\sum_{\text{each point } o} w_{o,c_j}^2} \qquad c_1 = \left(\frac{1^2\times3+0^2\times4+0.48^2\times9+0.42^2\times14+0.41^2\times18+0.47^2\times21}{1^2+0^2+0.48^2+0.42^2+0.41^2+0.47^2}, \frac{1^2\times3+0^2\times10+0.48^2\times6+0.42^2\times8+0.41^2\times11+0.47^2\times7}{1^2+0^2+0.48^2+0.42^2+0.41^2+0.47^2}\right) = (8.47, 5.12)$$

- Iteratively calculate this until the cluster centers converge or the change is small enough

Computing Mixture Models with EM

- Given n objects O = {o_1, …, o_n}, we want to mine a set of parameters Θ = {θ_1, …, θ_k} s.t.,P(**O**|**Θ**) is maximized, where $\theta_j = (\mu_j, \sigma_j)$ are the mean and standard deviation of the j-th univariate Gaussian distribution
- We initially assign random values to parameters θ_j, then iteratively conduct the E- and M- steps until converge or sufficiently small change
- At the E-step, for each object o_i, calculate the probability that o_i belongs to each distribution,

$$P(\Theta_j|o_i, \mathbf{\Theta}) = \frac{P(o_i|\Theta_j)}{\sum_{l=1}^{k} P(o_i|\Theta_l)}$$

- At the M-step, adjust the parameters $\theta_j = (\mu_j, \sigma_j)$ so that the expected likelihood P(**O**|**Θ**) is maximized

$$\mu_j = \sum_{i=1}^{n} o_i \frac{P(\Theta_j|o_i, \mathbf{\Theta})}{\sum_{l=1}^{n} P(\Theta_j|o_l, \mathbf{\Theta})} = \frac{\sum_{i=1}^{n} o_i P(\Theta_j|o_i, \mathbf{\Theta})}{\sum_{i=1}^{n} P(\Theta_j|o_i, \mathbf{\Theta})} \quad \sigma_j = \sqrt{\frac{\sum_{i=1}^{n} P(\Theta_j|o_i, \mathbf{\Theta})(o_i - u_j)^2}{\sum_{i=1}^{n} P(\Theta_j|o_i, \mathbf{\Theta})}}$$

Advantages and Disadvantages of Mixture Models

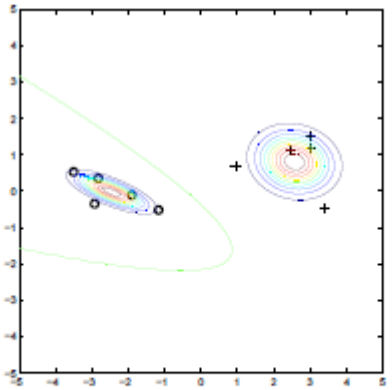
- Strength
  - Mixture models are more general than partitioning and fuzzy clustering
  - Clusters can be characterized by a small number of parameters
  - The results may satisfy the statistical assumptions of the generative models
- Weakness
  - Converge to local optimal (overcome: run multi-times w. random initialization)
  - Computationally expensive if the number of distributions is large, or the data set contains very few observed data points
  - Need large data sets
  - Hard to estimate the number of clusters

EM with generative mixture model

The MLE of θ without and with X_u is different.

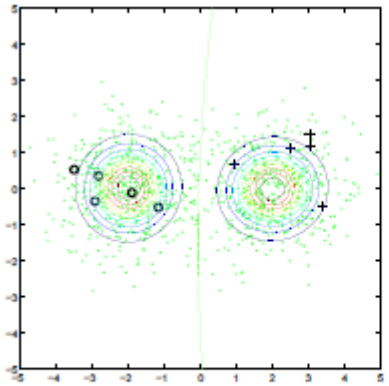
labeled data only

$$\log p(X_l, Y_l|\theta) = \sum_{i=1}^{l} \log p(y_i|\theta)p(x_i|y_i,\theta)$$



labeled and unlabeled

$$\log p(X_l, Y_l, X_u|\theta) = \sum_{i=1}^{l} \log p(y_i|\theta)p(x_i|y_i,\theta) + \sum_{i=l+1}^{n} \log \left(\sum_{y=1}^{c} p(y|\theta)p(x_i|y,\theta) \right)$$



In principle X_u is useful for other generative models too.

Generative model for semi-supervised learning

Assumption

knowledge of the model form $p(X, Y|\theta)$.

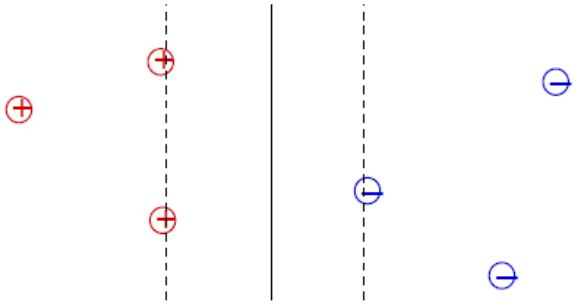
- joint and marginal likelihood

$$p(X_l, Y_l, X_u|\theta) = \sum_{Y_u} p(X_l, Y_l, X_u, Y_u|\theta)$$

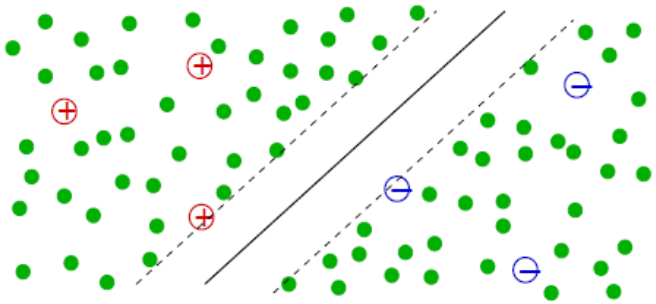
- find the maximum likelihood estimate (MLE) of θ, the maximum a posteriori (MAP) estimate, or be Bayesian
- common mixture models used in semi-supervised learning:
  - Mixture of Gaussian distributions (GMM) – image classification
  - Mixture of multinomial distributions (Naïve Bayes) – text categorization
  - Hidden Markov Models (HMM) – speech recognition
- Learning via the Expectation-Maximization (EM) algorithm (Baum-Welch)

Semi-supervised Support Vector Machines

SVMs



Semi-supervised SVMs (S3VMs) = Transductive SVMs (TSVMs)



Assumption: Unlabeled data from different classes are separated with large margin.

Semi-supervised Support Vector Machines

S3VMs

Assumption

Unlabeled data from different classes are separated with large margin.

S3VM idea:

- Enumerate all 2^u possible labeling of X_u
- Build one standard SVM for each labeling (and X_l)
- Pick the SVM with the largest margin

Advantages and Disadvantages of S3VMs

- Advantages
  - Applicable wherever SVMs are applicable.
  - Clear mathematical framework.
- Disadvantages
  - Optimization is difficult.
  - Can be trapped in bad local optima.
  - More modest assumption than generative model or graph-based methods, potentially lesser gain.

Entropy Regularization

- Assumption: if the two classes are well-separated, then $p(y|x)$ on any unlabeled instance should be close to 0 or 1.
- Entropy $H(p) = -p \log p - (1-p) \log(1-p)$ should be small
- entropy regularizer $\Omega(f) = \sum_{j=l+1}^{l+u} H(p(y=1|\mathbf{x}_j, \mathbf{w}, b))$
- semi-supervised logistic regression

$$\min_{\mathbf{w},b} \quad \sum_{i=1}^{l} \log\left(1+\exp(-y_i f(\mathbf{x}_i))\right) + \lambda_1 \|\mathbf{w}\|^2$$
$$+\lambda_2 \sum_{j=l+1}^{l+u} H(1/\left(1+\exp(-f(\mathbf{x}_j))\right))$$

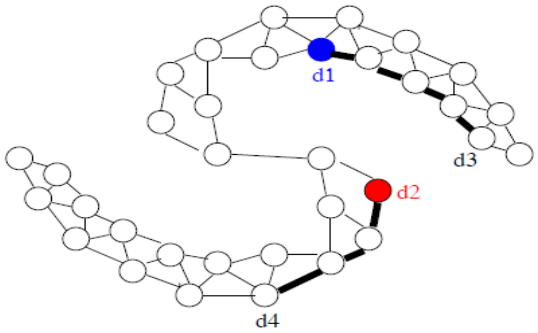
- The probabilistic counter part of S3VMs.

Graph-based semi-supervised learning

- Assumption
  - A graph is given on the labeled and unlabeled data. Instances connected by heavy edge tend to have the same label.

The graph

- Nodes: $X_l \cup X_u$
- Edges: similarity weights computed from features, e.g.,
  - k-nearest-neighbor graph, unweighted (0, 1 weights)
  - fully connected graph, weight decays with distance $w = \exp\left(-\|x_i - x_j\|^2/\sigma^2\right)$
- Want: implied similarity via all paths



Graph-based semi-supervised learning

- Some graph-based algorithms
  - Mincut
  - Harmonic
  - Local and global consistency
  - Manifold regularization

Graph-based semi-supervised learning

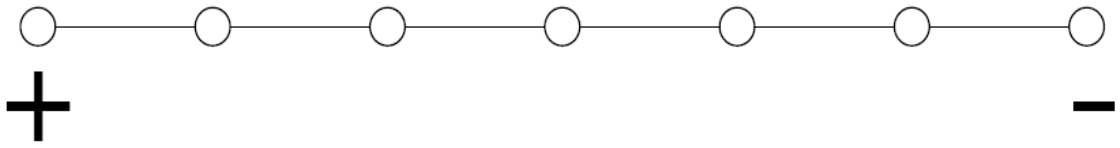
- The mincut algorithm

The graph mincut problem:

- Fix Y_l, find $Y_u \in \{0,1\}^{n-l}$ to minimize $\sum_{ij} w_{ij}|y_i - y_j|$.
- Equivalently, solves the optimization problem

$$\min_{Y\in\{0,1\}^n} \infty \sum_{i=1}^{l}(y_i - Y_{li})^2 + \sum_{ij} w_{ij}(y_i - y_j)^2$$

- Combinatorial problem, but has polynomial time solution.
- Mincut computes the modes of a Boltzmann machine
- There might be multiple modes
- One solution is to randomly perturb the weights, and average the results.



The harmonic function

Relaxing discrete labels to continuous values in $\mathbb{R}$, the harmonic function f satisfies

- $f(x_i) = y_i$ for $i = 1 \ldots l$
- f minimizes the energy

$$\sum_{i \sim j} w_{ij} (f(x_i) - f(x_j))^2$$

- the mean of a Gaussian random field
- average of neighbors $f(x_i) = \frac{\sum_{j \sim i} w_{ij} f(x_j)}{\sum_{j \sim i} w_{ij}}, \forall x_i \in X_u$

An algorithm to compute harmonic function

One way to compute the harmonic function is:

1. Initially, set $f(x_i) = y_i$ for $i = 1 \ldots l$, and $f(x_j)$ arbitrarily (e.g., 0) for $x_j \in X_u$.
2. Repeat until convergence: Set $f(x_i) = \frac{\sum_{j \sim i} w_{ij} f(x_j)}{\sum_{j \sim i} w_{ij}}, \forall x_i \in X_u$, i.e., the average of neighbors. Note $f(X_l)$ is fixed.

This can be viewed as a special case of self-training too.

Problems with harmonic function

Harmonic solution has two issues

- It fixes the given labels Y_l
  - What if some labels are wrong?
  - Want to be flexible and disagree with given labels occasionally
- It cannot handle new test points directly
  - f is only defined on X_u
  - We have to add new test points to the graph, and find a new harmonic solution

Local and global consistency

- Allow $f(X_l)$ to be different from Y_l, but penalize it
- Introduce a balance between labeled data fit and graph energy

$$\min_f \sum_{i=1}^{l} (f(x_i) - y_i)^2 + \lambda f^\top \Delta f$$

Manifold regularization

Manifold regularization solves the two issues

- Allows but penalizes $f(X_l) \neq Y_i$ using hinge loss
- Automatically applies to new test data
  - Defines function in kernel K induced RKHS: $f(x) = h(x) + b, h(x) \in \mathcal{H}_K$
- Still prefers low energy $f_{1:n}^\top \Delta f_{1:n}$

$$\min_f \sum_{i=1}^{l} (1 - y_i f(x_i))_+ + \lambda_1 \|h\|_{\mathcal{H}_K}^2 + \lambda_2 f_{1:n}^\top \Delta f_{1:n}$$

Manifold regularization algorithm

1. Input: kernel K, weights λ_1, λ_2, (X_l, Y_l), X_u
2. Construct similarity graph W from X_l, Xu, compute graph Laplacian Δ
3. Solve the optimization problem for $f(x) = h(x) + b, h(x) \in \mathcal{H}_K$

$$\min_f \sum_{i=1}^{l} (1 - y_i f(x_i))_+ + \lambda_1 \|h\|^2_{\mathcal{H}_K} + \lambda_2 f_{1:n}^\top \Delta f_{1:n}$$

4. Classify a new test point x by sign$(f(x))$

Pros and Cons of Graph-based SSL

- Pros
  - Clear mathematical framework.
  - Performance is strong if the graph happens to fit the task
  - The (pseudo) inverse of the Laplacian can be viewed as a kernel matrix
  - Can be extended to directed graphs
- Cons
  - Performance is bad if the graph is bad
  - Sensitive to graph structure and edge weights

Which semi-supervised learning method should I use?

Ideally, one should use a method whose assumptions fit the problem structure.

- Do the classes produce well clustered data?
  If yes, EM with generative mixture models may be a good choice.
- Do the features naturally split into two sets?
  If yes, co-training may be appropriate.
- Is it true that two points with similar features tend to be in the same class?
  If yes, graph-based methods can be used.
- Already using SVM?
  Transductive SVM is a natural extension.
- Is the existing supervised classifier complicated and hard to modify?
  Self-training is a practical wrapper method.

Future Direction

- Real SSL tasks
  - What tasks can be dramatically improved by SSL, so that new functionalities are enabled?
- New SSL assumptions

Generative models, multiview, graph methods, S3VMs

$$\sum_{i=1}^{l} \log p(y_i|\theta)p(x_i|y_i,\theta) + \lambda \sum_{i=l+1}^{n} \log \left(\sum_{y=1}^{c} p(y|\theta)p(x_i|y,\theta) \right)$$

$$\min_f \sum_{v=1}^{M} \left(\sum_{i=1}^{l} c(y_i, f_v(x_i)) + \lambda_1 \|f\|_K^2 \right) + \lambda_2 \sum_{u,v=1}^{M} \sum_{i=l+1}^{n} (f_u(x_i) - f_v(x_i))^2$$

$$\min_f \sum_{i=1}^{l} c(y_i, f(x_i)) + \lambda_1 \|f\|_K^2 + \lambda_2 \sum_{i,j=1}^{n} w_{ij}(f(x_i) - f(x_j))^2$$

$$\min_f \sum_{i=1}^{l} (1 - y_i f(x_i))_+ + \lambda_1 \|f\|_K^2 + \lambda_2 \sum_{i=l+1}^{n} (1 - |f(x_i)|)_+$$

Future Direction

What other assumptions can we make on unlabeled data? For example:

- label dissimilarity $y_i \neq y_j$

$$\sum_{i,j} w_{ij}(f(x_i) - s_{ij}f(x_j))^2$$

  w_{ij} edge confidence; $s_{ij} = 1$: same label, -1: different labels

- order preference $y_i - y_j \geq d$ for regression

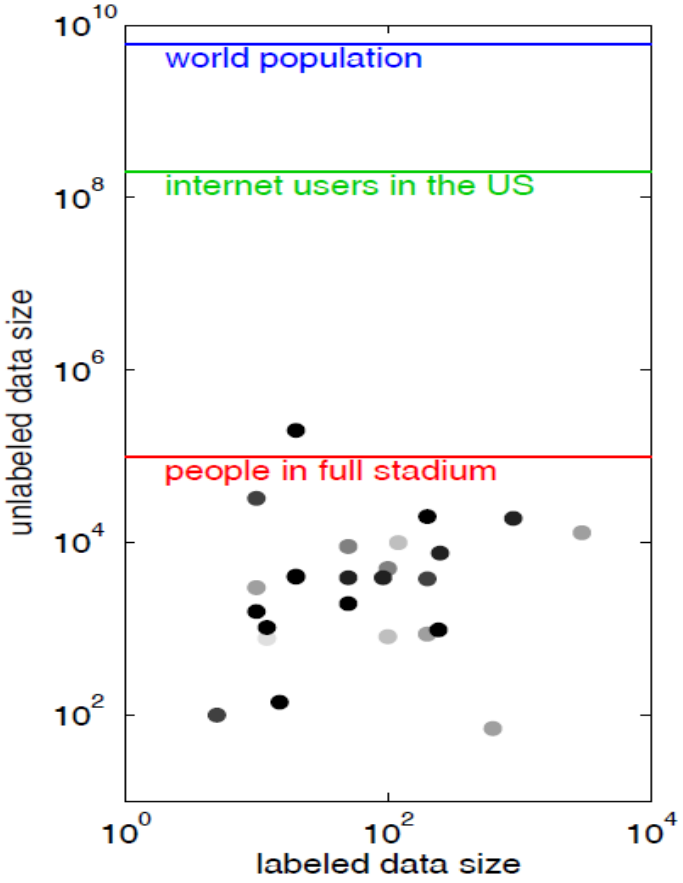
$$(d - (f(x_i) - f(x_j))_+$$

New assumptions may lead to new SSL algorithms.

Future Direction

- Efficiency on huge unlabeled datasets

  Some recent SSL datasets as reported in research papers:



Future Direction

Safe SSL

- How do we know that we are making the right model assumptions?
- Which semi-supervised learning method should I use?
- If I have labeled AND unlabeled data, I should do at least as well as only having the labeled data.

How can we make sure that SSL is "safe"?

References

1. Olivier Chapelle, Alexander Zien, Bernhard Schölkopf (Eds.). (2006). *Semi-supervised learning*. MIT Press.
2. Xiaojin Zhu (2005). *Semi-supervised learning literature survey*. TR-1530. University of Wisconsin-Madison Department of Computer Science.
3. Matthias Seeger (2001). *Learning with labeled and unlabeled data*. Technical Report. University of Edinburgh.

... and the references therein.

Thank you