Fuzzy Agglomerative Clustering

Michal Konkol

Natural Language Processing Group
Department of Computer Science and Engineering
University of West Bohemia
Univerzitni 8
306 14 Plzen
Czech Republic
nlp.kiv.zcu.cz
konkol@kiv.zcu.cz

Abstract. In this paper, we describe *fuzzy agglomerative clustering*, a brand new fuzzy clustering algorithm. The basic idea of the proposed algorithm is based on the well-known hierarchical clustering methods. To achieve the soft or fuzzy output of the hierarchical clustering, we combine the single-linkage and complete-linkage strategy together with a *fuzzy distance*. As the algorithm was created recently, we cover only some basic experiments on synthetic data to show some properties of the algorithm. The reference implementation is freely available.

Keywords: Hierarchical Clustering, Fuzzy Clustering, Agglomerative Clustering.

1 Introduction

Clustering algorithms became a very important tool for many fields including biology [1], marketing [2], or natural language processing [3]. They simply analyse the input data and create groups of similar objects. There are two basic types of clustering algorithms: hard and soft. The hard clustering algorithms are older and allow each object to be part of only one cluster. The soft clustering algorithms allow each object to be part of multiple clusters. The fuzzy clustering algorithms have the same output as soft clustering, but also include a measure of membership of objects in the clusters. Very often the terms soft and fuzzy are interchangeable. The fuzzy clustering seems to be more natural as the world is usually not black and white.

Two of the most common (if not the most common) hard clustering algorithms are k-means and various types of hierarchical clustering, such as single-linkage or complete-linkage. While the fuzzy counterpart of k-means is studied by many authors, the fuzzy hierarchical clustering remains almost unnoticed.

In this paper we propose *fuzzy agglomerative clustering*, which is a novel algorithm for fuzzy clustering based on hierarchical clustering principles. The fuzzy output of the algorithm is achieved through a combination of single-linkage and complete-linkage strategies in one algorithm together with a *fuzzy distance*. The fuzzy distance is used for points that are already included in a cluster and thus cannot be added to another cluster in a standard (hard) hierarchical clustering.

The rest of the paper is organized as follows. We summarize the related work in section 2. The detailed description of our algorithm is given in section 3. The experiments are covered in section 4. The last section summarizes our contribution and shows some paths of our future research.

2 Related Work

There are multiple types of algorithms referred to as hierarchical fuzzy clustering. The first type [4] are standard hard hierarchical algorithms extended for use on fuzzy data.

The second type [5–7] is based on another fuzzy clustering algorithm, which generates a high number of fuzzy clusters. These clusters are then merged using standard hierarchical clustering. Very often the fuzzy clusters are generated using fuzzy c means with a desired number of clusters much higher then the expected number of clusters in the data.

The last type are hierarchical clustering algorithms adapted to produce fuzzy clusters. Our algorithm belongs to this group. We are only aware of one (other) algorithm of this type [8]. This algorithm uses a very different approach and the results should be different from our algorithm.

3 Proposed Algorithm

First, we would like to introduce our mathematical notation and definition of the task. We follow with outline of some of the problems connected with fuzzy hierarchical clustering and present the basic idea behind our solution. Then we give an in-depth description of our algorithm.

The input of a clustering algorithm is a set of points $P = \{p_i \in \mathbb{R}^k | i \in \mathbb{N} \leq N\}$, where N is the number of data objects represented as k-dimensional vectors. The output is a set of clusters $C = \{C_i \subseteq P | i \in \mathbb{N} \leq M\}$, where M is the number of clusters in the output and each cluster C_i is a set of points. The clusters created by a standard hard clustering algorithm are partition of input points, i.e. $\bigcup C_i = P$, $C_i \cap C_j = \emptyset$ for $\forall i, j$. For fuzzy clustering $\bigcup C_i = P$, but the second property $C_i \cap C_j = \emptyset$ for $\forall i, j$ is not mandatory. The output of fuzzy clustering algorithm can be enhanced with membership values $\mu(p, C_i)\ \forall p \in C_i$, which determines how likely (or how much) the point belongs to a cluster. The indices used for points and clusters (e.g. i in C_i) are used only to distinguish different clusters and do not imply any order.

We use $d_{a,b}$ to denote distance or dissimilarity between a and b, where a (resp. b) can be both a point or a cluster. Further in the paper, we use only the term distance even though we do not need the triangle inequality property of distance metrics.

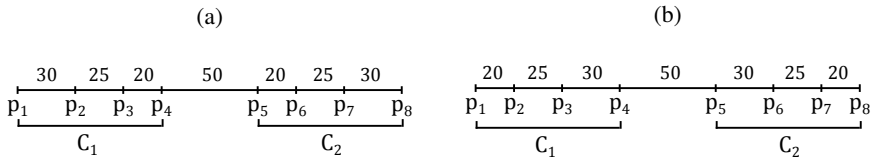
We use the symbol $=$ to denote both equality and assignment and the particular meaning should be clear from the context.

3.1 The basic idea

If we want to go from hard hierarchical clustering to fuzzy hierarchical clustering, many problems arise. We will focus on these problems and later we will describe our idea how to solve them.

Imagine we are clustering points in $\mathbb{R}^1$ using Euclidean distance and we are currently in situation (a) or (b) depicted on Fig. 1. In both situations, we have eight points $p_1, \ldots, p_8$ and two clusters $C_1 = \{p_1, p_2, p_3, p_4\}$, $C_2 = \{p_5, p_6, p_7, p_8\}$. The numbers on the top of the line sections are the distances between neighbouring points. This particular situation can arise using various strategies for the hard hierarchical clustering – *single-*, *complete-* or *average-linkage*. Now, if we want to have a fuzzy clustering we would (in some cases) like to have point p_5 simultaneously in clusters C_1 and C_2, i.e. we want to add point p_5 to cluster C_1.

Fig. 1: Example.



Now consider that we are using single-linkage. The distance d_{p_5,C_1} is equal to d_{C_1,C_2}. The problem is, that we need to decide, if we want to add point p_5 to cluster C_1 or if we want to merge clusters C_1 and C_2. The later is hard clustering. If we choose the former, in situation (a) we will gradually add points p_6, p_7, p_8 to C_1. In situation (b) have two options. We can update the distances between clusters as usually, which leads to the same situation as in (a). Or we can ignore the change of distances between clusters when adding points that already belong to some cluster and we will merge clusters C_1 and C_2 right in the next step. In all cases, we get C_1 and C_2 merged. There is no possibility to control the *fuzziness*, i.e. how many points should be added.

With the complete-linkage strategy $d_{p_5,C_1} \neq d_{C_1,C_2}$, but if we choose to add point p_5 to C_1, we will gradually add points p_6, p_7, p_8 to C_1 again, in both situations.

Similar problems are common for all hierarchical clustering algorithms as far as we know.

We solve these problems by combining single-linkage and complete-linkage in one algorithm. We use complete-linkage for distances between clusters (e.g. d_{C_1,C_2}). We use the single-linkage for distances between points and clusters (e.g. d_{p_5,C_1} or d_{p_1,p_2}). This is still not enough to solve these problems, but opens some possibilities. We propose a measure of membership $\mu(p_i, C_j)$ which measures the relatedness between cluster C_j and point p_i. The details will be covered in the next section. The important property is, that the membership of points decreases as we gradually add points, e.g. $\mu(p_5, C_1)$ could be 0.4, $\mu(p_6, C_1)$ could be 0.25, etc. We use this function to gradually increase distance between points, so d_{p_7,C_1} could become greater than d_{C_1,C_2} even though d_{C_1,C_2} is acquired with the complete-linkage strategy. To distinguish these altered distances, we call them *fuzzy distances* in this paper. Our approach allows parametrization which controls fuzziness.

We believe it is possible to switch from single-linkage to some other strategy, e.g. average-linkage. We use the single-linkage, because it is the most intuitive strategy.

3.2 The algorithm

At the beginning, we start with $C = \emptyset$. We will also need a set of points U, that are already in a cluster, i.e. $U = \{p_j \in P | \exists i \in \mathbb{N} \leq |C| : p_j \in C_i \in C\}$. We use $\bar{U} = P \setminus U$ for unused points. The $|C|$ denotes the current number of clusters during the algorithm. It is 0 at the beginning and M at the end.

We remember all distances between points $d_{p_i,p_j}, p_i, p_j \in P$, between points and clusters $d_{p_i,C_j}, p_i \in P, C_j \in C$, and between clusters $d_{C_i,C_j}, C_i, C_j \in C$. It is necessary to define a set of usable distances D, because we want to skip specific distances, e.g. distance d_{p_i,C_j} if $p_i \in C_j$. We finish the initialization by computing distances between points $d_{p_i,p_j}, p_i, p_j \in P, i \neq j$ and adding them to D. We assume that for each distance $d_{a,b} \in D$ we also know the a and b. The set D can be seen as distance matrix, where we use only some cells.

The algorithm loops until it converges. The loop consists of three steps – find minimum value $d_{a,b}$ in D, merge corresponding elements a and b (points and/or clusters), recompute distances where necessary. The first step in the loop, finding the minimum value of D, is straightforward. The interesting part is the second and third step. We distinguish four cases based on the minimum value $d_{a,b} = \min D$.

- merging two points, i.e. $a, b \in P$
- merging unused point and cluster, i.e. $a \in \bar{U}, b \in C$ (or $a \in C, b \in \bar{U}$)
- merging used point and cluster, i.e. $a \in U, b \in C$ (or $a \in C, b \in U$)
- merging clusters, i.e. $a, b \in C$

We cover these cases in the following subsections.

3.3 Merging two points

The minimal distance in D is $d_{a,b} = \min D$, where $a, b \in P$. Firstly, update the set of usable distances D. We do not want to merge the same points again, so we set $D = D \setminus \{d_{a,b}, d_{b,a}\}$.

We (optionally) remove combinations of a and b with other points from D. We do this because of implementation reasons as it is faster to find the minimum of D and merging a (resp. b) with some other point will usually not change the result. Mathematically, we set $D = D \setminus \{d_{x,p} \in D | (x = a \vee x = b) \wedge p \in P\}$. We mark the points a and b as used points, i.e. $U = U \cup \{a, b\}$, and create new cluster $C_{new} = \{a, b\}$.

Now we need to compute some new distances. We start with unused points $p \in \bar{U}$. For all these points we compute the distance using (1) and add this distance to D, $D = D \cup \{d_{p,C_{new}}\}$. Then we follow with the distances between our new cluster C_{new} and all the other clusters $C_{other} \in C$ using (2) and we again add this distance to D, $D = D \cup \{d_{C_{other},C_{new}}\}$. We also define distances $d_{a,C_{new}} = d_{a,b}$ and $d_{b,C_{new}} = d_{a,b}$, but we do not add these distances to D, because C_{new} already contains a and b.

$$d_{p,C_{new}} = \min\{d_{p,a}, d_{p,b}\} \tag{1}$$

$$d_{C_{other},C_{new}} = \max\{d_{C_{other},a}, d_{C_{other},b}\} \tag{2}$$

Then we need to compute the fuzzy distances. First, we need to define function $d_{min}(u)$ by (3), where $u \in U \subset P \wedge u \neq a \wedge u \neq b$. The membership $\mu(u, C_{new})$ of used point u in cluster C_{new} is defined by (4). The distance $d_{u,C_{new}}$ is defined by (5). The $t(\mu)$ function is defined by (6), where m is the fuzziness parameter. As the algorithm picks shortest distances first $d_{min}(u) < \min\{d_{u,a}, d_{u,b}\}$ and $\mu(u, C_{new}) \in [0, 0.5]$ and $t(\mu) > 1$. As $m \to \infty$ all points are added to all clusters with uniform membership. As $m \to 1$ the algorithm becomes a hard clustering. We have chosen the $t(\mu)$ function arbitrary, because of its desirable properties, e.g. $t(\mu) > 1$ for $\mu \in [0, 1)$, $t(\mu) \to \infty$ if $\mu \to 0$, smooth function, etc. The function $t(\mu)$ is shown on Fig. 2 for some values of m.

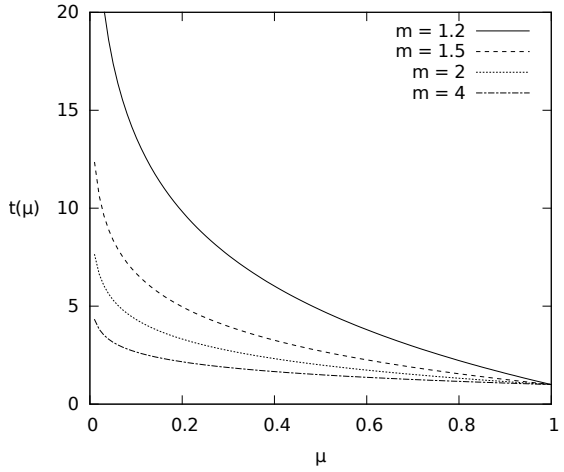
$$d_{min}(u) = \min\{d_{u,C_i} | u \in C_i \in C\} \tag{3}$$

$$\mu(u, C_{new}) = 1 - \frac{\min\{d_{u,a}, d_{u,b}\}}{\min\{d_{u,a}, d_{u,b}\} + d_{min}(u)} \tag{4}$$

$$d_{u,C_{new}} = t(\mu(u, C_{new})) \min\{d_{u,a}, d_{u,b}\} \tag{5}$$

$$t(\mu) = 1 - \log_m(\mu) \tag{6}$$

Fig. 2: The $t(\mu)$ function for some values of fuzziness parameter m.



We have computed new fuzzy distances for $d_{u,C_{new}}$. But we need to compute fuzzy distances $d_{a,C_{other}}$ (resp. $d_{b,C_{other}}$) for all $C_{other} \in C, other \neq new$ using (7) and (8). Note that $d_{min}(a) = d_{a,b}$. At the end we add the new cluster to C, i.e. $C = C \cup C_{new}$.

$$\mu(a, C_{other}) = 1 - \frac{d_{a,C_{other}}}{d_{a,C_{other}} + d_{a,b}} \tag{7}$$

$$d_{a,C_{other}} = t(\mu(a, C_{other}))d_{a,C_{other}} \tag{8}$$

3.4 Merging unused point and cluster

In this situation $a \in \bar{U}$ and $b \in C$, we will mark $C_b = b$ to show that b is a cluster. We merge a and C_b, $C_b = C_b \cup \{a\}$. Then we follow the same steps as in the previous section, but with slightly different formulas. We remove a from D, $D = D \backslash \{d_{a,C_b}\}$ and also (optionally) $D = D \setminus \{d_{a,p} \in D | p \in P\}$. We add a to used points, $U = U \cup \{a\}$.

Then we update the distances d_{p,C_b} for all $p \in \bar{U}$ using (9) and d_{C_{other},C_b} for all $C_{other} \in C, other \neq b$.

$$d_{p,C_b} = \min\{d_{p,a}, d_{p,C_b}\} \tag{9}$$

$$d_{C_{other},C_b} = \max\{d_{a,C_{other}}, d_{C_b,C_{other}}\} \tag{10}$$

We follow with the fuzzy distances d_{u,C_b} for all $u \in U, u \neq a$. The corresponding formulas (11) and (12) are slightly changed compared to the previous section, but have the same logic. And finally, we need to update fuzzy distances $d_{a,C_{other}}$ using formulas (13) and (14).

$$\mu(u, a) = 1 - \frac{d_{u,a}}{d_{u,a} + d_{min}(u)} \tag{11}$$

$$d_{u,C_b} = \min\{t(\mu(u, a))d_{u,a}, d_{u,C_b}\} \tag{12}$$

$$\mu(a, C_{other}) = 1 - \frac{d_{a,C_{other}}}{d_{a,C_{other}} + d_{a,C_b}} \tag{13}$$

$$d_{a,C_{other}} = t(\mu(a, C_{other}))d_{a,C_{other}} \tag{14}$$

3.5 Merging used point and cluster

In this situation $a \in U \subset P$ and $b \in C$, we will mark $C_b = b$ to show that b is a cluster. Firstly, we add a to C_b, $C_b = C_b \cup \{a\}$. Then we recompute distances again, but only d_{u,C_b} where $u \in U, u \neq a$ using (15) and (16). We use a different definition of membership in (16) if compared to (12). Instead of using the membership corresponding to a, we use membership corresponding to C_b. We do this, because there is always (at least) one point $p \in U$, that has a property $d(p, a) = d_{min}(p)$. For such a point, the

membership would be always 0.5. As the algorithm would continue, we would have such point in each step (of this type) and that would be undesirable behavior.

$$\mu(u, C_b) = 1 - \frac{d_{u,C_b}}{d_{u,C_b} + d_{min}(u)} \tag{15}$$

$$d_{u,C_b} = \min\{t(\mu(u, C_b))d_{u,a}, d_{u,C_b}\} \tag{16}$$

3.6 Merging two clusters

This is the simplest situation. We have $a, b \in C$. To denote that a and b are clusters, we use $C_a = a$ and $C_b = b$. We simply recompute distances d_{p,C_a} using (17), where $p \in P$. We use (18) for distances d_{C_{other},C_a}, where $C_{other} \in C, other \neq a$. Then we merge the clusters, $C_a = C_a \cup C_b$, and remove C_b from C, $C = C \setminus \{C_b\}$.

$$d_{p,C_a} = \min\{d_{p,C_a}, d_{p,C_b}\} \tag{17}$$

$$d_{C_{other},C_a} = \max\{d_{C_{other},C_a}, d_{C_{other},C_b}\} \tag{18}$$

3.7 Convergence and Complexity

The convergence is guaranteed in the worst case in $\frac{N(N-1)}{2}(N-2) + N$ steps. The $\frac{N(N-1)}{2}$ is the maximum number of clusters, that can be possibly created, i.e. all possible pairs in a set of N elements. The maximum number of clusters is multiplied by $N-2$, because we can add $N-2$ points to each pair of points, i.e. to have all points in all clusters. The additional N is needed to merge the clusters, where each cluster consists of all points. The convergence is usually much faster.

We propose a rule, that each point can merge with another point, only if both points are not used in another cluster. This rule reduces the $\frac{N(N-1)}{2}$ to $\frac{N}{2}$. This rule can affect the results in some cases, but only a very little for reasonably large data.

3.8 Implementation

We have implemented the proposed algorithm in the Brainy machine learning library [9]. The reference implementation is based on a distance matrix, that stores all the necessary distances.

4 Experiments

We have only tested our algorithm on basic synthetic datasets. Because of limited space we present only one example of output produced by our algorithm. Fig. 3 shows two clusters randomly sampled from two Gaussian distributions. Points that belong to first cluster are marked by a circle. Points that belong to second cluster are marked by cross.

Crossed circles are points that belong to both clusters. You can see the output for various settings of fuzziness.

We are aware, that this experiment is not sufficient for any claims or proofs. We only hope it shows some potential of the proposed algorithm. We are going to do proper experiments to compare the proposed algorithm with other algorithms, but it is out of the scope of this preliminary study.

5 Discussion and Future Work

In this paper, we proposed a fuzzy agglomerative clustering. Since we came up with this algorithm very recently, we carried out only a few synthetic tests of the algorithm. They show some basic behavior of the algorithm, but an in-depth experimental study is yet to be done.

The algorithm have some desirable properties, which come from its hierarchical nature. It is possible to use plenty of various distance and dissimilarity metrics. It is possible to use multiple stopping conditions, so it is not necessary to know (or guess) the number of clusters. The output is deterministic. At the moment, it is hard to tell, if these properties outweigh some of the disadvantages and if the algorithm will be used in practice.

There are many different ways to improve the algorithm. One of them is the definition of the $t(\mu)$ function, which was chosen arbitrary without any deep study. Another interesting path can be to change the single-linkage strategy (e.g. for average-linkage). The implementation of the algorithm can also be optimized.

Acknowledgements

This work was supported by grant no. SGS-2013-029 Advanced computing and information systems, by the European Regional Development Fund (ERDF). Access to the MetaCentrum computing facilities provided under the program “Projects of Large Infrastructure for Research, Development, and Innovations” LM2010005, funded by the Ministry of Education, Youth, and Sports of the Czech Republic, is highly appreciated.

References

1. Legendre, P., Legendre, L.: Numerical ecology. Volume 24 of Developments in Environmental Modelling. Elsevier B.V., Amsterdam, The Netherlands (2012)
2. Russell, S., Lodwick, W.: Fuzzy clustering in data mining for telco database marketing campaigns. In: Fuzzy Information Processing Society, 1999. NAFIPS. 18th International Conference of the North American. (Jul 1999) 720–726
3. Brychcín, T., Konopík, M.: Semantic spaces for improving language modeling. Computer Speech & Language **28**(1) (2014) 192 – 209
4. GhasemiGol, M., Sadoghi Yazdi, H., Monsefi, R.: A new hierarchical clustering algorithm on fuzzy data (fhca). International Journal of Computer and Electrical Engineering-IJCEE **2**(1) (February 2010) 134–140

5. Rodrigues, M.E.S.M., Sacks, L.: A scalable hierarchical fuzzy clustering algorithm for text mining. In: In Proceedings of the 5th International Conference on Recent Advances in Soft Computing. (2004)
6. Treerattanapitak, K., Jaruskulchai, C.: Generalized agglomerative fuzzy clustering. In Huang, T., Zeng, Z., Li, C., Leung, C., eds.: Neural Information Processing. Volume 7665 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 34–41
7. Frigui, H., Krishnapuram, R.: Clustering by competitive agglomeration. Pattern Recogn. **30**(7) (July 1997) 1109–1119
8. Bank, M., Schwenker, F.: Fuzzification of agglomerative hierarchical crisp clustering algorithms. In Gaul, W.A., Geyer-Schulz, A., Schmidt-Thieme, L., Kunze, J., eds.: Challenges at the Interface of Data Analysis, Computer Science, and Optimization. Studies in Classification, Data Analysis, and Knowledge Organization. Springer Berlin Heidelberg (2012) 3–11
9. Konkol, M.: Brainy: A machine learning library. In Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M., eds.: Artificial Intelligence and Soft Computing. Volume 8468 of Lecture Notes in Computer Science. Springer International Publishing (2014) 490–499

Fig. 3: Clustering points of two clusters randomly sampled from two Gaussian distributions using fuzziness (a) 1.01, (b) 1.1, (c) 1.2, (d) 1.3.