Learning from Labeled and Unlabeled Data using Graph Mincuts

Avrim Blum AVRIM@CS.CMU.EDU
Shuchi Chawla SHUCHI@CS.CMU.EDU
Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213 USA

Abstract

Many application domains suffer from not having enough labeled training data for learning. However, large amounts of unlabeled examples can often be gathered cheaply. As a result, there has been a great deal of work in recent years on how unlabeled data can be used to aid classification. We consider an algorithm based on finding minimum cuts in graphs, that uses pairwise relationships among the examples in order to learn from both labeled and unlabeled data. Our algorithm uses a similarity measure between data to construct a graph, and then outputs a classification corresponding to partitioning the graph in a way that minimizes (roughly) the number of similar pairs of examples that are given different labels. We give several theoretical justifications for this approach, and provide experiments on both synthetic and real datasets. This method is also seen to be robust to noise on the labeled examples.

1. Introduction

Learning algorithms often face a lack of sufficient labeled data. Whether the task is to classify text documents, web pages, or camera images, we often need a learning algorithm to do well with only a few labeled training examples. Luckily, in many cases, large numbers of unlabeled examples may be readily available. For instance, in document classification, one might have easy access to a large database of documents, only some of which have been classified by hand. As a result, there has been a good deal of work in recent years on how unlabeled data can be usefully employed in order to produce better predictions (Ratsaby & Venkatesh, 1995; Castelli & Cover, 1996; Nigam et al., 1998; Blum & Mitchell, 1998; Bennett & Demiriz, 1998; Hofmann, 1999; Zhang & Oles, 2000; Schuurmans, 1997).

Recently, a method based on graph mincuts has been proposed in the vision literature for the problem of cleaning up 3-D pixel images (Greig et al., 1989; Roy & Cox, 1998; Boykov et al., 1998; Snow et al., 2000). Given an initial noisy image created from a stereo camera, the goal is to improve the image by minimizing an appropriate "energy function." This energy function combines a term for each pair of neighboring pixels that are at different depths (encouraging the algorithm to "smooth" the image) and a term for the number of pixels changed from the original image (encouraging the algorithm not to change too many pixels). The insight of Greig et al. (1989) and Boykov et al. (1998) is that this energy function can be minimized by applying a graph mincut algorithm.[1]

In this paper, we show that this method can be applied to the machine learning problem of combining labeled and unlabeled data as well. Given a dataset of labeled and unlabeled examples, we construct a graph on the examples such that the minimum cut on this graph yields an "optimal" binary labeling of the unlabeled data according to certain optimization functions. Our approach is inspired by the work of Kleinberg and Tardos (2000) who connect the work in vision to a more general classification setting they call the "metric labeling problem". In fact, we will be converting the learning problems into a technically simpler version of their setting (a binary rather than multi-way classification) that can be solved exactly rather than just approximated. Thus, our focus will be on how to construct an appropriate graph rather than developing new algorithms for solving the graph problem as in Kleinberg and Tardos (2000).

As with most other approaches to combining labeled and unlabeled data, the high level idea of this method is to assign values to the unlabeled examples in order to optimize an associated objective function. For the mincut approach, the kinds of functions that can be optimized are limited to depend only on pairwise relationships among examples. What makes this approach especially appealing, however, is that for the functions we *can* handle, graph mincuts give a *polynomial-time* algorithm to find the true *global* optimum. Thus we trade off the generality of an approach such as EM or hill-climbing/gradient-descent (which can be applied almost anywhere) for confidence in finding the exact optimum. The natural question then is: how inter-

[1] A similar technique is used by Wu and Leahy (1993) for image partitioning. Shi and Malik (1997) give a more sophisticated approach based on *normalized* cuts.

esting are the objective functions this approach can represent? Do they make sense theoretically, and do they help experimentally?

In this paper, we provide results in both of these directions. We describe the mincut approach in detail and prove a number of theoretical guarantees. We also show experimentally that this method can indeed use unlabeled data to substantial advantage, though it is clear we have not yet found the best way to tune this approach. We will also see experimentally that the mincut approach tends to be robust to random noise. This is not surprising given its use for reduction of noise in images (Boykov et al., 1998; Snow et al., 2000).

For concreteness, here is an example of a kind of optimization that the mincut algorithm can perform.[2]

> Given a set of (positive and negative) labeled examples L, and a set of unlabeled examples U, find a labeling of the points in U that minimizes the leave-one-out cross-validation error of the nearest-neighbor algorithm, when applied to the entire dataset $L \cup U$.

Notice that this optimization problem is natural for an iterative-relabeling style algorithm or for greedy local optimization. However, by setting it up as a graph mincut problem, we can find the global optimum, and do so in polynomial time. In a sense, the graph mincut approach relates to nearest-neighbor style algorithms much like transductive SVM (Bennett & Demiriz, 1998; Hofmann, 1999) relates to the standard SVM algorithm: its goal is to assign labels to the unlabeled data in such a way as to make the underlying learning algorithm "happiest". We explore this further in Section 3.

One nice feature of having an algorithm that efficiently finds a global optimum is that we can compare it to a local optimization algorithm for the same objective function and see how the results differ. In particular, we are interested in (a) is the global optimum really better than a typical local optimum in terms of the value of the objective function, and (b) does this translate to a significant difference in terms of prediction accuracy. (I.e., does our objective function or generative model have anything to do with reality?) We perform experiments of this form as well.

One last point is that assuming the unlabeled training data comes from the same underlying distribution as the test data, there is really no difference between "unlabeled data" and "test data". Thus, the problem of how to use unlabeled data can also be viewed as the question: "given a large set of (unlabeled) test data, can properties of the entire test set be used to make better predictions than via the standard approach of fixing the learned hypothesis before any test data has been seen?" In particular, in our experiments we will put both the unlabeled training data and the test data into the same pot, then run the algorithms, and then read off the labels assigned to the test points as our predictions.

This paper is organized as follows. In section 2, we describe the Graph Mincut algorithm. In sections 3 and 4, we present some theoretical results to motivate the applicability of this algorithm. Section 5 contains experimental results, on both synthetic data and datasets from the UCI repository. Finally, in section 6, we present our conclusions.

2. The Graph Mincut Learning Algorithm

We now describe the Graph Mincut learning algorithm. First, let us introduce some notation. We are given a set L of labeled examples, and a set U of unlabeled examples. We assume we are in the setting of binary classification (labels are positive or negative) and use L_+ to denote the set of positive examples in L, and L_- to denote the set of negative examples in L. The algorithm is then as follows.

1. We construct a weighted graph $G = (V, E)$, where $V = L \cup U \cup \{v_+, v_-\}$, and $E \subseteq V \times V$. Associated with each edge $e \in E$ is a weight $w(e)$. We will call the vertices v_+ and v_- *Classification vertices*, and all other vertices *Example vertices*.

2. The classification vertices are connected by edges of infinite weight to the labeled examples having the *same* label as they do. Specifically, $w(v, v_+) = \infty$ for all $v \in L_+$ and $w(v, v_-) = \infty$ for all $v \in L_-$.

3. The edges between Example vertices are assigned weights based on some relationship between the examples, such as the similarity/distance between them. The specific choice of these edge weights will be discussed later. In the rest of the paper, the function assigning weights to edges between *Example nodes* will be referred to as the *Edge Weighting function* w.

4. Now we determine a minimum (v_+, v_-) cut for the graph; that is, we find the minimum total weight set of edges whose removal disconnects v_+ from v_-. This can be found using a max-flow algorithm in which v_+ is the source, v_- is the sink, and the edge weights are treated as capacities (see, e.g., (Cormen et al., 1990)). Removing the edges in the cut partitions the graph into two sets of vertices which we call V_+ and V_-, with $v_+ \in V_+$ and $v_- \in V_-$. For concreteness, if there are multiple minimum cuts, we can set the algorithm to choose the one such that V_+ is smallest (this is always well-defined and easy to obtain from the flow).

[2] This particular criteria may not be the best one in the world to optimize — in particular, the nearest-neighbor graph is likely to have isolated pockets — but we give better criteria in Section 3.

5. We assign a positive label to all unlabeled examples in the set V_+ and a negative label to all unlabeled examples in the set V_-.

The motivation for this algorithm is that if edges between examples which are similar to each other are given a high weight, then, two similar examples are likely to be placed in the same vertex subset obtained from the mincut. This conforms with the basic assumption of many learning algorithms (like nearest neighbor) that similar examples should be classified similarly.

This motivation in fact suggests how we should weight the edges. If we have a notion of "distance" between examples such that we expect nearby examples to generally have the same label (e.g., this might just be L_2 distance in the feature space), then a natural weighting function is to put high-weight edges between nearby examples, and low-weight edges (or no edges) between farther-away examples. If we are not initially handed such a distance function (e.g., we do not expect the straightforward one based on feature values to be very helpful) then we may first wish to feed the labeled data into some auxiliary learning algorithm that learns a distance function for us. For example, we could use the labeled data to weight the attributes based on information gain. Another freedom we have is to scale the weight of an edge (x, y) for $x \in U$ based on whether or not $y \in L$: this allows us to interpolate between putting unlabeled data on similar footing as the labeled data and ignoring unlabeled data completely. We will see later that the choice of edge weighting function can greatly influence the quality of output of the algorithm.

3. Motivation #1: minimizing LOOCV error

Why might the mincut approach be a reasonable one to try? In this section, we motivate this approach by considering the goal of assigning labels to the unlabeled data in order to maximize the "happiness" of some given learning algorithm A. We will prove two related but technically different kinds of results:

1. For certain learning algorithms A, we can define edge weights so that the mincut algorithm produces a labeling of the unlabeled data that (out of all possible such labelings) results in A having the least leave-one-out cross validation error when applied to the entire dataset $L \cup U$.
2. For certain (other) learning algorithms A, we can define edge weights so that the mincut algorithm's labeling results in A having zero leave-one-out cross validation error when only examples in U are held out.

The types of learning algorithms we will be able to handle are all of the nearest-neighbor style. We begin with a simple result of type (1) for the basic 1-nearest-neighbor algorithm.

Theorem 3.1 *Suppose we define edge weights between Example nodes in the following way: for each pair of nodes x and y, define $nn_{xy} = 1$ if y is the nearest neighbor of x, and $nn_{xy} = 0$ otherwise. Now let $w(x, y) = nn_{xy} + nn_{yx}$. Then, for any binary labeling of the examples $x \in U$, the cost of the associated cut is equal to the number of leave-one-out cross-validation mistakes made by 1-nearest neighbor on $L \cup U$.*

This theorem implies that minimizing the value of the cut corresponds to minimizing LOOCV error.

Proof: Fix some binary labeling $f(x)$ of the unlabeled examples $x \in U$. The LOOCV error of 1-nearest neighbor is simply the number of $x \in L \cup U$ such that the label of x is different from the label of x's nearest neighbor. This is the sum, over all ordered pairs $\langle x, y \rangle$ such that x and y have different labels, of nn_{xy}. But this is exactly the value of the cut produced by putting the positive examples into V_+, and putting the negative examples into V_-. ∎

It would seem natural to extend the above result to the k-nearest neighbor algorithm, but unfortunately the majority-vote operation of kNN causes a problem. What we can do instead is replace the majority-vote operation with averaging. Specifically, let's define *averaging kNN* to be the algorithm that examines the k nearest neighbors to a given test example x and predicts the fraction t/k, where t is the number of positive examples in that set (we are viewing positive examples as having label 1 and negative examples as having label 0). More generally, given a set of labeled examples S and a test example x, let's define *locally-weighted averaging* to be any algorithm that predicts the label of x to be a weighted average of the labels of the examples in S. So, we could use the k nearest examples to x as in averaging kNN, or, for instance, we could weight examples as some function of their distance from x.

Theorem 3.2 *Given a locally-weighted averaging algorithm A, we can define edge weights so that the minimum (v_+, v_-) cut yields a labeling of the unlabeled data that (out of all possible labelings of U) minimizes the L_1-norm LOOCV error of A.*

Proof: For each ordered pair of examples $\langle x, y \rangle$, define w_{xy} to be the weight given by A to example y when asked to classify example x. So, for each x, $\sum_y w_{xy} = 1$. Define the edge weight $w(x, y) = w_{xy} + w_{yx}$.

Now, fix some binary labeling $f(x)$ of the unlabeled examples $x \in U$. Let V_+ denote the set of positive examples in $L \cup U$, and let V_- denote the set of negative examples. Then, the cost of the (V_+, V_-) cut is equal to

$$\sum_{x \in V_+, y \in V_-} w(x, y) = \sum_{x \in V_+} \sum_{y \in V_-} w_{xy} + \sum_{y \in V_-} \sum_{x \in V_+} w_{yx}$$

$$= \sum_{x \in V_+} 1 - A(x) + \sum_{y \in V_-} A(y).$$

where $A(x)$ denotes the classification of x by the algorithm A.

This is the LOOCV error of A on the entire dataset in L_1 norm. Therefore, the labeling that minimizes the value of the cut also minimizes A's LOOCV error. ∎

For some algorithms A that we cannot represent exactly, we can at least achieve zero LOOCV error over just the *unlabeled* examples U. In particular, define Symmetric Weighted Nearest Neighbor to be any weighted nearest neighbor algorithm (the prediction on an example x is made by a weighted majority vote over the other examples in the dataset) but where the weights must be symmetric (the weight given to y when predicting on x is the same as the weight given to x when predicting on y). For example, the weights could be based on the distance between the examples. kNN is not symmetric because it is possible that y is a nearest neighbor of x but x is not a nearest neighbor of y.

Theorem 3.3 *Let w be the weight function used for the Symmetric Weighted Nearest Neighbor Algorithm. Then, if we use the same function w for weighting edges in Graph Mincut, the classification returned by Graph Mincut results in the algorithm having zero Leave-one-out Cross-validation error over U.*

The proof follows directly from the following lemma.

Lemma 3.4 *If f is the boolean classification returned by Graph Mincut on dataset $S = U \cup L$, where we view positive as $+1$ and negative as -1, then for all $x \in U$, $f(x) = Sgn(\Sigma_{u \in S-\{x\}} w(x,u) f(u))$, where $Sgn(z) = 1$ if $z > 0$ and $Sgn(z) = -1$ otherwise.*

Proof: The graph mincut will divide the set of examples S into a positive set V_+ and a negative set V_-. Suppose $x \in V_-$. Then, since we solved for a minimum cut, we have,

$$\sum_{u \in V_+} w(x,u) \quad \leq \quad \sum_{u \in V_-} w(x,u),$$

otherwise moving x from V_- to V_+ would strictly improve the value of the cut. This implies that

$$\sum_{u \in S-\{x\}} w(x,u) f(u) \quad \leq \quad 0$$

So we have $f(x) = Sgn(\sum_{u \in S-\{x\}} w(x,u) f(u))$ as desired. The case of $x \in V_+$ is the same, except we can now make the claim of strict inequality since V_+ is defined to be the *smallest* set such that (V_+, V_-) is a minimum cut. So, we again have $f(x) = Sgn(\sum_{u \in S-\{x\}} w(x,u) f(u))$ as desired. ∎

Proof of Thm 3.3: Immediate from Lemma 3.4. ∎

3.1 Discussion

The above results state that graph mincut will produce labelings of the unlabeled data that are in a sense self-consistent. If we think of a learning algorithm (in particular, its LOOCV error) as measuring how "nice" a dataset is, then mincut assigns labels to make the data nice for nearest-neighbor style algorithms.

This fact also points out a worry. If we have very few labeled examples and very many unlabeled examples, then this self-consistency can cause mincut to assign all the unlabeled examples to one class or the other. For instance, if we have just one labeled positive example and one labeled negative example, and we connect each example by edges of weight 1 to its three nearest neighbors, then labeling all the unlabeled points as negative gives a cut of value 3. This may well be the minimum cut unless the dataset really does separate into two distinct "blobs". This could well be a worse classifier than if the unlabeled data had been completely ignored. Shi and Malik (1997) address this problem in the context of image segmentation by using a normalized version of the mincut algorithm that attempts to equalize the sizes of the partitions (it is NP-hard to find the best 50/50 split in a graph, but their approach at least encourages some balance).

Another potential problem is that if the graph is too sparse, it could well have a number of disconnected components. For example, if we use the graph based on 1-nearest-neighbor, then two unlabeled examples very near to each other might form their own component. The mincut algorithm is then free to label that component however it likes (and according to our policy, would label it negative). So, it is important to use a weighting function that does not allow this to happen.

4. Motivation #2: a Generative Model

In the vision literature, the mincut approach and various extensions are motivated through a generative model known as a *Markov Random Field*. This model assumes the points (examples) are picked in advance, and then the labels are determined probabilistically according to a certain distribution (Pietra et al., 1997). This distribution is such that the probability of any given global labeling is the product of unary and pairwise terms: higher probability if nearby points are given the same labeling and lower probability when they have different labeling. When you take the log, you get the mincut objective function. This model makes sense in physical systems (e.g., the Ising model for how spins behave in magnetized iron) and perhaps for pixel images, but is less satisfying when considering learning from examples. In particular, it doesn't address how examples are chosen, and *why* similar examples ought to have similar labels.

Instead, we consider here a generative model that seems (to us, at least) more satisfying. In this model,

we assume that the underlying distribution over examples is a union of k regions, each of which has a unique label. E.g., there could be 3 positive regions and 3 negative regions. The regions are separated by some minimum distance δ. An example is picked by randomly choosing a point inside the union of the regions, then giving it its region's label.[3] The idea is that if we set edge weights appropriately, then weights *between* regions will be weak, whereas as we see more and more unlabeled examples, edges *within* the regions will grow stronger. Furthermore, if each region is well-shaped (e.g., not like a dumbbell) then regions will become highly connected and a single labeled example inside a region will allow us to correctly classify all the points inside it.

We now make this a bit more precise. Say the regions live in a D-dimensional space, and for normalization, assume that their total volume is 1.[4] Define the "δ-interior" of some region R to be the set of points in R whose distance to the boundary of R is at least δ. Define the "δ-tendrils" of R to be the set of points that are *not* within distance δ of the δ-interior. (These are the points that no ball of radius δ contained in R can touch.) We say that a region R is "(ϵ, δ)-round" if (a) at most an ϵ fraction of its volume is in the δ-tendrils, and (b) the δ-interior of R is connected and non-empty.

The following analysis borrows some nice proof ideas of Tenenbaum et al. (2000). Let V_r denote the volume of the D-dimensional ball of radius r.

Theorem 4.1 *Suppose that data is generated uniformly at random in the union of k $(\epsilon, \delta/4)$-round regions, such that the distance between any two regions is at least δ and the classification of a point depends solely on the region to which it belongs. Let the weighting function for graph mincut be $w(x,y) = 1$ if $d(x,y) < \delta$ and $w(x,y) = 0$ otherwise. Then $O((k/\epsilon)\log k)$ labeled examples and $O((1/V_{\delta/4})\log(1/V_{\delta/8}))$ unlabeled examples are sufficient to correctly classify a $1 - O(\epsilon)$ fraction of the unlabeled examples with high probability.*

Note: A similar but messier result can be achieved by using a weighting function such as $w(x,y) = e^{1/d(x,y)}$ that drops off sufficiently rapidly with distance.

Proof sketch: We can ignore regions of probability mass $\leq \epsilon/k$. The first property we need is that each remaining region has at least one labeled example, and this example is not in a $\delta/4$-tendril. $O((k/\epsilon)\log k)$ labeled examples are sufficient for this to occur with high probability.

[3] We could add noise in the labels at this point, but for simplicity we leave that out.

[4] We could generalize this to assuming regions are D-dimensional manifolds lying in some higher dimensional space as in (Tenenbaum et al., 2000; Roweis & Saul, 2000) without affecting the results.

Next, we need enough unlabeled examples in each region so that in the resulting graph, the $\delta/4$-interior essentially becomes a single connected component. To analyze this, given a region R, fill it as much as possible with balls of radius $\delta/4$ such that the center of any ball i is not inside any other ball j. Notice that this will fill the entire $\delta/4$-interior, since if any point p inside it is uncovered, we could just greedily add a new ball centered at p. These balls have two important features: first, if we get an unlabeled example in each of them, then we can get from any point not in the $\delta/4$-tendrils to the labeled example, using edges in the graph. Second, if we shrink the radii of these balls by a factor of 2, then they all become disjoint. The second fact means that over all the regions, the total number of balls used is at most $1/V_{\delta/8}$. Therefore, the number of unlabeled examples we need is $O((1/V_{\delta/4})\log(1/V_{\delta/8}))$. ∎

4.1 Discussion

It might seem that under the generative model in theorem 4.1, most nearest neighbor style algorithms would perform pretty well. However, this is not the case if the regions are long and thin, and labeled data is sparse. This is because in that case, examples might easily be closer to labeled examples in other regions than their own. Experimental results shown in figures 2 and 3 corroborate this fact by comparing mincut against 3-nearest neighbor on a synthetic dataset of this type. The next section contains more discussion of these figures.

5. Experimental Analysis

We tested the Graph Mincut Algorithm on standard datasets, both real and synthetic, as well as on our own synthetic dataset intended to fit the generative model of Section 4.

5.1 Standard datasets

We compared the mincut algorithm with two standard learning algorithms, ID3 and 3-nearest-neighbor, on datasets obtained from the UC Irvine Machine Learning Repository (UCI, 2000). The mincut algorithm has many degrees of freedom in terms of how the edge weights are defined. In order to make the experiments as clean as possible, we consider weighting functions specifically motivated by the analysis in the previous sections and the learning algorithms we compare against. These are as follows:

Mincut-3 For this algorithm, we connect each unlabeled example by an edge of weight 1 to its three nearest neighbors. However, to avoid having isolated components (see Section 3.1) we force one of these to be a labeled example. Specifically, each example is connected to its nearest labeled example and the two other nearest examples overall.

Mincut-δ For this algorithm, we use some metric to

Table 1. Classification accuracies of Graph Mincut and other algorithms on various datasets from the UCI repository. "MUSH" is the mushroom dataset, and MI, MII, and MIII are the (synthetic) Monks problems. Datasets with an asterix have a small amount of noise in the training set. The best result for each dataset is given in bold. In cases where the best is Mincut-δ_{opt} (which involves picking δ after the fact), the second-best is also given in bold.

DATASET	$\|L\|\&\|U\|$	NUMBER OF FEATURES	MINCUT				ID3	3-NN
			MINCUT-3	MINCUT-δ_{opt}	MINCUT-δ_0	MINCUT-$\delta_{1/2}$		
MUSH	20+1000	22	82.1	**97.7**	**97.7**	97.0	93.3	91.1
MUSH*	20+1000	22	74.2	**88.7**	56.9	**87.0**	80.8	83.3
TAE	10+100	5	86.0	**99.0**	96.0	**97.0**	86.0	80.0
TAE*	10+100	5	76.0	**96.0**	86.0	**94.0**	76.0	62.0
VOTING	45+390	16	89.1	**91.3**	66.1	83.3	86.4	**89.6**
MUSK	40+200	166	73.0	**92.5**	91.0	**92.5**	83.5	87.0
PIMA	50+718	8	63.8	**72.3**	48.8	**72.3**	70.0	68.1
IONO	50+300	34	71.0	81.6	78.0	77.6	**88.6**	69.6
BUPA	45+300	6	53.3	**59.3**	48.0	41.7	**55.3**	52.7
MI	124+432	6	70.0	64.4	64.4	64.4	**98.6**	81.1
MII	169+432	6	**68.6**	67.2	57.2	67.2	67.9	63.6
MIII*	122+432	6	79.1	80.6	64.8	80.6	**94.4**	83.6

compute distances between points. If two points are closer than δ to each other, they are connected with an edge. δ is a parameter that depends on the dataset.

- For **Mincut-δ_0**, we choose the maximum δ for which the graph has a cut of value 0.
- For **Mincut-$\delta_{1/2}$**, we use that value of δ for which the size of the largest connected component in the graph is half the number of datapoints.
- For **Mincut-δ_{opt}**, we choose the value of δ that corresponds to the least classification error in hindsight. The performance of this algorithm gives us a benchmark to measure performance of other **Mincut-δ** variants by.

In some of the datasets we used, all attributes are categorical, so we use an L_0 (Hamming) notion of distance. For other datasets, we use L_2 (Euclidean) distance. For datasets with a large number of discrete-valued attributes, we take a cue from ID3 and define a distance metric that weights the attributes based on information gain. Information gain is computed over the labeled data. Specifically, the weighting function used is:

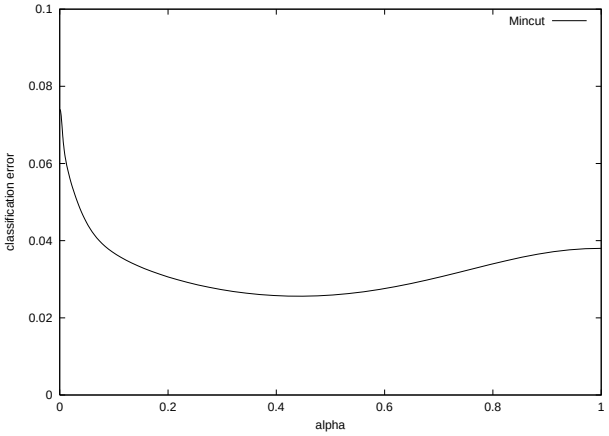
$$w(x,y) = \prod_{a \in attributes, x(a) \neq y(a)} \frac{c}{1 + e^{b \cdot gain_a}}$$

where b and c are constants. The specific choice of constants b and c does not affect algorithm performance, as it simply leads to convergence at a different value of δ.

Experimental results corresponding to the four Mincut variants described above are shown in Table 1.

As can be seen, **Mincut-δ_{opt}** outperforms other algorithms on most datasets (the main exceptions are the monks datasets). However, the best value of δ is highly problem dependent. In an attempt to have the algorithm automatically learn this value, we experimented with several techniques, two of which seemed reasonable for comparison. One of these is to examine the mincut value of the resulting graph and select δ based on this; for example, the **Mincut-δ_0** algorithm is of this form. However, **Mincut-δ_0** fails for inherently noisy datasets (compare for example the Mush and Mush* datasets and Tae and Tae* datasets). The other technique which performs better is to observe the size of the largest connected component in the resulting graph. The idea behind this technique is that when datasets are somewhat noisy, we should allow long distance dependencies in the graph to smoothen the effect of noise. This is the idea behind the **Mincut-$\delta_{1/2}$** algorithm. This technique works well for most datasets, and in particular, seems to do well in the presence of noise. As a future extension to this work, it would be interesting to explore new methods of finding a good value for δ.

Figure 1. Classification error of graph mincut on Mushroom dataset as a function of α



As observed before, graph mincut is very similar to nearest neighbor style algorithms. While nearest

neighbor bases its classification on only the labeled examples, graph mincut takes into account the unlabeled examples as well and treats them similarly. How would the performance of mincut differ if differential treatment was given to labeled and unlabeled examples? More precisely, if we scale down edge weights between two unlabeled examples by a factor of α, a value of 1 for α would result in the regular mincut algorithm, while a value of 0 would give no weight to unlabeled examples, making the algorithm resemble the underlying supervised learning algorithm. In some sense, α signifies our confidence in the unlabeled examples as compared to labeled examples. Figure 1 demonstrates the results of this experiment on the mushroom dataset for the **Mincut-δ** algorithm. Minimum classification error is achieved at $\alpha = 0.4$.

Figure 2. Classification errors of graph mincut and 3-NN on synthetic dataset as a function of number of labeled examples. Number of unlabeled examples = 5000.

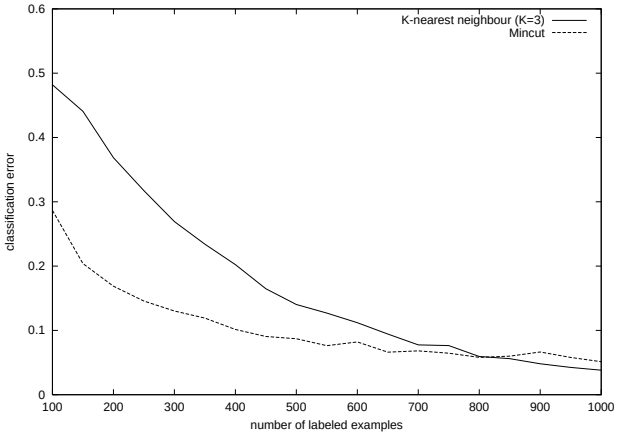
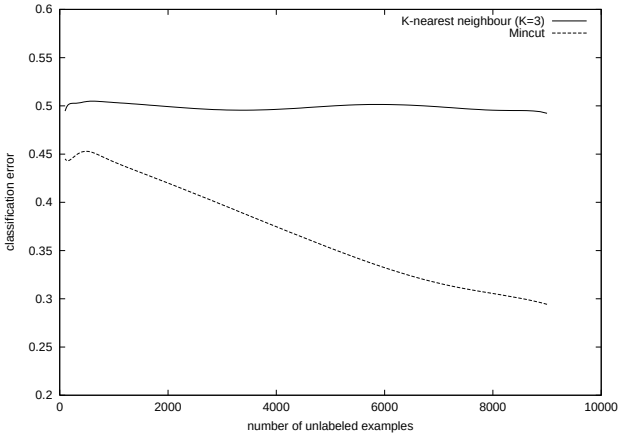


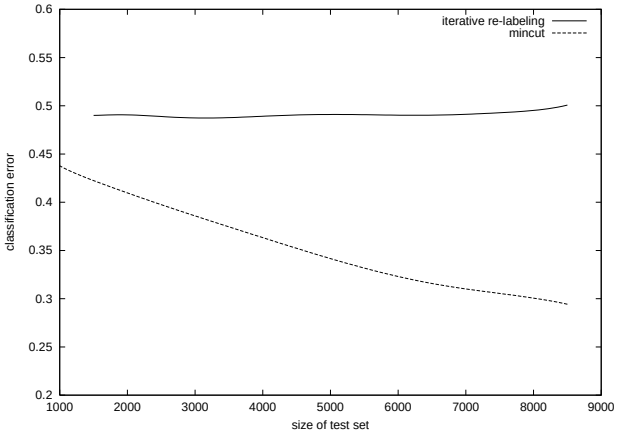
Figure 3. Classification errors of graph mincut and 3-NN on synthetic dataset as a function of number of unlabeled examples. Number of labeled examples = 50.



5.2 Synthetic datasets

In order to study various properties of the graph mincut approach more closely, we tested the algorithm on a synthetic dataset based on the k regions generative model described in Section 4. This dataset contains 8 regions in 3-dimensional space, which are separated from each other by a minimum distance δ. Specifically, each region is a 2-dimensional plane in yz space, located at x-coordinates $0, \delta, 2\delta, \ldots, 7\delta$ respectively. These regions alternate in sign, so that 4 of them are positive and 4 are negative. Data is generated uniformly at random from the union of these regions. For this dataset, we use L_2 norm for determining distance, and assign edge weights as an exponentially-decreasing function of distance. The dependence of classification error for graph mincut and 3-nearest neighbor on labeled and unlabeled examples is shown in Figures 2 & 3.[5] Figure 2 plots the performance of Mincut and 3-NN for a fixed number of unlabeled examples as the amount of labeled data varies. As can be seen in the figure, Mincut performs substantially better than 3-NN when there is very little labeled data, but this gap shrinks as the number of labeled examples increases, with both algorithms performing equally well at about 700 labeled examples. Figure 3 fixes the number of labeled examples at 50 and then increases the amount of unlabeled data. As seen in the figure, Mincut is able to use this unlabeled data to substantially improve performance. These results indicate that the real advantage of using graph mincut is achieved when there is a huge amount of unlabeled data, but a paucity of labeled data.

Figure 4. Classification error of graph mincut and iterative re-labeling on synthetic dataset as a function of number of unlabeled examples.



It is also instructive to compare the performance of graph mincut with an iterative re-labeling style algo-

[5]Each experiment involved 100 independent iterations of randomly selecting data and performing the classification algorithm. The reported values are means over these iterations.

rithm, which tries to minimize the same objective function as mincut, but finds a local rather than global optimum. In particular, one natural iterative approach is to begin with a random labeling, and then to perform hill-climbing, flipping labels so long as this reduces the cut value, until we reach a locally-optimal labeling. We compare the two algorithms on the same synthetic dataset as used before and find that mincut gives a far better performance (Figure 4). In fact, the iterative algorithm performs extremely poorly.

6. Conclusions

We describe a new method of utilizing unlabeled data for classification based on graph cuts. The essence of the approach is to assign values to the unlabeled examples in a way that optimizes consistency in a nearest-neighbor sense (i.e., that similar examples should be classified similarly). What makes this approach interesting from the theoretical point of view is that this is an optimization that can be performed in polynomial time. We motivate this approach both through self-consistency measures (minimizing LOOCV error of certain algorithms) and through a fairly natural generative model.

We find experimentally that the Graph Mincut algorithm performs reasonably well when compared to other learning algorithms that do not use the unlabeled data, especially when there are very few labeled examples. The underlying algorithm, however, has many degrees of freedom — in particular in the design of the edge-weighting function — and it may well be that we have not yet found the best way to compute a weighting function from the information available at the time of learning. For example, even for the simplest case of setting weights to 0 or 1 based on a single real-valued parameter δ, the experiments show that there was a significant gap between our rules for choosing δ in advance and the best δ in hindsight. Nonetheless, this is an efficient algorithm, and adds a new technique to our algorithm repertoire. We also find experimentally that the mincut approach is robust to noise. It would be interesting to use a variation of the method described here for noise reduction in real world datasets. The noise-robustness also suggests that the mincut approach could be used in conjunction with another learning algorithm: first the other learning algorithm would be used to give initial labels to the unlabeled data, and then the mincut algorithm would "clean up" this labeling by enforcing a kind of global consistency. We intend to explore this in future work. We would also like to compare the performance our algorithm with that of Transductive SVM.

Acknowledgements

We would like to thank Nikhil Bansal, Anubhav Gupta, and John Langford for a number of helpful discussions and suggestions. This research has been supported in part by NSF grants CCR-9732705 and CCR-0085982.

References

Bennett, K. P., & Demiriz, A. (1998). Semi-supervised support vector machines. *Advances in Neural Information Processing Systems 10* (pp. 368–374). MIT Press.

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. *Proceedings of the 1998 Conference on Computational Learning Theory.*

Boykov, Y., Veksler, O., & Zabih, R. (1998). Markov random fields with efficient approximations. *IEEE Computer Vision and Pattern Recognition Conference.*

Castelli, V., & Cover, T. (1996). The relative value of labeled and unlabeled samples in pattern-recognition with an unknown mixing parameter. *IEEE Transactions on Information Theory, 42*, 2102–2117.

Cormen, T., Leiserson, C., & Rivest, R. (1990). *Introduction to algorithms.* MIT Press.

Greig, D., Porteous, B., & Seheult, A. (1989). Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society, Series B, 51*, 271–279.

Hofmann, T. (1999). Text categorization with labeled and unlabeled data: A generative model approach. *NIPS 99 Workshop on Using Unlabeled Data for Supervised Learning.*

Kleinberg, J., & Tardos, E. (2000). Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. *40th Annual Symposium on Foundations of Computer Science.*

Nigam, K., McCallum, A., Thrun, S., & Mitchell, T. (1998). Learning to classify text from labeled and unlabeled documents. *Proceedings of the Fifteenth National Conference on Artificial Intelligence.* AAAI Press.

Pietra, S. D., Pietra, V. D., & Lafferty, J. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 19*, 380–393.

Ratsaby, J., & Venkatesh, S. S. (1995). Learning from a mixture of labeled and unlabeled examples with parametric side information. *Proceedings of the 8th Annual Conference on Computational Learning Theory* (pp. 412–417). ACM Press, New York, NY.

Roweis, S., & Saul, L. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science, 290*, 2323–2326.

Roy, S., & Cox, I. J. (1998). A maximum-flow formulation of the n-camera stereo correspondence problem. *International Conference on Computer Vision (ICCV'98)* (pp. 492–499).

Schuurmans, D. (1997). A new metric-based approach to model selection. *AAAI/IAAI* (pp. 552–558).

Shi, J., & Malik, J. (1997). Normalized cuts and image segmentation. *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (pp. 731–737).

Snow, D., Viola, P., & Zabih, R. (2000). Exact voxel occupancy with graph cuts. *IEEE Conference on Computer Vision and Pattern Recognition.*

Tenenbaum, J., de Silva, V., & Langford, J. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science, 290.*

UCI (2000). Repository of machine learning databases. http://www.ics.uci.edu/ mlearn/MLRepository.html.

Wu, Z., & Leahy, R. (1993). An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence, 15*, 1101–1113.

Zhang, T., & Oles, F. J. (2000). A probability analysis on the value of unlabeled data for classification problems. *Seventeenth International Conference on Machine Learning.*