# Fuzzy C-Means

Meritxell Arbiol López, Gerard Pons Recasens

April 2022

**Abstract**

In this paper we are going to explore what is behind the methods called probabilistic clustering, focusing especially on the fuzzy C-means algorithm, one of the most popular fuzzy clustering algorithms, where one point of our data can belong to more than one cluster.
We are going to discover and study in the literature the formulas that exist in the description it, and try to get an intuition of them and their effect on the results.
Finally we will compare using KNIME the result of its Fuzzy C-Means node with other available clusters using a toy dataset and exploring the effect of some of the parameters such as the crispness.

## 1 Introduction to Probabilistic Clustering

Clustering is an unsupervised learning technique which is usually seen as a way of generating subsets of a dataset based on the similarity of their individuals. Some of the most well known algorithm, such as k-means, approach the clustering problem in a hard manner: assign every data point to one and only one cluster. However, there also exist other approaches, named soft clustering, in which each data point can belong to more than one cluster: they are assigned a membership value $\in [0, 1]$ to each cluster. It can be easily seen that in a lot of scenarios, especially on those with overlapping clusters, this clustering approach is more natural than the traditional hard one. For instance, if we think about a classification problem where we need to identify the areas of some research papers, it is sensible to think that although some of the papers will belong to a clear research area, some other will fall in-between two or more topics.

One of the most well known soft-clustering algorithms is the Fuzzy C-means algorithm (FCM), developed by J.C. Dunn in 1973 [1] and improved by J. Bezdeck in 1981 [2], on which we are going to focus in this paper and explain in detail in the next section.

## 2 The FCM Algorithm

As explained in the previous section, with soft-clustering techniques we do not assign a cluster to each instance of the dataset, but we rather assign to each point the probability of belonging to each of the clusters. Hence, for each datapoint $i$ and cluster $j$ we define this probability as $w_{ij} \in [0, 1]$, which satisfies that $\sum_{j=1}^{k} w_{ij} = 1$. Being $C$ the set of $k$ centroids, we can recall that for original k-means we wanted to minimize the following equation:

$$\sum_{x_i \in X} min_{c_j \in C} ||x - c_j||^2 \tag{1}$$

Now, introducing the soft assignment of clusters, we can easily observe that the equation to be minimized is by weighting each contribution, using the defined $w_{ij}$ and the distances to all centroid. This term to minimize is usually referred as Fuzzy Within-Inertia:

$$\sum_{x_i \in X} \sum_{c_j \in C} w_{ij}^m ||x - c_j||^2 \tag{2}$$

We can observe here the hyper-parameter $m$, which is called the *fuzzification parameter*, whose effect on the clustering will be explained in the following sections.

Although the function to minimize is different, the algorithm structure and steps are not much different from the one of k-means:

1. First of all, the weights $w_{ij}$ are initialized, usually following a random approach. However, some of the popular initalization methods used in standard k-means (i.e. k-means++, MaxMin, Macqueen) can be easily changed to accommodate the fuzzy scenario[5].

2. Using the weights, we compute the $k$ centroids:

   As we can expect, the centroid $c_j$ of the cluster is the mean of all the points of the dataset weighted by $w_{ij}$:

   $$c_j = \frac{\sum_{x_i} w_{ij}^m x_i}{\sum_{x_i} w_{ij}^m} \tag{3}$$

3. Using now the computed centroids in Step 2, we recompute the weights $w_{ij}$. As a general intuition, one can suppose that a datapoint is likely to belong to a cluster if it is close to each centroid. Hence, the weight should be inversly propotional to the distance to it (notice the negative sign in the exponent). However, this weightings should be relative to all the distances to other clusters for the same datapoint as by Equation 1 they have to add up to one, hence the denominator ensures that.

   $$w_{ij} = \frac{||x_i - c_j||^{\frac{-2}{m-1}}}{\sum_{l=1}^{k} ||x_i - c_k||^{\frac{-2}{m-1}}} \tag{4}$$

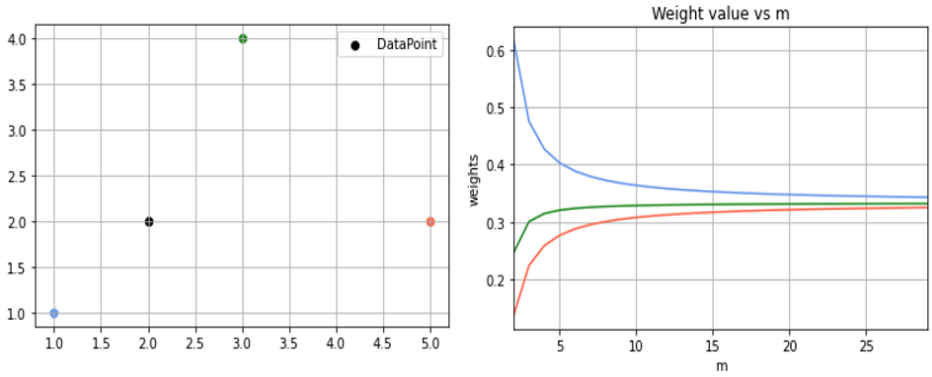


Figure 1: On the left hand side of the Figure we can observe the cluster centers (coloured) and the data point (in black) for which we will assign the probabilities. We can see on the right hand side how the weights to each cluster change as we increase the value of m, tending to 1/n.

We start again at Step 2 until convergence.

The formulas are quite intuitive with the exception of effect of the m parameter. We thought that the best way to observe that was by creating two simple examples, one for the update of the weights and another one for the update of the centroids.

## 2.1 Fuzzification Parameter and Weights

We present now a simple example, depicted in *Figure 1*. On the left hand side, we can observe the distribution of the problem, with 3 coloured centroids and one datapoint in black. On the left hand side, we computed the weights of this point with the three cluster centers, with different values of $m \in [2, 30]$, which is the most usual range. We can observe that with small values of m, the probabilities are distinct and proportional to the distance to the centroid, but as we increase m the values of the weights converge to the value $1/k$, making the clusters asymptotically equiprobable. Hence, we can clearly see the fuzzification effect, as the distinction between clusters is less clear once we increase $m$.

## 2.2 Fuzzification Parameter and Centroids

Now we are going to focus on the election of the centroid position by fixing weights of three datapoints, precisely $[0.5, 0.4, 0.3]$. We can observe in *Figure 2* that as the weights are values $\in [0, 1]$, raising them to the power of m, there is a great difference in attenuation: for instance, for m=3 the values $0.5^3$ and $0.4^3$ differ in 3 orders of magnitude. That is why we can observe that the centroid tends to the point with the higher probability. However, one must not be misled by this observation, because as we have seen with the weighting, high values of

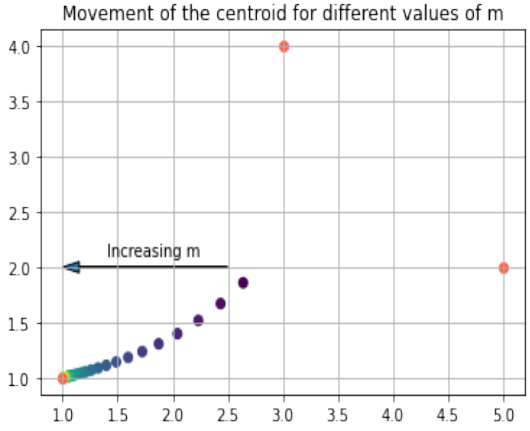


Figure 2: In this figure we have three datapoints (in red) and a cluster (note that we assume that there is at least one other cluster, but we do not show represent them). We represent where the centroid will be placed depending on the value of $m$.

$m$ will give similar values to them, resulting in the centroid laying close to the center of the three points.

### 2.3 But what is the Fuzzication Paramater?

After getting a visual intuition about what the parameter does in some simple and fixed scenarios, we can explain the purpose of that parameter. Briefly, it is a parameter which aims to make the algorithm robust to noise, as with it the effect of noisy data (i.e. the one that belongs to overlapping clusters or comes from background noise) can be decreased. It is a parameter that should be adjusted according to the dataset one is working with, as it is related to the maximum noise of the data. For it, some observations can be made:

- Although it is not properly defined when $m \to 1$ (see denominator of the exponent in *Equation 4*), one can see that in that case the algorithm will be exactly equal than the original k-means, assigning each point to just one cluster.
- As hinted in the previous subsections, when $m \to \infty$ all the points are assigned the same probabilities to each cluster.
- A large fuzzification value can decrease the effect of outliers, as more clusters share their objects, which would result in the outlying point membership being distributed among the different clusters, while the point that are in fact from a cluster will have their largest membership slightly decreased.

- With the previous point, one can see that in order to address correctly the problem and not over-penalize points that actually are from a cluster the value of $m$ should be appropriately chosen. This is not an easy task, an some methods have been proposed in the literature. For instance, in [3], distribution of distances between objects (by calculating the coefficient of variation) is used to determine the value of $m$, or in [4], where they generalize the problem by not using instances of the dataset but rather the dataset metrics, such as the number of dimensions or number of instances.

### 2.4 Comparison with K-Means

After understanding the algorithm, we can make some comparison of it with the well known K-Means algorithm:

**ADVANTAGES**

- In a lot of datasets, establishing a membership to different clusters is a more natural approach than only labelling each data point to one, as an overlap between clusters can exist depending on the domain.
- A well set fuzzification parameter can help to deal with noise and outlying observations.

**DISADVANTAGES**

- In each step of the algorithm there are more operations to be done, hence the time it takes to run the algorithm is greater.
- There is now another hyperparamter $m$, for which there is not a clear way to find an appropriate value.

**COMMON DRAWBACKS**

- They both suffer from the problem of initialization and the convergence to local minima.
- The number of clusters $K$ has to be specified as an hyper-parameter.

## 3 Fuzzy c-means in Action

While reading the literature, various sources pointed out that Fuzzy c-means has been widely used in protein identification and image segmentation. As we found these tasks interesting, we decided to do some research on how c-means helps with problems:

1. *Genes and Proteins:* the human genome is a very complex and wide topic, and a lot of studies have been focusing during the past years on having a better understanding about it. One of these tasks deals with gene expressions: it is common the case that proteins are involved in multiple

biological processes, so the genes producing them can be co-regulated (i.e. they regulated by at least one common known transcription factor) in different manners under different conditions. Approaching the problem in a traditional way, with hard clustering, is not sufficient to capture this behaviour. Here is where fuzzy c-means comes to play, and it has been widely used to capture these patterns [3][6].

2. *Image Segmentation:* in the image domain, partitioning the image in subsets is a difficult problem due to spatial resolution, poor contrast or overlapping intensities, among others. To deal with that, one of the most widely used techniques is Fuzzy c-means, due to its robustness to ambiguity. In that sense, even some modifications have been made to the algorithm, in order to deal with the image noise by introducing an spatial term to the objective function to minimize [7]. In the work, the introduced term takes into account the memberships of the neighbouring pixels. This change acts as a regularizer and biases the final solution towards an homogeneous labelling. It has particularly been proved to be effective when dealing with noisy scans.

# 4 Implementation of fuzzy k-means in KNIME

The Fuzzy C-means algorithm is implemented in KNIME by using the node called Fuzzy c-Means, present in the regular KNIME environment. The input to this node is the dataset used to which apply clustering (or just the train partition if the purpose is to train a clustering algorithm). It is important that the data have been previously normalized. The outputs of this node are the table with the classified training data (i.e. the cluster memberships of each data point) and the trained model in order to classify the test data using another KNIME node called Cluster Assigner.

There are a number of parameters that can be modified within the Fuzzy C-means node, such as the number of clusters we want, that is, the number of groups we want the algorithm to classify or we also have the maximum number of iterations we want the algorithm to do, whose default number in KNIME is 99. Another parameter is the Fuzzifier, where as we have explained above, this parameter indicates the level of overlapping allowed between clusters and it can take values ranging from greater than 1 to less than 10. We also have a checkbox if we want the initialization root to be random or not.

We can also choose whether we want to induce a noise cluster, and if we want to, we can let the delta parameter be updated at each iteration according to the average distances between points in our data. The other option we have is to let the delta parameter be the fixed distance from each point of our data to the noise cluster.

## 4.1 Comparison of different clustering methods in KNIME

In this project we will compare several clustering algorithms against fuzzy c-means using the KNIME tool. Among these algorithms we have the most commonly used ones called k-means and the K-medoids, very similar to the previous one but with some small changes.

The dataset with which we are going to perform the analysis is the one popularly known as Iris, where we have four numerical variables and one categorical variable, which is our target. Apart from its simplicity, understandability and familiarity, we have chosen the dataset because we know apriori the number of cluster/classes, 3, so that we can skip changing the K parameter and we can focus on the results.

In the previous sections we have already explained a little about the differences between C-Means and K-Means, but we do not have said anything about the K-Medoids algorithm nor covered it in the lectures. K-Medoids is an algorithm that is closely related to k-Means: breifly, the K-Means algorithm aims to minimize the squared-error while the K-Medoids algorithm minimizes the sum of dissimilarities between points and a point that is designated the center of that cluster. The K-Means algorithm does not define points in our data as the center, whereas K-Medoids does.

### 4.1.1 K-Means Analysis

The first algorithm that we have implemented in KNIME is the K-means algorithm, as we have mentioned before, it is important to have the data normalized first. We have done clustering and applied PCA to allow a comprehensible visualization and go from having 4 variables to work only with 2. By keeping the two variables with the maximum possible information, we can visualize more clearly the scatter plot on these two axes (PC1 and PC2) and get a more comprehensive view of the clustering distribution.

In *Figure 3* we can visualize the scatter plot with the resulting clusters. We can see that there is a difference between each of them: the orange cluster shows a clear separation with the others, while the boundary between the blue and green ones is not clear.

### 4.1.2 K-Medoids Analysis

To apply the k-medoids model, we followed the same steps as for the previous analysis (i.e. first normalizing the data) and we tried a couple of different distances to generate the model: first we used the Euclidean distance and then the Manhattan distance. Finally, we applied PCA to simplify again the complexity of our data by generating two components (PC1 and PC2) and visualizing the scatter plot of our already classified data on these two new components.

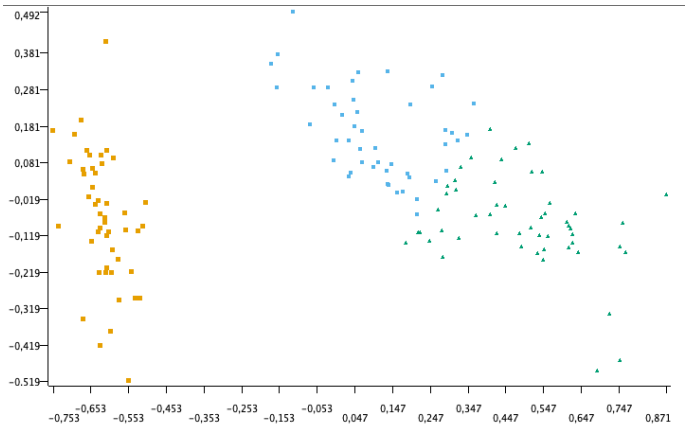


Figure 3: Scatter plot obtained with the k-means algorithm on the PC1 and PC2 axes (obtained by applying PCA).

In *Figure 4* and *Figure 5* we can see the results of our already classified data on PC1 and PC2: first with the Euclidean distance, and then with the Manhattan distance. We see that both results are almost identical except for a single difference. One of the points in our dataset (marked with a red circle) is part of the orange cluster when using the Euclidean distance, and when using the Manhattan distance this same point is colored green so it is part of the other cluster. Since this only happens in one point we can say that the results obtained are not under the influence of the chosen distance.

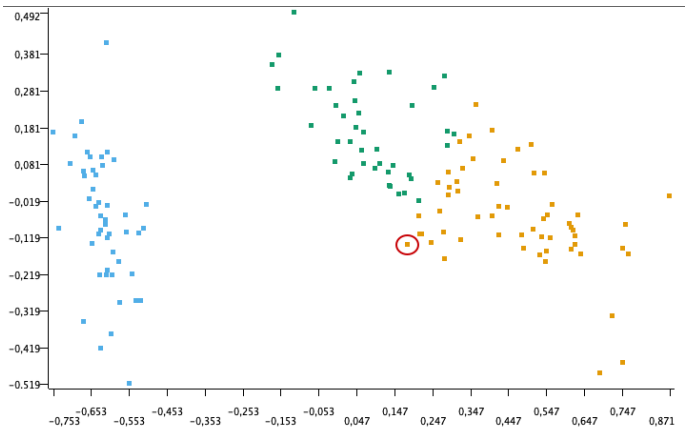


Figure 4: Scatter plot obtained with the k-medoids algorithm on the PC1 and PC2 axes using the Euclidean distance.

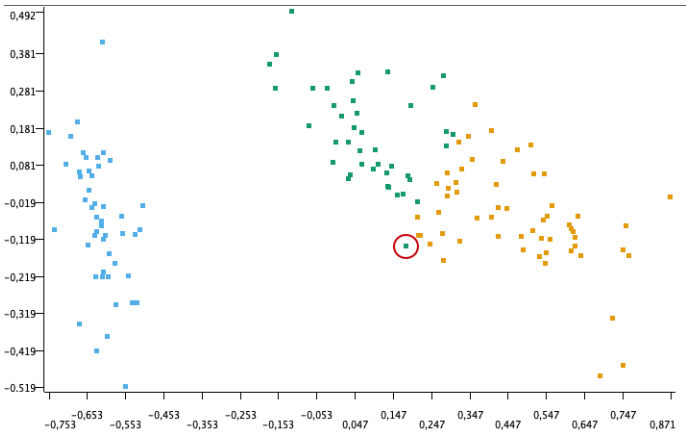


Figure 5: Scatter plot obtained with the k-medoids algorithm on the PC1 and PC2 axes using the Manhattan distance.

#### 4.1.3 Fuzzy C-Means Analysis

Finally, we have performed the analysis with the Fuzzy C-Means algorithm. For this purpose, we have also normalized the data previously. We have to remember that one of the most important properties to take into account when using the fuzzy c-means algorithm is that it can classify our data into more than one cluster at a time. For this purpose there is a parameter called *Fuzzifier*, as we have already explained above. This parameter can take values greater than 1 to less than 10. As this parameter increases, the overlap of belonging to more than one class is greater. Using this node, each of our points is assigned a weight for each of the clusters with a value between 0 and 1, finally we have a winning cluster, which is the cluster that for that point has the highest weight.

In the *Figure 6* we have the result obtained with the *Fuzzifier* parameter value = 8, so as we have explained before, the overlap of each point of our data between clusters is larger. Therefore, in *Figure 7* we see that the values taken by each point of our data for each cluster (*cluster_0, cluster_1 and cluster_2*) are quite large values. Finally among all these values we are left with the largest one, which is the winner cluster.

On the other hand, in *Figure 8* we see the results obtained with the value of *Fuzzifier* 2. We can clearly see the difference of weights of each point with each cluster in *Figure 9*. As now, the algorithm has much less "doubts" when classifying each data to a different cluster, and gives a more crisp cluster. The more we increase this parameter the more "doubts" the model will have when classifying.

If we compare the results obtained with both values of the *Fuzzifier* parameter (*Figure 6* and *Figure 8*), we see that the points that are on the border

between the orange and green class belong to one cluster or the other depending on the value set (the values marked inside the red circle). When the *Fuzzifier* parameter is 2 it seems that the orange cluster predominates, on the other hand when the value is 8, we see that some of these values belong to the green cluster.

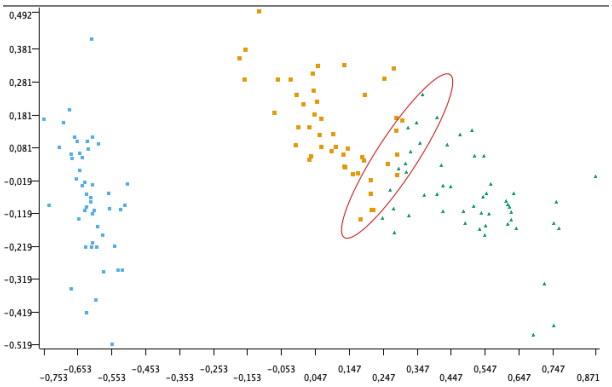


Figure 6: Scatter plot obtained with the fuzzy c-means algorithm on the PC1 and PC2 axes and Fuzzifier = 8.

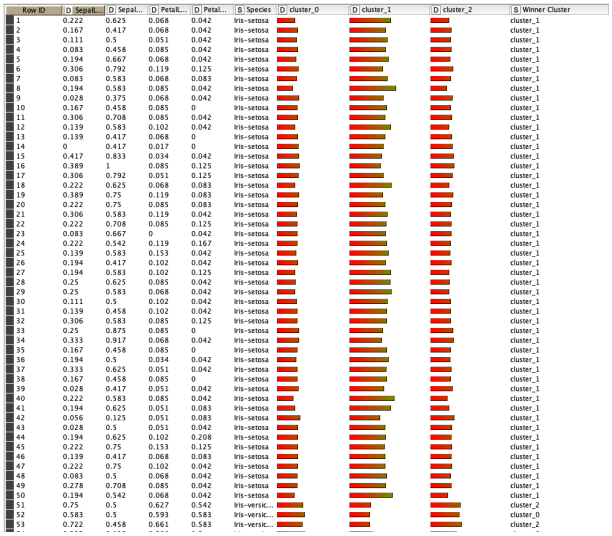


Figure 7: Cluster memberships with the fuzzy c-means algorithm with Fuzzifier = 8.

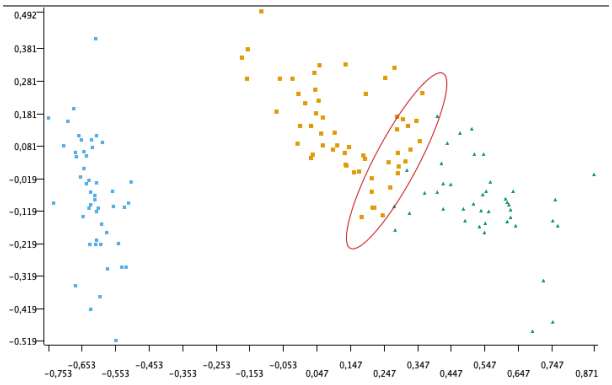


Figure 8: Scatter plot obtained with the fuzzy c-means algorithm on the PC1 and PC2 axes and Fuzzifier = 2.

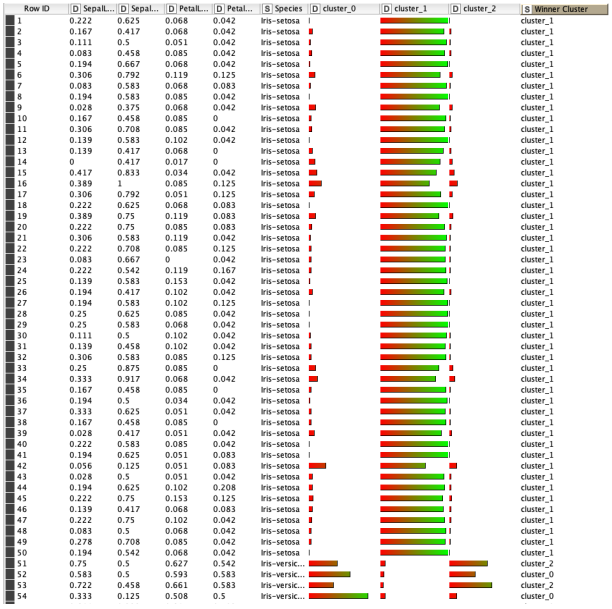


Figure 9: Cluster memberships with the fuzzy c-means algorithm with Fuzzifier = 2.

# 5 Summary and Conclusions

As we can see, throughout the study we have tested some clustering algorithms and compared them with each other. Despite the similarities, they all have small changes which include several parameter options that apply to some of them: the different distance measures used for the K-Medoids or the *Fuzzifier* parameter

of the C-Means that can make the final result vary more or less, especially in those points of our dataset that are on the border between two classes. We clearly saw this behaviour with the membership values for the small (i.e. 2) and larger (i.e. 8) value of the parameter, where the larger the value, the less crisp the clustering was. This happened to us between the orange and green classes, as these were spatially very close to each other. Thus, as explained in the first sections, we could treat the points in between the regions with their membership values and draw appropriate conclusions for them. On the other hand, with respect to the blue group, it has remained the same with all the algorithms we have used, since the distance separating this class from the other two was quite remarkable.

# References


[1] Dunn, J. C. (1973-01-01). "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters". Journal of Cybernetics. 3 (3): 32–57.

[2] J. C. Bezdek ,"Pattern Recognition with Fuzzy Objective Function Algoritms", Plenum Press, New York, 1981

[3] Dembélé,D. and Kastner,P. (2003) Fuzzy C-means method for clustering microarray data. Bioinformatics, 19, 973–980

[4] Schwämmle V, Jensen ON. A simple and fast method to determine the parameters for fuzzy c-means cluster analysis. Bioinformatics. 2010 Nov 15;

[5] Aybükë Oztürk, Stéphane Lallich, Jérôme Darmont, Sylvie Yona Waksman. MaxMin Linear Initialization for Fuzzy C-Means. 14th International Conference on Machine Learning and Data Mining (MLDM 2018), Jul 2018, New York, United States

[6] Gasch AP, Eisen MB. Exploring the conditional coregulation of yeast gene expression through fuzzy k-means clustering. Genome Biol 2002;3(11). RESEARCH0059.

[7] Ahmed, Mohamed N.,Yamany, Sameh M., Mohamed, Nevin, Farag, Aly A., Moriarty, Thomas (2002). "A Modified Fuzzy C-Means Algorithm for Bias Field Estimation and Segmentation of MRI Data". IEEE Transactions on Medical Imaging. 21: 193–199

# 6 Appendix

Below is a screenshot to better understand the scheme of how we have configured the KNIME nodes for each of the analyses we have performed.

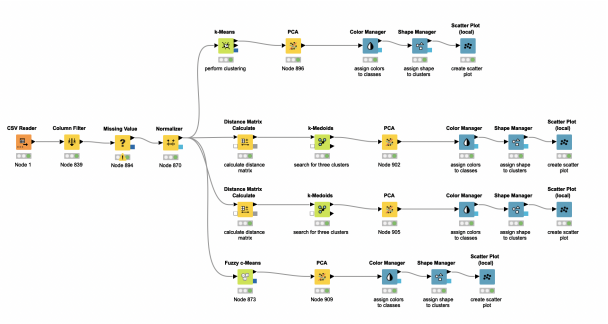


Figure 10: General schema from KNIME