Deterministic Assimilation Clustering

Andrew C Lea MA(Cantab)[1]

Abstract Clustering is an important and effective means of knowledge discovery. A key characteristic of a dataset may be the number of clusters present, but many techniques require this as a parameter, rather than yielding it as a result. Additionally, many effective techniques rely on initial random seeding, and are therefore non-deterministic. This paper introduces a new clustering algorithm – Assimilation Clustering - which is both deterministic and yields the number of clusters in the dataset.

1 Introduction to Clustering

Data clustering[1] is an important and effective means of knowledge discovery[2], and is a form of unsupervised learning. This section seeks to define clustering, and identify the problem that deterministic clustering seeks to address.

Clustering is only partly defined, in that what constitutes a cluster can itself have multiple definitions. When we ask an algorithm[3] to cluster data points, we are in effect asking it to cluster those data points in a way which our human, animal, or insect[4] pattern-recognition[5] has evolved to regard as clusters. Such clusters are likely to reflect an underlying reality, in that pattern recognition which recognises 'real' patterns confers an evolutionary advantage.

The **ideal clustering algorithm** would include these (and other) characteristics:

1. **deterministic**, and yield the same clusters on the same input data. Many clustering 'algorithms' (such as k-means) are heuristics, as the same input does not always yield the same output, depending on the initialisation[6].
2. **optimal**, in that rather than just find a local minimum (or set of clusters), it would find the global minimum (or 'best' set of clusters), and always therefore return the same set of clusters. Whilst an optimal clustering algorithm would imply determinism, a deterministic algorithm would not necessary be optimal.

[1]Primary Key Associates Ltd, andrew.lea@scientific.co.uk

3. **informative** and determine the number of clusters present from the data. Many techniques require this as an input, when it is normally unknown, and would be one of the most important facts we would like to determine.
4. **fast**, and have an algorithmic growth of Order[7] (N log N) or better. In some cases speed can be increased using parallel data processing[8].
5. **visually plausible**, and not appear counter-intuitive.

In practice, these requirements appear to be mutually exclusive. Two popular forms of clustering, in widespread industrial use, are:

1. **hierarchical clustering** [9], starts with each point as a separate cluster, and successively joins the two closest clusters, yielding a hierarchy. Visually effective and often highly informative, the hierarchical clusters are not distinct. It cannot be said, therefore, that *this* point and *that* point are in the same or different clusters. Though deterministic, hierarchical clustering but does not determine the number of clusters.
2. **k-means clustering** [10], in which each data point is initially assigned to one of k random clusters. The membership of each cluster is adjusted in successive iterations until the a local minimum is reached, when each point is a member of its closest cluster. k-means is fast, but not deterministic: the outcome depends on the initial random assignment.

Research Objective, Observations, and Algorithm Overview

This research sought to design an algorithm which is deterministic (criterion 1) and determines the number of clusters present (criterion 3).

This section discusses the algorithm developed, starting with key observations on which it depends, and then giving an overview of the algorithm, which is a type of agglomerative clustering algorithm.

This clustering algorithm uses the observation that not all points in a sample are necessarily part of a cluster. This is, in effect, an unspoken assumption in most clustering algorithms: they assume that *every* data point is part of a cluster.

Specifically, for a dataset of N points there could be anywhere between 0 to N clusters. It is entirely valid for a clustering algorithm processing N points to return L clusters, containing M points, with a remainder of N-M points which are *not part of any cluster*.

Secondly, clusters contain data points and therefore cover an area, even if it is convenient to represent that cluster as having a single centroid. Such a representation, however, hides the fact that sometimes different points in the same cluster may 'prefer' to be merged with different other clusters.

Specifically, **two clusters** – a smaller (A) and larger (B) - **should not be merged if a significant number of points in the smaller cluster** (A) **'belong' in a different partner cluster** (not B).

These observations are enabling principles of this new algorithm, which means that it does not have to join every singleton cluster or small cluster into larger clusters, although generally it does.

Initially, each point is a singleton cluster. Assimilation clustering progressively merges clusters, ordered with nearest first, to existing clusters, until no merges are possible, because any merge would imply a ***contradiction or significant ambiguity***, where sufficient points in the smaller of the clusters to be merged do not belong to the other, but would 'prefer' to be in a different cluster.

It is this progressive assimilation of smaller clusters into larger until a contradiction arises which gives the algorithm its name[2].

When complete, the algorithm will return C clusters, with R points remaining, which are not in any cluster (or are regarded as singletons). If there are N points, then (C + R) <= N.

Algorithm Implementation

The algorithm and a comparative k-means was implemented in Python. For purposes of evaluation, repeatable random test data was generated.

A **node** is a tuple of x and y position, ie (x,y); a **cluster** is a list of nodes; and the global **Clusters** is a dictionary of clusters. For example, a Python dictionary of two clusters, the first with a single node, and the second with two, would be represented as: Clusters = { 0: [(4,10)], 1: [(21,19), (23,17)] }

The **test data generator** returns a list of data nodes, using two possible models. If the model is “random” they are an evenly distributed set of nodes. If the model is “lumpy”, the nodes are distributed around a set of random centres and merged with a smaller set of random nodes, in order to produce noisy degraded clusters of nodes. By resetting the random seed the same 'random' dataset can be used in repeat tests. (Assimilation Clustering does not itself use random numbers.)

The clustering algorithm uses **metrics** which return the distance between two nodes, the distance between two clusters (the distance between the two nearest points in separate clusters), and the number of singleton clusters.

---

2 Star Trek fans may prefer to think of this as the 'Borg Algorithm'.

The clustering algorithm uses three core support routines, an initialisation routine, and the main clustering routine. The **support routines** are:

1. **ListClusterPairsInClosenessOrder()** lists pairs of clusters, such that the closest clusters are listed first, and without repetition.
2. **AllNodesExceptClusters(i, j)** yields all nodes in the dataset *except* those in cluster i or cluster j.
3. **CheckAffinity(shorter, longer)** checks that all the nodes in the shorter cluster do indeed belong in the longer cluster. In other words, it returns false if more than ***tolerance*** (typically 0 to 3) nodes in the shorter cluster would be better placed in a different cluster to the longer cluster.

```
def CheckAffinity(shorter, longer, tolerance):
 "Checks all nodes in candidate pair of clusters belong together"
 shorterCluster = Clusters[shorter]
 longerCluster  = Clusters[longer]
 objections = 0
 # Consider all the nodes in the shorter candidate cluster
 for nodeI in shorterCluster:
     # Find the distance to the nearest node in other cluster
     distToNearestPartnerNode = min(NodeDist(nodeI, nodeJ)
                                    for nodeJ in longerCluster)
     # Are there ANY other nodes, not in this pair,
     # which are nearer to the node being considered?
     if any(NodeDist(nodeI, otherNode) < distToNearestPartnerNode
       for otherNode in AllNodesExceptClusters(shorter, longer)):
         if objections > tolerance:
             return False
         else:
             objections += 1
     # ok to merge so long as there are not too many objections
     return objections <= tolerance
```

Clustering is initialised with InitialAssimilationClusters, which simply assigns each node to its own cluster:

ClusterRound() performs one round of clustering, and is called repeatedly until it can no longer merge clusters. It looks at all possible pairs of clusters (closest first) and if they belong together, merges them. The ***tolerance*** parameter is simply passed through to CheckAffinity. If ***strict***, then equal length clusters must *both* belong with each other, otherwise only one must belong with the other. Experiment suggests that ***strict*** is better as false. This routine exits after a merge since the cluster ordering must be re-calculated; and so that the drawing can occur.

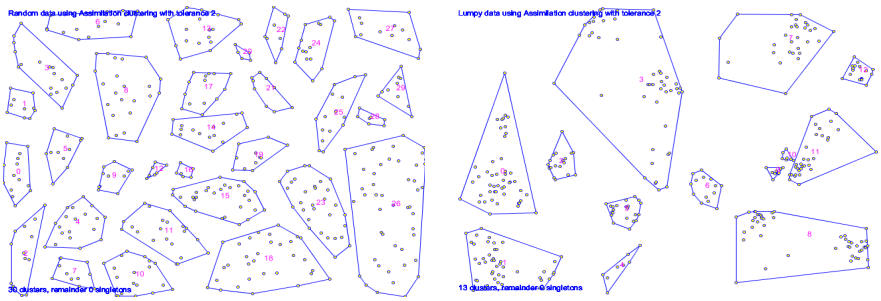
```
def ClusterRound(tolerance, strict):
    # Find shortest distance between any two clusters
    n = 0
    for d, i,j in ListClusterPairsInClosenessOrder():
        n += 1
        if len(Clusters[i]) < len(Clusters[j]):
            ok = CheckAffinity(i, j, tolerance)
```

```
        elif len(Clusters[i]) > len(Clusters[j]):
            ok = CheckAffinity(j, i, tolerance)
        else: # equal length clusters...
            if strict:
                ok = CheckAffinity(i, j, tolerance) and
                 CheckAffinity(j, i, tolerance)
            else:
                ok = CheckAffinity(i, j, tolerance) or
                 CheckAffinity(j, i, tolerance)
        if ok:
            MergeClusters(i, j)
            return False
    return True
```

Finally, the **supervisor** sets up the test data, and keeps calling the main clustering routine until it has finished, drawing a new frame each iteration.

Results

The results of Assimilation Clustering with tolerances of 2 against both random and lumpy data fields are shown below.



Conclusions

Compared to the initial five criteria, Assimilation Clustering is:

1. **deterministic**, and yields the same clusters on the same input data.
2. **neither optimal nor sub-optimal**, in that it is not seeking a minimum.
3. **informative** and determines the number of clusters present from the data.
4. **slow**, although an optimised fast version has not been ruled out.
5. **visually plausible**, to the author at least.

Further Work

Before Assimilation Clustering is ready for industrial deployment, further work would be desirable:

1. Assimilation Clustering should be compared with other techniques to characterise the scenarios in which it produces useful output.
2. Mathematical examination of the validity of the clusters produced.
3. Alternative interpretations of tolerance should be explored. In this paper an absolute tolerance is used, but a proportional or even adaptive tolerance could be used instead.
4. An optimised version of Assimilation Clustering should be written, and its Order of execution determined. Obvious optimisations include (i) incrementally adjusting the ordered list of cluster distances and (ii) when checking affinity, consider only nodes in nearby clusters.

References

1. Jain, Murty, and Flynn, "Data clustering: a review", 1999, ACM Computing Surveys (CSUR) vol 31 issue 3
2. Oded Maimon and Lior Rokach, "Data Mining and Knowledge Discovery Handbook", 2010 Springer ISBN: 978-0-387-09822-7 and 978-0-387-09823-4
3. Andeas Blass and Yuri Gurevich, "Algorithms: A Quest for Absolute Definitions", 2003, Bulletin of European Association for Theoretical Computer Science
4. M.V. Srinivasan, "Pattern recognition in the honeybee: Recent progress", 1993, Journal of Insect Physiology, Volume 40, Issue 3, Elsevier
5. Jain, Duin, Mao, "Statistical pattern recognition: a review", 2000, Pattern Analysis and Machine Intelligence, IEEE Transactions on (Volume:22 , Issue: 1)
6. J.M Peña1, J.A Lozano, P Larrañaga, "An empirical comparison of four initialization methods for the K-Means algorithm", 1998, Pattern Recognition Letters, Volume 20, Issue 10, Elsevier
7. Donald Knuth. "The Art of Computer Programming", Volume 1: Fundamental Algorithms, Third Edition. Addison–Wesley, 1997. ISBN 0-201-89683-4.
8. Anusha Vasudevan, M. Swetha, H. Hyba, G. Rajiv Suresh Kumar, "Map-Reduce Based High Performance Clustering On Large Scale Dataset Using Parallel Data Processing", 2014, Data Mining and Knowledge Engineering 6(5).
9. Segaran, "Programming Collective Intelligence", O'Reilly, ISBN 978-0-596-52932-1
10. Hastie, Tibshirani, Friedman, "The Elements of Statistical Learning: Data Mining, Inference, and Prediction", 2009, Springer