Fitting a Graph to Vector Data

Samuel I. Daitch SAMUEL.DAITCH@YALE.EDU
Yale University, New Haven, CT USA

Jonathan A. Kelner KELNER@MIT.EDU
Massachusetts Institute of Technology, Cambridge, MA USA

Daniel A. Spielman SPIELMAN@CS.YALE.EDU
Yale University, New Haven, CT USA

Keywords: Learning on Graphs, Transductive Classification, Transductive Regression, Clustering

Abstract

We introduce a measure of how well a combinatorial graph fits a collection of vectors. The optimal graphs under this measure may be computed by solving convex quadratic programs and have many interesting properties. For vectors in d dimensional space, the graphs always have average degree at most $2(d+1)$, and for vectors in 2 dimensions they are always planar. We compute these graphs for many standard data sets and show that they can be used to obtain good solutions to classification, regression and clustering problems.

1. Introduction

Given a collection of vectors $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n \in \mathbb{R}^d$, we ask the question,

"What is the right graph to fit to this set of vectors?"

In recent years, a number of researchers have gained insight by fitting graphs to their data and then using these graphs to solve clustering, classification, or regression problems on their data, *e.g.* (Ng et al., 2001; Zhu et al., 2003; Belkin & Niyogi, 2003; Joachims, 2003; Zhou & Schölkopf, 2004a; Coifman et al., 2005). They have employed simply defined graphs that are easy to compute, associating a vertex of the graph with each data vector, and then connecting vertices whose vectors are sufficiently close, sometimes with weights depending on the distance. Not surprisingly, different results are obtained by the use of different graphs (Maier et al., 2008), and researchers have studied how to combine different graphs in a way that tends to give heavier weight to the better graphs (Argyriou et al., 2006). In this paper, we study what can be gained by choosing the graphs with more care.

For a set of vectors $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$, we construct a weighted, undirected graph on n vertices, where $w_{i,j} = w_{j,i} \geq 0$ denotes the weight of edge (i, j), and $d_i = \sum_j w_{i,j}$ denotes the weighted degree of vertex i. When there is no edge (i, j), we have $w_{i,j} = 0$. We do not allow self-loops, so $w_{i,i} = 0$ for all i.

We measure how well the graph with weights $\boldsymbol{w}$ fits the vectors by how small it makes the following function, which is a weighted sum of the squared distance from each vertex to the weighted average of its neighbors:

$$f(\boldsymbol{w}) = \sum_i \left\| d_i \boldsymbol{x}_i - \sum_j w_{i,j} \boldsymbol{x}_j \right\|^2.$$

If we let X be the n-by-d matrix with ith row $\boldsymbol{x}_i$, and let L be the graph Laplacian matrix, defined as

$$L_{i,j} = \begin{cases} -w_{i,j} & \text{if } i \neq j \\ d_i & \text{if } i = j \end{cases},$$

then f may be rewritten as

$$f(\boldsymbol{w}) = \|LX\|_F^2,$$

where $\|M\|_F$ is the Frobenius norm $(\sum_{i,j} M_{i,j}^2)^{1/2}$.

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009. Copyright 2009 by the author(s)/owner(s).

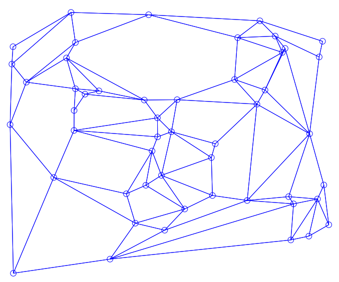

Figure 1. The hard graph for a random set of vectors in two dimensions.

Since $f = 0$ for a graph with no edges, we construct graphs that minimize f subject to constraints that bound the vertex degrees away from zero.

We define a *hard graph* of the vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ to be a graph minimizing f subject to each vertex having weighted degree at least 1. As some vectors could be outliers, we also consider allowing some vertices to have lower weighted degree. To this end, we define an α-*soft* graph of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ to be a graph minimizing f subject to

$$\sum_i (\max(0, 1 - d_i))^2 \leq \alpha n. \tag{1}$$

In special cases, such as when the vectors exhibit certain symmetries, the solutions to these programs will not be unique. Thus, when we refer to *a* hard graph or α-soft graph for a set of vectors, this is not intended to imply that it is the unique such graph.

Our measure of quality of fit $f = \|LX\|_F^2$ is similar to the one found in the locally linear embedding algorithm of (Roweis & Saul, 2000). If the graph is a collection of k disjoint cliques, with the weight on each edge being the reciprocal of the number of vertices in its clique, then f is just the value of the k-means objective function of the partition of the vectors into sets corresponding to the cliques. Also, the singular value decomposition of X may be understood to compute for each k the k-dimensional projection matrix minimizing $\|(I - \Pi)X\|_F$.

In Section 2, we sketch how we compute these graphs. In Section 3, we prove that the hard and α-soft graphs of a point set in $\mathbb{R}^d$ are sparse and that they are planar for two-dimensional data. We expect these graphs will be discovered to have other interesting combinatorial properties. In Sections 4.1, 4.2 and 4.3 we present the results of using these graphs to solve classification, regression, and clustering problems on many standard data sets. The classification and regression experiments are done in the transductive setting, where unlabeled data is used to construct the graph. Even with no parameters to choose, our graphs provide very good answers to many of these problems.

2. Computing the graphs

2.1. Hard Graph

Let us first show that a hard graph is obtained by solving a convex quadratic program.

Let E be the set of all possible edges in the graph, and let $m = |E|$. We define U to be the $n \times m$ matrix such that the column of U corresponding to edge $e = (i, j) \in E$ has exactly two nonzero entries: $U_{i,e} = 1$ and $U_{j,e} = -1$. We also define the length-m vector $\boldsymbol{w}$ to contain all the edge weights, and we let W be the $m \times m$ diagonal matrix with diagonal entries given by $\boldsymbol{w}$. The graph Laplacian may then be expressed as $L = UWU^T$.

Let $\boldsymbol{x}^{(k)}$ be the kth column of X. Define the vector $\boldsymbol{y}^{(k)} = U^T \boldsymbol{x}^{(k)}$, and let $Y^{(k)}$ be the diagonal matrix containing the entries of $\boldsymbol{y}^{(k)}$ on its diagonal.

Then the edge weights $\boldsymbol{w}$ of the hard graph are the weights satisfying $d_i \geq 1$ that minimize

$$f(\boldsymbol{w}) = \|LX\|_F^2 = \sum_{k=1}^{d} \left\| L\boldsymbol{x}^{(k)} \right\|^2 = \sum_{k=1}^{d} \left\| UWU^T \boldsymbol{x}^{(k)} \right\|^2$$
$$= \sum_{k=1}^{d} \left\| UW\boldsymbol{y}^{(k)} \right\|^2 = \sum_{k=1}^{d} \left\| UY^{(k)}\boldsymbol{w} \right\|^2 = \|M\boldsymbol{w}\|^2$$

where $M^T = \begin{bmatrix} Y^{(1)}U^T & \ldots & Y^{(d)}U^T \end{bmatrix}$.

Since there are $\binom{n}{2}$ edges that could appear in the hard graph, it is computationally infeasible to directly solve this quadratic program in all $\binom{n}{2}$ variables for even moderately large n. Instead, we solve the quadratic program on a small subset of edges. We then compute a small set of new edges that will improve the hard graph, add these in to the quadratic program, and compute the solution with the new edges, removing those edges whose weights have been set to zero. We repeat until the graph cannot be improved. Since these graphs are sparse, as will be proven in the next section, our quadratic programs never get too large.

To determine which edges to add to the quadratic program, we consider the Lagrange function

$$\Lambda(\boldsymbol{w}, \boldsymbol{z}) = f(\boldsymbol{w}) - \sum_i z_i(d_i - 1) = \|M\boldsymbol{w}\|^2 - \boldsymbol{z}^T(A\boldsymbol{w} - \mathbf{1}),$$

where A is the matrix obtained by taking the absolute values of the entries of U, so that $A\boldsymbol{w}$ gives the vector of weighted degrees.

The primal-dual solution pair $(\boldsymbol{w}, \boldsymbol{z}) \geq 0$ must satisfy the Karush-Kuhn-Tucker conditions, namely: $\frac{d\Lambda}{d\boldsymbol{w}} \geq 0$, $\frac{d\Lambda}{d\boldsymbol{z}} \leq 0$, $\boldsymbol{w}^T \frac{d\Lambda}{d\boldsymbol{w}} = 0$, and $\boldsymbol{z}^T \frac{d\Lambda}{d\boldsymbol{z}} = 0$, where

$$\frac{d\Lambda}{d\boldsymbol{w}} = 2M^T M\boldsymbol{w} - A^T \boldsymbol{z} \quad \text{and} \quad \frac{d\Lambda}{d\boldsymbol{z}} = \mathbf{1} - A\boldsymbol{w}.$$

When we solve the quadratic program on a subset of edges, we obtain a solution pair $(\boldsymbol{w}, \boldsymbol{z}) \geq 0$ that satisfy all of the KKT conditions on the full quadratic program, except that $\frac{d\Lambda}{dw_{(i,j)}}$ may be negative on the excluded edges. If there are any edges (i, j) for which $\frac{d\Lambda}{dw_{(i,j)}} < 0$, we add to our quadratic program the edges with the smallest $\frac{d\Lambda}{dw_{(i,j)}}$ values. If there are no such edges, then we have a solution to the full quadratic program, and we are done.

In our experiments, we use the Matlab package SDPT3 (Tütüncü et al., 2003; Toh et al., 1999) to solve the quadratic programs. In Table 1, we indicate how long it took to compute the hard graph for the data sets on which we performed experiments.

2.2. α-Soft graph

Let us define $\eta(\boldsymbol{w}) = \|\max(\mathbf{0}, \mathbf{1} - A\boldsymbol{w})\|^2$ to be the left-hand-side term in (1), which indicates to what extent the weighted degrees are smaller than 1. Since the value of $f(\boldsymbol{w})$ is always improved by uniformly scaling down all edge weights, it is clear that an α-soft graph must satisfy (1) with equality, that is, it must have $\eta(\boldsymbol{w}) = \alpha n$.

It is inefficient to directly solve the optimization problem that yields an α-soft graph. Instead, to compute the α-soft graphs, we solve optimization problems of the form

$$\min_{\boldsymbol{w}} \{f(\boldsymbol{w}) + \mu \cdot \eta(\boldsymbol{w}) : \boldsymbol{w} \geq 0\} \tag{2}$$

for various values of μ. For any given μ, if $\boldsymbol{w}$ is a solution to (2) then it must also be an α-soft graph for $\alpha = \eta(\boldsymbol{w})/n$. Furthermore, note that as μ increases, α decreases monotonically. Thus, to compute an α-soft graph, we solve (2) using an initial guess for the value of μ, and we then adjust μ up or down proportionally to how far $\eta(\boldsymbol{w})/n$ is from the desired value of α. We may repeat this until we are arbitrarily close to the desired α. In our experiments, when we construct 0.1-soft graphs, we actually search for graphs which have α in the range 0.1 ± 0.01.

To solve a convex program of the form in (2), note that it can be formulated as a non-negative least squares problem:

$$\min_{\boldsymbol{w},\boldsymbol{s}} \{\|M\boldsymbol{w}\|^2 + \mu \|\mathbf{1} - A\boldsymbol{w} - \boldsymbol{s}\|^2 : \boldsymbol{w}, \boldsymbol{s} \geq 0\}.$$

In our experiments, we solve these problems using Matlab's `quadprog` routine. As with the hard graphs, we use the technique described in section 2.1 to reach the solution to the non-negative least squares problem by solving a sequence of such problems on subsets of the edges.

3. Properties of the graphs

We begin by mentioning that for the set of vectors in $\{0, 1\}^5$ with an even number of ones, neither the hard graph nor the α-soft graph are unique. However, we conjecture that both graphs are unique for almost all sets of vectors (with probability 1 under arbitrarily small perturbations).

Let us prove that these graphs have average degree at most $2(d + 1)$. We suggest that the average degree of the hard or α-soft graph of a set of vectors may be a useful measure of the essential dimensionality of those vectors, and that it will be small if they lie close to a low-dimensional manifold of low curvature. We also conjecture that for every set of vectors of arbitrarily high dimension, there is a sparse, approximately optimal solution to the programs defining the hard and α-soft graphs.

Theorem 3.1. *For every $\alpha > 0$, every set of n vectors in $\mathbb{R}^d$ has a hard and an α-soft graph with at most $(d + 1)n$ edges.*

Proof. Recall that the objective function optimized by these graphs is given by a quadratic form $\|M\boldsymbol{w}\|^2$ on the weights $\boldsymbol{w}$, where the matrix M has dn rows. Let us again write the vector of degree sums as $A\boldsymbol{w}$, where A has n rows.

Suppose for the sake of contradiction that the minimum number of edges in a hard graph (or α-soft graph) is $m > (d+1)n$. Let $\boldsymbol{w}$ be the weights in such a graph.

Then there must be some some non-zero (but not necessarily positive) vector $\boldsymbol{w}^*$, with non-zero entries restricted to these m edges, such that $\begin{bmatrix} M \\ A \end{bmatrix} \boldsymbol{w}^* = 0$.

Clearly there is some r such that the weights $\boldsymbol{w}' = (\boldsymbol{w} + r\boldsymbol{w}^*)$ remain nonnegative and have at least one fewer positive edge than $\boldsymbol{w}$. Since $M\boldsymbol{w}' = M\boldsymbol{w}$ and $A\boldsymbol{w}' = A\boldsymbol{w}$, the score and degrees have not changed. So these new weights still form a hard (or α-soft) graph, but now with fewer than m edges. $\square$

Theorem 3.2. *For every $\alpha > 0$, every set of n vectors in $\mathbb{R}^2$ has a hard and an α-soft graph that are planar.*

We prove a more general statement, which implies Theorem 3.2 by treating edges as cliques of size 2.

Theorem 3.3. *Let $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and $T = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m\}$ be two sets of points the interior of whose convex hulls intersect. Then, for every $\alpha > 0$, any hard or α-soft graph of maximum total degree of a set containing $S \cup T$ either does not contain a clique on S or does not contain a clique on T.*

Proof. Let $\boldsymbol{z}$ be a point in the intersection of the interiors of the convex hulls of S and T. We know that there exist $\alpha_i > 0$ such that $1 = \sum_i \alpha_i$ and $\boldsymbol{z} = \sum_i \alpha_i \boldsymbol{x}_i$, and there also exists $\beta_i > 0$ such that $1 = \sum_i \beta_i$ and $\boldsymbol{z} = \sum_i \beta_i \boldsymbol{y}_i$.

Consider a hard graph (or α-soft graph) that contains a clique on S and a clique on T. Suppose that we decrease the weight on every edge $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ by $r\alpha_i\alpha_j$ and on every edge $(\boldsymbol{y}_i, \boldsymbol{y}_j)$ by $r\beta_i\beta_j$, while we increase the weight on every edge $(\boldsymbol{x}_i, \boldsymbol{y}_j)$ by $r\alpha_i\beta_j$. We choose r just large enough to eliminate some edge from one of the cliques. We will show that these changes increase the weighted degree of every vertex in $S \cup T$ and do not change the objective function, so we can always construct a hard (or α-soft) graph with an edge missing from one of the two cliques and larger total degree.

First let us confirm that the weighted degrees increase: indeed, the weighted degree of $\boldsymbol{x}_i \in S$ increases by

$$\begin{aligned}
&\sum_{j \in [m]} r\alpha_i\beta_j - \sum_{\substack{k \in [n] \\ k \neq i}} r\alpha_i\alpha_k \\
&= r\alpha_i \left(\sum_{j \in [m]} \beta_j - \sum_{k \in [n]} \alpha_k + \alpha_i \right) \\
&= r\alpha_i^2 > 0.
\end{aligned}$$

A symmetric argument holds for the vertices in T.

Now let us show that the score of the graph does not change. For each vertex $\boldsymbol{v}$, let us define the vector

$$\boldsymbol{\delta}(\boldsymbol{v}) = \sum_{(\boldsymbol{v}, \boldsymbol{v}') \in E} w_{(\boldsymbol{v}, \boldsymbol{v}')} (\boldsymbol{v} - \boldsymbol{v}'),$$

and note that the objective function is $\sum_{\boldsymbol{v}} \|\boldsymbol{\delta}(\boldsymbol{v})\|^2$. So it suffices for us to show that we are not changing the value of any $\boldsymbol{\delta}(\boldsymbol{x}_i)$ or $\boldsymbol{\delta}(\boldsymbol{y}_j)$. Indeed, the amount by which we change $\boldsymbol{\delta}(\boldsymbol{x}_i)$ is

$$\begin{aligned}
&\sum_{j \in [m]} r\alpha_i\beta_j(\boldsymbol{x}_i - \boldsymbol{y}_j) - \sum_{k \in [n]} r\alpha_i\alpha_k(\boldsymbol{x}_i - \boldsymbol{x}_k) \\
&= r\alpha_i \Big[\Big(\sum_{j \in [m]} \beta_j - \sum_{k \in [n]} \alpha_k \Big) \boldsymbol{x}_i \\
&\quad + \Big(\sum_{k \in [n]} \alpha_k \boldsymbol{x}_k - \sum_{j \in [m]} \beta_j \boldsymbol{y}_j \Big) \Big] \\
&= r\alpha_i \left[(1 - 1)\boldsymbol{x}_i - (\boldsymbol{z} - \boldsymbol{z}) \right] = 0.
\end{aligned}$$

□

4. Experimental Results

Table 1 lists the data sets we used in our experiments. For each data set, we provide the average degrees of the hard and 0.1-soft graphs as d_{hard} and d_{soft}. Before building our graphs, we always normalize the data by rescaling each dimension so that it has standard deviation one. Observe that the average degree of each graph is lower than that predicted by the analysis in Theorem 3.1. (Recall that the average degree of a graph is twice the number of edges, divided by the number of vertices.)

In our experiments, we compare our graphs with those obtained by standard approaches of constructing graphs from vectors. The most common approach is to use weighted k-nearest neighbor graphs (denoted KNN in the tables). These graphs are specified by two parameters, k and σ. Each vertex is connected to its k nearest neighbors, in Euclidean distance, with an edge weight of either 1, or with weight inverse exponential in the square of the edge of the length, divided by $2\sigma^2$. We tried all $k = 3, 5, 7, 10, 15, 20, 25, 30, 40$, and σ a power of 2 between 2^{-10} and 2^{10} times σ_0, where σ_0 is the mean distance across edges in the k-nearest neighbor graph. We also tried forming graphs by connecting all pairs of vertices within a given distance r and considered weighting the edges as we did for the k-nearest neighbor graphs (denoted THRESH in the tables). For r, we tried dividing the median inter-vertex distance by all powers of $2^{1/3}$ between 0 and 27.

4.1. Classification

We employed the simple algorithm of (Zhu et al., 2003) for learning labels of vertices in the graphs. We do not yet know if our results would be improved by using one of the algorithms from (Zhou & Schölkopf, 2004a; Sindhwani et al., 2005; Zhou & Schölkopf, 2004b; Zhou et al., 2003; M. Belkin, 2004; Wang et al., 2008). We first explain how we handle the case of two classes. Let S be the set of vectors whose labels we know, and let $c \in \mathbb{R}^n$ be a vector such that $c_i \in \{0, 1\}$ for each $i \in S$, depending on the class of vector i. We then solve for the vector x minimizing

$$x^T L x = \sum_{(i,j) \in E} w_{i,j}(x_i - x_j)^2, \tag{3}$$

subject to $x_i = c_i$ for $i \in S$. For every $j \notin S$, we then guess that the class of vector j is 0 if $x_j < 1/2$ and 1 otherwise. Note that x can be found by solving one linear equation in a diagonally-dominant matrix, so this can be done very quickly (Spielman & Teng, 2004).

When we have $k > 2$ classes, we construct a vector c^j

Table 1. Average degrees of our graphs for various data sets. SOURCE indicates whether we obtained the data from the UCI Machine Learning Repository (Asuncion & Newman, 2007) or from LIBSVM (Chang & Lin, 2001). TYPE indicates whether the data come from a classification problem, and if so how many classes, or whether they come from a regression problem. SOFT TIME and HARD TIME indicate how many seconds it took to compute the soft and hard graphs on a single core of a Dell Precision 690 workstation with an Intel Xeon 2.66 GHz processor and 4GB of RAM. For each data set, we normalized every column to have variance one.

DATA SET	SOURCE	TYPE	n	dim	d_{hard}	d_{soft}	SOFT TIME (sec)	HARD TIME (sec)
ABALONE	LIBSVM	REGRESSION	4177	8	13.1	12.7	1,582	49,986
GLASS	UCI	6 CLASSES	214	9	8.9	8.6	8	22
HEART	UCI	2 CLASSES	270	13	11.1	11.0	12	36
HOUSING	LIBSVM	REGRESSION	506	13	8.8	10.1	23	114
IONOSPHERE	UCI	2 CLASSES	351	34	13.0	11.9	72	148
IRIS	UCI	3 CLASSES	150	4	7.0	7.0	4	17
MACHINE	UCI	REGRESSION	209	6	8.9	8.0	11	26
MPG	LIBSVM	REGRESSION	392	7	7.6	8.8	16	43
PIMA	UCI	2 CLASSES	768	8	11.2	10.9	111	529
SONAR	UCI	2 CLASSES	208	60	12.7	13.0	38	50
VEHICLE	UCI	4 CLASSES	846	18	11.8	11.5	159	889
VOWEL990	LIBSVM	11 CLASSES	990	10	10.4	10.1	85	706
WINE	UCI	3 CLASSES	178	13	9.1	9.9	6	15

for class j that is 1 for labeled examples in the class and 0 elsewhere. We then solve (3) for each class to obtain a vector x^j, and we guess that an unlabeled vertex i belongs to the class j for which x_i^j is largest.

Table 2 presents the results of performing classification using 10-fold cross-validation. For each experiment, we compute the hard graph and the 0.1-soft graph once. We then randomly partition the vectors into 10 sets of size as equal as possible, and for each set we use the algorithm above to infer the classes, using the labels on the other nine sets. We repeat each of these experiments 100 times, and report the average error. There were no parameters to train.

For comparison, we also report results of experiments done using conventional graphs, LIBSVM (Chang & Lin, 2001), and from experiments reported in (Su & Zhang, 2006) and (Kotsiantis et al., 2006). In the experiments that we performed on competing algorithms, we used the other nine sets to train parameters. In the columns KNN and THRESH we compare to the graphs described above, and chose parameters by leave-one-out cross validation on the nine other sets. In each experiment with LIBSVM, we called the `easy` routine on the nine tenths of the data to choose parameters and train the support vector classifier. We then used this classifier on the remaining tenth of the data. The results are the averages of shifting over all 10 parts of the partition and repeating the whole process 50 times.

The results we copy from the (Su & Zhang, 2006) are for FBC: Full Bayes Classifier, AODE: Averaged One-Dependence Estimators (Webb et al., 2005), and HGC: the Hill Climbing BN Learning Algorithm (Heckerman). The results we copy from the (Kotsiantis et al., 2006) are for NB: Naive Bayesian networks, C4.5 (Quinlan, 1993), BP: Back Propagation, and SMO: Sequential Minimal Optimization.

These results suggest that our graphs provide very good classifiers. They perform exceptionally well on the ionosphere and sonar data sets. The only data set on which any algorithm performs significantly better is vehicle, on which LIBSVM does particularly well.

4.2. Regression

We again use the algorithm of (Zhu et al., 2003) to predict the values of the unlabeled vertices. Supposing that we know the values f_i for $i \in S$, we predict the remaining f_i values to be those that minimize

$$\boldsymbol{f}^T L \boldsymbol{f} = \sum_{(i,j) \in E} w_{i,j} (f_i - f_j)^2$$

subject to fixing the known f_i values.

Table 3 presents the results of the regression experiments using 10-way cross-validation. Again, for each experiment, we compute the hard graph and the 0.1-soft graph once. We then randomly partition the vectors into 10 sets of size as equal as possible, and for each set we use the algorithm above to infer the function. We repeat each of these experiments 50 times and report the average error. For the ABALONE data set, because of time constraints due to its larger size, we perform 2-way rather than 10-way cross-validation.

Again we observed what happens when we replace our

Table 2. Classification error (%), 10-fold cross validation. The best result for each data set is bold. The experiments that do not perform better than ours have a grey background.

DATA SET	HARD	0.1-SOFT	KNN	THRESH	LIBSVM	FBC	AODE	HGC	NB	C4.5	BP	SMO
GLASS	27.78	28.30	**26.92**	33.30	31.44	37.56	38.27	41.64	50.55	32.37	32.68	42.64
HEART	18.18	17.81	**16.05**	16.1	17.01	16.19	16.37	17.41	16.41	21.85	16.70	16.19
IONOSPHERE	**4.75**	5.57	18.50	6.34	6.20	9.20	8.26	6.60	17.83	10.26	12.93	12.07
IRIS	4.87	4.21	4.46	6.20	**3.87**	6.27	6.00	3.93	4.47	5.27	15.20	15.13
PIMA	26.64	26.61	24.54	26.45	23.24	25.15	23.43	24.08	24.25	25.51	22.96	**22.93**
SONAR	9.16	**8.64**	13.80	14.94	11.71	22.62	20.09	30.84	32.29	26.39	21.33	22.12
VEHICLE	23.03	22.47	27.70	29.98	**14.87**	25.77	28.35	31.90	55.32	27.72	18.89	25.92
VOWEL990	1.19	0.95	2.62	0.98	**0.64**	6.54	10.36	7.30	37.10	19.80	7.27	29.39
WINE	2.92	2.62	2.86	3.64	2.57				2.54	6.80	1.98	**1.24**

Table 3. Regression mean-square error, k-fold cross validation (k=2 for ABALONE, k=10 for other data sets). For each data set, the labels have been rescaled to have variance one. The best result for each data set is bold. The experiments that do not perform better than ours have a grey background.

DATA SET	HARD	0.1-SOFT	KNN	THRESH	EPSILON-SVR	GPROC
ABALONE	**0.479**	0.482	0.492	0.657		
HOUSING	0.136	0.138	0.224	0.507	0.138	**0.112**
MACHINE	0.170	0.185	**0.164**	0.608	0.394	0.890
MPG	0.120	**0.118**	0.137	0.145	0.128	0.129

graphs with the KNN and THRESH graphs described above. In the KNN and THRESH regression experiments, we ran each experiment 10 times. The hard and 0.1-soft graphs outperform these graphs for all but one data set, and even in that data set the performance is very close.

We again also ran experiments using LIBSVM (Chang & Lin, 2001). We modified the classification routine `easy` to search the same parameter ranges but do ϵ-support vector regression instead of classification. We fed this routine nine tenths of the data to choose parameters and train the support vector machine, which we then used to predict values on the remaining tenth of the data. We did this for each of the 10 partitions of data, and repeated the whole experiment 20 times.

We also ran 10-way cross validation experiments using the `gproc` Gaussian process regression algorithm from Spider (Weston et al., 2008), repeating each experiment 5 times.

Due to the size of ABALONE, we were unable to run regression tests on it with LIBSVM or Spider. On the other three sets of regression data that we tested, our graphs outperformed LIBSVM on all of them, and Spider on all but one.

4.3. Clustering

Given the graph associated with a set of unlabeled vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, one may obtain a clustering of the vectors into k subsets by finding a good k-partition of the corresponding graph. We did this with a spectral algorithm; it remains an interesting question whether other graph partitioning algorithms would improve the results.

The graph partitioning algorithm that we applied is essentially the same as that used by Ng, Jordan, and Weiss (Ng et al., 2001). We formed the normalized Laplacian $\widetilde{L} = D^{-1/2}LD^{-1/2}$, where D is the diagonal matrix whose i^{th} diagonal entry is the weighted degree of vertex i. We then found its eigenvectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$, where the $\boldsymbol{v}_i$ are sorted in increasing order of the corresponding eigenvalues.

Let $V = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t]$ be the $n \times t$ matrix whose columns are given by the first t eigenvectors. (We shall discuss the correct value of t below.) Following Ng, Jordan, and Weiss, we let W be the matrix obtained by scaling the rows of V to each have norm 1. The rows of W give us n points in $\mathbb{R}^t$, which we clustered using k-means. We then lifted this clustering back to the original vectors.

It is not immediately obvious how best to choose t. A standard answer, which was the one employed by Ng, Jordan, and Weiss, is to set it equal to the desired number of clusters k. However, we found that this was not the best strategy with our graphs. Intuitively, one may view the matrix V as a low-rank approximation to L. If one works in a model in which the graph is expected to have a very sparse cut that breaks it into

Table 4. Clustering error (%). We compare our results to k-means and the Ng-Jordan-Weiss algorithm. For both the hard and 0.1 soft graphs, we list the performance when $t = k$ and when t is chosen heuristically as described in the text. In the t CHOSEN columns, the selected value of t is listed in parentheses.

DATA SET	K-MEANS	NJW	HARD, $t = k$	HARD, t CHOSEN	0.1-SOFT, $t = k$	0.1-SOFT, t CHOSEN
GLASS	0.41	0.43	0.44	0.43 (12)	0.45	0.45 (12)
HEART	0.41	0.42	0.35	0.35 (2)	0.19	0.19 (2)
IONOSPHERE	0.29	0.33	0.26	0.09 (15)	0.33	0.09 (15)
IRIS	0.11	0.33	0.33	0.15 (5)	0.17	0.09 (8)
PIMA	0.34	0.35	0.35	0.35 (12)	0.35	0.35 (12)
SONAR	0.45	0.47	0.41	0.41 (35)	0.4	0.35 (35)
VEHICLE	0.55	0.62	0.58	0.54 (6)	0.56	0.53 (6)
VOWEL990	0.66	0.76	0.65	0.66 (30)	0.65	0.6 (30)
WINE	0.3	0.33	0.03	0.03 (3)	0.03	0.03 (3)

k large and well-connected pieces, L will be well approximated by its rank-k approximation, and taking $t = k$ is the right course of action. This occurs in most random graph models in which one assumes a good k-partition exists (McSherry, 2001). The theoretical basis for this arises from the existence in these models of a large gap between the k^{th} and $(k+1)^{\text{st}}$ eigenvalues. However, this gap arises from the very strong clustering assumed to exist in these models, and it does not always occur in practice, even when there is a good partition into k clusters.

Ideally, one wants to include only as many eigenvectors as are necessary to get a good approximation of the data. Let $\gamma_j = \sum_i (\boldsymbol{v}_j \cdot \boldsymbol{x}_i)^2$ be the total squared norm of the projections of the x_i into the j^{th} eigenspace. In practice, there tend to be some number of reasonably large γ_i, after which they fall off quite sharply. Setting t to be an index after which the γ_i are small yielded considerably better results than simply setting $t = k$. While the choice of the exact cutoff point is somewhat arbitrary, it tends to be fairly clear in practice, and the quality of the resulting clusterings tended not to be very sensitive to its exact value.

To evaluate the efficacy of this approach, we removed the class labels from the data sets from Section 4.1 and applied our algorithm. We then compared the resulting clustering to the actual correct classification and computed the fraction of the points that were misclassified. For comparison, we performed the same experiments using k-means and the Ng-Jordan-Weiss routines implemented in Spider (Weston et al., 2008).

We computed the classification error for both the hard and 0.1-soft graphs. In order to allow a fair comparison to the Ng-Jordan-Weiss algorithm, which used exactly k values, we computed the performance of our method with $t = k$. In addition, we computed the clustering error when t was chosen heuristically as described above. We note that we selected the t at which the γ_i appeared to become negligible, not the t that gave the lowest error rate (as doing so would require using the actual classification, which was not given to us in the problem).

For both the hard and 0.1-soft graphs, our algorithm with $t = k$ was never significantly worse than the Ng-Jordan-Weiss algorithm, and it was usually significantly better. As the two algorithms performed the same operations on their respective graphs, this provides strong support for the notion that our graphs improve upon the traditional ones. When t was chosen more carefully, our graphs tended to significantly outperform both k-means and Ng-Jordan-Weiss. In particular, they provided extreme improvements on the IONOSPHERE and WINE datasets.

5. Conclusions and Future work

We have suggested optimizing a graph to fit a data set, and found a measure of fitness under which the optimal graphs are sparse, have interesting combinatorial properties, and provide good answers to classification, regression, and clustering properties. We ask if there are other natural graphs to fit to a data set, and if our graphs can be improved for these learning problems. For example, we ask if one can incorporate labeled examples into the construction of the graph. We also ask if there is a natural way to use our graphs to infer labels of vectors that were not used in the graph construction, such as was done in the work of (Yu et al., 2004; Sindhwani et al., 2005; Coifman & Lafon, 2006).

Acknowledgments

This work was partially supported by NSF Grants CCF-0634957 and CCF-0843915. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science

Foundation.

References

Argyriou, A., Herbster, M., & Pontil, M. (2006). Combining graph laplacians for semi-supervised learning. *Advances in Neural Information Processing Systems*, 67–74.

Asuncion, A., & Newman, D. (2007). UCI machine learning repository. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

Belkin, M., & Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, *15*, 1373–1396.

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/∼cjlin/libsvm`.

Coifman, R. R., & Lafon, S. (2006). Geometric harmonics: A novel tool for multiscale out-of-sample extension of empirical functions. *Applied and Computational Harmonic Analysis*, *21*, 31 – 52.

Coifman, R. R., Lafon, S., Lee, A. B., Maggioni, M., Nadler, B., Warner, F., & Zucker, S. (2005). Geometric diffusion as a tool for harmonic analysis and structure definition of data, part i: Diffusion maps. *PNAS*, *102*, 7426–31.

Joachims, T. (2003). Transductive learning via spectral graph partitioning. *Proc 20th ICML* (pp. 290–297).

Kotsiantis, S. B., Zaharakis, I. D., & Pintelas, P. E. (2006). Machine learning: a review of classification and combining techniques. *Artif. Intell. Rev.*, *26*, 159–190.

M. Belkin, I. Matveeva, P. N. (2004). Regularization and semi-supervised learning on large graphs. *Proc. 17th COLT*.

Maier, M., von Luxburg, U., & Hein, M. (2008). Influence of graph construction on graph-based clustering measures. *Proc. 21st NIPS*.

McSherry, F. (2001). Spectral partitioning of random graphs. *Proc. 42nd IEEE FOCS* (pp. 529–537).

Ng, A. Y., Jordan, M. I., & Weiss, Y. (2001). On spectral clustering: Analysis and an algorithm. *Proc. 15th NIPS* (pp. 849–856).

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Francisco: Morgan Kaufmann.

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

Sindhwani, V., Niyogi, P., & Belkin, M. (2005). Beyond the point cloud: from transductive to semi-supervised learning. *Proc. 22nd ICML* (pp. 824–831).

Spielman, D. A., & Teng, S.-H. (2004). Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. *Proc. 36th ACM STOC*.

Su, J., & Zhang, H. (2006). Full bayesian network classifiers. *Proc. 23rd ICML*.

Toh, K. C., Todd, M. J., & Tütüncü, R. H. (1999). SDPT3—a MATLAB software package for semidefinite programming, version 1.3. *Optim. Methods Softw.*, *11/12*, 545–581. Interior point methods.

Tütüncü, R. H., Toh, K. C., & Todd, M. J. (2003). Solving semidefinite-quadratic-linear programs using SDPT3. *Math. Program.*, *95*, 189–217.

Wang, J., Jebara, T., & Chang, S.-F. (2008). Graph transduction via alternating minimization. *Proc. 25th ICML*.

Webb, G. I., Boughton, J., & Wang, Z. (2005). Aggregating one-dependence estimators. *Journal of Machine Learning*, *58*, 5–24.

Weston, J., Elisseeff, A., BakIr, G., & Sinz, F. (2008). The spider. http://www.kyb.mpg.de/bs/people/spider/.

Yu, K., Tresp, V., & Zhou, D. (2004). *Semi-supervised induction with basis functions* (Technical Report 141). Max Planck Institute for Biological Cybernetics.

Zhou, D., Bousquet, O., Lal, T. N., Weston, J., & Schölkopf, B. (2003). Learning with local and global consistency. *Proc. 17th NIPS*.

Zhou, D., & Schölkopf, B. (2004a). Learning from labeled and unlabeled data using random walks. *26th DAGM Symposium* (pp. 237–244).

Zhou, D., & Schölkopf, B. (2004b). A regularization framework for learning from graph data. *ICML Workshop on Statistical Relational Learning and Its Connections to Other Fields*.

Zhu, X., Ghahramani, Z., & Lafferty, J. D. (2003). Semi-supervised learning using gaussian fields and harmonic functions. *Proc. 20th ICML*.