CPSC 320 Notes, Clustering Completed

October 4, 2018

AS BEFORE: We're given a complete, weighted, undirected graph $G = (V, E)$ represented as an adjacency list, where the weights are all between 0 and 1 and represent similarities—the higher the more similar—and a desired number $1 \leq k \leq |V|$ of categories.

We define the similarity between two categories C_1 and C_2 to be the maximum similarity between any pair of nodes $p_1 \in C_1$ and $p_2 \in C_2$. We must produce the categorization—partition into k (non-empty) sets—that minimizes the maximum similarity between categories.

Now, we'll prove this greedy approach optimal.

1. Sort a list of the edges E in decreasing order by similarity.
2. Initialize each node as its own category.
3. Initialize the category count to $|V|$.
4. While we have more than k categories:
   (a) Remove the highest similarity edge (u, v) from the list.
   (b) If u and v are not in the same category: Merge u's and v's categories, and reduce the category count by 1.

1 Greedy is at least as good as Anything Else

We'll start by noting that any solution to this problem partitions the edges into the "intra-category" edges (those that connect nodes within a category) and the "inter-category" edges (those that cross categories).

1. **Getting to know the terminology:** Imagine we're looking at a categorization produced by our algorithm in which the inter-category edge with maximum similarity is e.

   Can our greedy algorithm's solution have an intra-category edge with **lower** weight than e? Either draw an example in which this can happen, or sketch a proof that it cannot.

2. Give a bound—indicating whether it's an upper- or lower-bound—on the maximum similarity of an arbitrary solution in terms of any one of its inter-category edge weights. (That is, I tell you that the solution has an inter-category edge with weight w. How much can you tell me so far about the solution's overall "goodness"?)

3. Give a bound on the maximum similarity of a solution produced by the greedy algorithm in terms of the weight of any one of the edges it considered in step 4.

4. Consider an arbitrary valid solution $\mathcal{S}$ to an instance of the problem. Prove that its intra-category edges cannot be a proper superset of the intra-category edges of the greedy solution to the same instance (i.e., cannot be the same edges plus at least one more intra-category edge).

   (**Assume** that the greedy algorithm produces valid solutions, i.e., partition V into exactly k subsets.)

5. Now, considering in decreasing order of weight whether each edge is inter- or intra-category, let e be the first edge where the greedy solution differs from $\mathcal{S}$. (If no such edge exists, then they have the same set of categories, are the same solution, and so our greedy algorithm is "at least as good".)

   Use the behaviour of the greedy algorithm and your work in the previous part to explain why e cannot be inter-category in the greedy solution but intra-category in $\mathcal{S}$.

6. Continuing the previous part, we now know that e is intra-category in the greedy solution but inter-category in $\mathcal{S}$. Use your previous work to finish the proof that the greedy solution is "at least as good" according to our metric as the arbitrary solution $\mathcal{S}$ (and therefore the greedy solution is optimal).

2 Challenge

1. We can also give an "exchange" argument starting: "Consider a greedy solution $\mathcal{G}$ and an arbitrary solution $\mathcal{S}$. If they're different, then some edge (u, v) must be inter-category in $\mathcal{S}$ but intra-category in $\mathcal{G}$. In that case, we can make $\mathcal{S}$ more similar to $\mathcal{G}$ without decreasing $\mathcal{S}$'s goodness by..."

   Finish the proof. (Warning: even if you select (u, v) more carefully, you may not **just** want to move u into v's category or vice versa.)

2. Switching to MSTs:

   (a) What does the "cut" property tell us if we partition the graph into a "cluster" of: an already connected set of nodes, and everything else?

   (b) Use this insight to create an efficient algorithm somewhat similar to Kruskal's algorithm except that it adds **many** edges to the MST-so-far at each step of the outer loop.

   (c) Analyse the performance of your algorithm, including determining what data structures to use. (Make each data structure as simple as possible while still obtaining the best bound rather than making each as efficient as possible.)