1 Background

Graph's are intuitively useful for clustering and labeling in models that support a notion of edges. Natural connections exist between the nodes in social networks, linked sets of web pages, paper citation corpuses, etc. By representing these connections in a weighted or unweighted graph it is possible to predict a node’s characteristics from its neighbors or to cluster data based on the graph’s structure.

However, researchers have also attempted to extend the use of graphs to understand data that is not naturally connected. Instead, the data is simply a set of vectors - a more common input structure in machine learning problems. Without edge information the task becomes two-fold:

1. Create a graph by constructing edges between data points based on (potentially high-dimensional) spatial information.

2. Use the ‘artificial’ graph to perform learning operations on the data (cluster data using spectral techniques, label unlabeled vectors based on neighbors, etc.)

Since research has primarily focused on the second step, fairly simple algorithms are typically used for graph construction [2]. For example, exponentially weighted $k$-nearest neighbor graphs and graphs that construct edges between all vertices within a given threshold distance are popular.

More recently, attempts have been made to improve initial graph construction. I will be considering one system in particular that was presented in a 2009 paper titled “Fitting a Graph to Vector Data” by Daitch, Kelner, and my project advisor, Daniel Spielman [1]. In this paper an attempt is made to find a natural graph by constructing edges in such a way that vertices are spatially well approximated by their neighbors.

Consider a problem with $n$ data vectors, $\{x_1, x_2, \ldots, x_n\}$. We can express a candidate graph by constructing the $n^2$ element vector $w$, which contains the weights for every possible edge between data points. If we do not want an edge between vertices $x_i$ and $x_j$, we simply set $w_{i,j} = w_{j,i} = 0$. Furthermore, there are no self-loops so $w_{i,i} = 0 \forall i$. The authors seek the graph that minimizes the objective function:

$$f(w) = \sum_{i=1}^{n} \left\| d_i x_i - \sum_{j=1}^{n} w_{i,j} x_j \right\|^2$$

Where $d_i$ is the weighted degree of vertex $x_i$. This function computes a sum of the squared distances between each vertex and the weighted average of its neighbors. The sum is weighted by $d_i$, so higher degree vertices are more important to the objective function.

Of course this function can be minimized trivially by setting all weights to 0, so the authors suggest 2 possible restrictions to avoid degenerate solutions. Most simply, we can restrict our solution such that $d_i = \sum_{j=1}^{n} w_{i,j} \geq 1$ for all vertices. The graph resulting from this constraint is called the...
“hard graph.” Alternatively, we can allow additional flexibility (some vertices with weighted degree below 1) by constraining the problem such that

$$\sum_{i=0}^{n}(\max(0, 1 - d_i))^2 < \alpha n$$

for some constant $\alpha$. The graph correspond to this second constraint is called the $\alpha$-soft graph.

## 2 Project Focus

In addition to an intuitive motivation, it turns out that graphs constructed according to the system above have some desirable properties. Most importantly, they perform practically well in several benchmark classification, regression, and clustering problems and thus the construction seems worth pursuing further.

### Implementation and Acceleration

Initially my project will focus on efficiently finding $w$ to minimize the objective function, $f(\cdot)$. This turns out to be a convex optimization problem and Daitch et al. show how to formulate it as a quadratic program, for hard constraints, or more specifically, a non-negative least squares problem, for $\alpha$-soft constraints.

Unfortunately, direct optimization becomes intractable as $n$ increases since we have to solve for the weight of every potential edge in the graph, leaving $\binom{n}{2}$ free variables. However, it turns out that the graphs proposed are sparse - it can be shown that every set of vertices has an optimal soft and hard graph with less than $(d + 1)n$ edges, where $d$ is the dimensionality of the data vectors [1]. This suggests a much faster method for finding the optimal graph: it is possible to solve the quadratic program or non-negative least squares problem on a subset of the potential edges, setting all other edge weights to 0. For example, we might start by only allowing non-zero weight on the edges in a quickly computed $k$-nearest neighbors graph. Next, we consider the derivative of the Lagrange function (or least squares function) at the current solution with respect to each $w_{i,j}$ that was forced to zero. If the derivative is non-negative for all $w_{i,j}$, we have found a solution to our program. Otherwise, we can reduce the objective function by adding weight to some $w_{i,j}$. So we simply re-solve, allowing the previously excluded edges with the lowest derivatives to vary as well. By continuing in this way we’ll eventually find a solution, hopefully without having to consider all possible edges.

However, this process is still somewhat inefficient, so I will begin by trying to improve the experimental runtime of the optimization on large graphs. Specifically, I am going to first look into more refined ways of choosing which vectors to add to the program after a solution is found to be non-optimal. Hopefully it is possible to avoid computing a derivative for every edge fixed at zero in each iteration. Additionally, it might make sense to add or excluded certain edges in combination, regardless of the magnitudes of their derivatives. On a related note, it is probably worth more carefully considering how many edges are added in each iteration so that we can find an ideal balance between iteration speed and the number of iterations required to reach a minimum. Finally, I would like to consider the choice of an initial edge subset after examining its importance on the algorithm’s runtime.

Working towards improvements will require significant experiments on hand-generated and real data, so time towards the beginning of the semester will be spent getting a clean implementation running and setting up several test cases. Additionally, I hope that my work leads to ideas for further improvements to the algorithm presented in [1] as I become more familiar with the problem.
Theoretical Exploration

In addition to working on an efficient code base for computing graphs, I would like to spend time thinking about some of the more interesting properties of the model proposed by Daitch et al. Several open questions came out of the initial research, including the problem of uniqueness. Although it is possible to construct highly symmetric data sets that lead to multiple $f(\cdot)$ minimizing graphs, it has been conjectured, although not proven, that a unique solution will exist for all sets of vectors after small random perturbations. I will spend some time looking into this issue.

I would also like to gain a better understanding of how the value of the objective function converges as we move through iterations of the linear program. It might be possible to halt the algorithm before all negative derivative edges are added and still achieve a good approximate solution.

3 Deliverables

Code

I will be writing code, initially in MATLAB, that hopefully improves on the code currently being used to construct hard and soft graphs for vector data sets.

Benchmark Testing

Any new code will be tested against the current optimization algorithm and significant improvements presented.

Experiments

I plan on running and recording the results of experiments on hand-generated and natural data. One goal is to gain a better understanding of the order in which edges are added to the solution vector under the current algorithm. Hopefully results could inform an improved procedure for choosing edges to add to the quadratic program. Experiments on convergence and change propagation (when new edges are introduced) might also be helpful.

Theoretical Results, Theorems, Bounds

Ideally, I will make some progress on the open theoretical questions as well, but naturally this deliverable is more up in the air.

References
