Semi-Supervised Learning: Lecture Notes

William W. Cohen

March 30, 2018

1 What is Semi-Supervised Learning?

In *supervised learning*, a learner is given a dataset of m labeled examples

$$\{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^m, y^m)\}$$

and constructs a function f which predicts a label y for an unseen test example $\mathbf{x}$, i.e., $\hat{y} = f(\mathbf{x})$. In *semi-supervised learning* (SSL) the learner has an additional input: a set of n unlabeled examples $\{\mathbf{x}^{m+1}, \ldots, \mathbf{x}^{m+n}\}$. Figure 1 suggests how this can be helpful.

There are several different paradigms for SSL [6]. Here I will focus on *graph-based SSL* [2]. Here the examples (both labeled and unlabeled) are organized in a graph. I will assume W is a matrix such that $W[i, j]$ is the weight of the edge between $\mathbf{x}_i$ and $\mathbf{x}_j$, and weights are always positive. This graph can be constructed in many ways, but usually it is sparse (i.e., most edges have weight zero). Here are some common cases.

- *Similarity edges:* All nodes in the graph are examples, and $W[i, j]$ measures similarity between examples i and j. Figure 2(A) shows an example of this, where the examples are images of digits, taken from [5]. Note that some of the nodes in the graph have labels, and some do not.

- *Network edges:* All nodes in the graph are examples, $W[i, j]$ indicates that some relationship holds between the examples. For instance, the examples are web sites, and $W[i, j] = 1$ if there is a hyperlink from from site i to site j.

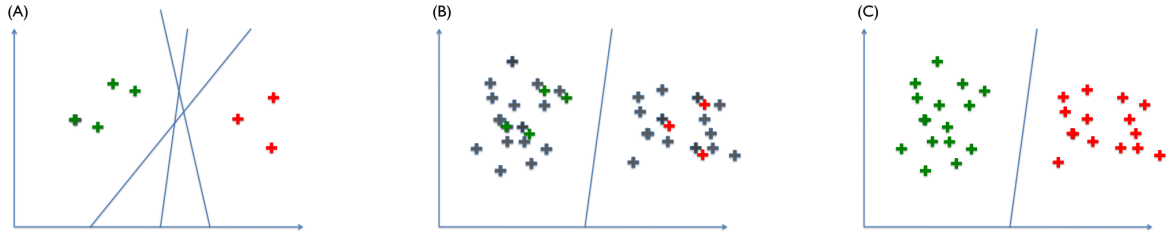

Figure 1: Cartoon suggesting why unlabeled examples might be helpful in learning. (A) If there are few labeled examples, learning a classifier is unconstrained—many different classifiers separate the data equally well. (B) If unlabeled examples are added, then the choice of classifier may become more obvious; for instance, if we assume that the labeled examples are drawn from two well-separated clusters, learning becomes nearly as constrained as in supervised learning with many examples (C).

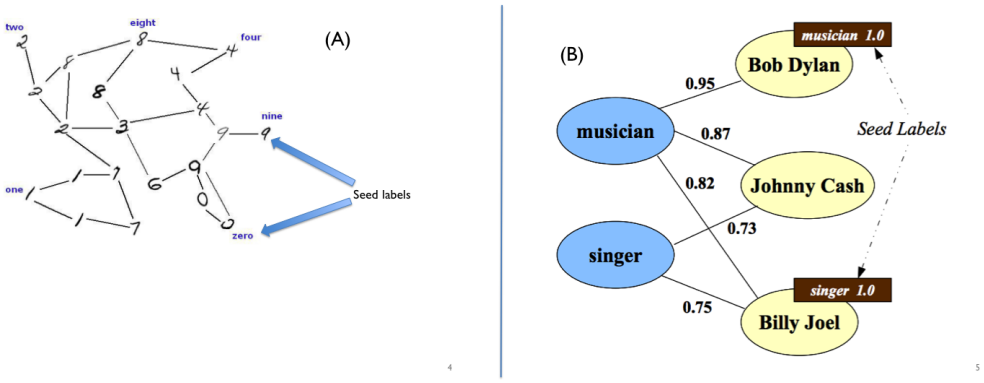

Figure 2: Graphs based on similarity edges (A) and coupling edges (B).

- *Coupling-node edges:* The graph is a bipartite graph, and some nodes are examples, and the other nodes correspond to sets of example. As an example, in [3] text-mining was performed on a large corpus, looking for phrases like "songwriters such as Bob Dylan" or "singers such as Billy Joel". This was used to create a graph with nodes of two types: categories ("songwriters", "singers", ...) and examples ("Bob Dylan", "Billy Joel", ...). Edges connect a category nodes to member of that category, with weights derived from frequency in the corpus. Figure 2(B) shows an example of this from [3].

Note that only some of the nodes in the graph are labeled. The initially-labeled nodes are sometimes called "seeds". Graph-based SSL is often called *label propagation* (LP), with the intuition that labels "propagate" from the seeds through the graph along the edges, thus associating labels with initially-unlabeled nodes. Label propagation is generally a "soft" iterative process, where in each iteration, some real-number scores, each representing confidence in a particular label at a particular node, are propagated through the graph.

2 MultiRank Walk: A Sample LP Algorithm

Let us first look at this problem intuitively, and ask: when does it make sense that a label from $\mathbf{x}_i$ should be "propagated" (transferred) to a connected node $\mathbf{x}_j$? More interestingly, when should labels be propagated over a path consisting of multiple edges?

Figure 3 shows three subgraphs, each of which contains two seeds with red and green labels, and a starred node which is connected to both red and green seeds. However, in each case, we argue it is more plausible to associate the starred node with the green label.

- In subgraph (A), the path to the green seed is shorter than the path to the red seed.

- In subgraph (B), every path to a seed is of length two, but there are more paths to the greed seed than to the red seed. Put another way, define $\mathcal{N}(i)$ to be the set of nodes that are neighbors of i, and suppose that shared neighbors of two graph nodes indicate similarity, such as shared words of two documents indicate similarity. The starred node

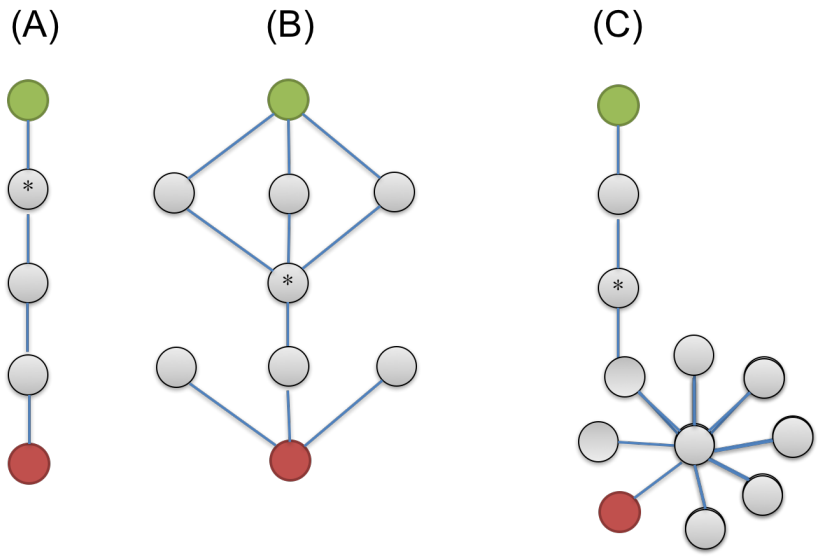

Figure 3: Some sample graphs to motivate a particular graph propagation scheme

has three common neighbors with the green seed, and only one common neighbor with the red one.

- In subgraph (C), there is exactly one length-two path to each of the seeds, and the starred node has exactly one common neighbor with each seed. But there is an important difference: the common neighbor with the red node is a common neighbor for many pairs of nodes. Again treating common neighbors as analogous to words, this would be like two documents sharing a common word (like "the") rather than a rare one (like "aardvark"). This suggests that the starred node should ultimately receive the green label, not the red label, after propagation.

We now ask: is there any simple algorithmic definition of graph similarity that is consistent with these intuitions? It turns out there is: they are all consistent with random-walk-with-reset (RWR) distance, also known as personalized PageRank.

Let's start by defining the following random process, parameterized by a set of seed nodes S and a reset probability c:

1. Draw node V_0 uniformly at random from S (notice V_0 is a random variable here)

2. For $t = 0, \ldots, T$:

   (a) with probability c, draw node V_{t+1} uniformly at random from S;

   (b) otherwise, draw[1] node V_{t+1} from the neighbors of V_t, weighted by the weight of the edge from V_t to V_{t+1}.

This is the same as the random surfer model used in PageRank, except that the "teleport" steps always jump to a seed. Ignore for a minute the parameter c, and define

$$RWR_S(j) \equiv \lim_{T \to \infty} \Pr(V_T = j)$$

Finally the vector $\mathbf{RWR}_S$ is the vector of these values over all nodes, i.e.

$$\mathbf{RWR}_S = \langle RWR_S(1), RWR_S(2), \ldots, RWR_S(m+n) \rangle$$

where the nodes of the graph are named with integers $1, \ldots, m+n$.

One way to approximate $\mathbf{RWR}_S$ is let $\mathbf{s}$ be a row vector which is uniform over the seeds S, and let W' be a column-normalized version of W: i.e.,

$$W'[i,j] = \frac{W[i,j]}{\sum_k W[i,k]}$$

Let $\mathbf{v}^0 = \mathbf{s}$, and then repeatedly do this update:

$$\mathbf{v}^{t+1} = c\mathbf{s} + (1-c)\mathbf{v}^t W'$$

If $\mathbf{s}$ was uniform over the whole graph (not just the nodes in S) then this would be exactly the update used to compute ordinary PageRank. In practice this converges quickly: the effect of long paths on RWR_S is small because the probability of not doing a reset in n steps is $(1-c)^n$, which shrinks exponentially. As discussed in class, this update operation can also be done using map-reduce for very large graphs.

Now let's look go back to Figure 3 and consider two seeds sets, G and R (for green and red) which contain just one node each. Let i^* be the index of the starred node, and also suppose that each edge is bidirectional, so our random walks don't hit any "dead ends", and that the edge weights are all equal. We can see that propagating labels using RWR satisfies the intuitions discussed above:

[1]Hence we are assuming here that every node has some outlinks. Sometimes this is enforced by having every node link to itself.

- In subgraph (A), $RWR_G[i^*] > RWR_R[i^*]$ because RWR is dominated by the number of short random walks, and the path from i^* to the green seed is shorter than the path to the red seed.
- In subgraph (B), $RWR_G[i^*] > RWR_R[i^*]$ because RWR is dominated by the short random walks, and there are *more* short paths to the green seed.
- In subgraph (C), $RWR_G[i^*] > RWR_R[i^*]$ because a random walk from the red seed needs to go through a high-fanout "hub" node, and the probability of moving toward the starred node from this hub is low.

This suggests that we can use $RWR_G[i]$ (respectively $RWR_R[i]$) a measure of how much the the green (respectively red) label should be propagated to i. So we have now motivated the following simple LP method.

1. For classes $y_1, \ldots, y_K$:
   (a) Let S_k be the seeds for class y_k
   (b) Compute $\mathbf{RWR}_{S_k}$
2. For each unlabeled graph node j:
   - Predict the class y_k where $k = argmax_{k'}\ \mathbf{RWR}_{S'_k}[j]$

This approach is called the *MultiRank Walk* (MRW) algorithm, and it works reasonably well in a wide range of settings [1].

Note that in MRW, nodes don't need to have any features (although features of the examples $\mathbf{x}_i$ might have been used in creating the graph initially—e.g., in the graph might include similarity edges computed where similarity was computed using features.) This is also true for the other LP methods we'll discuss here.

3 Graph-based SSL as optimization

For MRW, we took an existing propagation method (based on PageRank) and argued that it was an intuitively reasonable way to propagate labels. Some LP schemes also arise as solutions to certain simple ptimization criterion.

Remember in our notation, the first m examples are labeled and examples $m+1 \ldots m+n$ are unlabeled. For a binary classification task, let us use

$y = +1$ for a positive examples and $y = -1$ for a negative example. Labeling the unlabeled examples (softly) simply means that we will assign a real-number "soft" prediction $\hat{y}_i$ every label. One plausible goal would be to make these predictions consistent with the labeled examples, also try to minimize degree to which connected nodes are given different labels—i.e., to minimize the following loss on the unlabeled examples:

$$Loss = \sum_{i>m,j>m} W[i,j](\hat{y}_i - \hat{y}_j)^2 \tag{1}$$

This loss expresses a sort of "smoothness" constraint on our labels: the predicted labels for nodes connected by an edge should be similar.

I will now assume that $W[i,j] = W[j,i]$—which makes sense if a weight indictes similarity of two nodes. With a little algebra, we can write the smoothness penalty L in matrix form. To simplify I'll write w_{ij} for $W[i,j]$, define $d_i = \sum_j w_{i,j}$, and let D be a diagonal matrix where $D_{i,i} = d_i$. I will also ignore the complicated indexing $i > m, j > m$ below. Then

$$\begin{aligned}
Loss &= \sum_{i,j} w_{i,j}(\hat{y}_i - \hat{y}_j)^2 \\
&= \sum_{i,j} w_{i,j}\hat{y}_i^2 + \sum_{i,j} w_{i,j}\hat{y}_j^2 - 2\sum_{i,j} w_{i,j}\hat{y}_i\hat{y}_j \\
&= \sum_i (\sum_j w_{i,j})\hat{y}_i^2 + \sum_j (\sum_i w_{i,j})\hat{y}_j^2 - 2\sum_{i,j} w_{i,j}\hat{y}_i\hat{y}_j \\
&= \sum_i d_i\hat{y}_i^2 + \sum_j d_j\hat{y}_j^2 - 2\sum_{i,j} w_{i,j}\hat{y}_i\hat{y}_j \\
&= 2\sum_i d_i\hat{y}_i^2 - 2\sum_{i,j} w_{i,j}\hat{y}_i\hat{y}_j \\
&= 2(\sum_i d_i\hat{y}_i^2 - \sum_{i,j} w_{i,j}\hat{y}_i\hat{y}_j) \\
&= 2(\hat{\mathbf{y}}^T D\hat{\mathbf{y}} - \hat{\mathbf{y}}^T W\hat{\mathbf{y}}) \\
&= 2\hat{\mathbf{y}}^T(D - W)\hat{\mathbf{y}}
\end{aligned}$$

We'll go ahead and drop the factor of 2, which doesn't make a difference since at the end we want to just minimize subject to a constraint, and let S be another diagonal matrix which tells where the seeds are: i.e., $S_{i,i} = 1$ for $i \leq m$ and $S_{i,i} = 0$ otherwise. Then the final optimization problem is

$$\text{minimize } \hat{\mathbf{y}}^T(D - W)\hat{\mathbf{y}} \text{ subject to the constraint } S\hat{\mathbf{y}} = S\mathbf{y} \tag{2}$$

(A side note: $\hat{\mathbf{y}}^T(D-W)\hat{\mathbf{y}}$ is often written as $\hat{\mathbf{y}}^T L\hat{\mathbf{y}}$ where the matrix $L=(D-W)$ is called the "graph Laplacian". I'm going to avoid this notation for now so we don't confuse this L with a loss function.)

There is a simple iterative algorithm to solve Eq 2.

1. Let $\hat{\mathbf{y}}^0$ be any label assignment consistent with the seed labels.
2. For $t=0,\ldots,T$:
   (a) For every unlabeled node $i>m$, let $\hat{y}_i^{t+1}=\frac{1}{d_i}\sum_j w_{i,j}\hat{y}_j^t$
   (b) For every labeled node $i\leq m$, let $\hat{y}_i^{t+1}=y_i$ (where y_i is the seed label for example i).

This algorithm is very simple and natural: in each iteration, you simply replace each node's value with the weighted average of its neighbor's values (while keeping the seed nodes unchanged). This method has been invented at least three times with different motivations, and has many names.

- Zhu, Gharamani and Lafferty [5] motivate it as an optimization problem (as we did above). They called the method *Gaussian harmonic fields* (GHF), because after you converge, in iteration T, it should be true that
$$\hat{y}_i^T=\frac{1}{d_i}\sum_j w_{i,j}\hat{y}_j^T$$
or in other words, each prediction $\hat{y}_i^T$ is the weighted average of the predictions for its neighbors. This is called the "harmonic property". Not everyone liked that name, and other researchers have cited [5] and called the method "harmonic fields" (HF) or "ZGL-LP".

- Around the same time, Mackassy and Provost [**?**] presented results with the same method, which they called the *relational neighbor* algorithm (RN) and later used the even catchier name *weighted-vote relational neighbor* algorithm (wvRN) [**?**]

- Earlier, Nigam and Ghani [**?**] presented a variant of co-training called co-EM which is algorithmically the same (although the presentation assumes a coupling-node graph).

4 Other optimization criteria

There are many variations of this optimization-based algorithm. For instance, Eq 2 can be extended to multilabel tasks by introducing K different label vectors $\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_K$ and solving this optimization problem:

$$\text{minimize } \sum_{k=1}^{K} \hat{\mathbf{y}}_k (D - W) \hat{\mathbf{y}}_k \text{ subject to the constraint } \forall k, S\hat{\mathbf{y}}_k = \mathbf{y}_k$$

You can relax the hard constraint that the predictions match the seeds precisely. For instance you can simply minimize the loss

$$\begin{aligned} Loss_2 &= \mu_1 \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 + \mu_2 \hat{\mathbf{y}}^T (D - W) \hat{\mathbf{y}} \\ &= \mu_1 (\hat{\mathbf{y}} - \mathbf{y})^T S (\hat{\mathbf{y}} - \mathbf{y}) + \mu_2 \hat{\mathbf{y}}^T (D - W) \hat{\mathbf{y}} \end{aligned} \tag{3}$$

Here μ is a parameter that determines how to trade off the "smoothness" part of the loss with the soft constraint on matching seeds.

Eq 2 doesn't make any distinction between high-fanout "hub" nodes and other nodes. One approach to making such a distinction is to modify the smoothness penalty of Eq 1 to

$$Loss_3 = \sum_{m+1}^{n} \sum_{i,j} w_{i,j} \left(\frac{\hat{y}_i}{f(d_i)} - \frac{\hat{y}_j}{f(d_j)} \right)^2 \tag{4}$$

where $f(d)$ is some appropriate way of scaling down the impact of high-degree nodes based on their degree: for instance, $f(d) = \sqrt{d}$ is used in the *local and global consistency* algorithm [4]. This formula downweights label disagreements that involve one or more high-degree node.

Another approach to handling "hubs" is to introduce vector of biases $\mathbf{r}_k$ that pushes each node toward a particular class, and trade closeness to the bias vectors off against the usual smoothness criterion.

$$Loss_4 = \sum_{k=1}^{K} \mu_1 \hat{\mathbf{y}}_k^T (D - W) \hat{\mathbf{y}}_k + \mu_2 (\hat{\mathbf{y}}_k - \mathbf{r}_k)$$

These bias vectors can be used to push the labels of certain unlabeled nodes towards a special "unknown" or "dummy" label. If one uses $\mathbf{r}$ to push high-degree nodes toward a dummy label, then an effect similar Eq 4 is obtained.

In [3] this loss function is combined with the agreement constraint of Eq 3 to get the following loss

$$Loss_4 = \mu_1(\hat{\mathbf{y}} - \mathbf{y})^T S(\hat{\mathbf{y}} - \mathbf{y}) + \mu_2 \hat{\mathbf{y}}^T (D - W)\hat{\mathbf{y}} + \mu_3(\hat{\mathbf{y}_k} - \mathbf{r}_k)$$

All of these optimization criterion have iterative update schemes similar to the ones shown above.

References

[1] Frank Lin and William W Cohen. Semi-supervised classification of network data using very few labels. In *Advances in Social Networks Analysis and Mining (ASONAM), 2010 International Conference on*, pages 192–199. IEEE, 2010.

[2] Amarnag Subramanya and Partha Pratim Talukdar. Graph-based semi-supervised learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(4):1–125, 2014.

[3] Partha Pratim Talukdar and Fernando Pereira. Experiments in graph-based semi-supervised learning methods for class-instance acquisition. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 1473–1481. Association for Computational Linguistics, 2010.

[4] Denny Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Advances in neural information processing systems*, pages 321–328, 2004.

[5] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919, 2003.

[6] Xiaojin Zhu and Andrew B Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009.