COS 435: Notes on Clustering Algorithms for 3/30/06

On 3/28/06 I presented on the board single-pass agglomerative clustering and agglomerative clustering for "single link" cluster distance using minimum spanning trees. These are covered in the assigned reading in essentially the same way. On 3/30/06 we covered general hierarchical agglomerative clustering, the k-means algorithm, and divisive clustering. Martin Makowiecki's slides are posted for his presentation on the k-means algorithm. In this note, I am summarizing my development on the board of general hierarchical agglomerative clustering and divisive clustering. These may vary somewhat from the presentations in the book. – there are many variations that use the same principles. I am also adding a *small* amount of detail that I skipped due to time constraints in class.

General hierarchical agglomerative algorithm

Uses any computable cluster similarity measure sim(Ci, Cj):

1. For n objects v1, …, vn, assign each to a singleton cluster Ci = {vi}.
2. repeat {
   a. identify two most similar clusters Cj and Ck (could be ties – chose one pair)
   b. delete Cj and Ck and add (Cj U Ck) to the set of clusters

   } until only one cluster

Dendograms diagram the sequence of cluster merges.

Running time if must compare all clusters each time to do 2a:

$$\sum_{i = 1 \ldots (n-1)} \Big(\text{cost of 2b for } i^{th} \text{ merge} + \text{cost to compute/look up sim(Cj, Ck) for all clusters Cj, Ck in the } i^{th} \text{ repetition of the loop} \Big)$$

If one keeps a matrix of sim(Cj, Ck) for all current pairs, then the running time is:

$$\sum_{i = 1 \ldots (n-1)} \Big(\text{cost of 2b for } i^{th} \text{ merge and update of sim(Ci, Cj) matrix} + (n-i+1)(n-i)/2 \Big)$$

If the update of the sim(Cj, Ck) matrix can be done in $O(n^2)$ steps, the whole algorithm is $O(n^3)$.

For the single-link distance measure, recall that one algorithm builds a minimum spanning tree by sorting all the edges between objects and then processing the edges in sorted order, smallest first, and keeping for the MST any edge that does not form a cycle. This algorithm is doing what the general hierarchical agglomerative algorithm does – the partial MSTs are the clusters. By pre-sorting all the edges between objects, the combined cost of step 2a over all repetitions is $O(n^2)$ – assuming we can detect in constant time that an edge would cause a cycle. This is easy with a random access table in main memory giving the cluster for each object and a challenge if the amount of data forces some of the

tables onto disk. The cost of step 2b to update such a table is O(n) in main memory. This gives a total running time of $O(n^2 \log n)$ assuming random access tables can be used.

Space: for a general similarity matrix at least space $\frac{1}{2}n^2$ is required, assuming sim(vi,vj) = sim(vj,vi). If one can calculate sim(vi,vj), e.g. using a cosine metric for vector objects, then one might need only O(n) space However, this might cost in running time because one might have to recalculate all similarities between objects or clusters each time one needed a most similar pair. O(n) space does not provide enough space to store the similarities between all pairs of clusters until the number of clusters has been reduced to about $\sqrt{n}$.

For "complete linkage" and "all pairs average distance" similarity measures for clusters, we can update the similarity between clusters after a merge in constant time by taking advantage of the following equations (in terms of distance):

- For "complete linkage",
  dist (Cj U Ck , Cp) = max (dist (Cj, Cp) , dist (Ck , Cp))
- For "all pairs average distance",
  dist (Cj U Ck , Cp) = 1/ (|Cj|+|Ck|) * (|Cj|*dist (Cj, Cp) + |Ck|*dist (Ck , Cp))

Then we can maintain a matrix of sim(Cj, Ck) and update it in $O(n^2)$ steps per merge.

Divisive algorithms and cut-based measures of cluster similarity

Cut-based measures of similarity measure the weakness of the connection between objects in different clusters rather the strength of the connection of objects within a cluster.

Divisive algorithms start with one cluster of all objects and recursively split up the cluster and subsequent clusters. Since this is a cutting operation, cut-based measures seem to be a natural choice. However, it is not necessary to use a cut-based measure with a divisive algorithm.

There are many cut-based similarity measures for clusters. We looked at one:

Let U be the set of all objects, i.e. U = {v1, …, vn}. For any clustering C1, C2, … Ck of the objects, $U = \bigcup_{i=1, \ldots, k} Ci$. Define:

$$\text{cutcost}(Cp, U\text{-}Cp) = \sum_{(vi \text{ in } Cp,\ vj \text{ in } U\text{-}Cp)} \text{sim}(vi, vj).$$

$$\text{intracost}(Cp) = \sum_{(vi,\ vj \text{ in } Cp)} \text{sim}(vi, vj).$$

Then the cost of a clustering C1, … Ck is defined to be:

$$\text{cost}(C1, \ldots, Ck) = \sum_{p=1,\ldots,k} \left(\text{cutcost}(Cp, U\text{-}Cp) / \text{intracost}(Cp) \right)$$

For this measure, the cost of a cluster measures the total similarity to objects outside the cluster (its cutcost) relative to the total similarity among items in the cluster. Finding a clustering that minimizes cost (C1, … , Ck) is called *mi- max cut* clustering.

An iterative improvement algorithm to heuristically find a clustering with low min-max cut is as follows:

```
Choose initial partition C1, … , Ck
repeat {
        unlock all vertices
        repeat {
                choose some Ci at random
                choose an unlocked vertex vj in Ci
                if moving vj to any other cluster improves min-max cut cost
                        move vertex vj to cluster giving best improvement
                "lock" vertex vj
        } until all vertices locked
}until converge
```

Convergence is usually determined by improvement being smaller than some chosen threshold between executions of the inner "repeat" loop. Vertex "locking" just insures that all vertices are examined before examining any vertex twice. There are many variations on this iterative improvement algorithm.

This iterative improvement algorithm shares several properties with the k-means algorithm: the number of clusters, k, is chosen in advance; an initial clustering is chosen (possibly at random); iterative improvement is used to modify a clustering to a better clustering. The most important difference is that the min-max cut cost is minimized rather than the sum of the squares of the distances from vertices to their cluster centroids.