Matthew Piatetsky WDD Disagreement Distance CS 565

Clustering is a useful tool for creating groups of similar data. It is also useful to compare different clusterings of data and determine their similarity, or their disagreement distance. However, clusters can have many dimensions and disagreement distance can be determined in a multitude of different ways. Different disagreement distance functions can be useful for different tasks. It is important to know what you want to determine about a pair of clusterings when designing a disagreement measure.

One possible way to examine two clusterings is to look at the aspect of membership. Let our two possible clusterings be clustering A and clustering B. One can compare whether both A and B place any given pair of points in the same cluster, or in different clusters. If A and B place every pair of points the same way, one could say that A and B are identical in terms of membership. For example, if you had clusterings which say which patients have received certain tests, you might only be interested in checking whether both A and B put each patient in the same cluster. On the other hand, if you wanted to know the degree to which each patient is at risk of getting a disease, concepts of distance become more important. Thus, a membership disagreement distance could lose potentially useful information, especially if two clusterings have similar memberships but different distances between points.

My first step in devising my own measure was to look for a way to incorporate not only membership disagreement, but also distance, into a disagreement measure. My first instinct was to compare the distance between each point and every other point. This would essentially be a pairwise Euclidean distance.[1] $PairwiseEuclideanDistance(A, B) = \sqrt{\frac{\sum_i^A \sum_j^B (A_i - B_j)^2}{|A| \times |B|}}$ However, since this would be comparing every point's distance, it would need to be normalized to be a metric, as in the formula below. $NormalizedPairwiseEuclideanDistance(A, B) = abs(\frac{2*ED(A,B)-ED(A,A)-ED(B,B)}{2})$ This measure would better capture degrees of disagreement, especially among clusters with similar membership. However, this measure has some problems as well. For one, such a measure could be impractical to compute with very large datasets. With many fields and many points, computation could be too slow, since comparing each point to every other point would make the algorithm O(n^2). Plus, implementing this measure would be much more difficult than implementing the previous measure. The next step for me was to find a measure that would be simpler in terms of computations and implementation.

I compromised on a measure based on centroid distances. I determined that I could find the centroid for each field and sum the Euclidean distance between centroids. This would be much simpler and faster to compute than pairwise distances, and would still retain information related to distances between points. This measure would likely lose information related to the dispersion of points within a cluster, since it only uses the centroid, but it still has advantages over a pairwise distance measure. One more modification to consider is that we care more about centroid distances where two clusters disagree. On the other hand, we don't care about distances when two clusters agree completely. Thus, the centroid distance can be weighted based on disagreements where disagreements represent the

[1] Note: The formulas below use the following symbols for $|Cardinality|$ and $\overline{Average}$

fraction of points that are different between clusters. This makes this measure better reflect the actual disagreement between two clusterings.

$$WeightedDisagreementDistance(A,B) = \frac{|Disagreements|}{|Points|} * \sqrt{\sum_{i=1}^{|Fields|} (\overline{Field(A)} - \overline{Field(B)})^2}$$

In order to implement this measure in an algorithm, I needed to determine how I would address clusterings with different numbers of clusters. A simple and reasonable way to address this would be to map each cluster to a similar cluster in terms of membership and then compute my disagreement measure. This way my algorithm would work for any number of clusters. In addition, such a mapping would make logical sense, since if a larger cluster was split into two, it would end up being compared to the two smaller clusters which most likely map to it.

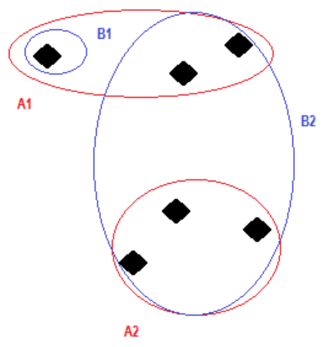
I determined the following basic pseudocode for my algorithm. 1. Read in data file with points, and two files with cluster assignments. 2. Pass through points and compute sums of each field. 3. Compute membership counts for each cluster. 4. For each cluster in clustering1 find the best match in clustering2. 5. Compute weighted cluster disagreement distance.

This algorithm is O(n) in terms of running time and storage space. In terms of running time, I can pass once through all the points once which would be O(n). Here n is the number of points, multiplied by the number of fields, since each point is a vector with a value for each field. Thus, one could say the running time for the one pass is O(points*fields), but this section is still linear, since it passes through each value once. Sums can be computed as the algorithm passes through the data, and averages can be computed later once the correct mapping has been determined. Thus, it is necessary to store enough data for each centroid. There would be a centroid for each field in each cluster, so this section would require O(field*cluster) storage space, which would be even less than the O(fields*points) running time. The rest of the algorithm, which involves determining the correct mapping of clusters, is O(n^2) since each cluster is being compared to each other cluster. However, here the n is only the number of clusters, which should generally be much smaller than the number of points. Thus, the dominant section of the algorithm will be O(n) in terms of running time, and even less than that in terms of storage space. This is very good since clusterings of very large datasets can have lots of points and an n^2 algorithm would be significantly slower than this algorithm which has efficient storage and is close to linear time.

One potential flaw of this algorithm is that it is not a metric; however, it still behaves like a metric in the average case. The algorithm always satisfies non-negativity, since it is a modified Euclidean distance, and satisfies the triangular inequality for the same reason. The algorithm also satisfies isolation due to the weight on the average, since if the clusterings are the same, the disagreement weight will be 0. However, in the worst case, the algorithm does not satisfy symmetry. We can imagine the following membership matrix. In this situation, A1 will be mapped to B2, and A2 will be mapped to B2. On the other hand, B1 will be mapped to A1, and B2 will be mapped to A2. Thus, in this case the measure is not symmetric. If it was necessary to have a measure that was symmetric, one could simply modify the distance to be (WDD(A,B)+WDD(B,A))/2. However, this should not even be necessary because in the average case, the measure will behave like a metric. Note that this

Clusters	B1	B2
A1	1	2
A2	0	3

example is artificially constructed. A simple clustering algorithm like k-means (the one I used for my input) would probably not create clusters like B1 and B2. A1 and A2 are far more natural clusters, and with reasonable clustering, this measure would behave symmetrically the large majority of the time.



This algorithm is fairly simple and efficient, and the measure it uses can have many applications. One potentially very useful application is related to a concept we discussed in class – finding the ideal number of clusters (i.e. finding the k for k-means, k-medoid or a similar problem). One way to potentially address this problem is to find a number of clusters where the clusterings are most stable. One can repeatedly cluster a dataset with different seeds, and use this disagreement measure to compare clusterings. If you eventually find a number of clusters where the clusterings remain mostly the same, even with different seeds, then it is likely that you have found the ideal k/number of clusters.

I performed several experiments on this data which yielded some interesting results.[2] I compared my WDD measure against membership disagreements, for both datasets. I noted, as below, that my measure preserved the same relative ordering as the measure based on membership, but the values were perhaps more precise in terms of distance, which could be useful when dealing with larger numbers of clusters/clusterings. I confirmed that my measure satisfied isolation and non-negativity and that it accurately assigned lower values to more similar clusterings than more distant ones. I further noted that my measure can in fact be used to measure clustering stability as described above. Below are the tables for several different clusterings (A,B,C…) of the wine data for both 3 and 5 clusters respectively. When comparing the clusterings, one can see that my measure shows that clusterings of 3 clusters are far more similar than clusterings of 5 clusters for the wine data. This suggests that 3 is a better k to use for k means; indicating that the data probably describes 3 types of wines. For the communities' dataset, I noted that both 3 and 5 seemed to be stable cluster numbers, suggesting that there was more than one legitimate k.

mdis	A	B	C	D
A	0	2	7	7
B		0	9	9
C			0	0
D				0

WDD	A	B	C	D
A	0	0.0416	0.1409	0.1409
B		0	0.1791	0.1791
C			0	0
D				0

mdis	A	B	C
A	0	54	30
B		0	37
C			0

WDD	A	B	C
A	0	0.547	0.379
B		0	0.4466
C			0

In conclusion, I developed a measure and an algorithm which can convey the disagreement between clusterings. The algorithm is efficient in terms of running time and storage space and it behaves like a metric in the average case. This algorithm can also have practical applications in determining the ideal number of clusters for problems such as k-means clustering. The measure also behaves similarly to a membership disagreement count, but differs when distances are important and when some fields need to be weighted more than others.

[2] The tables with the data for most of these experiments can be found in the last sheet of the files wnorm-matrix.xslx and comm-matrix.xslx