Advanced Machine Learning Techniques

Fuzzy Clustering:

Fuzzy C-Means and Gustafson-Kessel Fuzzy C-Means

Author: Aleix Solanes
Master in Artificial Intelligence
UPC, UB, URV
January 2017

Contents

Fuzzy C-means

Fuzzy C-means is one of the most used fuzzy clustering algorithm. It has a very similar algorithm with respect to k-means.
The main idea is:

1. Choose a number of clusters
2. Assign randomly to each point, the coefficients for being in the clusters
3. Repeat until convergence (the coefficients' change between two iterations is no more than a given epsilon, also called the sensivity threshold).
    a. Compute centroids for each cluster
    b. Compute the coefficients of being in the clusters for each point

The centroid is calculated by averaging the weight of belonging to the cluster for each point.
Fuzzy C-means aims to minimize the objective function:

$$\underset{C}{\arg\min} \sum_{i=1}^{n} \sum_{j=1}^{c} w_{ij}^{m} \left\| \mathbf{x}_i - \mathbf{c}_j \right\|^2$$

where each element of Wij tells the degree to which element Xi belongs to cluster Cj.

$$w_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{\|\mathbf{x}_i - \mathbf{c}_j\|}{\|\mathbf{x}_i - \mathbf{c}_k\|} \right)^{\frac{2}{m-1}}}$$

Differences: FCM vs K-means

Both algorithms try to minimize an objective function, the main difference is that FCM adds the parameters Wij (membership values) and m (fuzzyfier). A large m, results in smaller membership values.
Both algorithms try to minimize intra-cluster variability, but both share the same problems[1]:

- Minimum is local minimum
- Results depend on initial choice of weights.

Fuzzy C-means implementation

In order to run a simple FCM clustering algorithm, like scikit-learn other clustering algorithms, all it is necessary to do is first instantiate the classifier:

```
from FCM import FCM
fc = FCM(num_clusters=5, m=2, plot_level=1,seed=5)
```

Once we have an instance with the desired parameters specified, we can fit the model:

```
memberships=fc.fit(data)
```

The variable memberships, now will contain a matrix with all the memberships per sample:

```
Memberships obtained:
[[ 0.71053374  0.02124392  0.04934133  0.17669329  0.04218772]
 [ 0.46594556  0.05661617  0.31039262  0.12904206  0.03800359]
 [ 0.10361575  0.03653273  0.01954002  0.82615808  0.01415342]
 ...,
 [ 0.16555209  0.2233164   0.05693435  0.52041155  0.03378562]
 [ 0.10112099  0.58273981  0.05752954  0.21434965  0.04426   ]
 [ 0.11105771  0.5846696   0.0643575   0.1354051   0.1045101 ]]
```

This is the output obtained, as it can be seen, for each point we obtain a list with a weight corresponding to the probability that this point belongs to each cluster. For example, the first row (the first point analyzed) should belong to cluster number 1, because the weight is of 0.71 out of 1.0. In the second one, it is not such clear, because cluster 1 has 0.46 and cluster 3 has 0.31. This is the singularity of Fuzzy clustering, that for each point you can know how much reasonable is to include a point to a cluster.

The function includes a parameter to know the label assigned (the one with the higher weight value). This parameter is the `labels':

```
Labels = fc.labels
>> [0,3,2,1,...,3,2,0]
```

Each number of the output corresponds to one cluster assigned.

Input parameters

- Num_clusters : number of clusters expected
- M : fuzzifier value
- Plot_level (only 2D data):
  - 0: no plot while fitting (default)
  - 1: plot all clusters centers in a single plot, with the shape of the cluster
  - 2: plot all clusters separate, useful to know the number of each cluster
- Seed : for reproducibility

Cluster validation: Silhouette Coefficient

As a measure of validation, the method returns the Silhuette Coefficient. This value is calculated using the mean intra-cluster distance (a) and the mean nearest non belonging cluster distance (b) for each sample. So, this value for a sample is:

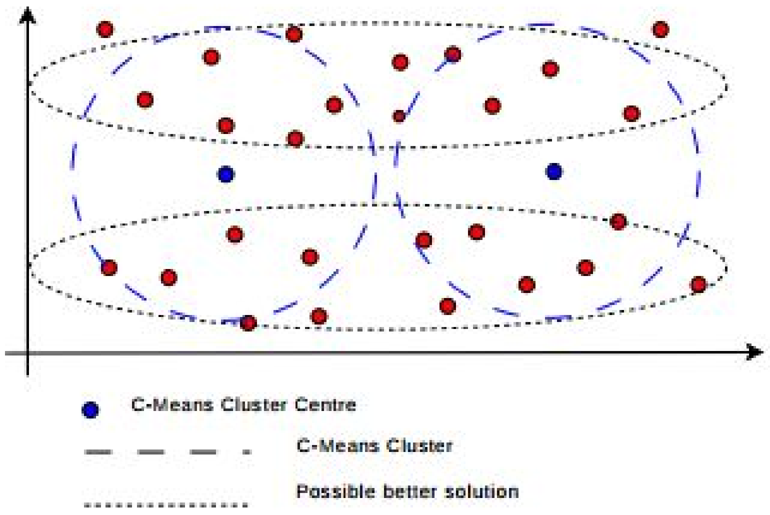
$$\frac{b-a}{max\,(a,b)}$$

To obtain this value, it is as simple as a call to the score parameter:

```
Silhouette_coefficient = fc.score
>> 0.46
```

Gustafson-Kessel Fuzzy C-means

This variation of Fuzzy C-means is quite similar to Fuzzy C-means (FCM). The main difference is the way the distance is calculated.
FCM uses euclidean distances, while Gustafson-Kessel uses Mahalanobis distances.[1,2,3]



The use of this distances is due to the fact that C-Means assumes circle clusters, while GKFCM will consider the option of this clusters being ellipsoids.
The distance (mahalanobi) will be calculated as follows:

$$d_{ik}^2 = (\mathbf{x}_i - \mathbf{v}_k)^T A_i (\mathbf{x}_i - \mathbf{v}_k) \qquad A_i = \sqrt[p]{\det S_i} S_i^{-1}$$

The Si parameter, is the fuzzy covariance matrix, and corresponds to the following formula[1,2,3]:

$$S_i = \sum_{e=1}^{n} \mu_{ie}^m (\mathbf{x_e} - \mathbf{v_i})(\mathbf{x_e} - \mathbf{v_i})^T$$

Gustafson-Kessel Fuzzy C-means implementation

Like in FCM, in order to run a GKFCM clustering algorithm, all it is necessary to do is first instantiate the classifier:

from GKFCM import GKFCM

fc = GKFCM(num_clusters=5, m=2, plot_level=1,seed=5)

Once we have an instance with the desired parameters specified, we can fit the model:

memberships=fc.fit(data)

The variable memberships, now will contain a matrix with all the memberships per sample:

```
Memberships obtained:
[[ 0.71053374  0.02124392  0.04934133  0.17669329  0.04218772]
 [ 0.46594556  0.05661617  0.31039262  0.12904206  0.03800359]
 [ 0.10361575  0.03653273  0.01954002  0.82615808  0.01415342]
 ...,
 [ 0.16555209  0.2233164   0.05693435  0.52041155  0.03378562]
 [ 0.10112099  0.58273981  0.05752954  0.21434965  0.04426   ]
 [ 0.11105771  0.5846696   0.0643575   0.1354051   0.1045101 ]]
```

Like in FCM, for each point we obtain a list with a weight corresponding to the probability that this point belongs to each cluster.

The function includes a parameter to know the label assigned (the one with the higher weight value). This parameter is the `labels':

```
Labels = fc.labels
>> [0,3,2,1,...,3,2,0]
```

Each number of the output corresponds to one cluster assigned.

Input parameters

- Num_clusters : number of clusters expected
- M : fuzzifier value
- Plot_level (only 2D data):
  - 0: no plot while fitting (default)
  - 1: plot all clusters centers in a single plot, with the shape of the cluster
  - 2: plot all clusters separate, useful to know the number of each cluster
- Seed : for reproducibility
- Det : the determinator value to calculate Mahalanobi distance, concretely the matrix Ai

Cluster validation: Silhouette Coefficient

As a measure of validation, the method returns the Silhuette Coefficient. This value is calculated using the mean intra-cluster distance (a) and the mean nearest non belonging cluster distance (b) for each sample. So, this value for a sample is:

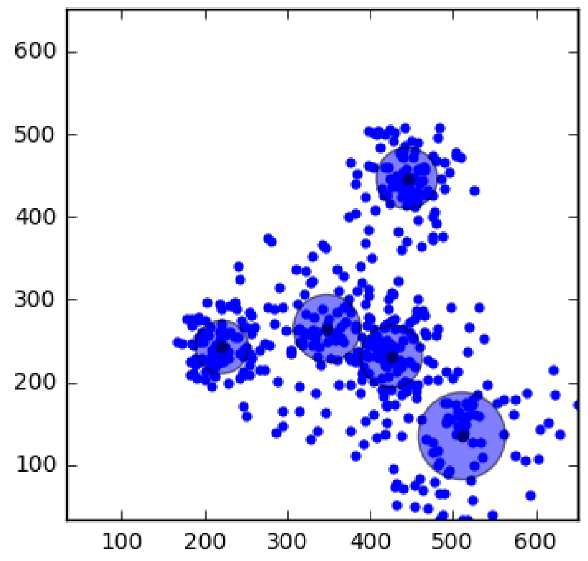
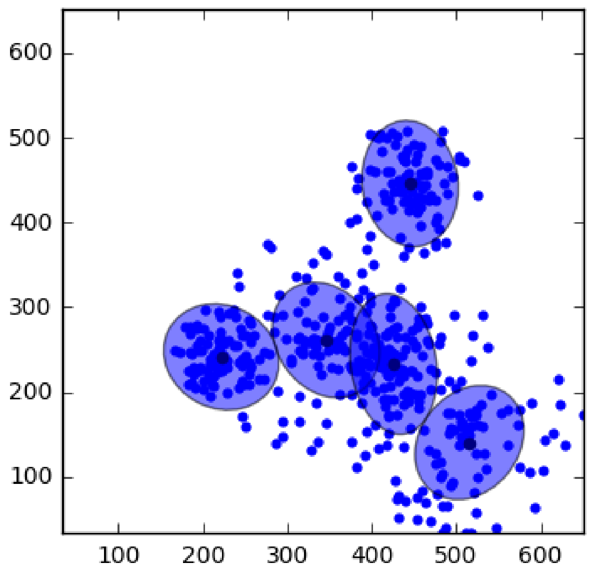
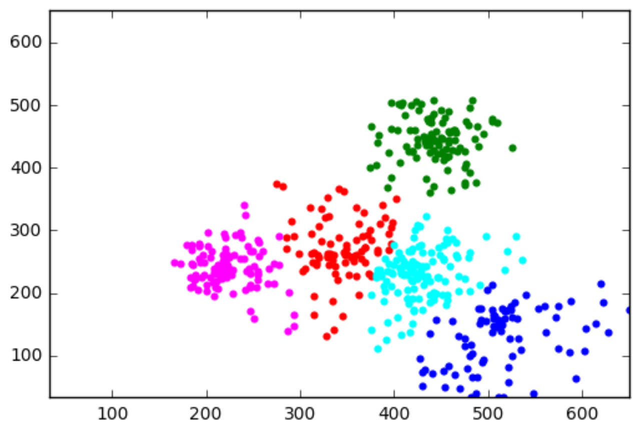
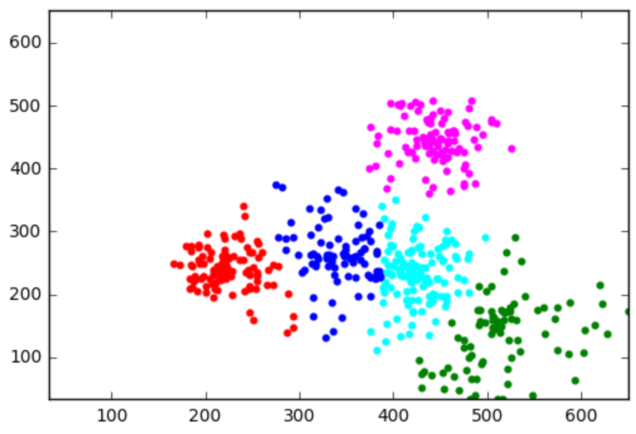
$$\frac{b-a}{max\,(a,b)}$$

To obtain this value, it is as simple as a call to the score parameter:

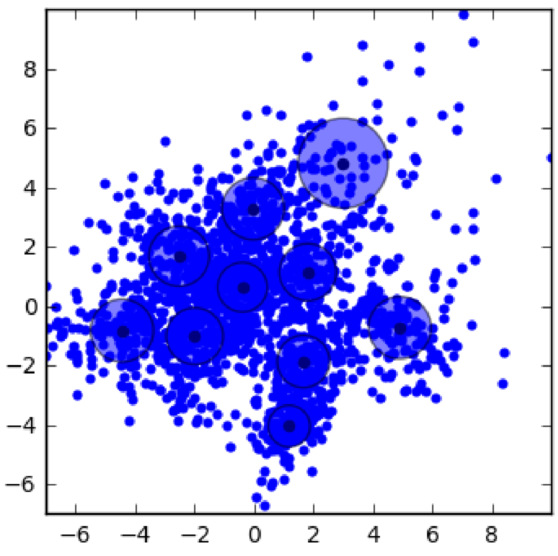
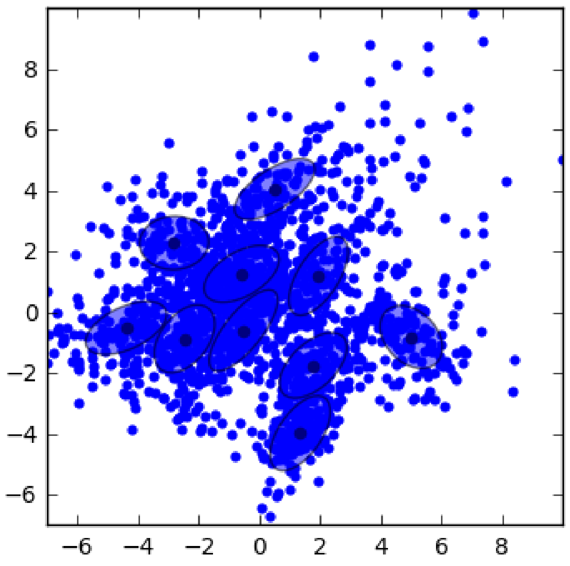
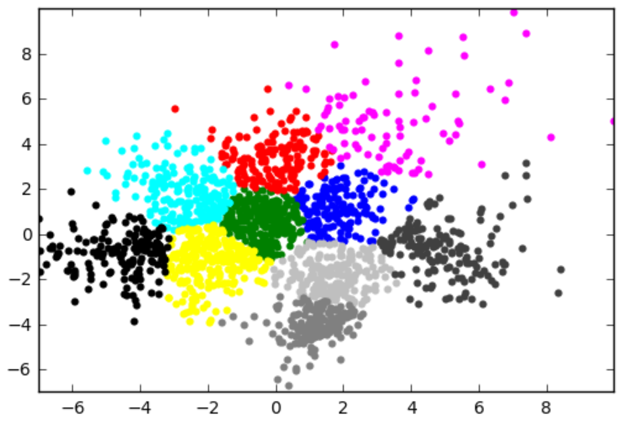
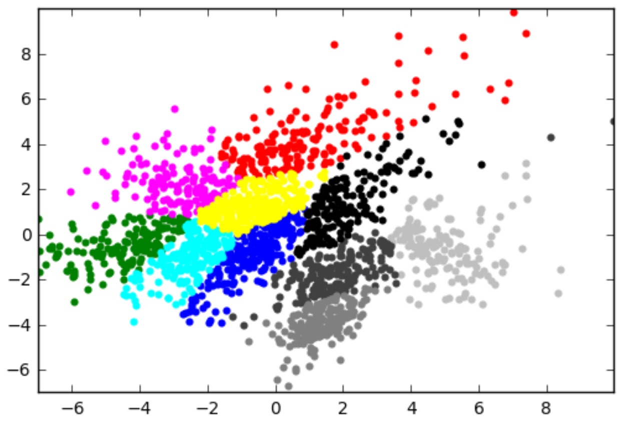
```
Silhouette_coefficient = fc.score
>> 0.46
```

Comparison: FCM vs GKFCM

Let’s see some plots comparing the results obtained with both methods.

Fake data: Summary table	
FCM	GKFCM
	
	

We can see in this plot, that the shape of the clusters are different, being the GKFCM ones, ellipsoid shaped clusters. A few points change from one cluster to another between both algorithms.

MNIST data (PCA): Summary table	
FCM	GKFCM
	
	

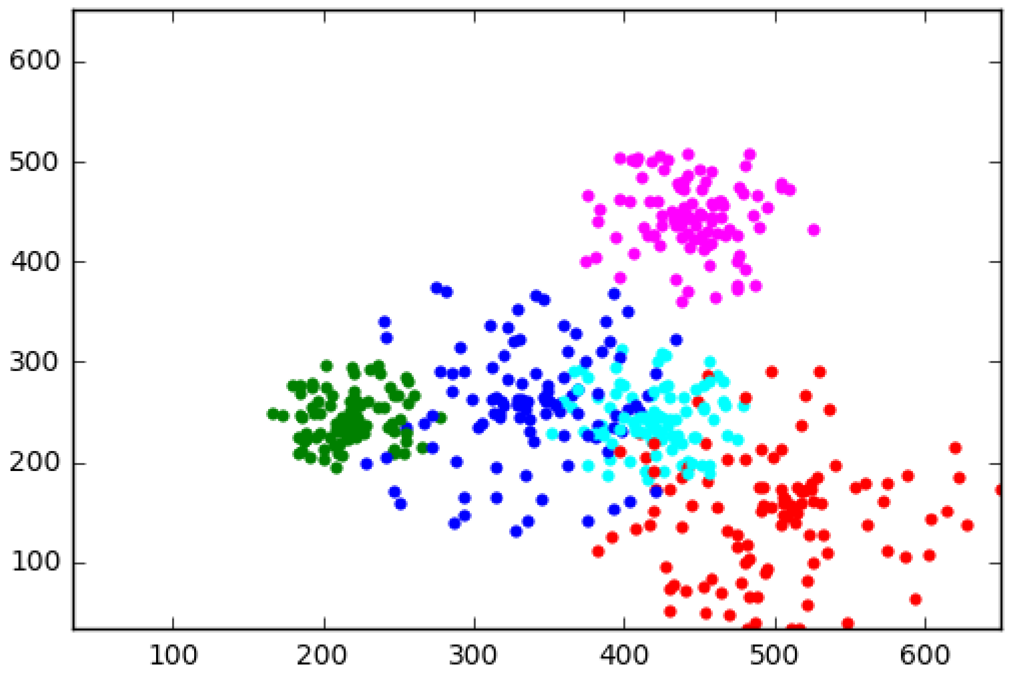
In this example, we can see the MNIST data reduced to 2 dimensions by the use of Principal Components Analysis (PCA). In the resulting clusters it is easily seen that clusters in GKFCM are quite different from the ones in FCM. In FCM clusters are circle-shaped, and so, every single point is related to the nearest cluster, while in GKFCM clusters are more elongated, this is the effect of the Mahalanobi distance, that produce ellipsoid clusters.

NOTE: it is important in every single clustering process, to try to understand the data, and if the objective is to classify and separate correctly the clusters, to find a correct representation of the data. In this example, PCA has been used with 2 components, to show the cluster shapes, but it is not the best representation.
In PCA it is recommended to reach a 95% of variance of the data, and with the use of only two components, this example reached only 21% of variance. In the following section we will try another approach to reach a better approach.

Results

In order to compare performance among different algorithms, different metrics have been used.
The methods used can be differentiated in two groups. The first one, being for datasets from which we have the ground truth, and the second one, for unsupervised data without ground. Usually in unsupervised clustering, the ground is not known, and so, it has non-sense to use a measure like the accuracy. In this case, it is used to compare how the different methods fit the original cluster distribution.
A simple algorithm that is commonly used when clustering in an unsupervised manner, is the K-means algorithm. In order to show the performance of FCM in clustering tasks, some metrics between these different methods have been used.

Artificial data



FCM
Cluster 0 has 18.0 incorrect samples classified from 100.0 (accuracy: 82.0%)
Cluster 1 has 2.0 incorrect samples classified from 100.0 (accuracy: 98.0%)
Cluster 2 has 28.0 incorrect samples classified from 100.0 (accuracy: 72.0%)
Cluster 3 has 18.0 incorrect samples classified from 100.0 (accuracy: 82.0%)
Cluster 4 has 6.0 incorrect samples classified from 100.0 (accuracy: 94.0%)
Total accuracy: 85.6
Adjusted rand index: 0.6943865301724527
Silhuette coefficient: 0.48926213738659535

GKFCM

```
Cluster 0 has 29.0 incorrect samples classified from 100.0 (accuracy: 71.0%)
Cluster 1 has 15.0 incorrect samples classified from 100.0 (accuracy: 85.0%)
Cluster 2 has 6.0 incorrect samples classified from 100.0 (accuracy: 94.0%)
Cluster 3 has 19.0 incorrect samples classified from 100.0 (accuracy: 81.0%)
Cluster 4 has 2.0 incorrect samples classified from 100.0 (accuracy: 98.0%)
Total accuracy: 85.8
Adjusted rand index: 0.6992439091017185
Silhuette coefficient: 0.47649106295772514
```

K-means

```
Cluster 0 has 32.0 incorrect samples classified from 100.0 (accuracy: 68.0%)
Cluster 1 has 21.0 incorrect samples classified from 100.0 (accuracy: 79.0%)
Cluster 2 has 2.0 incorrect samples classified from 100.0 (accuracy: 98.0%)
Cluster 3 has 6.0 incorrect samples classified from 100.0 (accuracy: 94.0%)
Cluster 4 has 16.0 incorrect samples classified from 100.0 (accuracy: 84.0%)
Total accuracy: 84.6
Adjusted rand index: 0.6744332696090998
Silhuette coefficient: 0.49181325218755
```

	FCM	GKFCM	K-means
Accuracy	85.6%	85.8%	84.6%
Adjusted Rand Index	0.69	0.7	0.67
Silhuette coefficient	0.49	0.48	0.49

In this simple example, the results of FCM as well as GKFCM outperforms K-means in terms of accuracy (it is a strange measure, because usually the ground is not provided and has non-sense to calculate this metrics for this kind of algorithms). In terms of clustering metrics, all three performs in a similar manner.

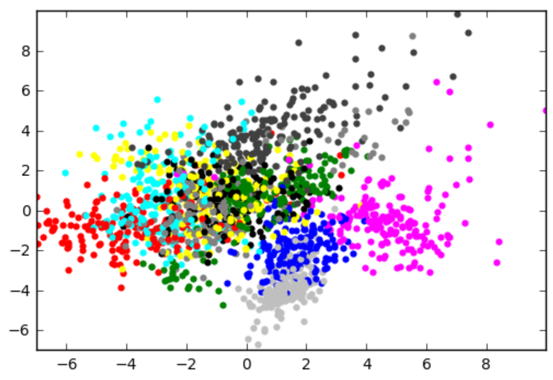
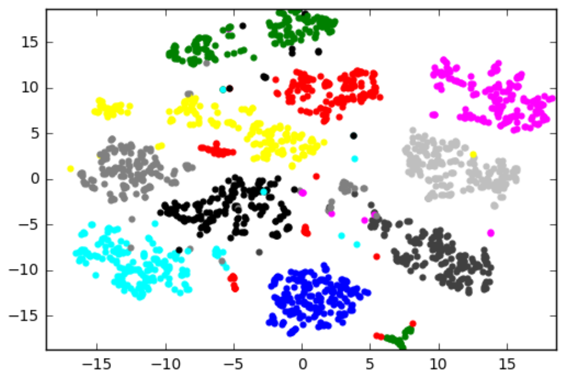
Real data : MNIST

In order to test how well these algorithms would perform in a real world example, the MNIST handwritten digits database has been used. It consists in hundreds of handwritten digits from 0 to 9, and for each image a label indicating which number is is provided.

Two approaches to show the importance of the preprocessing of the data have been used. The first one was to consider a simple PCA of the data, to reduce up to 2 components (in order to make it easy to plot). The second one is to optimize the distances between clusters in order to separate as much every single cluster, and also make it visible in a 2D plot.

How could the data be represented in a way that each cluster is as well characterized as possible? An approach could be to reduce dimensionality by the use of *t-distributed Stochastic Neighbor Embedding* (T-SNE).

The package sklearn.manifold.TSNE is an implementation that it will be used in MNIST in order to try to improve the separability of the clusters.

MNIST with PCA (2 components)	MNIST with T-SNE
	
Variance explained (ratio): Component 1: 0.12033916 Component 2: 0.09561054	

As it can be seen, the use of T-SNE makes it easier to clusterize data, even visually it is simpler to classify it.

PCA (2 components)	FCM	GKFCM	K-means
Accuracy	54%	53%	54.2%
Adjusted Rand Index	0.32	0.33	0.33
Silhuette coefficient	0.37	0.31	0.38

T-SNE	FCM	GKFCM	K-means
Accuracy	84%	80%	88.26%
Adjusted Rand Index	0.71	0.7	0.77
Silhuette coefficient	0.47	0.47	0.49

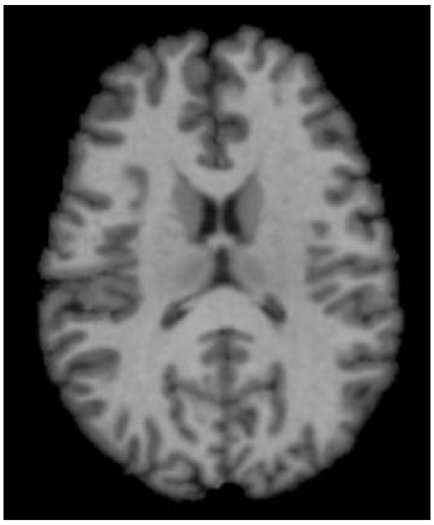
In PCA all perform in a similar way, while in T-SNE seems that K-means outperforms fuzzy methods in all metrics used. In other words, K-means in this case suits better the distribution of the data.

Obviously the use of GKFCM and FCM in a “Hard Clustering” mode is not the objective of both algorithms, so the silhouette coefficient is the best metric to be used in unsupervised clustering.

Real-life example: MRI data

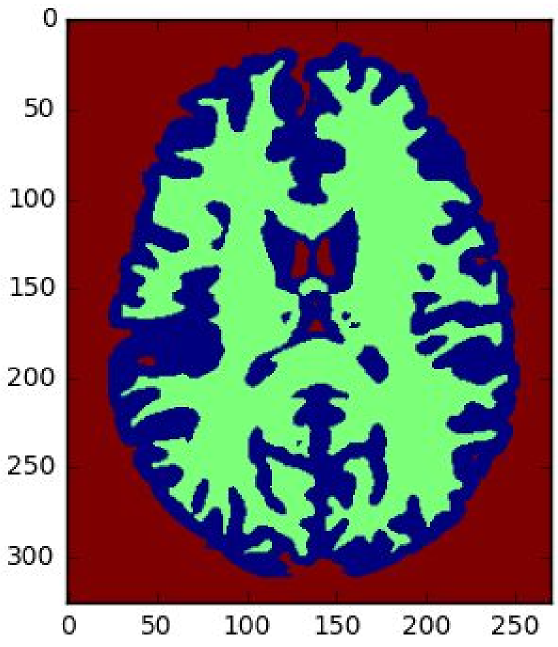
Fuzzy clustering is oftenly used in tasks like segmentation, due to the fact that every voxel can belong to different clusters, and this can introduce new options to differentiate for example the tissues of a brain.

In this example, a single T1-weighted MRI image has been used in order to show how FCM could be used (and in fact, it is used) in neuroimaging.
The image considered is the following one taken from a Siemens 1.5T from the hospital Sant Joan de Déu.
In this case, in order to make it easy to view, it has been used a single slice of the brain. FCM is a slow algorithm, and for a single image containing 88020 pixels takes more than a minute, so considering that the resolution of this MRI scanner produces more than 150 axial slices, it could be a really slow process to apply it to a whole brain image without any optimization.
The original image:

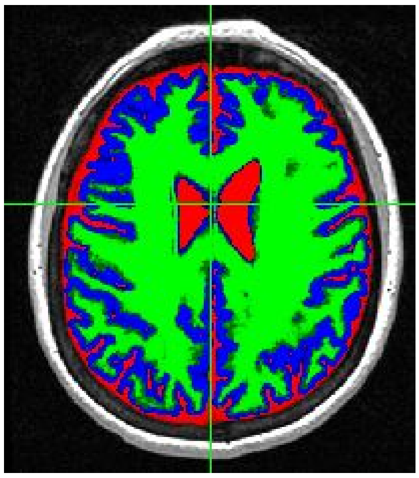


In MRI segmentation, there are three main components that are important to make further analysis in neuroimaging:

1. White matter: High intensity signal (white-ish)
2. Grey matter: intermediate signal intensity (grey-ish)
3. Cerebrospinal fluid: Low intensity values (black-ish)



This is the segmented image by using FCM, so that for each pixel we have a ratio of belonging to white matter, to grey matter or to cerebrospinal fluid. The generated image has been created considering the highest probability cluster.
So, in this example, brown-red color corresponds to CSF (or the out-brain zone, that in real-world examples would have been masked out).
Green color corresponds to white matter, and dark blue corresponds to grey matter.



This image corresponds to a segmentation performed by one of the most famous and used softwares to segment images, Statistical Parametrical Mapping (SPM).
As can be seen, the segmentation seems to have similar results, the differences between both images are due to the fact that SPM also performs a register to a common space, and that the slice is not exactly the same.
Obviously we have done a simple clustering, and SPM also uses different templates and statistical maps to assign each voxel to the different tissues (or clusters in our case).

Structure of the code

The code is structured in three main files.
A base file that contains common methods of both algorithms and also some other functionalities like the random center generation, called BaseClustering.py.
This base file is called from the other two files: FCM.py and GKFCM.py.
FCM contains the class FCM which corresponds to the Fuzzy C-means algorithm, and GKFCM contains the class GKFCM which corresponds to the Gustafson-Kessel FCM algorithm.

By placing the files in the same folder and importing FCM and GKFCM, it is as simple as write ‘import FCM, GKFCM’ to be able to call the functions implemented inside both methods.

Conclusion

Fuzzy C-means and Gustafson-Kessel Fuzzy C-means performs similar to K-means when used as a Hard clustering approach. But this is not the main objective of these algorithms. The best part of using fuzzy clustering is to obtain a soft-clustering approach, so to have ratios of each point belonging to every single cluster. In the examples shown in this document, different methods have been proved to demonstrate that the results obtained from these algorithms are quite good related to one of the most known clustering algorithms (K-means).
The main drawback of these algorithms is the computation time, because the amount of calculation done is much important in fuzzy clustering than in hard clustering.

One of the reasons to use these methods, is when we can make use of other information to combine with the memberships obtained to define the final clusters.
For example, in brain MRI segmentation, it is commonly considered to use a statistical map, where each region has a probability of being White Matter, Grey Matter or Cerebrospinal Fluid. So by using Fuzzy Clustering we will obtain a probability of each region being one tissue or another, and by combining the statistical information with the information obtained from the membership function we can improve the segmentation.

Between both algorithms, we can consider one or another by knowing the nature of the data. FCM considers the clusters to be circle-shaped, and GKFCM considers ellipsoid-based shapes. Both approaches can perform good depending on the data, so, as always, the knowledge of how data is structured, and choosing the right algorithm will let us improve our clustering approaches.

References

1. Graves, D. and Pedrycz, W.: Fuzzy C-means, Gustafson-Kessel FCM, and Kernel-based FCM: A Comparative Study (2007)
2. Marie-Jeanne Lesot and Rudolf Kruse: Gustafson-Kessel-like clustering algorithm based on typicality degrees (2007)
3. E. Gustafson and W. Kessel. Fuzzy clustering with a fuzzy covariance matrix. In Proc. of IEEE CDC, (1979).