Fuzzy Clustering Algorithms for Image Segmentation

Release 1.00

Alberto Rey[1], Alfonso Castro[1] and Bernardino Arcay[1]

December 9, 2011

[1]Faculty of Computer Science, University of A Coruña, Spain

Abstract

In this document we present the implementation of three fuzzy clustering algorithms using the Insight Toolkit ITK www.itk.org. Firstly, we developed the conventional Fuzzy C-Means that will serve as the basis for the rest of the proposed algorithms. The next algorithms are the FCM with spatial constraints based on kernel-induced distance and the Modified Spatial Kernelized Fuzzy C-Means. Both of these introduce a Kernel function, replacing the Euclidean distance of the FCM, and spatial information into the membership function.

These algorithms have been implemented in a threaded version to take advantage of the multicore processors. Moreover, providing an useful implementation make it possible that classes work with 2D/3D images, different kernels and spatial shapes.

We included the source code as well as different 2D/3D examples, using several input parameters for the algorithms and obtaining the results generated on 2D/3D CT lung studies.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3331]

Contents

1 Introduction

The basis of fuzzy set theory, established by Zadeh [18], introduced the idea of uncertainty of belonging described by a membership function. Since this article, several research areas have used his technique to solve and model complex problems, due its good results in classifying the ambiguous information. Fuzzy logic has also been applied to computerized image analysis taking advantages of its insensibility to noise and the ability to easily handle multidimensional features presented in most digital images.

Traditionally, clustering algorithms divide up a data set into classes or clusters, where objects assigned to the same cluster are homogeneous according to some features, such as intensity or texture; and where the objects classified to different regions are not similar. Nevertheless, there is usually no sharp boundary between clusters so the clasical hard clustering methods, such as the k-means [12], are not the most suitable for this kind of data. By contrast, within fuzzy logic, one of the most applied techniques are fuzzy clustering algorithms. These algorithms have considerable benefits than hard ones, because they could retain much information from the input image and they are not forced to classify each pixel of an image exclusively to one class. In this sense, fuzzy clustering algorithms allow objects to belong to multiple clusters, using a membership factor, that indicates with which grade is classified each object into each segmented cluster.

Different based fuzzy clustering algorithms, particurlarly Fuzzy C-means and its variants, has been widley applied in the task of image segmentation [3]. These algorithms classify the image into clusters, by iteratively minimizing a cost function in the feature domain that is dependent on the distance of the pixels to the clusters centers, also named centroids.

The conventional FCM algorithm [2] had been improved to take into consideration local spatial information of pixels, based on the ground that the pixels in the nearby neighborhood has similar feature data. Chuang et al. [7], incorporated a second phase to the FCM where the membership of each pixel is updated using spatial information that may fortify or reduce the weighting of a pixel to a cluster. By constrast, Zhang et al. [20], modified the objective function incorporating a penalty term that contain spatial neighborhood information acting as a regularizer and biases the solution toward piecewise-homogeneous labeling.

On the other hand, in the last years several linear methods have been generalized to deal with the problem

of nonlinear separability of classes by projecting the input data to a higher dimensional feature space in a nonlinear manner using Mercer kernels. For example, Supervised Vector Machines (SVM) [15, 9], kernel Principal Component Analysis (KPCA) [16] and kernel Fisher linear discriminat analysis [14]. This idea has been applied in fuzzy clustering algorithms [10] and in particular on the FCM [17, 19, 6, 5, 20] to remove of its constraints. Zhang et al. [20] replace the original Euclidean norm metric of the FCM with a new kernel-induced distance where prototypes reside in the feature space. They have demonstrated that this method is more robust to noise and resolve the tendency of the FCM to partition data points into cluster of hyperspherical shape with an equal number of data points in each cluster. Wu et al. [17], proposed a Fuzzy kernel C-Means which integrates a Mercel kernel function and where the prototypes are located in the kernel space. It is not only suitable for clusters with the spherical shape, but also other shapes.

At present the tendency of development fuzzy clustering algorithms is to combine several algorithms in order to improve the results obtained with each single algorithm. In this sense, we introduce two algorithms that follow this rule in addition to the FCM that serve as a basic for these. This paper introduces the development of three fuzzy clustering algorithms: FCM , KFCM_S, MSKFCM.

2 Algorithms proposed

In this section we describe the basic concepts of each method implemented.

2.1 Fuzzy C-Means (FCM)

The Fuzzy C-Means was introduced by Bezdek [3], it permits divide a finit colection of elements $X = \{x_1,...,x_n\} \subset R^P$ into c fuzzy sets by minimizing the following objective function:

$$JS_m^{\phi} = \sum_{i=1}^{c}\sum_{k=1}^{N} u_{ik}^m \|x_k - v_i\|^2 \tag{1}$$

where c is the number of clusters, N the number of data points, m is a weighting parameter that determines the amount of fuzziness of the resulting clasification. The matrix U, that contains the elements u_{ik}, represents a partition matrix satisfying $U \subset \{u_{ik} \in [0,1] / \sum_{i=1}^{c} u_{ik} = 1, \forall k \; and \; 0 < \sum_{k=1}^{N} u_{ik} < N, \forall i\}$. The elements $\{v_i\}_{i=1}^{c}$ are the centers or prototypes of the clusters.

The main steps of this algorithm are:

1. Fix c, $t_{\max}$, $m > 1$ and $\varepsilon > 0$. Initiliaze the prototypes $V^{(0)}$.
2. For $t = 1,2,3,\ldots,t_{\max}$,

   Obtain the membership factor of each element to each cluster:

   $$u_{ik} = \left(\sum_{j=1}^{c} \frac{\|x_k - v_i\|}{\|x_k - v_j\|}^{\frac{2}{m-1}} \right)^{-1} \tag{2}$$

   Calculation of the new centroids of the image:

   $$v_i = \frac{\sum_{k=1}^{N} u_{ik}^m \, x_k}{\sum_{k=1}^{N} u_{ik}^m} \tag{3}$$

3. If the threshold remains below a specific value $\|V^t - V^{t+1}\| \leq \varepsilon$, stop. Otherwise, return to Step 2.

2.2 FCM with spatial constraints based on kernel-induced distance (KFCM_S)

This was proposed in [6] and is the kernelized version of the FCM_S (Fuzzy C-Means with spatial constraints) algorithm presented in the same work, which applies a penalty factor that contains spatial neighborhood information. Kernelized methods introduce a kernel function that allows us to transform the original low dimension input space into a higher dimensional feature space, where complex nonlinear problems can be treated and solved in a better way, as probed by Cover [8]. In this method the prototypes are determined in the original input space and are implicity mapped to the kernel feature space through a kernel function.

The present paper only considers the radial basis function kernel (Eq.4) and specifically two Gaussian derivations (Eq.5):

$$K(x,y) = exp\left(-\frac{(\|x-y\|^a)^b}{\sigma^2}\right) \tag{4}$$

$$K(x,y) = exp\left(-\frac{\|x-y\|^2}{\sigma^2}\right) \ , \ K(x,y) = exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right) \tag{5}$$

If we decided to use other Kernel functions that satisfy the Mercel conditions [13], new expressions for the algorithm will be derived.

Therefore, modifying the objective function of the FCM in order to introduce the kernel function and add the penalty factor, we obtain the final objective function:

$$JS_m^{\phi} = \sum_{i=1}^{c}\sum_{k=1}^{N} u_{ik}^m(1-K(x_k,v_i)) + \frac{\alpha}{N_R}\sum_{i=1}^{c}\sum_{k=1}^{N} u_{ik}^m \sum_{r\varepsilon N_k}(1-K(x_r,v_i)) \tag{6}$$

where N_k represents a window (in this work could have different shapes: box, sphere, ball, annulus, ...; and 2D or 3D dimension) which includes the neighbors of pixel x_k (without considering it), N_r is the cardinality of N_k, and α ($0<\alpha<1$) is a parameter that controls the effect of the penalty term. Deriving the Eq. 6 respect the variables u_{ik} and v_i obtains the following two conditions that minimize the function. An iterative algorithm can be derived from the above conditions.

In the initialization of the algorithm, the parameters c, i.e. the number of clusters, the initial class centroids, the initial memberships, the threshold epsilon, α and m must be determined.

In the first step of the iterative process, the memberships are calculated as follows:

$$u_{ik} = \frac{\left((1-K(x_k,v_i)) + \frac{\alpha}{N_R}\sum_{r\varepsilon N_k}(1-K(x_r,v_i))^m\right)^{\frac{-1}{(m-1)}}}{\sum_{j=1}^{c}\left((1-K(x_k,v_j)) + \frac{\alpha}{N_R}\sum_{r\varepsilon N_k}(1-K(x_r,v_j)^m\right)^{\frac{-1}{(m-1)}}} \tag{7}$$

Finally, the centroids are updated as follows:

$$v_i = \frac{\sum_{k=1}^{n} u_{ik}^m\left(K(x_k,v_i)x_k + \frac{\alpha}{N_R}\sum_{r\varepsilon N_k}K(x_r,v_i)x_r\right)}{\sum_{k=1}^{n} u_{ik}^m\left(K(x_k,v_i) + \frac{\alpha}{N_R}\sum_{r\varepsilon N_k}K(x_r,v_i)\right)} \tag{8}$$

As in the previous algorithm, repeat these steps until condition $\|V^t - V^{t+1}\| \leq \varepsilon$ is satisfied, where epsilon is a determined threshold.

2.3 Modified Kernelized Spatial Fuzzy C-Means (MSKFCM)

This algorithm, is a variant of the one proposed in [4] and consists in a combination of the algorithms SFCM [7] and KFCM [20]. The main difference lies in the use of a different implementation of the KFCM, in this regard, is solved the large computational cost presented in the original implementation of the algorithm, allowing to parallelize the code and significantly reduce the execution time, indispensable when it comes to execution of comprehensive studies. As in the previous implemented algorithm, the prototypes are determined in the original input space and we only consider the Gaussian radial basis function kernel.

The main objective of this implementation is to combine the core strengths of each of the two algorithms used, that is, homogeneity and insensitivity to noise. Combining the main steps of these algorithms is achieved an iterative method which consists of the following phases:

1. Selection of initial parameters: the number of clusters in which you want to split the image, a sample of each cluster and the values for the parameters p, q for the calculation of spatial membership.
2. Calculation of the membership factor:

$$u_{ik} = \frac{(1 - K(x_k, v_i))^{-1/(m-1)}}{\sum_{j=1}^{c} (1 - K(x_k, v_j))^{-1/(m-1)}} \tag{9}$$

3. Updating the membership factor, applying:

$$u'_{ik} = \frac{u_{ik}^p h_{ik}^q}{\sum_{j=1}^{c} u_{jk}^p h_{jk}^q} \tag{10}$$

   where $h_{ik} = \sum_{z \in NB(x_k)} u_{iz}$. NB represents a window that could have different shapes and dimension.

4. Calculation of new centroids:

$$v_i = \frac{\sum_{k=1}^{n} {u'_{ik}}^m K(x_k, v_i) x_k}{\sum_{k=1}^{n} {u'_{ik}}^m K(x_k, v_i)} \tag{11}$$

5. If the error remains below than a certain threshold $\|V^t - V^{t+1}\| \leq \varepsilon$, stop. In other case, return to Step 2.

3 Implementation

In the following subsections will be presented the ITK classes coded to implement the previous fuzzy algorithms and several helper classes needed by these classifiers. Essentially, there are three global types of classes: the base class for all fuzzy classifiers *itk::FuzzyClassifierInitializationImageFilter*, classes that implement each algorithm named as *itk::FCMClassifierInitializationImageFilter, itk::SKFCMClassifierInitializationImageFilter and itk::MSFKCMClassifierInitializationImageFilter* , the class that implements the defuzzification on a membership image *itk::FuzzyClassifierImageFilter*, kernel functions implementations and other auxiliary classes.

The base class of all the implemented clustering algorithm is *itk::FuzzyClassifierInitializationImageFilter*. It derives from *itk::ImageToImageFilter*, which provide most of the functionality needed for handle images.

This is a template class that takes an image as input and produce a membership image as output. It takes three template parameters:

- The type of the input image.
- The type of the membership factors. Dafaults to *double*.
- The type of the values of the centroids. Dafaults to *double*.

It has the following attributes:

- *m_NumberOfClasses* is the number of clusters in which the input image will be classified. Its type is *unsigned int*.
- *m_Centroids* is a vector with the initial values of the centroids. Its type is *std::vector<itk::Vector<CentroidType,InputImageDimension>>*.
- *m_MaximumNumberOfIterations* is the maximum number of iterations allowed. Its type is *unsigned int*.
- *m_MaximumError* is the error threshold used to stop the iterative process. Its type is *double*.
- *m_Error* is the error obtained between each iteration of the algorithms. Its type is *double*.
- *m_M* is a parameter that control the degree of fuzziness in the clusters. Its value must be greater than 1. If *m_M>1* the degree of fuzziness increases among points in the decision space. Its type is *double*.
- *m_IgnoreBackgroundPixels* indicates whether to ignore the background pixels. Its type is *bool*.
- *m_BackgroundPixel* is the pixel value that represent the background in the input image. Its type is *InputImagePixelType*.
- *m_ImageToProcess* is an internal image obtained by a numeric conversion of the pixels of the input image to the types of the centroids. It avoids to cast each accesed pixel to the centroids type. Its type is *itk::Image<itk::Vector<CentroidType,InputImageDimension>,InputImageDimension>*.

The output image is represented as an *itk::VectorImage*, in which each pixel is a vector that contains the memberships factors of each original pixel to the corresponding centroid. If the user set the variable *m_IgnoreBackgroundPixels* to true, the membership values for the pixels that correspond to background are set to -1.

The process to convert the fuzzy output, in this case a *itk::VectorImage* of membership factors, into a quantiafiable scalar output (label image) is made by the class *itk::FuzzyClassifierImageFilter*. This process is called defuzzification.

The class *itk::FuzzyClassifierImageFilter* is derived from *itk::ImageToImageFilter* and basically performs the assignment of each pixel of the fuzzy output, obtained after applying the algorithms presented, to a particular cluster. It takes two template parameters:

- The type of the input image.
- The type of the labels. Dafaults to *unsigned char*.

This assignment is performed by using the class *itk::Statistics::MaximumDecisionRule*, which returns the class label with the largest discriminant score for each pixel of the fuzzy image. The range of values for labels is [0, *m_NumberOfClasses*] and are assigned taking into account the following situations:

- If the membership factors of a pixel have the value -1, it is assume that this pixel correspond with a background pixel and must be ignored. The label value assigned for this pixels is the number of classes.
- In other case, the label value assigned to the query pixel correspond with the class returned by the decision rule. i.e. 0 to (m_NumberOfClasses-1) .

On the other hand, this base class provides to the derived classes the support for multithreading process. For that, we implement the methods *ThreadedGeneratedData(), BeforeThreadedGeneratedData() and AfterGenerateData()* in each subclass. The implementation of *GenerateData()* provided by the *itk::FuzzyClassifierInitializationImageFilter* handles memory allocation for the output image and the internal image use through the classification process, also is responsible for calling the *Initialize()* method that could be implemented in the subclasses and is used to initialize the parameters of some of the implemented algortihms.

3.1 Fuzzy C-Means (FCM)

In the Figure 1 is presented the inheritance diagram for the FCM algorithm presented in 2.1. As we commented previously, all of the implemented algorithms derived from the base class *itk::FuzzyClassifierInitializationImageFilter*.

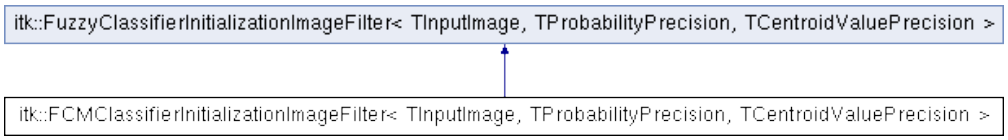


Figure 1: Inheritance diagram for the FCM algorithm.

The class *itk::FCMClassifierInitializationImageFilter* implements the FCM algorithm presented in this paper. This defines the methods *ThreadedGeneratedData(), BeforeThreadedGeneratedData() and AfterGenerateData()* to support multithreading process. In essence, in the *ThreadedGeneratedData()* method are implemented, in a threaded manner, the equations defined in the step 2.

It has the following attributes:

- *m_DistanceMetric* is a pointer to the Euclidean distance metric. Its type is *itk::Statistics::EuclideanDistanceMetric<CentroidType>*.
- *m_CentroidsModificationAttributesLock* is a mutex lock used to protect the modification of attributes through different threads. Its type is *itk::FastMutexLock*.

3.2 FCM with spatial constraints based on kernel-induced distance (KFCM_S)

Figure 2 shows the inheritance diagram for the KFCM_S algorithm. As the previous algorithm it is derived from the base class *itk::FuzzyClassifierInitializationImageFilter* and it is developed to support multithreading processes.

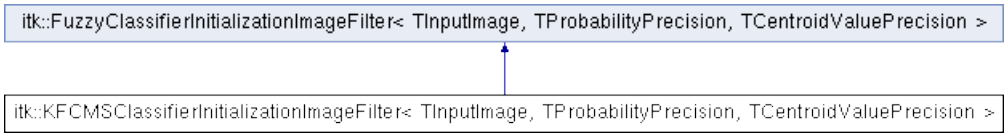


Figure 2: Inheritance diagram for the KFCM_S algorithm.

By contrast, this algorithm uses a different distance metric, replacing it by a kernel function that in this paper is the Gaussian radial basis function kernel implemented in *itk::Statistics::RBFKernelInducedDistanceMetric* as a general radial basis function. This function is a sublcass of *KernelInducedDistanceMetric* that is derived from *itk::Statistics::DistanceMetric* and declares a common interface for the kernel distance metrics, supporting the type of the pixels of the membership matrix, original matrix and the centroids. In this paper we only developed two kinds of kernel functions:

- *RBFKernelInducedDistanceMetric*. Is the Radial basis function presented in a general manner. Represent the first function in Equation 4 and also could be derived the first function of Equation 5.
- *GRBFKernelInducedDistanceMetric*. Gaussian radial basis function with different denominator. Represent the second function in Equation 5.

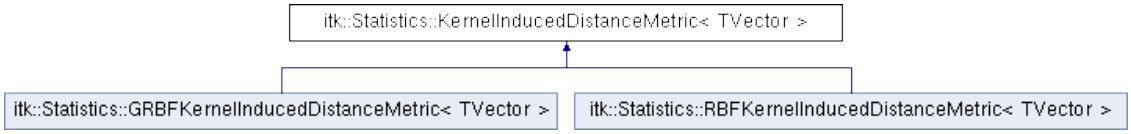


Figure 3: Inheritance diagram for the RBF kernel function.

The use of different kernel functions that satisfy the Mercer Theorem, i.e polynomial, sigmoid,..., will modify the equations presented in the Section 2 so they can not be used with the proposed implementation of this paper.

This algorithm defines in the Equation 7 and 8 a penalty factor that controls the influence of the neighboring of each pixel. In this sense, we define a *itk::ConstShapedNeighborhoodIterator* that allows the user to set different shapes for the neighborhood and iterate over it. This region is defined by using a *itk::FlatStructuringElement* with a particular shape (*Box*, *Ball*, *Cross*, *Annulus* or *Polygon*) and size (2D/3D element with different radius size in each direction), defined by the user. This elements specify which neighbors are active and which are inactive, but this method never consider the central pixel of the region so we call the method *DeactivateOffset(CentralIndex)* to remove it from the active index list.

This class has the following attributes:

- *m_Alpha* is a parameter that control the influence of the penalty factor. Its type is *double*.
- *m_KernelDistanceMetric* is a pointer to the kernel distance metric. Its type is *itk::Statistics::KernelInducedDistanceMetric<CentroidType>*.
- *m_CentroidsModificationAttibutesLock* is a mutex lock used to protect the modification of attributes through different threads. Its type is *itk::FastMutexLock*.

3.3 Modified Spatial Kernelized Fuzzy C-Means (MSKFCM)

This algorithm is implemented in *itk::MSKFCMClassifierInitializationImageFilter* following the same procedure as the previous algorithms to support multithreading and different shapes for the neighborhood. In Figure 4 is shown the inheritance diagram of this class.

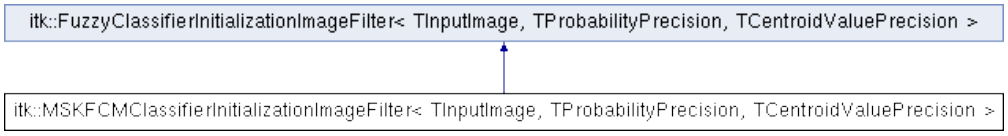


Figure 4: Inheritance diagram for the MSKFCM algorithm.

This class has the following attributes:

- *m_P* is a parameter that control the relative relevance of the membership factors. Its type is *double*.
- *m_Q* is a parameter that control the relative relevance of the neighborhood information. Its type is *double*.
- *m_KernelDistanceMetric* is a pointer to the kernel distance metric. Its type is *itk::Statistics::KernelInducedDistanceMetric<CentroidType>*.
- *m_CentroidsModificationAttibutesLock* is a mutex lock used to protect the modification of attributes through different threads. Its type is *itk::FastMutexLock*.
- *m_Barrier* is a standard barrier for synchronizing the execution of threads.

4 Usage

In this section are presented different examples with the use of the implemented algorithms in different situations.

4.1 FCM 2D classification pipeline

In this example, we read an image to classfiy it, applying the FCM algorithm, in a number of clusters defined by the user. The output image represent the label image with each pixel classified into its respectively cluster. The user must provide the values for the input parameters of this algorithm. This source code is avaliable in the file *FCMClassification2D.cxx*.

Firstly, are imported the headers of the FCM algorithm and the defuzzification class.

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "itkFCMClassifierInitializationImageFilter.h"
#include "itkFuzzyClassifierImageFilter.h"
```

Then are declared the types of the input/output images, in this case 2D images, and are assigned the parameters for the FCM object.

```
int
main(int argc, char * argv[])
{

  if (argc < 7)
    {
      std::cerr << "usage: " << argv[0] << " input output nmaxIter error m "
        "numThreads numClasses { centroids_1,...,centroid_numClusters } "
        "[ -f valBackground ]" << std::endl;
      exit(1);
    }

  const int dim = 2;
  typedef signed short IPixelType;

  typedef unsigned char OPixelType;

  typedef itk::Image<IPixelType, dim> IType;
  typedef itk::Image<OPixelType, dim> OType;

  typedef itk::ImageFileReader<IType> ReaderType;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  typedef itk::FuzzyClassifierInitializationImageFilter<IType>
      TFuzzyClassifier2D;

  typedef itk::FCMClassifierInitializationImageFilter<IType> TClassifierFCM;

  TClassifierFCM::Pointer classifier = TClassifierFCM::New();
  classifier->SetMaximumNumberOfIterations(atoi(argv[3]));
  classifier->SetMaximumError(atof(argv[4]));
  classifier->SetM(atoi(argv[5]));
  classifier->SetNumberOfThreads(atoi(argv[6]));

  int numClasses = atoi(argv[7]);
  classifier->SetNumberOfClasses(numClasses);
```

The centroids are initialized usign the values provided bu the user, with one value for each cluster specified.

```
  TFuzzyClassifier2D::CentroidArrayType centroidsArray;

  int argvIndex = 8;
  for (int i = 0; i < numClasses; i++)
    {
      centroidsArray.push_back(atof(argv[argvIndex]));
      ++argvIndex;
    }
  classifier->SetCentroids(centroidsArray);
```

If the user specifiy that the background must be ignored in the classification process, is called the method SetIgnoreBackgroundPixels(true) and set the background value to ignore those pixels.

```
  if ( (argc-1 > argvIndex ) && (strcmp(argv[argvIndex + 1], "-f") == 0) )
    {
    classifier->SetIgnoreBackgroundPixels(true);
    classifier->SetBackgroundPixel(atof(argv[argvIndex + 1]));
    }
```

Finally, is created the class that performs the defuzzification of the output of the FCM algorithm and construct the final image with the pixels classified in each cluster.

```
  classifier->SetInput(reader->GetOutput());

  typedef itk::FuzzyClassifierImageFilter<TClassifierFCM::OutputImageType>
      TLabelClassifier2D;
  TLabelClassifier2D::Pointer labelClass = TLabelClassifier2D::New();
  labelClass->SetInput(classifier->GetOutput());

  typedef itk::ImageFileWriter<OType> WriterType;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(labelClass->GetOutput());
  writer->SetFileName(argv[2]);

  try
    {
    writer->Update();
    }
  catch( itk::ExceptionObject & excp )
    {
    std::cerr << "ExceptionObject caught !" << std::endl;
    std::cerr << excp << std::endl;
    return EXIT_FAILURE;
    }

  return EXIT_SUCCESS;
}
```

4.2 KFCM_S 2D classification pipeline

In this example is classified an image using the KFCM_S algorithm. As we commnent in the Section 2.2, this algorithm needs that the user specify a Kernel function and a the shape of the neighborhood iterator. In this case, we use a Gaussian Radial Basis Function as the kernel and a Box shape as the neighborhood.

The source code is avaliable in the file *KFCMSClassification2D.cxx*.

Firstly, is imported a diferent header for this fuzzy algorithm and declared its *typedef*.

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkSimpleFilterWatcher.h"

#include "itkSKFCMClassifierInitializationImageFilter.h"
#include "itkFuzzyClassifierImageFilter.h"

int
```

```
main(int argc, char * argv[])
{

  if (argc < 10)
    {
      std::cerr << "usage: " << argv[0] << " input output nmaxIter error m alpha"
        "numThreads numClasses { centroids_1,...,centroid_numClusters } sigma radius"
        "[ -f valBackground ]" << std::endl;
      exit(1);
    }

  const int dim = 2;
  typedef signed short IPixelType;

  typedef unsigned char OPixelType;

  typedef itk::Image<IPixelType, dim> IType;
  typedef itk::Image<OPixelType, dim> OType;

  typedef itk::FuzzyClassifierInitializationImageFilter<IType>
      TFuzzyClassifier2D;

  typedef itk::SKFCMClassifierInitializationImageFilter<IType> TClassifierKFCMS;

  typedef itk::ImageFileReader<IType> ReaderType;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  TClassifierKFCMS::Pointer classifier = TClassifierKFCMS::New();
  classifier->SetMaximumNumberOfIterations(atoi(argv[3]));
  classifier->SetMaximumError(atof(argv[4]));
  classifier->SetM(atoi(argv[5]));
  classifier->SetAlpha(atof(argv[6]));
  classifier->SetNumberOfThreads(atoi(argv[7]));

  int numClasses = atoi(argv[8]);
  classifier->SetNumberOfClasses(numClasses);

  TFuzzyClassifier2D::CentroidArrayType centroidsArray;

  int argvIndex = 9;
  for (int i = 0; i < numClasses; i++)
    {
      centroidsArray.push_back(atof(argv[argvIndex]));
      ++argvIndex;
    }

  classifier->SetCentroids(centroidsArray);
  itk::SimpleFilterWatcher watcher(classifier, "KFCMS classifier");
```

In the following block we declare the Gaussian radial basis kernel function setting the values A=2 and B=1 to the general equation of the RBF function defined in 4. The sigma value is defined by the user.

```
  typedef TFuzzyClassifier2D::CentroidType TCentroid;
```

```
typedef itk::Statistics::RBFKernelInducedDistanceMetric<TCentroid>
    RBFKernelType;
RBFKernelType::Pointer kernelDistancePtr = RBFKernelType::New();
kernelDistancePtr->SetA(2.0);
kernelDistancePtr->SetB(1.0);
kernelDistancePtr->SetSigma(atoi(argv[argvIndex]));
classifier->SetKernelDistanceMetric(kernelDistancePtr);
```

Then is defined the shape of the structuring element needed by the neighborhood iterator. In this case, we declare an *itk::FlatStructuringElement* with *Box* shape. The radius of this element is defined by the user in the command line.

```
typedef typename itk::FlatStructuringElement<
    dim> StructuringElement2DType;
typename StructuringElement2DType::RadiusType elementRadius;
for (int i = 0; i < dim; i++)
  {
  ++argvIndex;
  elementRadius[i] = atoi(argv[argvIndex]);
  }
StructuringElement2DType structuringElement = StructuringElement2DType::Box(
    elementRadius);
classifier->SetStructuringElement(structuringElement);
```

The following code is the same as the previous example.

```
if ( (argc-1 > argvIndex ) && (strcmp(argv[argvIndex + 1], "-f") == 0) )
  {
    classifier->SetIgnoreBackgroundPixels(true);
    classifier->SetBackgroundPixel(atof(argv[argvIndex + 2]));
  }

classifier->SetInput(reader->GetOutput());

typedef itk::FuzzyClassifierImageFilter<TClassifierKFCMS::OutputImageType>
    TLabelClassifier2D;
TLabelClassifier2D::Pointer labelClass = TLabelClassifier2D::New();
labelClass->SetInput(classifier->GetOutput());

typedef itk::ImageFileWriter<OType> WriterType;
WriterType::Pointer writer = WriterType::New();
writer->SetInput(labelClass->GetOutput());
writer->SetFileName(argv[2]);

try
  {
  writer->Update();
  }
catch (itk::ExceptionObject & excp)
  {
  std::cerr << "ExceptionObject caught !" << std::endl;
  std::cerr << excp << std::endl;
  return EXIT_FAILURE;
```

```
    }

  return EXIT_SUCCESS;
}
```

4.3 MSKFCM 2D classification pipeline

In this example is presented the main differences between the code needed by the MSKFCM classfication algorithm and the previous ones. Basically, those differences lie in the use of another import headers and *typedef* declarations, the rest of the code is the same as the KFCM_S example.

The source code is avaliable in the file *MSKFCMClassification.cxx*.

```
#include "itkMSKFCMClassifierInitializationImageFilter.h"
#include "itkFuzzyClassifierImageFilter.h"

...


  typedef itk::FuzzyClassifierInitializationImageFilter<IType>
      TFuzzyClassifier2D;

  typedef itk::MSKFCMClassifierInitializationImageFilter<IType>
      TClassifierMSKFCM;

  TClassifierMSKFCM::Pointer classifier = TClassifierMSKFCM::New();
  classifier->SetMaximumNumberOfIterations(atoi(argv[3]));
  classifier->SetMaximumError(atof(argv[4]));
  classifier->SetM(atoi(argv[5]));
  classifier->SetP(atof(argv[6]));
  classifier->SetQ(atof(argv[7]));


...


  typedef itk::FuzzyClassifierImageFilter<TClassifierMSKFCM::OutputImageType>
      TLabelClassifier2D;


...
```

4.4 MSKFCM 3D classification pipeline

In this example, we read a 3D CT study and we classify it using the MSKFCM algorithm. Also, is selected a 3D ball as the structuring element for the neighborhood. Finally, we will write the labeled images obtained after the defuzzification process.

The source code is avaliable in the file *MSKFCMClassification3D.cxx*.

Fisrtly, we declare the classes needed to manage dicom studies and the classes needed by the clustering algorithm.

```
#include "itkGDCMImageIO.h"
#include "itkGDCMSeriesFileNames.h"
#include "itkImageSeriesReader.h"
#include "itkImageSeriesWriter.h"
#include "itkSimpleFilterWatcher.h"

#include "itkFuzzyClassifierInitializationImageFilter.h"
#include "itkMSKFCMClassifierInitializationImageFilter.h"
#include "itkFuzzyClassifierImageFilter.h"
```

Then, are defined the types of the input and output images and are created the objects needed to read a serie of 2D images, that correspond to the CT study.

```
int
main(int argc, char * argv[])
{

  if (argc < 11)
    {
      std::cerr << "usage: " << argv[0] << " input output nmaxIter error m P Q"
        "numThreads numClasses { centroids_1,...,centroid_numClusters } sigma"
        "radius [ -f valBackground ]" << std::endl;
      exit(1);
    }


  const int dim = 3;
  typedef signed short IPixelType;

  typedef unsigned char OPixelType;

  typedef itk::Image<IPixelType, dim> IType;
  typedef itk::Image<OPixelType, dim> OType;

  typedef itk::Image<OPixelType, 2> OType2D;

  typedef itk::GDCMImageIO DicomIOType;

  typedef itk::ImageSeriesReader<IType> DicomReaderType;

  typedef itk::GDCMSeriesFileNames NamesGeneratorType;

  DicomReaderType::Pointer reader = DicomReaderType::New();
  DicomIOType::Pointer dicomIO = DicomIOType::New();
  reader->SetImageIO(dicomIO);
  NamesGeneratorType::Pointer namesGenerator = NamesGeneratorType::New();
  namesGenerator->SetUseSeriesDetails(true);
  namesGenerator->SetDirectory(argv[1]);

  typedef std::vector<std::string> IdsContainerType;
  IdsContainerType ids;
  try
    {
    ids = namesGenerator->GetSeriesUIDs();
```

```
  }
catch (itk::ExceptionObject & excp)
  {
  std::cerr << "ExceptionObject caught !" << std::endl;
  std::cerr << excp << std::endl;
  return EXIT_FAILURE;
  }

std::string identifier = ids.begin()->c_str();

typedef std::vector<std::string> NamesContainerType;
NamesContainerType names = namesGenerator->GetFileNames(identifier);

reader->SetFileNames(names);
```

After to read the 3D image, is created the algorithm and are assigned its parameters,introduced by the user, by the same way as the previous examples.

```
typedef itk::FuzzyClassifierInitializationImageFilter<IType>
    TFuzzyClassifier2D;

typedef itk::MSKFCMClassifierInitializationImageFilter<IType>
    TClassifierMSKFCM;

typedef itk::FuzzyClassifierInitializationImageFilter<IType>
    TFuzzyClassifier;

TClassifierMSKFCM::Pointer classifier = TClassifierMSKFCM::New();
classifier->SetMaximumNumberOfIterations(atoi(argv[3]));
classifier->SetMaximumError(atof(argv[4]));
classifier->SetM(atoi(argv[5]));
classifier->SetP(atof(argv[6]));
classifier->SetQ(atof(argv[7]));
classifier->SetNumberOfThreads(atoi(argv[8]));

int numClasses = atoi(argv[9]);
classifier->SetNumberOfClasses(numClasses);

TFuzzyClassifier2D::CentroidArrayType centroidsArray;

int argvIndex = 10;
for (int i = 0; i < numClasses; i++)
  {
    centroidsArray.push_back(atof(argv[argvIndex]));
    ++argvIndex;
  }

classifier->SetCentroids(centroidsArray);
itk::SimpleFilterWatcher watcher(classifier, "MSKFCM classifier");
```

We use the Gaussian radial basis kernel.

```
typedef TFuzzyClassifier2D::CentroidType TCentroid;
typedef itk::Statistics::RBFKernelInducedDistanceMetric<TCentroid>
```

```
    RBFKernelType;
RBFKernelType::Pointer kernelDistancePtr = RBFKernelType::New();
kernelDistancePtr->SetA(2.0);
kernelDistancePtr->SetB(1.0);
kernelDistancePtr->SetSigma(atoi(argv[argvIndex]));
classifier->SetKernelDistanceMetric(kernelDistancePtr);
```

In this case, the structuring element is a Ball with a radius size specified by the user and with three components. So, this is a 3D ball.

```
typedef typename itk::FlatStructuringElement<dim> StructuringElement2DType;
typename StructuringElement2DType::RadiusType elementRadius;
for (int i = 0; i < dim; i++)
  {
    ++argvIndex;
    elementRadius[i] = atoi(argv[argvIndex]);
  }
StructuringElement2DType structuringElement = StructuringElement2DType::Ball(
    elementRadius);
classifier->SetStructuringElement(structuringElement);
```

If we deal with large 3D images, we can reduce a lot of proccess time ignoring the background.

```
if ( (argc-1 > argvIndex ) && (strcmp(argv[argvIndex + 1], "-f") == 0) )
  {
    classifier->SetIgnoreBackgroundPixels(true);
    classifier->SetBackgroundPixel(atof(argv[argvIndex + 2]));
  }

classifier->SetInput(reader->GetOutput());
```

Finally, are writen the labeled images obtained after the defuzzification:

```
typedef itk::FuzzyClassifierImageFilter<TFuzzyClassifier::OutputImageType>
    TLabelClassifier2D;
TLabelClassifier2D::Pointer labelClass = TLabelClassifier2D::New();
labelClass->SetInput(classifier->GetOutput());

typedef itk::ImageSeriesWriter<OType,OType2D> DicomWriterType;
DicomWriterType::Pointer writer = DicomWriterType::New();
writer->SetInput(labelClass->GetOutput());
writer->SetImageIO(dicomIO);
namesGenerator->SetOutputDirectory(argv[2]);
writer->SetFileNames(namesGenerator->GetOutputFileNames());
writer->SetMetaDataDictionaryArray(reader->GetMetaDataDictionaryArray());

try
  {
  writer->Update();
  }
catch (itk::ExceptionObject & excp)
  {
  std::cerr << "ExceptionObject caught !" << std::endl;
```

```
    std::cerr << excp << std::endl;
    return EXIT_FAILURE;
    }

  return EXIT_SUCCESS;
}
```

4.5 MSKFCM Opening 2D pipeline

In this example we extend the code provided in the example 4.3 to remove the objects, of a specified class of the labeled image obtained with the MSKFCM algorithm, with a size value greater than a given value provided by the user.

The source code is avaliable in the file *MSKFCMOpening2D.cxx*.

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkSimpleFilterWatcher.h"

#include "itkMSKFCMClassifierInitializationImageFilter.h"
#include "itkFuzzyClassifierImageFilter.h"

#include "itkBinaryShapeOpeningImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc < 14)
    {
      std::cerr << "usage: " << argv[0] << " input output nmaxIter error m P Q"
        " lambda backClass openClass numThreads numClasses { centroids_1,...,"
        "centroid_numClusters } sigma radius [ -f valBackground ]" << std::endl;
      exit(1);
    }

  const int dim = 2;
  typedef signed short IPixelType;

  typedef unsigned char OPixelType;

  typedef itk::Image<IPixelType, dim> IType;
  typedef itk::Image<OPixelType, dim> OType;

  typedef itk::ImageFileReader<IType> ReaderType;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  typedef itk::FuzzyClassifierInitializationImageFilter<IType>
      TFuzzyClassifier2D;
  typedef itk::MSKFCMClassifierInitializationImageFilter<IType>
      TClassifierMSKFCM;
```

```
TClassifierMSKFCM::Pointer classifier = TClassifierMSKFCM::New();
classifier->SetMaximumNumberOfIterations(atoi(argv[3]));
classifier->SetMaximumError(atof(argv[4]));
classifier->SetM(atoi(argv[5]));
classifier->SetP(atof(argv[6]));
classifier->SetQ(atof(argv[7]));
classifier->SetNumberOfThreads(atoi(argv[11]));

int numClasses = atoi(argv[12]);
classifier->SetNumberOfClasses(numClasses);

TFuzzyClassifier2D::CentroidArrayType centroidsArray;

int argvIndex = 13;
for (int i = 0; i < numClasses; i++)
  {
    centroidsArray.push_back(atof(argv[argvIndex]));
    ++argvIndex;
  }

classifier->SetCentroids(centroidsArray);
itk::SimpleFilterWatcher watcher(classifier, "MSKFCM classifier");

typedef TFuzzyClassifier2D::CentroidType TCentroid;
typedef itk::Statistics::RBFKernelInducedDistanceMetric<TCentroid>
    RBFKernelType;
RBFKernelType::Pointer kernelDistancePtr = RBFKernelType::New();
kernelDistancePtr->SetA(2.0);
kernelDistancePtr->SetB(1.0);
kernelDistancePtr->SetSigma(atoi(argv[argvIndex]));
classifier->SetKernelDistanceMetric(kernelDistancePtr);

typedef typename itk::FlatStructuringElement<dim> StructuringElement2DType;
typename StructuringElement2DType::RadiusType elementRadius;
for (int i = 0; i < dim; i++)
  {
    ++argvIndex;
    elementRadius[i] = atoi(argv[argvIndex]);
  }
StructuringElement2DType structuringElement = StructuringElement2DType::Box(
    elementRadius);
classifier->SetStructuringElement(structuringElement);

if ( (argc-1 > argvIndex ) && (strcmp(argv[argvIndex + 1], "-f") == 0) )
  {
    classifier->SetIgnoreBackgroundPixels(true);
    classifier->SetBackgroundPixel(atof(argv[argvIndex + 2]));
  }

classifier->SetInput(reader->GetOutput());

typedef itk::FuzzyClassifierImageFilter<TClassifierMSKFCM::OutputImageType>
    TLabelClassifier2D;
TLabelClassifier2D::Pointer labelClass = TLabelClassifier2D::New();
```

```
  labelClass->SetInput(classifier->GetOutput());
```

The following block code creates an itk::BinaryShapeOpeningImageFilter filter that allow us to manipulate the label image obtained with the defuzzification phase and remove the objects with a size lower than a specific value. By contrast, If we had set SetReverseOrdering(true), objects with a size greater than a specific value would have been removed from the image.

```
  typedef itk::BinaryShapeOpeningImageFilter< OType > LabelOpeningType;
  LabelOpeningType::Pointer opening = LabelOpeningType::New();
  opening->SetInput( labelClass->GetOutput() );
  opening->SetBackgroundValue( atoi(argv[9]) );
  opening->SetForegroundValue( atoi(argv[10]) );
  opening->SetLambda( atoi(argv[8]) );
  opening->SetReverseOrdering( false );
  opening->SetAttribute( "PhysicalSize" );

  typedef itk::ImageFileWriter<OType> WriterType;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(opening->GetOutput());
  writer->SetFileName(argv[2]);

  try
    {
    writer->Update();
    }
  catch (itk::ExceptionObject & excp)
    {
    std::cerr << "ExceptionObject caught !" << std::endl;
    std::cerr << excp << std::endl;
    return EXIT_FAILURE;
    }

  return EXIT_SUCCESS;
}
```

5 Results

In this section we present the results obtained after to applied several classification examples over several preprocessed CT lung images.

The initial images were provided by the Lung Image Database Consortium (LIDC) database [1]. These images are stored using DICOM standard and has a size of 512x512 pixels.

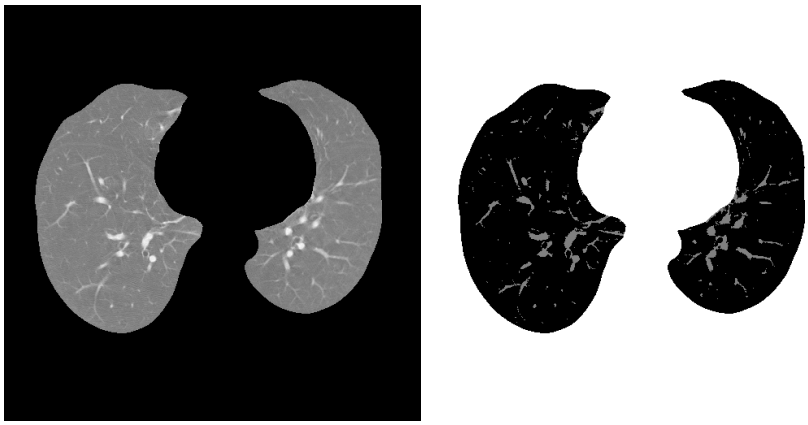
The code of the algorithms and all the examples included were tested both ITK 3.20.1v and 4.0-rc03v.

5.1 FCM result

The result presented in 5 can be obtained by using the example *FCMClassification2D.cxx* on the input image *inputIm1.dcm* and with the following input parameters:

```
./FCMClassification2D inputIm1.dcm outIm1.dcm 500 0.001 2 4 2 -940 -450 -f -2000
```

In this example, we want to classify the image in three cluters, that represent the lung tissue, possible nodules and the background; but as the pixels of the background are ignored only have to be specified the intial value of the two centroids.



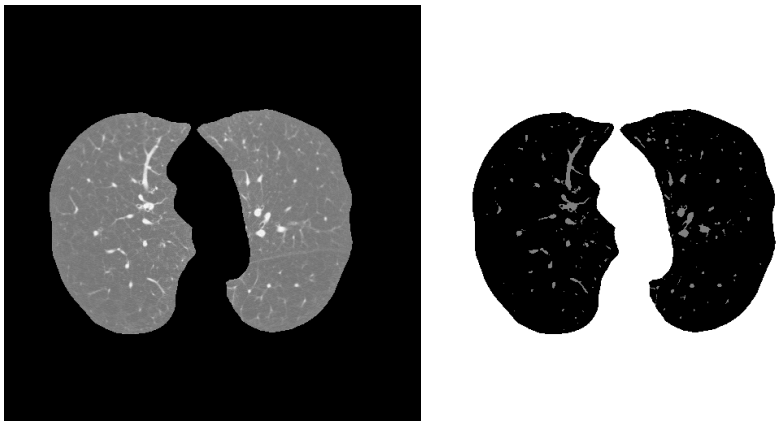
(a) Input Image.

(b) Defuzzied image obtained with the FCM algorithm.

Figure 5: Parameters: MaxIterations = 500, ε = 0.001, α = 2, NumOfThreads = 4, NumOfClasses = 2, $\{V^1$=-900 V^2=-450$\}$, backgroundValue = -2000 .

The result shown in 6 is obtained with following the comand line:

```
./FCMClassification2D inputIm3.dcm outIm3.dcm 500 0.001 2 2 2 -840 -350 -f -2000
```



(a) Input Image.

(b) Defuzzied image obtained with the FCM algorithm.

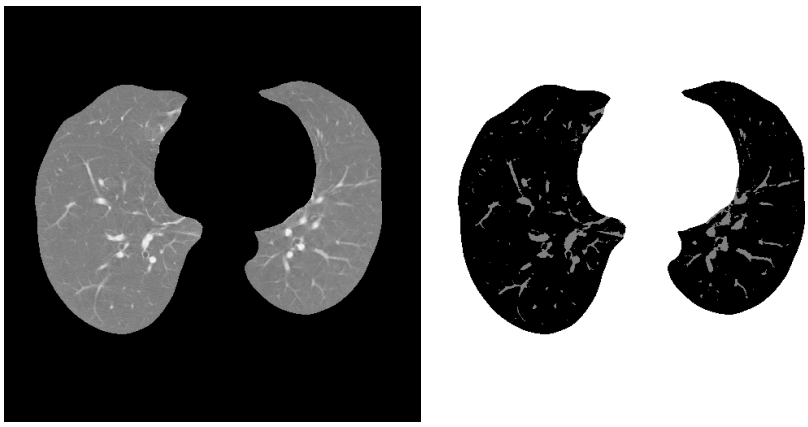
Figure 6: Parameters: MaxIterations = 500, ε = 0.001, α = 2, NumOfThreads = 2, NumOfClasses = 2, $\{V^1$=-840 V^2=-350$\}$, backgroundValue = -2000 .

5.2 KFCM_S result

The result shown in Image 7 can be obtained by using the example *KFCMSClassification2D* on the input image *inputIm1.dcm* and with the following input parameters:

```
./KFCMSClassification2D inputIm1.dcm outIm1.dcm 500 0.001 1.5 1 4 2 -940 -450 750 1 1 -f -2000
```

As in the previous example we want to classify the image in three clusters. In this case is used a structuring element with a radius of 1x1 and the pixels of the background are ignored.



(a) Input Image.

(b) Defuzzied image obtained with the KFCM_S algorithm.

Figure 7: Parameters: MaxIterations = 500, ε = 0.001, m = 1.5, α = 1, NumOfThreads = 4, NumOfClasses = 2, $\{V^1$=-940 V^2=-450$\}$, σ = 750, StructRadius = 1x1, backgroundValue = -2000 .

The result shown in 8 is obtained with the following comand line:

```
./KFCMSClassification2D inputIm3.dcm outIm3.dcm 500 0.001 1.4 1 4 2 -840 -350 850 1 1 -f -2000
```

5.3 MSKFCM Opening result

The result presented in Image 9 can be obtained by using the example *MSKFCMOpening2D.cxx* on the input image *inputIm1.dcm* using the following input parameters:

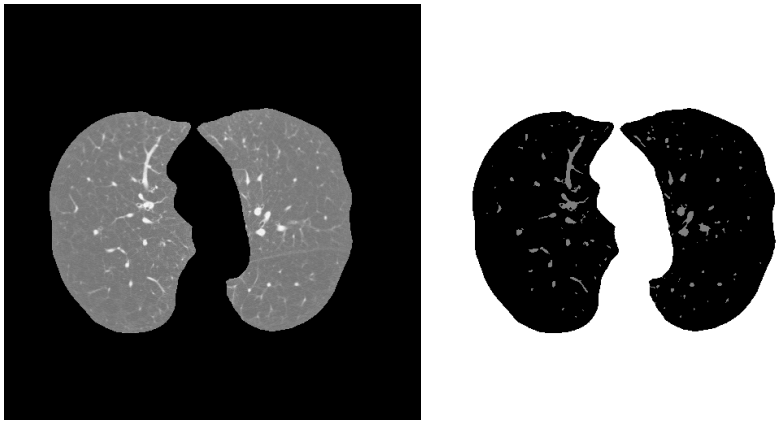
```
./MSKFCMOpening2D inputIm1.dcm outIm1.dcm 500 0.001 2 2 1 25 0 2 4 3 -1000 -940 -450 450 3 3
```

In this case, the image will be classifed in three clusters but the pixels of the background are not ignored so we must to specify the value for each of the three inital centroids. By other hand, we use a Ball with a radius of 3x3 as structuring element.

On the other hand, a valid example of the command line used to execute the example *MSKFCMClassification3D.cxx* is:

```
./MSKFCMClassification3D ./inputImDir/ ./outputImDir/ 500 0.001 2 2 1 4 2 -939 -474 650
3 3 3 -f -2000
```

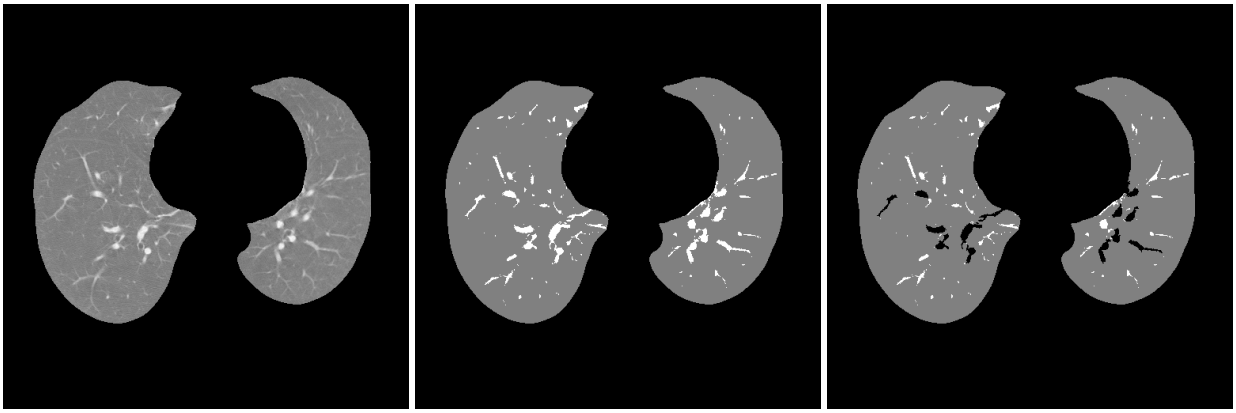
Note that as the image is 3D we set a radius of 3x3x3.



(a) Input Image.

(b) Defuzzied image obtained with the KFCM_S algorithm.

Figure 8: Parameters: MaxIterations = 500, ε = 0.001, m = 1.4, α = 1, NumOfThreads = 4, NumOfClasses = 2, $\{V^1$=-840 V^2=-350$\}$, σ = 850, StructRadius = 1x1, backgroundValue = -2000 .



(a) Input Image.

(b) Defuzzied image obtained with the MSKFCM algorithm.

(c) Image with objects from cluster one with a Physical size smaller than 25 Physical Units (PU) removed.

Figure 9: Parameters: MaxIterations = 500, ε = 0.001, m = 2, P = 2, Q = 1, lambda = 100, backgroundClass = 2, openClass = 1, NumOfThreads = 4, NumOfClasses = 3, $\{V^1$=-1000 V^2=-900 V^2=-450$\}$, σ = 450, StructRadius = 3x3 .

6 Conclusions

This paper describes the implementation of three fuzzy clustering algorithms, that work with both 2D and 3D information and they deal with a type of fuzzy techniques that are not still included in ITK. Moreover, it defines a base to develop new fuzzy clustering algorithms that are being proposed within this high researched area.

On the other hand, we introduced new characteristics to these algorithms allowing to classify each pixel attending to the spatial information present within its neighborhood that may have different shapes. In this way, the spatial term of these algorithms is made flexible.

Finally, we proposed a threaded version of these algorithms to reduce the computing time, particularly for those executions that have as an input a 3D study composed by several images, usually more than 150 images.

7 Acknowledgments

This work was supported by project 09SIN006PR from the I+D Suma Call of the Galician Plan for Research, Development and Technological Innovation (INCITE) from Xunta de Galicia.

We thank Juan Sanchez and Juan Rodrigo Cantorna for providing and adapting parts of the code to ITK v4.

References

[1] S.G. Armato III, G. McLennan, and M.F. McNitt-Gray. Lung Image Database Consortium: Developing a Resource for the Medical Imaging Research Community. *Radiology*, 232:739–748, 2004. 5

[2] J. C. Bezdek. *Fuzzy Mathematics in Pattern Classification*. PhD thesis, Applied Math. Center, Cornell University, 1973. 1

[3] J. C. Bezdek, Mikhil R. Pal, and James Keller. *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Kluwer Academic Publishers, 1999. 1, 2.1

[4] A. Castro and B. Arcay. Comparison of various fuzzy clustering algo- rithms in the detection of ROI in lung CT and a modified kernelized- spatial fuzzy c-means algorithm. In *Proc. of 10th IEEE Int. Conf. on Inf.Tech. and Appl. in Biom., Corfu, Greece*, 2010. 2.3

[5] Long Chen, C.L.P. Chen, and Mingzhu Lu. A Multiple-Kernel Fuzzy C-Means Algorithm for Image Segmentation. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 41(5):1263–1274, 2011. 1

[6] S. C. Chen and D. Q. Zhang. Robust image segmentation using FCM with spatial constraints based on new kernel-induced distance measure. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(4):1907–1916, 2004. 1, 2.2

[7] Keh-Shih Chuang, Hong-Long Tzeng, Sharon Chen, Jay Wu, and Tzong-Jer Chen. Fuzzy c-means clustering with spatial information for image segmentation. *Comp. Med. Imag. and Graph.*, pages 9–15, 2006. 1, 2.3

[8] T.M. Cover. Geomeasureal and statistical properties of systems of linear inequalities in pattern recognition. *IEEE Trans. Electron. Comput.*, 14:326–334, 1965. 2.2

[9] N. Cristianini and B. Schölkopf. Support vector machines and kernel methods: the new generation of learning machines. *AI Magazine*, 23(3):31–41, 2002. 1

[10] Maurizio Filippone, Francesco Camastra, and Francesco Masulli. A survey of kernel and spectral methods for clustering. *Pattern Recogn.*, 41(1):176–190, 2008. 1

[11] L. Ibanez, W. Schroeder, L. Ng, and J. Cates. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-15-7, http://www.itk.org/ItkSoftwareGuide.pdf, second edition, 2005.

[12] J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the fifth berkeley symposium on mathematical statistics and probability*, pages 281–297, 1967. 1

[13] K-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, 2001. 2.2

[14] V. Roth and V. Steinhage. Nonlinear Discriminant Analysis Using Kernel Functions. In *NIPS'99*, pages 568–574, 1999. 1

[15] B. Schölkopf. *Support vector learning*. GMD-Bericht ; 287. Oldenbourg, München, Germany, 1997. 1

[16] B. Schölkopf, AJ. Smola, and K-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998. 1

[17] Zhong-dong Wu, Wei-xin Xie, and Jian-ping Yu. Fuzzy C-Means Clustering Algorithm Based on Kernel Method. In *Proc. of the 5th Int. Conf. on Computational Intelligence and Multimedia Applications*. IEEE Computer Society, 2003. 1

[18] L. Zadeh. Fuzzy sets. *Information and Control*, pages 338–353, 1965. 1

[19] D. Q. Zhang and S. C. Chen. Fuzzy clustering using kernel method. In *Proc. of the Int. Conf. on Control and Automation*, pages 123–127, 2002. 1

[20] D. Q. Zhang and S. C. Chen. A novel kernelized fuzzy c-means algorithm with application in medical image segmentation. *Artificial Intelligence in Medicine*, 32:326–334, 2004. 1, 2.3