Problem Set 4

Problem 1. Consider agglomerative clustering on n points on a line. Explain how you could avoid n^3 distance computations. How many distance computations does your scheme use?

Answer. One algorithm that operates in $nlog(n)$ is the following. Construct a heap to keep track of the $n-1$ intervals induced by the n points. Also construct an index of pointers for each interval, that provides the left and right neighboring intervals. In each iteration:

1. extract the min-sized interval from the heap
2. use the pointers mapping to neighboring intervals to update their endpoints
3. reheapify

Building the heap and the index takes $O(n)$ time. In the loop, there are n steps, each of which takes at most $O(log(n))$ steps, so the algorithm has a total running time of $O(nlog(n))$.

Problem 2. Consider a set of documents that we wish to group into k clusters. Among all possible k-clusterings of these documents, does agglomerative clustering produce the highest Value clusters? (Refer to slide 29 of lecture 8 for the definition of "Value" of a clustering)

Answer. Notice that as per the definitions in slide 29 of lecture 8, for a fixed k (i.e., for a given number of clusters), the value of a k-clustering is maximized when the total benefits of the clusters is maximized. We will now describe a simple counter-example that demonstrates that agglomerative clustering does not always produce a k-clustering with maximum total benefit.

Consider the set $\{1, 2, 3, 7, 8, 9, 16\}$ of 7 documents in 1-d space that we wish to partition into $k = 2$ clusters. For the purposes of this example, we will define the similarity between two documents as the negative of the absolute distance between them[1]. Now, applying agglomerative clustering, it is easy to see that for $k = 2$ we will end up with the clusters $\{1, 2, 3, 7, 8, 9\}$ and $\{16\}$ with a total benefit of -18.[2] On the other hand, the 2-cluster $\{1, 2, 3\}$, $\{7, 8, 9, 16\}$ results in a higher total benefit of -14, thus providing the necessary counter-example.

Problem 3. Suppose you have n points in d dimensions with each point labeled either red or green. How large does n need to be (as a function of d) in order to create an example with the red and green points not linearly separable? (Assume that the set of n points are *general*, i.e., every subset of d points from the set is linearly independent.)

Answer. You need at least d+2 points.

```
First we provide d+2 points that are not linearly separable, for any d.
d+1 red points:
(0,0,0,...,0)
(1,0,0,...,0)
(0,1,0,...,0)
...
```

[1] Note that the same idea can be extended to accommodate cosine similarity as well.

[2] Benefits are computed by summing the similarities between each document and the centroid of the cluster to which it belongs.

```
(0,0,0,...,1)
1 green point:
1/d * (1,1,1,...,1)

Next we show every set of d+1 points are linearly separable.

Any hyperplane that satisfies the following is a separator:
<w,x_i> + b < 0
<w,x_j> + b > 0

where w is a vector of variables of length d, and each x is a vector of
length d, where i iterates over the green pts, j over the red pts.
You can rewrite the above as

<w,x_i> + b = 1
<w,x_j> + b = -1

Now shift the axes so that x_1 is the new origin.  We are left with
b = 1
<w,x_i> = 0 (for all green pts except i = 1
<w,x_j> = -2
```

Treating b as the constant 1, we are left with d linearly independent equations and d unknowns, which is guaranteed to have a solution. Thus we can construct a hyperplane separating the green and red points.

Problem 4. This problem explores assigning labels to clusters. Assume we have generated clusters hierarchically, and we are currently assigning labels to each of 3 subclusters within a computer-related cluster. Given below are some components of the centroid vector of each subcluster (columns represent centroid vectors, rows represent the term axes). Generate a labelling for each centroid by first reweighting each of its components by multiplying the term weight by the Inverse Cluster Frequency of the term. In other words, for each term dimension, multiply the term weight by $log\frac{N}{n}$, where N is the number of subclusters in the current parent cluster, and n is the number of subclusters in which the term has been assigned a nonzero term weight. Then choose the two terms with the highest weight in each centroid to be the label for the cluster corresponding to that centroid. What are the labels? What would the labelling have been if we hadn't reweigted with ICF?

$$\begin{pmatrix} & CentroidA & CentroidB & CentroidC \\ computer & 4 & 3 & 3 \\ disk & 3 & 0 & 0 \\ spreadsheet & 0 & 1 & 2 \\ java & 0 & 3 & 1 \\ API & 0 & 4 & 0 \\ CPU & 2 & 2 & 1 \\ drives & 2 & 0 & 0 \end{pmatrix}$$

Answer. If we don't reweight with ICF, the labels are A='computer disk', B='API java' or 'API computer', and C='computer spreadsheet'. The ICF weights are

- $ICF(computer) = log(3/3) = 0$

- $ICF(disk) = log(3/1) = 1.09$
- $ICF(spreadsheet) = .405$
- $ICF(java) = .405$
- $ICF(API) = 1.09$
- $ICF(CPU) = 0$
- $ICF(drives) = 1.09$

Applying these weights, the new labelling would be A='disk drives', B='API Java', C='spreadsheet Java'.