Acta Scientific COMPUTER SCIENCES

Volume 4 Issue 2 February 2022

Research Article

# Reducing Time Complexity of the Fuzzy C-Means Algorithm: Case Studies

**Amrita Bhattacherjee[1]* and Sugata Sanyal[2]**

[1]*Department of Statistics, St. Xavier's College, Kolkata, India*

[2]*Professor (Retired), School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai, India*

***Corresponding Author:** Amrita Bhattacherjee, Department of Statistics, St. Xavier's College, Kolkata, India.

**Received:** November 30, 2021

**Published:** January 18, 2022



## Abstract

The Fuzzy C-Means clustering technique is one of the most popular soft clustering algorithms in the field of data segmentation. However, its high time complexity makes it computationally expensive, when implemented on very large datasets. Kolen and Hutcheson (2002) [1] proposed a modification of the FCM Algorithm, which dramatically reduces the runtime of their algorithm, making it linear with respect to the number of clusters, as opposed to the original algorithm which was quadratic with respect to the number of clusters. This paper proposes further modification of the algorithm by Kolen., *et al*. by suggesting effective seed initialisation (by Fuzzy C-Means++, proposed by Stetco., *et al*. [2]) before feeding the initial cluster centers to the algorithm. The resultant model converges even faster. Empirical findings are illustrated using two synthetic and two real-world datasets.

**Keywords:** Clustering; Fuzzy Partitions; Time Complexity; Fuzzy C-Means Algorithm; Unsupervised Machine Learning

## Introduction

Cluster analysis or clustering is a method of grouping data points into different clusters or categories such that objects within the same cluster are more similar to each other than objects in different clusters. The objects are grouped together based on some similarity measure, which is specified depending on the data at hand and the objective of the task. This method has widespread application, ranging from pattern recognition and market segmentation to image processing and various other fields of data analysis.

The Fuzzy C-Means algorithm is one such clustering algorithm which facilitates soft partitioning of the objects in the dataset. The earliest applications of clustering primarily focused on 'crisp' partitions of objects, where each point either fully belongs to a category or does not belong to a category at all. This approach relied on the idea that an object in a category does not bear any resemblance to any of the categories except to the one it belongs to. Soft partitions, on the other hand, rely on the idea that each object is characterised by the extent to which they belong to all the clusters/categories. A measure of this extent of an object's resemblance to each cluster is introduced by Zadeh (1965) [11] in the form of what is now known as a 'membership function'. The final goal is to create partitions or clusters with soft or fuzzy margins. As stated by Bezdek., *et al*. [3]: "A fuzzy c-partition of (the dataset) X is one which characterizes the membership of each sample point in all the clusters by a membership function which ranges between 0 and 1". The detailed definition of fuzzy c-means (FCM) partitioning and the corresponding algorithm, as proposed by James Bezdek, Robert Ehrlich and William Full (1983) [3], is given in Section 3.1.

The main limitation of this algorithm is its time complexity and memory requirements. The algorithm alternates between estimating cluster centers from the membership matrix and updating the membership matrix based on the cluster centers. As such, the membership matrix, which is of the order of the number of objects to be clustered, is repeatedly accessed and updated, on every iteration. This greatly affects the speed of the algorithm when the dataset is very large. This problem has been widely addressed in the literature. This paper focuses on the modification proposed by Kolen and Hutcheson (2002) [1], where the membership matrix is not generated (or updated) iteratively. This modification generates an algorithm which has a time complexity of O(ncp) as opposed to Bezdek's original FCM Algorithm, which had a time complexity of $O(nc^2p)$, where n is the number of objects in the dataset, c is the number of clusters and p is the number of features of each object/point in the data. Let us call this algorithm FCM-U, where U refers to the membership matrix.

This paper employs the FCM-U algorithm and pairs it with the popular approach of effective seed initialisation for even faster convergence. Here, the FCM++ algorithm (proposed by Stetco., et al. [2]) is implemented for effective seed initialization. On clubbing these two algorithms together, the model runs faster and empirically converges earlier than the FCM-U algorithm. The following section discusses some related works in reducing time complexity of the FCM Algorithm, followed by short descriptions of the original FCM algorithm, the FCM++ approach and the FCM-U algorithm. Then, the proposed model is defined, followed by a comparative analysis of the results obtained when this algorithm is employed for clustering datasets. Finally, some further scopes of improvement are discussed.

## Related works

Several researchers have proposed methods to tackle the problem of high computational cost that comes with implementation of the Fuzzy C-Means algorithm.

In 1986, Cannon, Dave and Bezdek [4] proposed an Approximate Fuzzy C-Means algorithm where the exact variates in the equation are replaced with integer/real-valued estimates. Tolias and Panas [5] applied spatial constraints on image segmentation problems using a fuzzy rule-based system, which showed reduced computational time.

In 1994, Kamel and Selim [6] proposed two algorithms that converged faster than the FCM algorithm, having adopted a continuous process where the algorithm starts updating the membership values as soon as a part of cluster centers are updated.

In 1998, Cheng., *et al*. [7] proposed a multi-stage random sampling approach where the cluster centers are estimated after taking repeated random samples from the data. Then, the centroids are initialised over the entire data. This process reported a speed-up of 2-3 times than the original algorithm. Hore., *et al*. [8] proposed a single-pass fuzzy c-means algorithm using weighted point calculation.

In 2002, Kolen and Hutcheson [1] proposed a modification which eliminates the task of repeatedly updating the membership matrix, this reducing the time complexity to a linear function of the number of clusters; as opposed to the original algorithm which was a quadratic function of the number of clusters. This was particularly beneficial for large datasets. In fact, this paper implements this approach in the proposed algorithm along with effective seed initialisation.

Another angle of attack adopted by researchers is manipulating the data itself. Hung and Yang [9] proposed the psFCM algorithm which used a simplified subset of the original data to speed up the convergence. Several approaches were made to eliminate initial bias and reduce the time taken for convergence of the FCM algorithm. These research works mainly focused on modifying the initial centroids which are passed to the algorithm. Effective seed initialisation shows promising result in removing initial bias of the FCM algorithm. In 2015, Stetco, Zeng and Keane [2] extended the idea of K-Means++ [10] algorithm into the standard version of Fuzzy C-Means.

## The algorithm

Section 3.1 briefly describe the idea of fuzzy partitions, followed by the Fuzzy C-Means algorithm which was originally proposed by James C. Bezdek [3]. Section 3.2 describes the concept of FCM++, as proposed by Stetco., *et al*. [2], and the modifications proposed by Kolen and Hutcheson [1]. Section 3.3 finally states the combined algorithm (with some modifications) that this paper implements.

### The fuzzy c-means algorithm

Let  be a set of  points in , the p-dimensional Euclidean space. For 1 the set of natural number, a fuzzy c-partition of  is represented by  where,  is a matrix of order , that is –

$$U = \left(\left(u_{ij}\right)\right)_{n\times c}$$

Where,  denotes the membership value of the  point in  to the fuzzy set. Here,  and .

The values of the membership matrix are subject to the following conditions –

- $0 \leq u_{ij} \leq 1, \quad \forall\, i, j$
- $\sum_{j=1}^{c} u_{ij} = 1, \quad \forall\, i$
- $0 < \sum_{i=1}^{n} u_{ij} < n, \quad \forall\, j$

The FCM algorithm defines a constant , which is called the fuzziness parameter and corresponds to the degree of fuzziness of the clusters.

By convention, we take . The FCM Algorithm then defines 'cluster centers' , as –

$$v_j = \frac{\sum_{i=1}^{n} x_i u_{ij}^{m}}{\sum_{i=1}^{n} u_{ij}^{m}} \quad \dots 3.1.1$$

The membership function is typically defined as –

$$u_{ij} = \left( \sum_{k=1}^{c} \left( \frac{d_{ij}}{d_{ik}} \right)^{\frac{2}{m-1}} \right)^{-1},$$

$$\text{for } 1 \leq i \leq n \text{ and } 1 \leq j \leq c \quad \dots 3.1.2$$

Where,

$d_{ij} = \|x_i - v_j\|$ is the distance of the $i^{th}$ point in $X$ to the $j^{th}$ cluster center.

The cost function is defined as –

$$J_m(U, V; X) = \sum_{i=1}^{n} \sum_{j=1}^{c} u_{ji}^{m} \|x_i - v_j\|^2 \quad \dots 3.1.3$$

Therefore, the Fuzzy C-means algorithm as proposed by Bezdek is given by –

Algorithm – 1: FCM

## Effective seed initialisation and eliminating the u-matrix

The Fuzzy C-Means ++ algorithm as proposed by Stetco*., et al*. uses effective seed initialisation to determine the starting values for the FCM algorithm.

| |
|---|
| **1.** Fix c, m. Choose an initial membership matrix $U^{(0)}$ |
| **2.** At step k, compute the means $v_j$, $1 \leq j \leq c$ using equation 3.1.1 |
| **3.** Update membership matrix $U^{(k)}$, using equation 3.1.2 |
| **4.** Repeat steps 2 and 3 until: $\|\| U^{(k+1)} - U^{(k)} \|\| < \epsilon$. Or, until k reaches the maximum number of permissible iterations |

**Algorithm 1**

Before stating the algorithm, we state some notations:

c: Number of clusters

p: Dimension of the datapoints

s: The spreading factor

V: The c x p prototype matrix

X: The n x p data matrix.

They defined a value , corresponding to the $i^{th}$ data point in X, given by –

$$P_i = \frac{d^s(x_i, V)}{sum(d^s)}$$

Where, $d^s(x_i, V)$ denotes the distance (raised to the power s) from a point to its closest representative in R.

The value of s controls the spreading factor of the algorithm. A small value of s will choose centers which are very close to each other, whereas a very large value of s might lead to the choice of outliers as cluster centers. When s is taken to be zero, the algorithm reduces to random seed initialisation.

Further, the first point is randomly chosen and determines the selection of all the other centers.

With the values and parameters defined above, the FCM++ algorithm by Stetco., *et al*. is as follows:

Algorithm – 2: FCM++ initialisation

```
function FCM++(X,c)
begin
      V=V ∪ random point from dataset
      while sizeOfV < k do
      begin
            choose xi ∈ X with probability Pi
            V = V ∪ xi
      end
      return V
end   //FCM++ ends here
```

**Algorithm 2**

We now state the algorithm as proposed by Kolen., *et al*. which constitutes the main body of the algorithm. In 2002, John Kolen and Tim Hutcheson proposed a modification in the algorithm which reduced the time of computation drastically. They eliminated the storage of the membership matrix at every iteration, and directly computed the updated cluster centers. The algorithm is stated below.

### Notations

c: Number of clusters

p: Dimension of the datapoints

n: Number of data points

m: The fuzziness coefficient

V: The c x p prototype matrix

J: The current cost measure

X: The n x p data matrix.

### Algorithm – 3: Eliminating U-Matrix

```
function UpdateV(V,c,X,p,n,m)
begin
        //save the current V matrix
        oldV=V

        //Initialise cost at 0
        J = 0
        rowsumU = 0      //c-dimensional vector
        V = 0            //initialise new V matrix to zero

        for k=1 to n do
        begin
                //Calculate the distances from the current datapoint X[i] to the centers in oldV
                //Calculate the numerators and denominators of equation 3.1.2 for this data point

                //initialise accumulator for denominator in equation 3.1.2
                denom3 = 0

                for i = 1 to c do
                begin
                        //calculate distance between current datapoint and ith cluster center
                        dsqr[i] = (||X[k]-oldV[i]||)^2

                        //save numer3[i] for future use
                        numer3[i] = (dsqr[i])^(1/m-1)
                        //Update denom3
                        denom3 = denom3 + 1/numer3[i]
                end

                for i = 1 to c do
                begin
                        u = (numer3[i]*denom3)^(-m)
                        //Update the cost (optional)
                        J = J + dsqr[i]*u

                        //Update the numerator of prototype centers
                        V[i] = V[i] + u*X[k]    //p-vector operation

                        //Update the future denominators of the centers
                        rowsumU[i] = rowsumU[i] + u

                end     //for i = 1 to c
        end     //for k = 1 to n

        //Combine numerator and denominators
        for i = 1 to c do
                V[i] = V[i]/rowsumU[i] //p-vector operation

        return V,J
end     //UpdateV ends here
```

**Algorithm 3**

### The proposed algorithm

This paper implements an algorithm which combines Sections 3.1 and 3.2 into a single implementation. In other words, we first generate a prototype matrix using effective seed initialisation (FCM++), and then use this initial prototype matrix as the starting point of the algorithm as stated in Section 3.2.

Additionally, some modifications were made so that the algorithm works even when the cluster centers are points from the dataset itself. The algorithm is as stated below:

- **Step 1:** Run algorithm 2 to obtain initial cluster centers
- **Step 2:** Pass V obtained in Step 1 to algorithm 3. Run algorithm 3 with some modifications. The modified version is given below.

```
function ModifiedUpdateV(V,c,X,p,n,m)
begin
      oldV=V
      J = 0
      rowsumU = 0
      V = 0

      for k=1 to n do
      begin
            denom3 = 0
            flag = -1 //flag to check equality of points with cluster center

            for i = 1 to c do
            begin
                  dsqr[i] = (||X[k]-oldV[i]||)^2
                  if dsqr[i]==0:
                        flag = i
                        continue to next i

                  numer3[i] = (dsqr[i])^(1/m-1)
                  denom3 = denom3 + 1/numer3[i]
            end
```

```
        for i = 1 to c do
        begin
            if i==flag:
                u=1
                V[i] = V[i]+X[k]
                rowsumU[i] = rowsumU[i]+1
            else:
                u = (numer3[i]*denom3)^(-m)
                J = J + dsqr[i]*u
                V[i] = V[i] + u*X[k]
                rowsumU[i] = rowsumU[i] + u
        end  //for i = 1 to c
    end  //for k = 1 to n

    for i = 1 to c do
        V[i] = V[i]/rowsumU[i]
    return V,J
end  //UpdateV ends here
```

**Algorithm a**

## Empirical results

Kolen and Hutcheson [1] illustrated the performance impacts of their modification in great detail. The algorithm implemented in this paper shows further improvement in computation speed owing to effective seed initialisation. The results are illustrated on 4 datasets – the Iris Dataset, the Wine Dataset and 2 synthetic datasets generated from gaussian distributions. The empirical findings are tabulated in tables 1 to 4.

## IRIS dataset

The Iris Dataset [12] contains a total of 150 datapoints where each point corresponds to attributes of a particular species of the Iris flower. A single point is a 4-dimensional feature vector containing measurements of – Sepal Length, Sepal Width, Petal Length and Petal Width of a flower. The corresponding label indicates one of three categories to which the flower belongs; the categories being Iris Setosa, Iris Virginica and Iris Versicolor. The time for convergence (to reach the same cost value) was measured (in seconds) for both the original FCM algorithm and the proposed modified algorithm while varying the number of clusters and the results are tabulated below.

The values indicate that the proposed algorithm provides a considerable gain in time due to faster convergence with the same cost value.

| Number of clusters | Algorithm Used | |
|---|---|---|
| | Original FCM | Proposed FCM |
| 2 | 0.058 | 0.021 |
| 3 | 0.147 | 0.047 |
| 4 | 0.193 | 0.072 |
| 5 | 0.299 | 0.093 |
| 6 | 0.384 | 0.101 |

**Table 1:** Time for convergence for the Iris dataset.

The time taken to converge for each algorithm is plotted in graph 1. The black points are the time taken (in seconds) by the original algorithm, plotted against the number of clusters specified to the algorithm. To compare the rate of change in time taken for each algorithm, a simple linear regression is fitted for each of them. The following graph gives a visual representation of the results obtained.

**Graph 1**

The regression equations obtained are –

$$\text{Time}_{\text{Original}} = (0.0804 \times \text{N}) - 0.1054$$

$$\text{Time}_{\text{Proposed}} = (0.0206 \times \text{N}) - 0.0156$$

Where, N represents the number of clusters. The regression is done keeping the number of features in the dataset constant.

It can be noted visually from the graph that the time taken by the original algorithm is consistently higher than that by the proposed algorithm. In addition, the rate of increase in time as the number of clusters increases can be obtained from the regression equations as follows –

$$\text{Slope for Original Algorithm} = 0.0804$$

$$\text{Slope for Proposed Algorithm} = 0.0206$$

Clearly, the rate of increase in time for a unit increase in the number of clusters is approximately 4 times higher for the original algorithm than that for the proposed algorithm.

This validates a considerable amount of savings in time, especially for higher number of clusters.

The time taken are recorded while keeping the cost value constant for a given number of clusters, which enables a fair comparison. The cost is calculated using equation 3.1.3. For perspective, the performance of the proposed algorithm in predicting the correct clusters can be visually estimated by looking at the following graphs. Graph 2 represents the true clusters as available in ground truth labels of the dataset.

**Graph 2**

Graph 3 represents the predicted clusters, as obtained by the proposed algorithm.

The red stars indicate the predicted cluster centers.

## Wine dataset

The Wine Dataset [13] contains data on the results of a chemical analysis of 3 different types of wine grown in the same region in Italy.

**Graph 3**

The 13 different features for each datapoint are actually the amount of each of the 13 different constituents found in the analysis. The attributes are real-valued numbers. There is a total of 178 datapoints. The time for convergence (to reach the same cost value) was measured (in seconds) for both the original FCM algorithm and the proposed modified algorithm while varying the number of clusters and the results are tabulated below.

| Number of clusters | Algorithm Used | |
|---|---|---|
| | Original FCM | Proposed FCM |
| 2 | 0.184 | 0.074 |
| 3 | 0.619 | 0.321 |
| 4 | 0.794 | 0.288 |
| 5 | 2.493 | 0.504 |
| 6 | 2.501 | 0.915 |

**Table 2:** Time for convergence for the Wine dataset.

Similar to its performance in the Iris dataset, the proposed algorithm, once again, shows significant economy in terms of time taken till convergence. A similar study is done to obtain simple linear regression equations for each of the algorithms. The regression lines are plotted against the number of clusters in graph 4.

The regression equations obtained are –

$$\text{Time}_{\text{Original}} = (0.6508 \times \text{N}) - 1.285$$

$$\text{Time}_{\text{Proposed}} = (0.1865 \times \text{N}) - 0.326$$

**Graph 4**

Where, N represents the number of clusters. The regression is done keeping the number of features in the dataset constant.

It can be noted visually from the graph that the time taken by the original algorithm is consistently higher than that by the proposed algorithm. In addition, the rate of increase in time as the number of clusters increases can be obtained from the regression equations as follows.

$$\text{Slope for Original Algorithm} = 0.6508$$
$$\text{Slope for Proposed Algorithm} = 0.1865$$

Here, the rate of increase in time for a unit increase in the number of clusters is approximately 3.5 times more for the original algorithm than that for the proposed algorithm.

The time taken are recorded while keeping the cost value constant for a given number of clusters, which enables a fair comparison. The cost is calculated using equation 3.1.3.

Following are two graphs representing the true clusters and the predicted clusters for the Wine dataset.

Upon implementing the FCM++(without U-matrix) algorithm, the predicted clusters are obtained as follows.

### Gaussian dataset (Type 1)

Isotropic gaussian blobs are generated using Python's Scikit-learn library. The dataset generated for this problem contains 3 clusters where cluster centers are generated at random from the interval (-10, 10). The standard deviation for each cluster is set at 1 (to maintain homoscedasticity). The random state is fixed at '0'. Under the above conditions, 300 points are generated, each having 3 features.

**Graph 5**

**Graph 6**

The points are plotted on a 2-dimensional space for visualisation.

The time for convergence (to reach the same cost value) was measured (in seconds) for both the original FCM algorithm and the

**Graph 7**

proposed modified algorithm while varying the number of clusters and the results are tabulated below.

| Number of clusters | Algorithm Used | |
|---|---|---|
| | Original FCM | Proposed FCM |
| 2 | 0.115 | 0.043 |
| 3 | 0.126 | 0.037 |
| 4 | 0.461 | 0.126 |
| 5 | 0.642 | 0.225 |
| 6 | 0.840 | 0.267 |

**Table 3:** Time for convergence for the Gaussian dataset (Type 1).

When the time taken is regressed over the number of clusters to be found, we get the following result.

**Graph 8**

The regression equations obtained are –

$$\text{Time}_{\text{Original}} = (0.197 \times \text{N}) - 0.345$$

$$\text{Time}_{\text{Proposed}} = (0.064 \times \text{N}) - 0.115$$

Where, N represents the number of clusters. The number of features in the dataset is kept constant.

It can be noted visually from the graph that the time taken by the original algorithm is consistently higher than that by the proposed algorithm. In addition, the rate of increase in time as the number of clusters increases can be obtained from the regression equations as follows.

$$\text{Slope for Original Algorithm} = 0.197$$

$$\text{Slope for Proposed Algorithm} = 0.064$$

Here, the rate of increase in time for a unit increase in the number of clusters is approximately 3 times more for the original algorithm than that for the proposed algorithm. Hence, we can conclude that the proposed algorithm facilitates a significant amount of savings in time to converge to the same clustering result.

The following graph illustrates the true clusters of this simulated dataset. Here, each true cluster essentially represents a distinct distribution (normal distribution with a different center).

**Graph 9**

The above points are then clustered using the proposed algorithm and the following predicted clusters were obtained.

**Graph 10**

The red stars indicate the predicted cluster centers.

### Gaussian dataset (Type 2)

Here, samples from 4 gaussian distributions of varying means and standard deviations are taken to create overlapping clusters. For this particular evaluation, the means of the 4 distributions are taken as (-3,1), (2,2), (1,-3) and (5,4) with respective standard deviations 1, 0.5, 1.5 and 2 respectively. 250 points are generated from each of these distributions (making a total of 1000 datapoints). The points are 2-dimensional in nature. The points are plotted on a 2-dimensional space for visualisation.

**Graph 11**

The time for convergence (keeping the cost same) is measured in seconds for both the original and the proposed algorithm. They are tabulated below for comparison.

| Number of clusters | Algorithm Used | |
|---|---|---|
| | Original FCM | Proposed FCM |
| 2 | 0.738 | 0.321 |
| 3 | 1.973 | 0.619 |
| 4 | 1.264 | 0.442 |
| 5 | 5.288 | 0.910 |
| 6 | 12.016 | 2.402 |

**Table 4:** Time for convergence for the Gaussian dataset (Type 2).

The time taken by each of the 2 algorithms is regressed separately on the number of clusters, and two regression equations are obtained.

**Graph 12**

Note that even though the regression line seems to suggest that, for 2 clusters, proposed algorithm takes more time than the original algorithm, it can be seen from the plotted points that, in the data, the proposed algorithm does in fact take less time for all clusters.

The regression equations obtained are –

$$\text{Time}_{\text{Original}} = (2.587 \times \text{N}) - 6.093$$

$$\text{Time}_{\text{Proposed}} = (0.445 \times \text{N}) - 0.842$$

Where, N represents the number of clusters. The number of features in the dataset is kept constant.

It can be noted that the rate of increase in time as the number of clusters increases can be obtained from the regression equations as follows.

$$\text{Slope for Original Algorithm} = 2.587$$

$$\text{Slope for Proposed Algorithm} = 0.445$$

Here, the rate of increase in time for a unit increase in the number of clusters is approximately 5 times more for the original algorithm than that for the proposed algorithm, which is especially pronounced for high number of clusters. Hence, we can conclude that the proposed algorithm facilitates a significant amount of savings in time to converge to the same clustering result.

For visualisation, the true clusters are plotted below, followed by a graph illustrating the predicted clusters.

**Graph 13**

**Graph 14**

The red stars indicate cluster centers as estimated by the algorithm.

## Conclusion

Comparative analyses of the time taken for Algorithm 2 and Algorithm 3, when implemented individually are already elaborated in [2,3] respectively. This paper combines these algorithms and compares its performance with the original Fuzzy C-Means algorithm to empirically confirm that it indeed accelerates the speed of the algorithm, which becomes more evident for larger datasets and higher number of clusters. In fact, the cluster accuracy stays intact (and in some cases, improves over the original FCM algorithm).

The tabulated values and graphs indicate faster convergence with very high cluster accuracy (as confirmed by Adjusted Rand Index during runtime).

## Further scope

One can be interested in tailoring the algorithm to the specific data in hand. In this context, feature normalisation, feature engineering, sampling from the dataset could be viable options for further speeding up the convergence.

The FCM algorithm largely depends on the initial centers selected. Further attempts could be made to eliminate the initial bias to ensure that the algorithm converges to a better solution. FCM++ has been proven to be a good approach in this context. However, testing other methods of effective seed initialisation (preferably along with Hutcheson and Kolen's [1] algorithm) might yield promising results.

Combining other time-reduction approaches like random sampling of the datapoints or multi-stage random sampling [7] have been proven to be very successful. Pairing this strategy with the proposed algorithm is expected to perform extremely well for large datasets.

Another open field of application is image segmentation. FCM algorithm finds manifold implementations in image segmentation problems, where the image sizes are quite high. In such a scenario, modifying the algorithm to accommodate image data and effectively reducing its runtime will open new avenues. The authors of this paper are looking into a similar implementation on image data, and tailor the time complexity reduction approach towards image-segmentation problems.

## Bibliography

1. Kolen John F and Tim Hutcheson. "Reducing the time complexity of the fuzzy c-means algorithm". *IEEE Transactions on Fuzzy Systems* 10.2 (2002): 263-267.
2. Stetco Adrian., *et al*. "Fuzzy C-means++: Fuzzy C-means with effective seeding initialization". *Expert Systems with Applications* 42.21 (2015): 7541-7548.
3. Bezdek James C., *et al*. "FCM: The fuzzy c-means clustering algorithm". *Computers and Geosciences* 10.2-3 (1984): 191-203.
4. Cannon Robert L., *et al*. "Efficient implementation of the fuzzy c-means clustering algorithms". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2 (1986): 248-255.
5. Tolias Yannis A and Stavros M Panas. "On applying spatial constraints in fuzzy image clustering using a fuzzy rule-based system". *IEEE Signal Processing Letters* 5.10 (1998): 245-247.
6. Kamel Mohamed S and Shokri Z Selim. "New algorithms for solving the fuzzy clustering problem". *Pattern Recognition* 27.3 (1994): 421-428.
7. Cheng Tai Wai., *et al*. "Fast fuzzy clustering". *Fuzzy Sets and Systems* 93.1 (1998): 49-56.
8. Hore Prodip., *et al*. "Single pass fuzzy c means". 2007 IEEE International Fuzzy Systems Conference. IEEE (2007).
9. Hung Ming-Chuan and Don-Lin Yang. "An efficient fuzzy c-means clustering algorithm". Proceedings 2001 IEEE International Conference on Data Mining. IEEE (2001).
10. Arthur David and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding". Stanford (2006).
11. Zadeh Lotfi A. "Fuzzy Sets, Information and Control". *MathSciNet zbMATH* 8 (1965): 338-353.
12. https://archive.ics.uci.edu/ml/datasets/iris
13. https://archive.ics.uci.edu/ml/datasets/wine