Final Paper

Implementation of Sublinear Clustering Algorithm

By James Camacho and Joseph Camacho

Overview

We implement the algorithm "Sublinear Time and Space Algorithms for Correlation Clustering via Sparse-Dense Decompositions". This clusters data in a (+/-)-labeled graph, where pluses mean two vertices are correlated and minuses mean anticorrelation. For example, a graph of all English words could have pluses between synonyms and minuses between antonyms. In other datasets, such as document clustering, these graphs can be incredibly large and dense, leading to a superlinear number of edges compared to the vertices. Even a single pass through the edges can be infeasible. This algorithm presents an approximation scheme that is $\widetilde{O}(n)$ in the number of vertices, often sublinear in the size of the dataset.

It requires access to the plus-labeled edges in an adjacency list, and works as follows:

1. Sample a constant number of edges from each vertex.
2. Choose a random (sublinear) number of vertices and query every edge of these vertices.
3. Filter these vertices using the sampled edges from step one to find the densest ones.
4. Create candidates for the clusters based on the dense vertices and their neighbors.
5. Find approximate clusters from the candidates.

If the constants are chosen right, there should be a high probability that the candidate clusters in step four form a laminar group, i.e. each candidate is either disjoint or a subset to every other candidate. Then, step five consists of assigning each vertex to the largest candidate it's contained in. If it isn't contained in any candidate clusters, it is assumed to be pretty isolated and assigned to its own cluster.

We can define a cost for the clustering---each time two vertices are put in different clusters but have a plus-edge between them, the cost is increased by one. Similarly if two vertices share a minus-edge but get put in the same cluster, the cost increases by one. It turns out the minus-edges are not too relevant, as two vertices with a minus-edge should only be clustered together if they share a lot of neighbors in the plus-subgraph, in which case the cost would increase whether or not they get clustered together. So, by only using the plus-subgraph, it is possible to determine good clusters with an $O(1)$ approximation to the optimum.

The accuracy of the algorithm depends on the details. How many edges do you sample from each vertex? What is the probability you sample all the edges for a particular vertex? What bounds do you use to filter dense vertices?

The paper suggests sampling $t = O(\varepsilon^{-2} \log n)$ vertices for some constant ε. In step two, a vertex v is chosen with probability $O(\log n / \deg(v))$. Note that this rewards sparser vertices—in a complete graph with n vertices you would only expect $O(\log n)$ vertices, and $O(n \log n)$ edges to be sampled, while a tree would have nearly every vertex chosen. Overall, steps one and two only require us to sample $\widetilde{O}(n\varepsilon^{-2})$ edges.

The filtering in step three is a little more complicated. First, all vertices that have much denser neighbors should be filtered out. This is according to the formula

$$\varepsilon \deg(v) < |\{u \text{ neighbors } v : \deg(u) > (1+\varepsilon)\deg(v)\}|.$$

Then, sparse vertices are filtered out. Call the *low* neighbors of v the vertices u where

$$\deg(u) < (1 + 7\varepsilon)\deg(v).$$

A low neighbor u is *isolated* if

$$|\text{sampled neighbors}(u) \cap \text{low}(v)| < (1 - 4\varepsilon)t,$$

i.e. very few of its neighbors are also low neighbors of v. If a large fraction, at least $2\varepsilon \deg(v)$, of the low neighbors of v are isolated, then v is considered sparse. We're left with only the densest vertices from our sample in step two.

In this paper we explore how the choice of ε and constant multiplier in $t = O(\varepsilon^{-2} \log n)$ affect the operation count and accuracy of the algorithm. We answer questions like, exactly how close are the candidate clusters to a laminar set family? How many intersections can we expect? Do things change rapidly at $\varepsilon = 1/4$? How do sparser or denser datasets change these results?

Operation Count