Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Μελέτη αλγορίθμων ομαδοποίησης σε περιβάλλον προγραμματισμού Python** **Python based study of clustering algorithms**
Ονοματεπώνυμο Φοιτητή	**Αθανάσιος Σκουρτανιώτης**
Πατρώνυμο	**Ευάγγελος**
Αριθμός Μητρώου	**ΜΠΠΛ/ 14075**
Επιβλέπων	**Άγγελος Πικράκης, Επίκουρος καθηγητής**

Ημερομηνία Παράδοσης **Νοέμβριος 2016**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Άγγελος Πικράκης Επίκουρος Καθηγητής	Χαράλαμπος Κωνσταντόπουλος Επίκουρος Καθηγητής	Μιχαήλ Ψαράκης Επίκουρος Καθηγητής

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον καθηγητή μου και επιβλέποντα της Μεταπτυχιακής μου διατριβής κ. Άγγελο Πικράκη για την καθοδήγησή του στη συγγραφή της.

Αφιερώνω αυτή την εργασία στη γυναίκα μου, Αφροδίτη, και στη νεογέννητη κόρη μας.

Abstract

Cluster analysis is the field of unsupervised learning that includes processes which divide data into groups according to a proximity measure. We briefly review the theoretical foundations of the field and provide a description of the programming concepts and tools used throughout this study. We also describe, build and use statistical techniques and indices suitable for the evaluation of clustering results. We implement seven different data clustering algorithms which can be organized into three different categories and test each one of them on three different datasets of synthetic data. In the final chapter, which can be considered a second distinctive part, we apply some of these algorithms combined together to accomplish image segmentation analysis tasks. We execute our algorithms on a set of images and measure the performance of those clustering-based segmentation results with reference to human made segmentation. We finally propose and construct a merging technique based on depth first search algorithm that when applied to an already clustered image, raises the performance dramatically.

Περίληψη

Η ανάλυση συστάδων είναι ο τομέας εκείνος της χωρίς επίβλεψη μηχανικής μάθησης που περιλαμβάνει διαδικασίες διαχωρισμού δεδομένων σε ομάδες σύμφωνα με κάποιο μέτρο εγγύτητας. Συνοπτικά εξετάζουμε το θεωρητικό υπόβαθρο του τομέα αυτού και παρέχουμε μία περιγραφή των εννοιών και των εργαλείων που χρησιμοποιούνται στην παρούσα εργασία. Επίσης περιγράφουμε και υλοποιούμε στατιστικές τεχνικές και δείκτες κατάλληλους για την αξιολόγηση των αποτελεσμάτων διαχωρισμού σε συστάδες. Υλοποιούμε εφτά διαφορετικούς αλγορίθμους ανάλυσης συστάδων που δύνανται να οργανωθούν σε τρεις διαφορετικές κατηγορίες και εξετάζουμε κάθε έναν από αυτούς σε τρία διαφορετικά σύνολα τεχνητά δημιουργηθέντων δεδομένων. Στο τελευταίο κεφάλαιο, που μπορεί να θεωρηθεί ως δεύτερο ξεχωριστό μέρος, εφαρμόζουμε κάποιους από τους υλοποιημένους αλγορίθμους συνδυαστικά μεταξύ τους, στον τομέα της ανάλυσης κατάτμησης εικόνας. Εκτελούμε τους αλγόριθμούς μας πάνω σε ένα σετ από εικόνες και μετράμε την απόδοση των αποτελεσμάτων μας με βάση αναφοράς τα αποτελέσματα που έχουν προκύψει από την κατάτμηση που πραγματοποίησε κάποιος άνθρωπος στην ίδια εικόνα, χρησιμοποιώντας μόνο την αίσθηση της όρασής του. Τέλος, προτείνουμε και υλοποιούμε μία τεχνική ενοποίησης βασισμένη στον αλγόριθμο αναζήτησης κατά βάθος η οποία όταν εφαρμόζεται σε μία εικόνα ήδη χωρισμένη σε συστάδες, αυξάνει δραματικά την απόδοση του αποτελέσματος.

Contents

Chapter 1
Introduction

1.1 Cluster analysis

Cluster analysis can be found in the bibliography with several names. Data clustering, unsupervised learning, segmentation analysis are some of them. There are also many overlapping definitions about what cluster analysis is and what it does. According to the classical book of the field, (Jain & Dubes, 1988), cluster analysis is a field of exploratory data analysis which *organizes data by abstracting underlying structure either as a grouping of individuals or as an hierarchy of groups*. According to (Anderberg, 1973) and (Bezdek J. , 1981), cluster analysis searches for structures within data. According to (Guojun Gan, 2007) it is *a method of creating groups of objects, or clusters, in such a way that objects in one cluster are very similar and objects in different clusters are quite distinct*. Finally, (Jain A. K., 2008) provides a slightly more formal definition of a clustering procedure by suggesting that *given a representation of n objects, one must find K groups based on a measure of similarity such that objects within the same group are alike but the object in different groups are not alike*.

It is obvious that in order to give an adequate definition of the cluster analysis, we should try first to find a definition for the most important building block of this field, the cluster structure. This is the subject of the following section.

1.1.1 Definition of cluster

The most essential structure in cluster analysis is obviously the "cluster". We should attempt to reach to a definition of the term "cluster" by reviewing two examples where such a definition would come handy.

In the first example we suppose that we need to classify[1] some books. If we are free to choose any rule to apply to this classification task then the basic question raised is which would be this rule and which would be the procedure to come up to it. The result of the classification would obviously be different depending on whether it is conducted based on each book's type or the release year of each book, or the nationality of the author, or the size of each book or even something seemingly irrational such as which is the second letter of each book's title etc. Namely, there are countless ways to classify books, some definitely more functional than others in certain cases. However, is there a commonly accepted way to measure this functionality and define the most appropriate clustering criterion for classifying books?

(Bonner, 1964) defines this criterion as the one that satisfies the most the value judgment of the user and produces some value to him. Following this very general definition of the clustering criterion, the term "cluster" is

[1] Nowadays, it has prevailed to use the terms "classify" and "classification" in order to refer to the methodology applied to supervised learning problems where, contrary to unsupervised learning, each vector contains a feature that is a target class and takes only discrete values. In the current context though we use these terms to refer to the unsupervised classification which most of the times is called "clustering".

described as a term that lacks a formal definition and is only required to "produce a value to the investigator". Although it is correct, this definition's disadvantage is that it does not distinguish between different cluster partitions. As long as a partition "produces value" to a researcher it is valid. It disregards, or not mentions at all, probably in order to be more general, the fact that some partitions are more natural than others. This can be easily seen in the second example.

In this example let us move one step further by considering the scatter plot (a) below, where the separate instances – which could be the books of the first example - are represented as dots in the R^2 space:

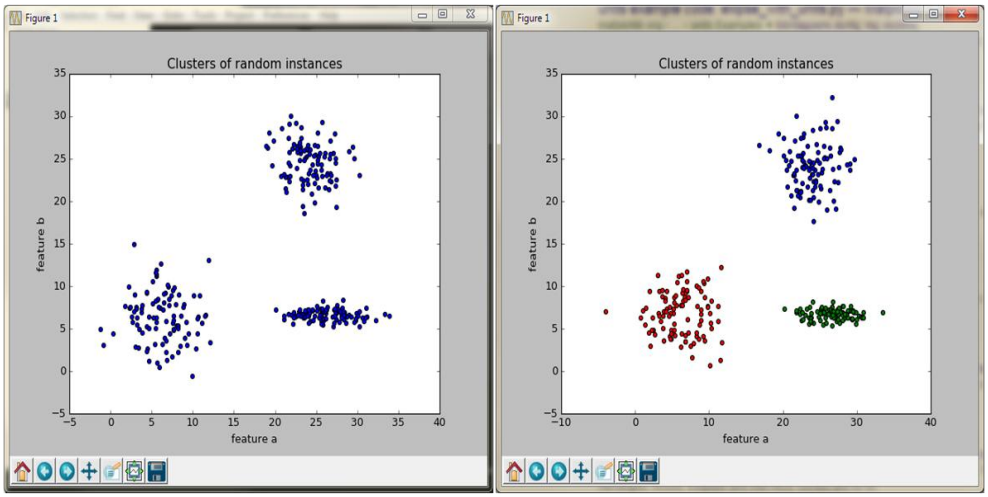

Figure 1 - a) A scatter plot of random vectors, b) The same set of vectors divided into 3 groups represented by different colors

Compared to the first example, here we have already chosen the criteria to classify the instances, we have quantified them into two features, a and b, and we just need to make a decision on which is the best classification. Obviously, there are not many people who would argue that, intuitively, the best way to classify the instances is the one presented in the scatter plot (b) where the instances are divided in three different categories discriminated by different colors. This realization indicates that practically, the choice of a specific partition over the others is not a totally subjective task as (Bonner, 1964) indicated. Furthermore, it answers the question imposed in the previous paragraph. There is actually a way, a method to define the most appropriate classification. For the time being we can call it "human perception".

It is the human perception that makes us choose out of the very large number[2] of possible partitions of the set of instances a specific partition.

[2] The number of possible partitions of a set of n instances to m clusters can be found by computing the Stirling numbers of the second kind either recursively based on the below formulas:

Base cases: $S(0,0) = 1, S(n,0) = S(0,n) = 0$

Recursive case: $S(m, n) = S(m-1, n-1) + nS(m-1, n)$

or by using the formula:

The next step is to quantify the "human perception" intrinsic mechanism into some measurable amount. This is of course something that has little to do with data analysis, but is rather a matter of other scientific fields.

Indeed, the first field to study the way our visual perception works was psychology. A German movement named the Gestalt school of psychology which was based on Berlin roughly in the first half of the 20th century, stressed the importance of perceptual organization. The main idea of the Gestalt psychology was that the whole is different from the sum of its parts(Irvin Rock, 1990). (Wertheimer, 1923), one of the three founders of Gestalt psychology defined the grouping laws that can be used to build the whole structures out of their parts. There are several different laws, however we are interested only in two of them that according to (Feldman, 1995) can be used to distinguish the clusters of figure (a). These are the laws of proximity and good continuation. The law of proximity stresses that an individual perceive two objects that are close to each other as a group. The law of good continuation indicates that elements that form smooth continuations tend to be perceptually grouped together. In our case of figure (a) only the law of proximity applies. In more complex datasets we shall use both laws. All these ideas were unified at the second half of the 20th century under the scientific field of "Cognitive Science", and its subfield, "Perception".

(Cormack, 1971) makes also an excellent effort to provide a synopsis of some of the most important early attempts during 1940 – 1970, to assign a definition to the term "cluster". The two basic ideas on which those attempts were based can be summarized into two properties that every cluster should possess, "internal cohesion" and "external isolation". These ideas are analogous to the concepts stressed by the adapted to the clustering issues Gestalt psychology.

What becomes clear from this partly historical, partly empirical review of the attempts to define the term cluster is that there cannot be one unique formal definition for it. On the contrary, the definition depends on two things. The first is the problem that is attempted to be solved and the second is the methodology, in other words the algorithm used to solve it. Different problems or different algorithms will most likely lead to different clustering results, providing in such a way a different definition of the term "cluster". At the same time though, we accept that in order to proceed to a clustering task we must introduce some commonly accepted properties that characterize a clustering effort as a "good" one. These two properties are the ones described in the previous paragraph.

1.1.2 Process of cluster analysis

There have been many suggestions of the steps that a clustering task is consisted of. (Buhmann, 2002) has introduced a model divided into four stages, data representation, modeling, optimization and validation. However, the model

$$S(m,n) = \frac{1}{n!}\sum_{i=0}^{n}(-1)^{i}\binom{n}{i}(n-i)^{m}$$

In the case where the number m of the clusters is undefined, then one has to compute the Stirling numbers for all possible values of m, something that is extremely computationally demanding and most of the times cannot be accomplished.

provided by (Jain, Murty, & Flynn, 1999), expanded by (Theodoridis & Koutroumbas, 2009) and slightly amended by our own observations during constructing and using clustering algorithms describes the clustering task in a clearer way. It is the following:

- **Data Collection:** This is the initial phase of every clustering task. The researcher usually ends up with a bunch of unprocessed raw data which are unsuitable for immediate process.
- **Data Preparation:** This stage includes the feature selection, the data preprocessing and the data transformation. All these procedures' purpose is to convert the raw data of the previous step to a properly meaningful dataset that will be fed into the clustering algorithm. This is one of the most important steps of the clustering task. Many times the success of an algorithm depends on the preparation of the dataset conducted during this stage.
- **Proximity measure:** The definition of a distance metric takes place at this stage. It is a decision that depends a lot on the type of cluster representative, so this is another thing that has to be defined at this stage. A very common combination that is being used in this thesis is the Euclidean distance as a distance measure and the point as a cluster representative.
- **Clustering criterion:** (Theodoridis & Koutroumbas, 2009) use this term to refer to the definition of the kind of cluster structures that can be discovered in the dataset (for example compact or elongated). It is the same stage that in (Buhmann, 2002) is described as "modeling".
- **Clustering algorithm:** This stage includes the selection of the clustering algorithm, the procedure of defining its requested parameters and its execution. We should note here that the definition of the values of the parameters of the clustering algorithm is usually achieved through the execution of validity indices, as we will see in practice in the following chapters. However this procedure should not be confused with the last stage of the cluster analysis process, the validation.
- **Validation:** In this final stage the evaluation of the clustering results takes place.

1.1.3 Applications of cluster analysis

There are numerous applications of cluster analysis algorithms and techniques to real world problems. (Jain A. K., 2008) mentions image segmentation, document clustering, grouping of customers into different types for efficient marketing, studying of genome data in biology. According to the writer, if we attempted to categorize all the possible uses of cluster analysis, then we would end up to three main purposes:

- Compression: organizing and summarizing data.
- Natural classification: identify the degree of similarity among forms or organisms.
- Underlying structure: to gain insight into data.

1.1.4 Categories of Clustering Algorithms

Several categorization efforts of the numerous clustering algorithms have been suggested. Most of these categorizations divide them into two main categories, hierarchical and partitional.

- Hierarchical algorithms.

The name of the algorithms of this category derives from the fact that they divide a set of instances to a number of clusters which have an hierarchical structure. Hierarchical structure in this context means that every cluster is a subset of another cluster with the exception of one which contains all the others and is not part of any other. As this structure can be easily represented mentally and visually with the aid of a tree, researchers use a kind of diagram called dendrogram in order to observe the results of any hierarchical clustering.

These algorithms are further divided into two categories based on the way they build the clusters, agglomerative and divisive. Agglomerative algorithms start by assigning each vector in a cluster and then merge these clusters at each step into larger ones. Divisive algorithms conduct the exact opposite process by assigning all vectors in one cluster and then further dividing this cluster into smaller ones.

- Partitional algorithms.

Contrary to hierarchical clustering, partitional algorithms result in grouping a set of vectors into a set of disjoint clusters without the hierarchical structure. One proposed categorization (Theodoridis & Koutroumbas, 2009) is:

  - Sequential: Algorithms that process the data sequentially.
  - Cost function optimization: Produce a clustering result by trying to minimize a cost function
  - Graph theory based: They regard data instances as nodes of a graph and then they apply a criterion to partition it.
  - Branch and bound clustering algorithms.
  - Genetic clustering algorithms.
  - Stochastic relaxation methods
  - Valley-seeking clustering algorithms.
  - Competitive learning algorithms.
  - Algorithms based on morphological transformation techniques.
  - Density-based algorithms.
  - Subspace clustering algorithms.
  - Kernel-based methods.

1.2 General programming Implementation Notes

Every algorithm that is described in this document has been implemented programmatically in Python. The implementation files are indicated in the proper position in each algorithm’s description. The fact that this code deals

with data manipulation differentiates it a bit from "traditional programming" and maybe makes it peculiar for someone that sees it for the first time. The reason for this differentiation and some general comments on the implementation are described below.

Programming in python is widely considered "high-level", meaning that the programmer does not have to consider about issues such as for example direct memory management. Everything is managed by the built-in mechanisms of the language, letting the programmer consider only the aspects of the problem he is trying to solve. However as with many things in life, in programming everything is relative. Therefore, for certain tasks, including data manipulation, programming in "pure" python is considered extremely "low level". This is due to the fact that all scientific fields that deal with data make heavy use of concepts of Linear Algebra. The most important reason for this is that the matrix, the basic structure defined in Linear Algebra provides an ideal basis for organizing and manipulating, through matrix operations, the data under investigation. Cluster analysis, seen as a subfield of Data analysis, is not an exception to this rule.

The use of Linear Algebra raises the need for an implementation of the matrix structure programmatically. However, the introduction of the matrix structure in a program modifies decisively the whole approach to the data. The program does not deal with data as separate elements anymore, but as a collection of elements, stored in matrices, which are manipulated as a total. Even more specifically, the program "exchanges" loops that affect individual data, for matrix operations, that affect collections of data.

The following example demonstrates this concept. Let us examine the difference in the implementation of a simple mathematical formula, the one used to find the euclidean distance between a list of points and a point. The two functions below do the same thing and return the same result, in different format, but still the same:

Function 1:

```
def euclidean_distance(data, point):
    data_distances = []
    for d in data:
        sum_of_squares = 0.0
        for i in range(0, len(d)):
            sum_of_squares += pow(d[i] - point[i], 2)
        data_distances.append(sqrt(sum_of_squares))
    return data_distances
```

Function 2:

```
import numpy as np

euclidean_distance = lambda data, point: np.sqrt(np.sum(np.power(data - point, 2), axis =
1).reshape((len(data), 1)))
```

The first function uses good old "high-level" python programming to transform the data element by element, whereas the second represents an even "higher-level" programming by utilizing the data structure of matrix, provided by the third party library Numpy in order to perform some matrix operations on the data[3]. One can easily observe that the two for loops of the first function

[3] This discussion about the low or high level of programming could go on, seemingly with no end, as Numpy array structures, in their turn, are considered low level compared to other

have been squeezed somewhere in the matrix operation functions of the Numpy library. It is these functions' responsibility to undertake the optimization of the operations they conduct, including the two for loops, at the low level.

Although it does not constitutes a new programming paradigm the difference of programming by manipulating matrices is so critical that the concept "Array Programming" has been introduced to refer to programming with the use of arrays and matrices and there is also a discrete category of array programming languages. The basis of these languages was the paper (Iverson, 1962) which was implemented as a language with APL programming language.

All the algorithms provided with the current thesis are implemented with the use of the described coding philosophy, into the frame of the functional programming paradigm.

1.3 Notes on Proximity Measures and Cluster Representatives

A very important concept to the majority of the clustering algorithms is the distance between certain elements of the clustering task, for example between two vectors or between a vector and a cluster representative, such as a centroid. This concept can be found in the bibliography under the name "proximity" and the different choices of distance measures are called "proximity measures".

These measures are divided into two major groups. The first group includes the "dissimilarity measures" which we can describe as the ones that take larger values, the larger the distance between two vectors becomes. It is obvious that these measures correspond to our intuition about how distance is measured. An example of a dissimilarity measure is the Euclidean distance. The second group includes the "similarity measures" which have the exactly opposite features. The larger the distance between two vectors is, the smallest value a similarity measure produces.

There is a lot of discussion about the different proximity measures which can be applied in the clustering algorithms. We should note however that in this thesis we have not tried to apply any other measure in the algorithms described and implemented, other than the Euclidian distance. So the terms "proximity", "distance" and "Euclidean distance" should be used here interchangeably.

Furthermore, distance measures are not the only clustering task characteristics that come in a large variety of types. The same happens with cluster representatives. We can describe a cluster representative as a geometric structure than, as its name reveals, describes the cluster and is used in the several calculations as the cluster it represents. The most important types of representatives are point, hyperplane and hyperspherical representatives (Theodoridis & Koutroumbas, 2009). In the current thesis though we are restricted to point representatives.

libraries. One such library is Pandas, which offers the DataFrame data structure which supports operations similar to Microsoft Excel spreadsheet.

Chapter 2
Cluster Validity

Clustering procedures on datasets lead to specific results. In case the vectors of the datasets are defined in the two dimensional space it is easier to evaluate these results by visual observation. However in cases of data with higher dimensionality such an evaluation is not that easy. This is mostly what created the need for the introduction of some standard procedures in order to evaluate the final result of a clustering algorithm. All such procedures that were developed were categorized under the umbrella term "cluster validity".

The reference source for cluster validity procedures is (Jain & Dubes, 1988). The writers have divided the cluster validity indices into two main categories, statistical testing and non statistical testing. Statistical testing is, in its turn, divided into two categories that are named *internal* and *external criteria* for cluster validity, depending on the data they are based on, whereas non statistical testing contains all *relative criteria*. Below there is a description of how the methodology of each category works.

2.1 Statistical Testing

Let us go back in the basic question cluster validity attempts to answer, namely, if a clustering result is "good" or "bad". The methodology followed by the statistical validity testing has two steps. The first step to a satisfying answer is to define an index that will measure this "good" or "bad" grade. The second step, which is a more challenging task, is to define a specific threshold, above or below which the index will be describing a "good" or "bad" clustering. We shall see the above steps under the frame of the subcategories of the statistical testing methods that we mentioned, internal and external criteria.

2.1.1 Internal Criteria

The target of evaluating a clustering partition using internal criteria is accomplished by using only the data themselves, contrary to as we will see, external criteria. The methodology includes running a clustering algorithm once on the dataset, and after that evaluating its performance.

After choosing and calculating the index, we also need to define a threshold for it. This is where the statistical part of the specific procedure is revealed, as we need to adopt the Hypothesis testing methodology and adjust it to our clustering task. As this methodology suggests we must define two hypotheses, H_0 and H_1 that would reflect the two states of the problem we seek to solve. The alternative hypothesis H_1 would suggest that there is a certain structure in our data set, whereas the null hypotheses H_0 would suggest that our data set is completely random and consequently it presents no structure at all.

There are three different ways we can choose to express the randomness in the null hypotheses and each one leads to different implementation. The first is to use "**Random position hypothesis**". This is our choice in the accompanying implementation code of this thesis. As the name implies, in order to represent a totally random data set we pick random data in the same space and range

where our initial data is defined, according to the uniform distribution. The second way is the **"Random Graph hypothesis"** and it is implemented by randomly creating different proximity matrices for the null hypothesis' datasets. Finally, the third way is **"Random label hypothesis"**, where new random partitions of the original dataset are constructed.

After choosing the way to express the null hypotheses we need to take sample distributions of the randomly chosen datasets and construct the probability density function of the index we have chosen. Unfortunately, this is a very computationally demanding task, so the only option we have is to take a limited amount of sample distributions and use them to extract the approximate probability density function of the index. This is accomplished by the so called Monte Carlo technique.

After constructing the probability density function of the index under the null hypothesis, we usually assume that it is normally distributed. We place the index value of the dataset under consideration in the same histogram and we calculate its p-value, which is the probability that this value or an even more extreme value would come up as a result. If the p-value is less than a certain significance level that we empirically define at the level of 5%, then we reject the null hypothesis, meaning that we do not believe that the index value we have obtained is coming out of the random dataset's probability distribution, so our dataset presents a structure.

With regards to the indices to be used in the internal criteria validation, the bibliography ((Jain & Dubes, Algorithms for Clustering Data, 1988), (Theodoridis & Koutroumbas, 2009)) does not propose the same variety of indices as in the case of external, or relative criteria. Therefore we limit ourselves only to the implementation of one index, the **Hubert's Gamma Statistic**. This is defined as:

$$G = \frac{1}{M}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} X(i,j)Y(i,j) \qquad (2.1)$$

where X, Y are two matrices with the same dimensions. It is based in the same philosophy of the relation (2.14) used in (Bezdek & C, 1973) to find the average coupling between two subsets, as we will see in the section of the Partition Coefficient index. The difference is though that here we are talking about a "coupling", a relation, between two matrices instead of two fuzzy subsets.

In the case of internal criteria, X is the proximity matrix of the data set and Y is a N x N matrix defined as:

$$Y(i,j) = \begin{cases} 1, if\ x_i and\ x_j\ belong\ to\ different\ clusters \\ 0, otherwise \end{cases}$$

It's easy to see that the larger the value of Hubert's Gamma, the largest the correlation between the two matrices.

2.1.2 External Criteria

Every algorithm presented in this thesis is tested against several synthetic datasets. These datasets are not created completely randomly. On the contrary, they are created as separate sets of vectors, in other words as clusters of vectors that are unified in one big dataset. This is due to the fact that an existing structure in the datasets is desirable in order to test whether our algorithm would succeed in revealing it or not. The fact is that we know this structure, we know in other words the solution –or at least a good solution – to the clustering task we attempt to solve, before we even solve it. Many problems in real life are similar to this situation. A certain amount of prior field knowledge usually exists and can be translated into an external partition of our dataset. In such case this partition can be used to validate the results of a clustering algorithm and this kind of validation is considered to belong to the external criteria.

The procedure used in the external criteria validation is exactly the same as in the internal criteria. We pick up a meaningful index of our choice, we get a measure of it in our dataset and then we form the null hypothesis of position randomness by using a monte carlo simulation on a set of 100 datasets. The simulation results in getting the probability distribution of the index which we consider a normal distribution and we finally use it in order to reject or accept the null hypothesis.

The very sensible question raised at this point is why, since we have all the external information available, we aren't just comparing the clustering result returned by our algorithm with this external information. We do not even need a sophisticated index, not to mention a statistically retrieved threshold, to express the similarity between the two partitions, the one we come up to and the external. A simple index, for example the Jaccard index, is adequate for a simple comparison of the two partitions. The answer is that we could perfectly evaluate the clustering only by a single comparison to the external data. However, in case we chose to go through all the complicated procedure described above, we then would be extending the testing to the quality of the external information we possess.

In order to make it clearer, let us assume that someone expressed some doubt on the quality of our external partition knowledge. After all, who can guarantee that in every case our external partition is not some meaningless, random partition without real structure? Then every successful comparison to it, would lead to accepting a bad clustering result. This is where the described monte carlo sampling placed under the frame of statistical hypothesis testing gives us a hint of whether the external partition is actually not a random one.

The indices we have implemented in the category of external criteria are four, the most usual ones used. In order to evaluate the first three of them we have to construct first a N x N utility table, where N is the number of the vector of the dataset. We define this table as a strictly upper(or lower) triangular matrix, since we are not going to fill in and use all its elements. Each element x_{ij} of the table would correspond to a pair of vectors x_i, x_j of the dataset and would take one of the four following values {SS, DD, SD, DS}. If we define C as the result of our clustering algorithm and as P the external partition we have available, then the first letter of each value is S(ame) if the two elements belong to the same cluster in C or D(ifferent) if they belong

to different clusters. The second letter of each value is assigned in a similar manner, however with regards the P partition.

After constructing the utility table we define the following measures which result by counting the different occurrences of the four values:

a = count(SS), b = count(SD), c = count(DS), d = count(DD).

We finally define M = a + b + c + d= N(N-1)/2 which is the total number of possible pairs. After this preprocessing, the calculation of the following indices is very easy:

- **Rand Statistic:** This statistic is defined as **(a + d)/M** and was proposed by (Rand, 1971). It measures the percentage of the number of pairs of vectors that are either in the same cluster in both partitions, C and P, or belong to different clusters in both partitions. In other words, if the two partitions are exactly the same, the index will take its largest value, which is 1. In the opposite case, it can be easily seen that the value of the index would be 0.
- **Jaccard Coefficient:** This statistic is defined as **a/(a + b + c).** In a similar manner as Rand Statistic does, it measures the percentage of the pairs of vectors that are in the same cluster, but does not take the vectors in different clusters under consideration. Its values can vary in the range [0, 1].
- **Fowlkes and Mallows index:** This statistic is defined as $\sqrt{\frac{a}{a+b}\frac{a}{a+c}}$ and was proposed by (Fowlkes & Mallows, 1983). Since the index is proportional to a, the larger value it takes, the greater the similarity is between C and P.

The final index we consider in this category is **Hubert's Gamma Statistic**. We have already described this index in the section of the internal criteria. The only thing that is different here is the input data used to calculate the index. Instead of using the proximity matrix as the one of the two input arrays, we use and array X that has the value 1 if x_i and x_j belong to the same cluster with regards to the external clustering or 0 otherwise. The second input array remains the same.

2.2 Non statistical Testing

Although still a testing procedure, the non statistical testing philosophy is different than the one of statistical testing. In statistical testing we saw that we are executing a clustering algorithm over a dataset once and then we evaluate the clustering result. (Bezdek, Keller, Krisnapuram, & Pal, 2005) regard this procedure as a "parametric estimation method", meaning that we estimate the parameters of some model and the validity indices measure the goodness of fit of the estimated parameters.

On the other hand, the so called non-statistical testing has more the interpretation of "exploratory data analysis". Every index we calculate describes one or more parameters of our model and measures its quality, defined by the partitioning of the dataset. (Jain & Dubes, Algorithms for Clustering Data, 1988) refer to the non statistical testing with the term "Relative Criteria" as opposed to the internal and the external criteria that we described in the previous sections. Practically though, in the frame of cluster analysis, conducting "non statistical or relative criteria testing" is synonym for "searching for the ideal number of clusters the dataset can be

partitioned to", or even more generally "searching for the best value of the clustering algorithm's parameters".

This is why practically we place the relative criteria testing not in the *validity stage* but in the stage defined as *clustering algorithm in 1.1.2*. In the following chapters where several algorithms are described and tested over synthetic data, the relative criteria are used at the beginning of each clustering task in order to define the necessary parameters. The methodology used includes the execution of the the clustering algorithms sequentially, several times, each one with different parameter values and calculate the values of each index at the end of each execution. The values of the parameters for which the indices take their optimal values are then used in order to re-execute the algorithm and pass the results to the internal and external validity criteria, at the validity stage.

Finally, one important question which could be raised with regards to non statistical testing is that it seems we are dealing with the problem of picking up the best clustering in an indirect way. We first cluster the dataset, we then measure one or more indices and finally we pick up the clustering where the indices have their minimum or maximum value. Why couldn't we proceed directly to the optimization of the indices and simply choose the cluster for which the indices have their optimal value?

(Bezdek, Keller, Krisnapuram, & Pal, 2005) answer that no indices can capture all the properties of a clustering that is considered "good". On the other hand, even if it could, many indices cannot be optimized easily. This is why we use them after the clustering procedure.

2.2.1 Hard clustering indices

The hard clustering indices described below apply to datasets that have been clustered with an algorithm which leads to crisp or hard clusters. Once this criterion is fulfilled, the choice of the specific algorithm does not have any importance at all. It is the final – crisp – result that we feed into each index for evaluation.

- **The Dunn index:** This index was proposed by ***(Dunn J. C., 1974)***. It is defined as:

$$DI = \frac{min_{1\le q\le c} min_{1\le r\le c, r\neq q} dist(C_q, C_r)}{max_{1\le p\le c} diam(C_p)} \qquad (2.2)$$

where $dist(C_q, C_r)$ is the distance between two clusters, C_q, C_r and it is defined as:

$$dist(C_q, C_r) = min_{x\in C_q, y\in C_r} d(x, y) \qquad (2.3)$$

where x, y are vectors in the dataset and $diam(C_p)$ is the diameter of cluster C_p and is defined as

$$diam(C_p) = max_{x,y\in C} d(x, y) \qquad (2.4)$$

In other words, the Dunn index is the value resulting after dividing the minimum distance among all pairs of clusters by the maximum diameter, which is obvious that it is essentially a measure of density. Consequently, we are seeking for as large values of the Dunn index as possible.

- **The Davies-Bouldin index**: ***(Bouldin & Davies, 1979)*** proceeded to the definition of some attributes that should be possessed by a general index which aims to measure the separation of the clusters produced after executing a clustering algorithm on a dataset. Based on them, the writers provide a theoretical formulation of a clustering measure and at the same time, an example of such a measure which as they note, is one of the simplest satisfying their definitions.

We can describe the proposed index with the below equations:

$$R_{ij} = \frac{S_i + S_j}{M_{ij}} \tag{2.5}$$

where S is called the dispersion of a cluster C_i and is defined as:

$$S_i = \left(\frac{1}{n_i} \sum_{x \in C_i} \|x - w_i\|^r \right)^{\frac{1}{r}} \tag{2.6}$$

and $M_{i,j}$ is the distance between two clusters which can be the euclidean distance between their representatives.

We also define for every cluster R_i as

$$R_i = max_{j=1,..,m, j \neq i} R_{ij} \tag{2.7}$$

and finally the Davies Bouldin index is defined as:

$$DB_m = \frac{1}{m} \sum_{i=1}^{m} R_i \tag{2.8}$$

The dispersion of a cluster measures the average distance of the cluster's vectors from its representative. It is a measure of the compactness of the cluster. This means that small values of R reveal compact (small dispersion) and clearly separated (large distance) clusters. For every cluster we choose the maximum R, the most penalizing case of R and we use it to calculate the average R which consists of the Davies Bouldin index. It is obvious that the minimum value of the index for different values of the number of clusters reveals the ideal number of clusters.

- **The silhouette index:** This index is described on ***(Kaufman & Rousseeuw, 2005)*** and is based on the notion of silhouette, introduced in ***(Rousseeuw, 1986)***. In order to define the silhouette, one must first define the value s(i):

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)} \qquad (2.9)$$

where i = 1 … N ,where N is the total number of vectors in the dataset, a_i is the average distance of x_i to all other vectors belonging to the same cluster with x_i and b_i is the minimum average distance of x_i to all vectors belonging to its closest cluster.

For each cluster we define the silhouette width as the average of the s_i for all the vectors that belong to it and finally we define the global silhouette index which is the average silhouette width of all the clusters the dataset has been partitioned to.

We can intuitively think of b_i, the cluster which has minimum average distance from x_i, as the second best choice for clustering x_i, after of course a_i. The larger b_i is, compared to a_i, the worse choice it becomes and so the better is our clustering partition. The opposite holds when b_i is small compared to a_i. This fact along with the observation that s_i takes values in the range [-1, 1] results in considering values of the silhouette index close to 1 as indication of a good clustering, whereas values close to -1, as indication of bad clustering.

- **The Gap statistic:** The first step in defining the Gap statistic index is to define two basic measures. The first is the sum of the distances between all available pairs of all the vectors X belonging to a cluster C. That is:

$$D_q = \sum_{x_i \in C_q} \sum_{x_j \in C_{q,i \neq j}} d(x_i, x_j) \qquad (2.10)$$

The second is the measure W which is defined as:

$$W_m = \sum_{q=1}^{m} \frac{1}{2n_q} D_q \qquad (2.11)$$

Obviously, the most compact the clusters are, the lower are the distances between the pairs of the vectors that belong to them and the lower is the value of W.

The problem raised at this point is the complete lack of a threshold to define whether a value of W is large or small. In order to overcome this issue (Tibshirani, Walther, & Hastie, 2001) who proposed the index resorted to the same technique we used in order to obtain thresholds in the case of internal and external indices. They suggested using a monte carlo technique in order to get datasets uniformingly distributed at the same space our clustering task is defined and use these datasets as reference distributions. The writers call this procedure a "standardization" of $\log(W_m)$ which takes place by comparing $\log(W_m)$ with its expectation under the reference distribution $E_n(\log(W_m))$. Mathematically this can be written as:

$$Gap_n(m) = \mathrm{E_n}(\log(\mathrm{W_m})) - \log(\mathrm{W_m}) \qquad (2.12)$$

which is the Gap index. The furthest $\log(W_m)$ is from $E_n(\log(W_m))$, that is the maximum value Gap takes for every value of m, gives an estimation for the optimal number of clusters.

2.2.2 Fuzzy clustering indices

Fuzzy clustering indices apply to datasets that have been partitioned by the fuzzy clustering algorithm. The common characteristic that they all have is that they introduce into their calculations a basic structure of the fuzzy clustering algorithm, which we will study at the relative chapter, the partition matrix.

- **Partition Coefficient:** This index was proposed by ***(Bezdek & C, 1973)*** and the only parameter it is based upon is the partition matrix U formed after the implementation of the fuzzy clustering algorithm. It is defined as:

$$PC(U) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m} u_{ij}^2 = \frac{\|U\|^2}{N} = \frac{tr(UU^T)}{N} \qquad (2.13)$$

and it is based on the definition of average coupling between two subsets of a partition, provided in the same paper:

$$\frac{1}{N}\sum_{k=1}^{n} u_{ik}u_{jk} \qquad (2.14)$$

for all $1 \le i, j \le c$ with $i \ne j$.

It is a really simple index, both with regards to its concept and its implementation. The values of the index vary in the range $[\frac{1}{m}, 1]$ where $\frac{1}{m}$ indicates that the clustering can be described as so "fuzzy" that it probably has no clustering structure at all, whereas 1 indicates that the clustering is crisp. As (Bezdek, Keller, Krisnapuram, & Pal, 2005) stresses, PC maximizes on every crisp partition produced by the fuzzy clustering algorithm, so it would make sense to choose the clustering that gives the index its maximum value, for every $m \ge 2$. However, (Theodoridis & Koutroumbas, 2009) underline that the index exhibits a negative dependence on the number of clusters m and instead of searching for a maximum point one should search in the index's plot for a "knee" like formation. Specific examples of the algorithm will be presented in following sections.

- **Partition Entropy Coefficient:** (Bezdek J. C., Mathematical Models for Systematics and Taxonomy, 1975) extended the concept of the fuzzy entropy of (Deluca & Termini, 1972) and used it to describe the fuzziness of a partition U. It is defined as

$$PE(U) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m}\left(u_{ij}log_a u_{ij}\right) \qquad (2.15)$$

and its values vary in the range $(0, log_a m)$. Its behavior is similar to the partition coefficient index, so the same comments are valid for this index too, although in the opposite direction, since this index increases as m increases.

- **Xie-Beni:** ***(Xie & Beni, 1991)*** realized that one of the major disadvantages of the partition coefficient index was that, apart from the monotonic decreasing tendency with m that is mentioned above, it also did not give any importance at all to geometrical properties of the data set. Their effort to introduce such a geometric property measure into a cluster validity index led to the Xie-Beni index.

The paper explains thoroughly the procedure which results to the below definition of the index:

$$S = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\mu_{ij}^2\|V_i - X_i\|^2}{\mathrm{n}\min_{i,j}\left\|V_{i-}V_j\right\|^2} \qquad (2.16)$$

where the nominator is defined as σ = $\sum_{i=1}^{m}\sum_{j=1}^{n}\mu_{ij}^2\|V_i - X_i\|^2$ and is the total variation of the data set under consideration and $s = \mathrm{d}^2{}_{\min} = \min_{i,j}\left\|V_{i-}V_j\right\|^2$ is called the separation of the fuzzy partition. With the appropriate transformations, for m = 2, the index can be written in the form:

$$S = \frac{J_2}{n\, d_{min}{}^2} \qquad (2.17)$$

which indicates that the minimization of the cost function leads to the minimization of S. Consequently, this is what we are looking for in this index, the number of clusters m where the index takes its minimum value.

- **Fukuyama – Sugeno index:** This is an index which, as in the case of the Xie-Beni, takes under consideration the geometric properties of the data set, by introducing into its definition two distance measures. These are the compactness of the clusters $\|\boldsymbol{x_i} - \boldsymbol{w_i}\|^2$ and the distance of the cluster representatives from the mean vector $\left\|\boldsymbol{w_j} - \boldsymbol{w}\right\|^2$. The definition of the index is:

$$FS_q = \sum_{i=1}^{N}\sum_{j=1}^{m} u_{ij}^q \left(\|x_i - w_i\|^2 - \left\|w_j - w\right\|^2\right) \qquad (2.18)$$

The minimum of the index indicate a good clustering.

Chapter 3
Cost Function Minimization Clustering Algorithms

If we assume a set of N vectors $\{x_i \in \mathbb{R}^k \mid 0 \le i \le N,\ k \in \mathbb{R}\}$ and its cost function

$$J(U,\theta) = \sum_{i=1}^{N}\sum_{j=1}^{m} u_{ij}^{q} d(x_i,\theta_j) \qquad (3.1)$$

where θ is the unknown parameter vector which describes each cluster, m is the a priori known number of clusters, $U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1m} \\ u_{21} & u_{22} & \dots & u_{21} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1} & u_{n2} & \dots & u_{nm} \end{bmatrix} = (u_{ij}) \in \mathbb{R}^{n\,x\,m}$, is a weight vector, q∈ℝ, and $d(x_i,\theta_j)$ is the distance metric from a vector x_i to the cluster representative θ_j, then the minimization of this function with regards to θ determines the values of θ that give us the best clustering of our data set.

Before proceeding to the task of minimizing the cost function, we should stress the important role played by the weight matrix U. It is the existence of this matrix in the definition of the cost function that provides great flexibility to define different types of clustering algorithms based on the same cost function. As we are going to see, only by amending the constraints imposed to the elements of the weight matrix makes us able to define three different types of clustering algorithms, namely fuzzy, possibilistic and hard clustering. The specific constraints for each one of them will be presented in the relative sections.

For now, let us first plot the values J of the cost function against all the possible values of the θ vector, considering that our data set is divided into two clusters, so the θ vector consists of two representatives, $\theta = [\theta_1, \theta_2]^T$. The U matrix is assigned some random values in the space [0, 1].

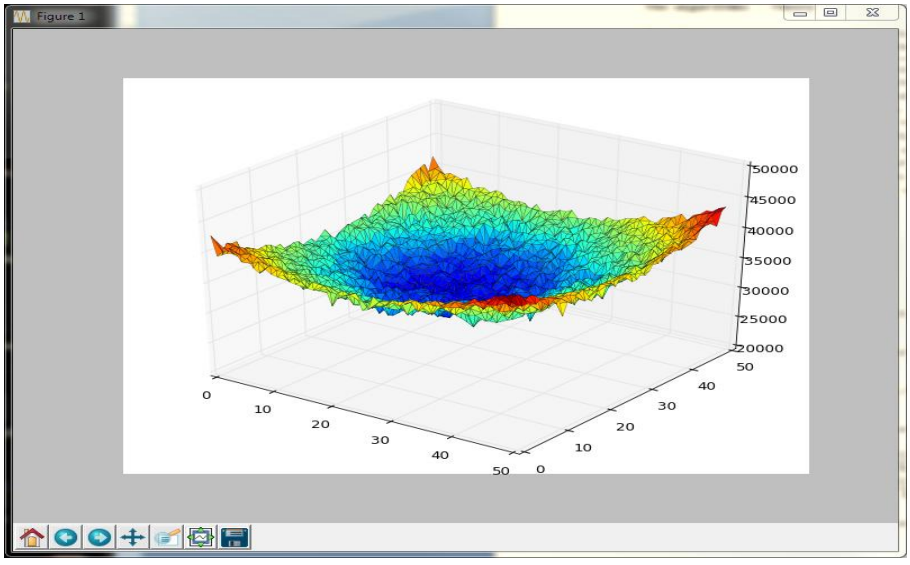

Figure 2 - Cost function minimization - file: minimizing_cost_function.py

It is obvious from the 3-D plot that the function converges to a global minimum. If the values of the weights of the U matrix were known a priori, as we assumed in order to construct this plot, then the task of minimizing the function would be easy and limited to the calculation of the gradient of J(θ) with respect to θ. However, the weights are not known, so we will approach the problem by using an iteration algorithm integrated in our algorithms. We shall see this algorithm, called the generalized expectation maximization algorithm, customized for each one of our algorithms in the following sections.

3.1 Fuzzy Clustering Algorithm

(Zadeh, Fuzzy Sets, 1965) stressed the important role played by imprecisely defined "classes" in human thinking. The two examples of such classes he mentions are "the class of beautiful women" and "the class of tall men". Obviously, there are not strictly defined criteria in order to classify a woman as beautiful or a man as tall. Another more general example of the importance of vagueness in everyday situations is mentioned in (Bezdek J. C., 1993). When we are passengers in a car and we want to advice the driver when to apply the brakes in case of a red light, we will prefer to say "Apply the brakes soon", instead of "Apply the brakes 15 meters before the light".

Zadeh named these classes "fuzzy sets", in contrast to ordinary sets(in part of the bibliography named as "crisp"), and he also provided a mathematical model for them. The cornerstone of his model is the membership function. This is a function that characterizes every fuzzy set and assigns to every object of the space under investigation, a real number k, k Ɛ [0,1] , which measures the "grade of membership" of the specific object to the specific set. Namely, in a space Ω, ∀ x Ɛ Ω, the membership function is defined as f: Ω → [0,1].

It may seem that fuzzy set theory coincides with Probability theory, since they both provide a model to measure uncertainty. In fact there has been a lot of discussion on this issue(Zimmermann, 2001), (Bezdek J. C., 1993). The two theories however are totally different both mathematically and semantically. What is important here is to stress that fuzzy clustering model is not considered a probabilistic model.

According to (Guojun Gan, 2007) the first researchers who applied fuzzy set theory to clustering were (Bellman, Kalaba, & Zadeh) and (Ruspini, 1969). Later, (Dunn J. , 1973) proposed a "fuzzy" variation of the k-means algorithm. The proposed algorithm was later improved by (Bezdek J. , 1981). The way the algorithm applied the theory of fuzzy sets to the task of clustering was by implementing the basic fuzzy set theory concept of membership function to the clustering problem. Specifically, in fuzzy clustering k-means (FCA), a vector is possible to belong to more than one cluster at the same time. The grade of membership of the corresponding vector i to the corresponding cluster j is defined by the corresponding element u_{ij} of the weight matrix U. There is also the constraint that the sum of the memberships – weights of the same vector with regards to all the clusters must be equal to 1.

Using mathematical notation we can express the above constraints:

$$u_{ij} \in [0,1], \qquad \mathrm{i} = 1 \dots \mathrm{N} \tag{3.2}$$

$$\sum_{j}^{m} u_{ij} = 1, \qquad i = 1 \dots N \tag{3.3}$$

$$0 < \sum_{i=1}^{N} u_{ij} < N, \qquad j = 1 \dots m \tag{3.4}$$

Since we want to optimize a function under a constraint, we form the following Lagrangian function:

$$L(U, \theta) = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^{q} d(x_i, \theta_j) - \sum_{i=1}^{N} \lambda_i \left(\sum_{j=1}^{m} u_{ij} - 1 \right) \tag{3.5}$$

Taking the gradient of this function and setting it equal to 0:

$$\nabla L = \begin{bmatrix} \frac{\partial L(U,\theta)}{\partial u_{ij}} \\ \frac{\partial L(U,\theta)}{\partial \theta_j} \\ \frac{\partial L(U,\theta)}{\partial \lambda_i} \end{bmatrix} = 0 \tag{3.6}$$

Expanding the function L makes the calculation of the partial derivatives a bit easier:

$$L(U, \theta) = u_{11}{}^{q} d(x_1, \theta_1) + u_{12}{}^{q} d(x_1, \theta_2) + u_{13}{}^{q} d(x_1, \theta_3) + \dots + u_{21}{}^{q} d(x_1, \theta_1) + u_{22}{}^{q} d(x_1, \theta_2) + u_{23}{}^{q} d(x_1, \theta_3) + \dots - \lambda_1 (u_{11} + u_{12} + \dots - 1) - \lambda_2 (u_{21} + u_{22} + \dots - 1) + \dots$$

Consequently:

$$\frac{\partial L(U, \theta)}{\partial u_{ij}} = q u_{ij}^{q-1} d(x_i, \theta_j) - \lambda_i = 0 \tag{3.7}$$

$$\frac{\partial L(U, \theta)}{\partial \theta_j} = \sum_{i=1}^{N} u_{ij}^{q} \frac{\partial d(x_i, \theta_j)}{\partial \theta_j} = 0 \tag{3.8}$$

$$\frac{\partial L(U, \theta)}{\partial \lambda_i} = \left(\sum_{i=1}^{N} u_{ij} - 1 \right) = 0 \tag{3.9}$$

From equations (3.3), (3.7), for every cluster s, where $1 \leq s \leq m$

$$u_{ij} = \frac{1}{\sum_{j=1}^{m} \left(\frac{d(x_i, \theta_s)}{d(x_i, \theta_j)} \right)^{\frac{1}{q-1}}} \qquad (3.10)$$

Unfortunately, the three equations above cannot give closed-form solutions so, as usually happens in such cases, an iterative scheme comes to our rescue. This is the generalized Expectation Maximization scheme that consists of two steps with the same name:

The expectation step (E-step) where each vector is assigned to the cluster with the most likelihood to belong to and the maximization step (M-step) where the parameters(centroids) are recalculated.

The algorithm can be described as follows:

Input values: data, number of clusters, fuzzifier

- initialize θ_j for every j randomly
- while termination condition:
  - E-step: for i = 1 to N
    - for j = 1 to m
      - $u_{ij} = \frac{1}{\sum_{j=1}^{m} \left(\frac{d(x_i, \theta_s)}{d(x_i, \theta_j)} \right)^{\frac{1}{q-1}}}$
  - M-step: $\theta_j(t) = \frac{\sum_{i=1}^{N} u_{ij}^{q}(t-1) x_i}{\sum_{i=1}^{N} u_{ij}^{q}(t-1)}$

Returns: clustered data

3.1.1 Implementation Notes

At the first part of the function ***fuzzy(data, no_of_clusters, centroids_initial = None, q = 1.25)***, all the necessary structures are being initialized. These are:

The partition matrix: The numpy array U that contains the weights assigned to each vector with regards each centroid.

The centroids_initial: A numpy array containing the initial values of the centroids. These can be generated in a random order, which is the most usual scenario, or provided as a parameter to the algorithm. The necessity that led us to provide this second way of setting up the centroids came from the relative criteria of cluster validity that demanded running the algorithm several times, by amending only certain parameters of it, but not the position of the centroids which had to be the same for the consecutive executions of the program.

Using the numpy function ***choice*** along with the parameter ***replace*** set to ***False***, assures that the centroids would be unique. The program will throw an error if the number of clusters is larger than the length of the array ***np.arange(np.min(data), np.max(data), 0.1)*** which is highly unlikely.

centroids_new: The termination criterion of the algorithm demands to compare the consecutive values of the centroids. This array serves as a structure which will hold the newly created centroid values.

centroids_history: For better visualization of the results we present all the previous positions of the centroids. This array serves as a structure which will hold them together.

The second part of the algorithm includes the iteration until the algorithm converges to a local or global minimum. The first ***for*** loop inside the main ***while*** loop is the expectation step where the values of the partition matrix U are calculated by matrix operations. The second loop is the maximization step where the centroids are being updated with new values. The last important part of the main while loop is where the termination condition is basically checked, as we are implementing a "python version" of the do – while loop, as the algorithm needs to make at least one iteration in any case.

The algorithm calculates also the η(ita) value, which is necessary as a parameter for the execution of the possibilistic clustering algorithm as we will see in the next section.

Finally, each vector i is assigned to the cluster j with the highest u_{ij}.

3.1.2 Disadvantages of the algorithm

(Nikhil, Kuhu, Keller, & Bezdek, 2005) define the disadvantage of the FCA as the fact that noisy points, meaning points that their absolute distance from a centroid has a relatively large value, may be given a large value of membership, even though we should expect these points to have a small, if any, such value. This comes straight from the fact that in the definition of the algorithm we have imposed the constraint (3.3), which practically means that we are obliged to give a membership to every vector in any case, and also the membership of a vector to a cluster depends on the inverse relative distance from the cluster's centroid, as in (3.10), and not of the absolute distance. The writers provide an excellent example in order to demonstrate this point.

3.1.3 Algorithm's testing on synthetic data

- **Blobs**

We have chosen to test the FCA in a challenging dataset. It consists of 4 blobs of 500 vectors and two of the blobs are so close to each other that can be wrongfully assumed to be one cluster. After running the relative fuzzy indices on the dataset we get the following plots:

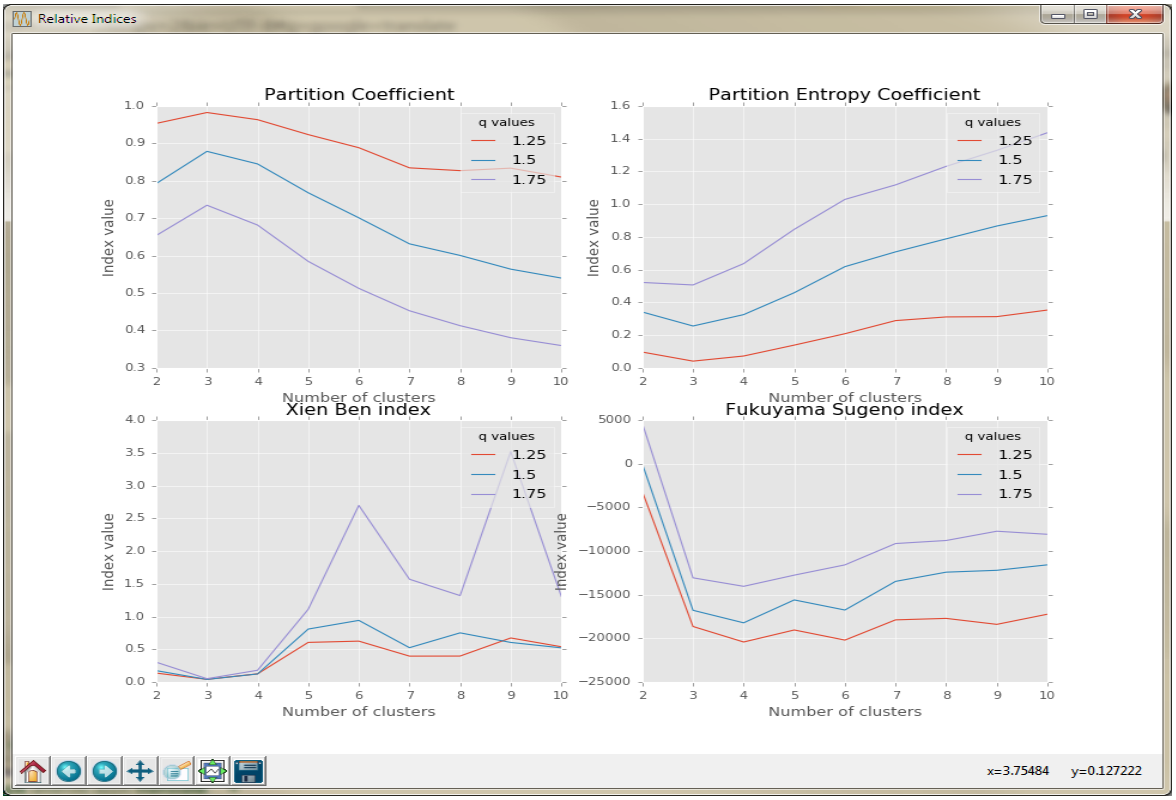

Figure 3 - Relative indices for FCA for 4 blobs of 500 nodes, seed = 46

We notice that three of the four indices (Partition Coefficient, Partition Entropy, Xien – Ben) take their optimal values at m = 3 indicating that 3 is the best number of clusters. The values of these indices at m = 4 however are not very far from their optimal ones. On the other hand, the Fukuyama Sugeno index is the only index that indicates that the optimal number of clusters is 4, something which agrees with the parameters we used when we constructed the dataset. Let us execute the FCA for m = 3 and m = 4.

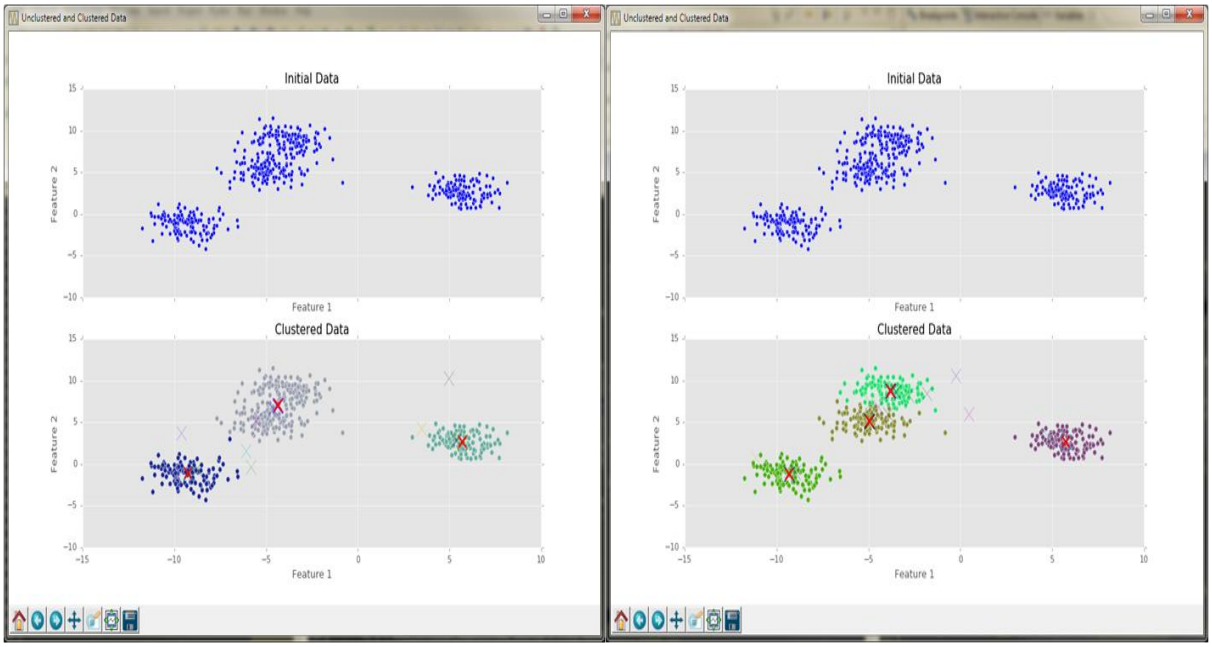

Figure 4 - a) Execution of FCA for m = 3, seed = 46, b) Execution of FCA for m – 4, seed = 46

It is in such datasets where the subjective nature of cluster analysis is revealed. Here we are not in place to provide a clear answer as to whether 3 or 4 clusters are the optimal solution, at least visually. If we rely on the indices things do not become much clearer since they reach to different results according to the methodology they use. One solution to this is always the external criteria indices that compare our clustering result with an external partition and in this case are obviously in favor of the four-clustered solution which we accept as the correct one.

In such case, we distinguish the Fukuyama Sugeno index as the only one which gave a correct, with reference to the external partition, indication of the number of clusters. This does not mean of course that this index will give correct results for all datasets, compared to the other indices. Different indices might perform differently in different datasets.

The internal and external indices for the last execution of the FCA are:

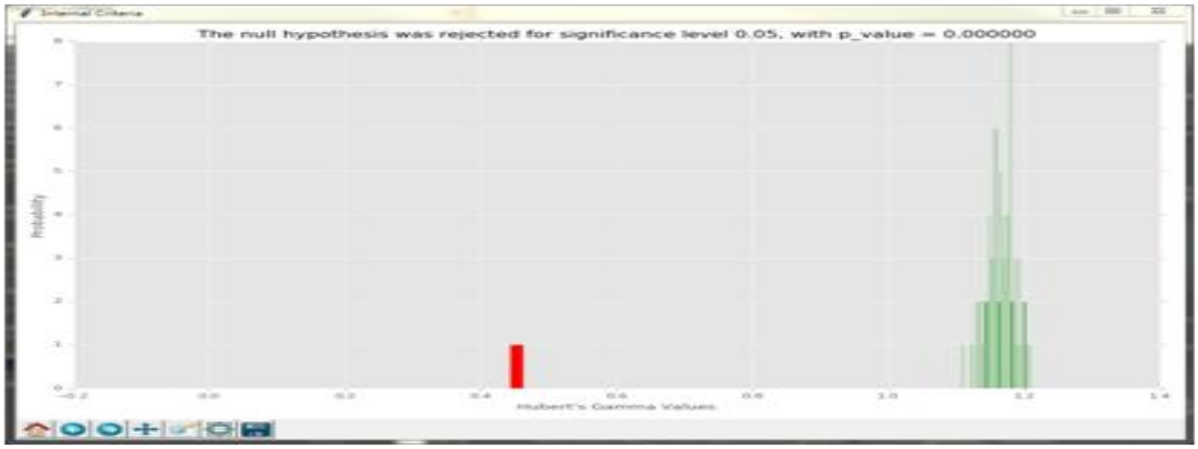

Figure 5 - Internal Criteria, FCA, Gamma index for dataset of 4 blobs, 500 nodes, seed = 46

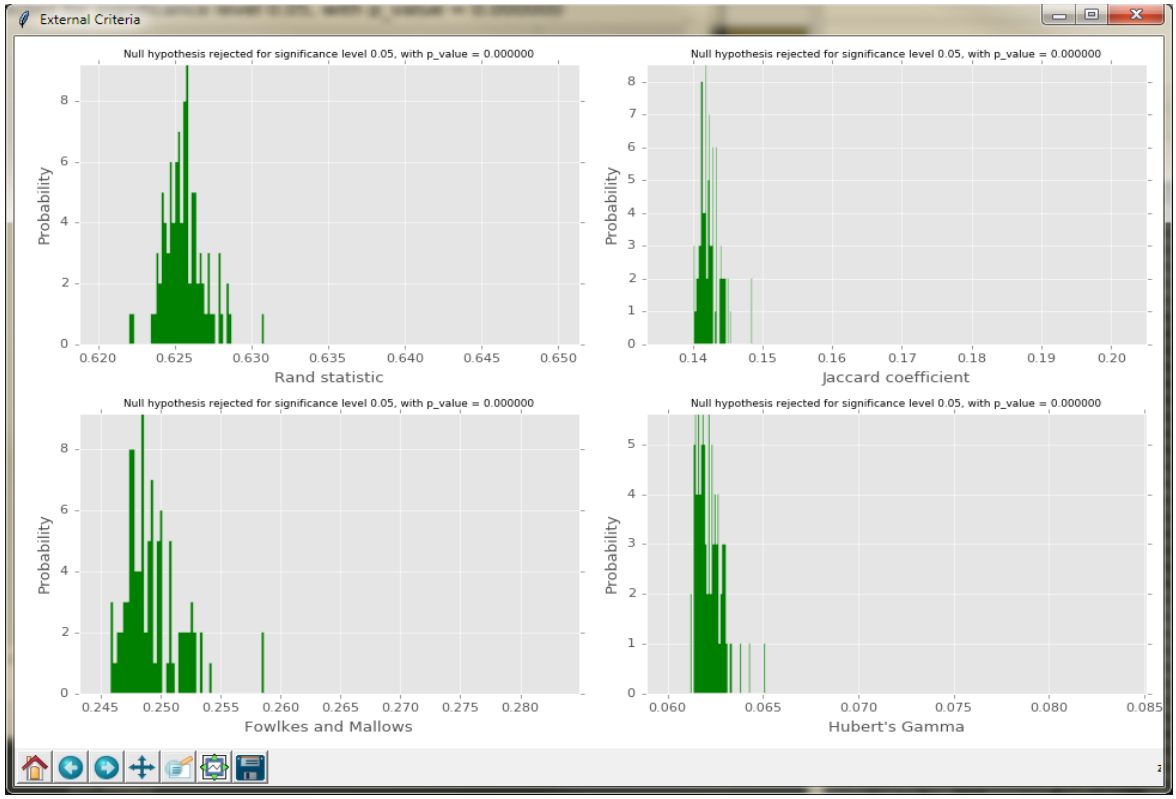

Figure 6 - External Criteria for FCA on dataset of 4 blobs, 500 nodes, seed = 46

The internal criteria reject the null hypothesis of randomness as we expected, so the clustering result is acceptable according to them. With regards to the external criteria, we should check the four indices we are calculating on figure 6 for two things. The first is the absolute value of the indices, apart from the gamma index which is not normalized, and then whether their value accept or reject the null hypothesis. The absolute value of the Rand statistic, the Jaccard coefficient and the Fowlkes and Mallows are very close to 1 (something that cannot be seen in the current printscreen, as it is zoomed-in), which was expected and also they reject the null hypothesis so it is clear that we have an indication of the existence of a structure in our dataset.

- **Concentric Circles**

The FCA fails to reveal the correct clustering structures in cases where the shape of the data is non-spherical. This is not due to the algorithm itself but because of the fact that in the current setup we are using centroids as the cluster representatives. Different representative structures, such as hyperplane or hyperspherical shapes would probably lead to better results in this type of datasets.

Having noted the above, we see the relative indices to simply follow the monotonicity of m and not being able to provide trustworthy indications for the best clustering, since it is impossible to have one.

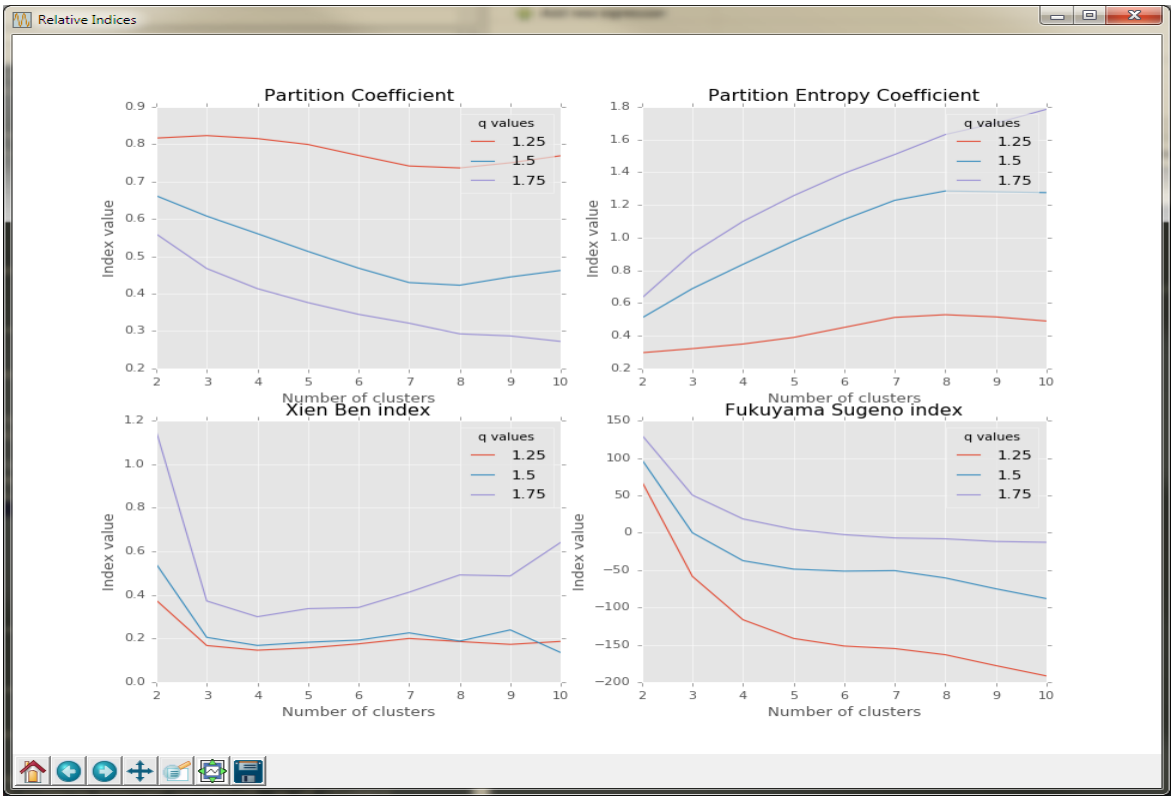

Figure 7 - Relative indices for FCA for 2 concentric circles of 500 nodes, seed = 10

Below we can see the clustering result along with the internal and external indices. What is important is that the external criteria correctly give the indication of a failed clustering attempt by accepting the null hypothesis. The internal criteria on the other hand wrongfully reject the null hypothesis but with a value for the gamma index very close to the values of the uniformingly distributed datasets created during the monte carlo simulations.

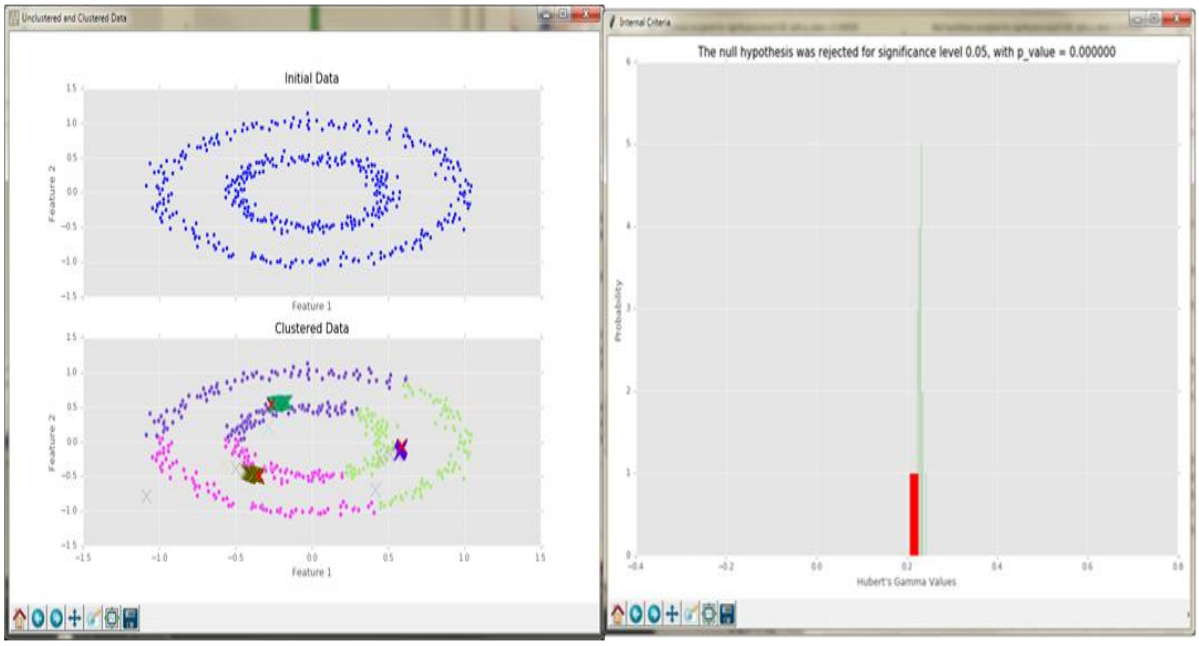

Figure 8 – a) Execution for FCA with m = 3, 2 concentric circles, seed = 10 , b) Internal criteria for FCA with m = 3, 2 concentric circles, seed = 10

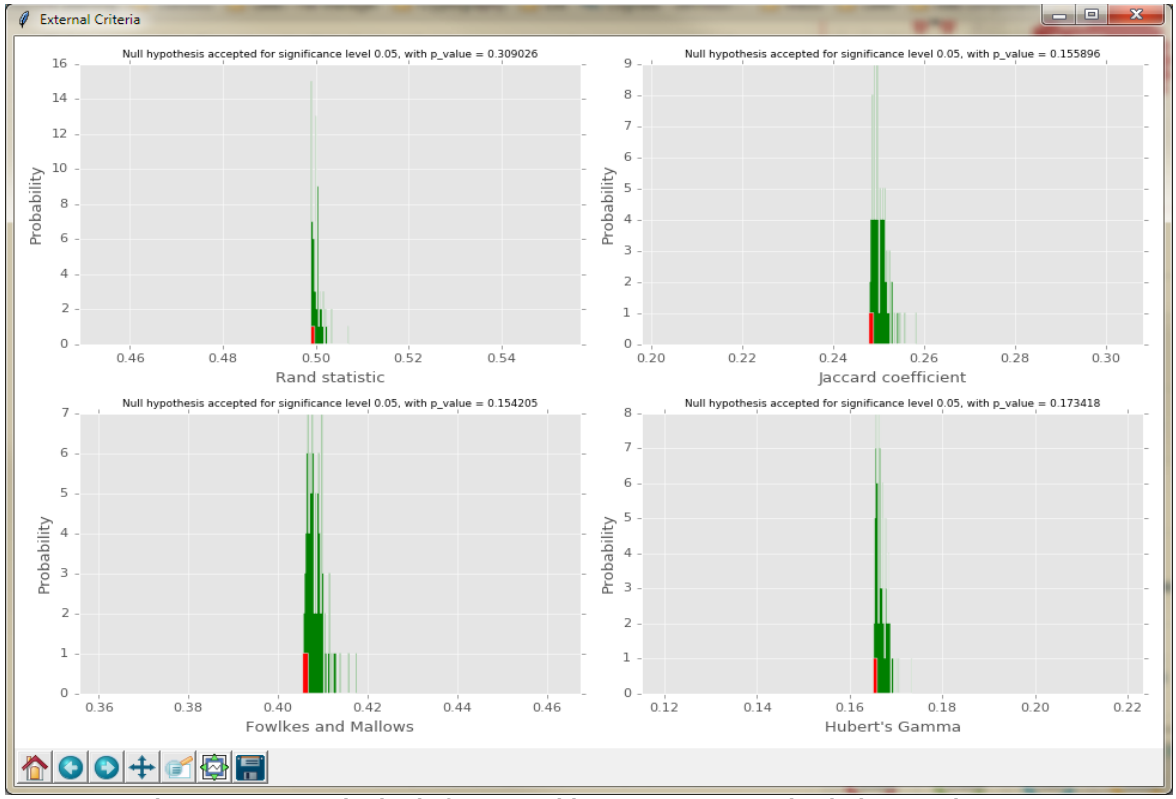

Figure 9 - External criteria for FCA with m = 3, 2 concentric circles, seed = 10

3.2 Possibilistic Clustering

(Zadeh, 1978) attempted to define a theory analogous to probability theory. He called it possibility theory and based it on fuzzy sets definition. He considered as the main application of his new theory the replacement of the probability on the statistical field of communication, which is nowadays known as information theory.

One of the features of possibility theory which will concern us here is the notion of the possibility distribution function. We will not provide a definition, however a description of possibility distribution function is that it assigns to each variable a possibility value which, for each value u of X, is equal to the membership function of u.

The possibilistic clustering algorithm (PCA) was proposed by (Krishnapuram & Keller, 1993) in order to overcome the disadvantage on the noisy data manipulation of the FCA, described in the relative section. As we noted, it is the constraint (3.3) that wrongfully raises the importance of the noisy vectors in the execution of the FCA. Consequently, the basis of the PCA is the relaxation of this constraint. Now, every membership value can belong in the range [0,1] and also:

$$0 < \sum_{i=1}^{N} u_{ij} \leq N \qquad (3.11)$$

Although it may seem like a small amendment, it actually changes the whole interpretation of the elements of the U matrix as defined in the FCA. Under the frame of PCA, each u_{ij} expresses the possibility the vector x_i to belong to cluster c_j. All the values of u_{ij} for $0 \leq i \leq N$ consist of the possibility distribution of the variable X for $X = \{x_1, x_2, \dots, x_N\}$. Another term to name u elements is "typicality" of x_i relative to cluster c_j.

Furthermore, the writers, in order to avoid trivial solutions, suggested that the cost function to be minimized should be added one more term:

$$L(U,\theta) = \sum_{i=1}^{N}\sum_{j=1}^{m} u_{ij}^{q} d(x_i,\theta_j) - \sum_{i=1}^{m} \eta_i \sum_{i=1}^{N} (1-u_{ij})^{q} \qquad (3.12)$$

The differentiation of the above relation with respect to u_{ij} and after some calculations gives us:

$$u_{ij} = \frac{1}{1+\left(\frac{d(x_i,\theta_j)}{\eta_j}\right)^{\frac{1}{q-1}}} \qquad (3.13)$$

This is exactly the part of differentiation between FCA and PCA and at the same time the point that raises all the strengths and weaknesses of the latter. According to the relation above, u_{ij} is inversely proportional to the absolute distance between the vector x_i and the centroid vector c_j. This practically means that the larger the distance, the smaller the value of u_{ij} and the less impact it would have during the next step of the iteration scheme

of the algorithm, at the updating of the centroids. Although this solves the problem of the noisy vectors, as we shall see at the disadvantages of the PCA, makes the algorithm very sensitive to different initialization parameters.

This is the reason the writers propose the initialization of PCA to be based on the output values produced from the FCA, after running on the dataset. These include the initial values of the centroid vectors and the values of the η parameter. More specifically, the values of the η parameter can be estimated as:

$$\eta_j = \frac{\sum_{i=1}^{N} u_{ij}^{q} d(x_i, \theta_j)}{\sum_{i=1}^{N} u_{ij}^{q}} \qquad (3.14)$$

Practically, if we choose to apply this initialization technique, then PCA can be viewed not as a separate algorithm but as an adjustment to the FCA results which moves the centroids more towards the high density areas of each cluster.

Finally, the algorithm described by pseudocode can be found below:

Input values: data, number of clusters, fuzzifier

- initialize $\theta_{j,}, \eta_j$ for every j randomly, or by running FCA
- while termination condition:
  - E-step: for i = 1 to N
    - for j = 1 to m
      - $u_{ij} = \frac{1}{1+\left(\frac{d(x_i,\theta_j)}{\eta_j}\right)^{\frac{1}{q-1}}}$
  - M-step: $\theta_j(t) = \frac{\sum_{i=1}^{N} u_{ij}^{q}(t-1) x_i}{\sum_{i=1}^{N} u_{ij}^{q}(t-1)}$

Returns: clustered data

3.2.1 Disadvantages of the algorithm

The disadvantage of the PCM is that it depends heavily on the initialization values of the centroid vectors. It their values are close to each other, then the result of the algorithm is highly likely to be coincident clusters, although there are cases where this can be an advantage(Nikhil, Kuhu, Keller, & Bezdek, 2005). This is the reason why we usually initialize the algorithm by using the output of the FCM.

3.2.2 Implementation Notes

From the point of view of the programmatical implementation, PCA and FCA are structurally the same, except for some differences in the calculations of the values of their corresponding data structures. This means that the same comments and remarks made at the corresponding section of the FCA are still valid here.

3.2.3 Algorithm's testing on synthetic data

- **Blobs**

We initiate the execution of the algorithm on the synthetic data by executing the relative indices which will give us an indication of the parameters to use. Possibilistic clustering can be considered a type of fuzzy clustering in the sense that every vector can simultaneously belong to two different clusters. Therefore, it makes sense to use the fuzzy clustering indices introduced in section 2.2.2.

Once we get to use them, we notice however that the fuzzy indices do not present similar patterns to the ones produced during the execution of the FCA. They are rather characterized by an increasing tendency with regards to the number of the clusters. It is not difficult to understand the reason.

In FCA the constraint (3.3) guarantees that the sum of all the elements of a partition matrix for one specific vector will always be equal to 1, regardless of the number of the clusters. The value of the fuzzy indices in the consecutive executions of the FCA takes under consideration a steady total amount of partitions of a vector to clusters. In the case of the PCA however the same constraint is eliminated. Now it is possible for a vector to belong in more than one clusters with high values of membership to each one of them. In fact, the larger the number of clusters, the larger is the value of the indices.

One way to overcome this difficulty is proposed in (Miin-Shen & Kuo-Lung, 2005). The authors propose a way to normalize the fuzzy indices in order to use them in the PCA framework. If we denote an index as μ, the calculation of its value for c different number of clusters, which implies c different memberships of each vector, then we can define the generalized validity indices as:

$$\mu'_{ij} = \frac{\mu_i(x_j)}{\sum_{k=1}^{c} \mu_k(x_j)}$$

for i = 1,….c and j = 1,…n.

Finally, we noted in section 3.2 that PCA can be seen as an adjustment to the results obtained by FCA. Let us see what this practically means by executing the PCA in the same dataset, using the centroids returned by FCA as initialization values for the centroids in the PCA.

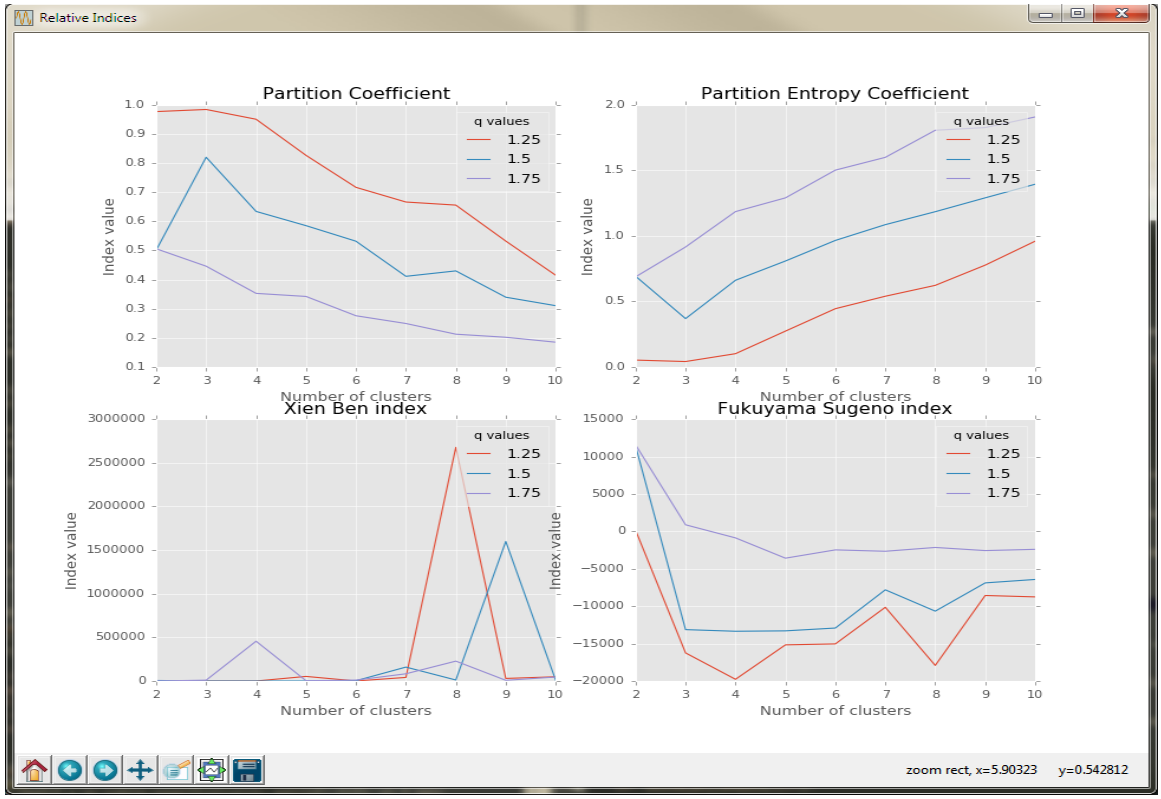

Figure 10 - Relative indices for PCA for 4 blobs of 500 nodes, seed = 46

Here we have a repetition of the comments we made at the FCA section where we executed the algorithm on blobs. We can see that Partition and Partition Entropy Coefficient indicate that 3 is the best number of clusters, however not for q = 1.75. The same is valid for Xien - Bien (although it is not easily seen in the plot). Fukuyama Sugeno again takes its optimal value for number of clusters equal to 4.

We choose to execute the PCA with parameters q = 1.25 and m = 3. The result is:

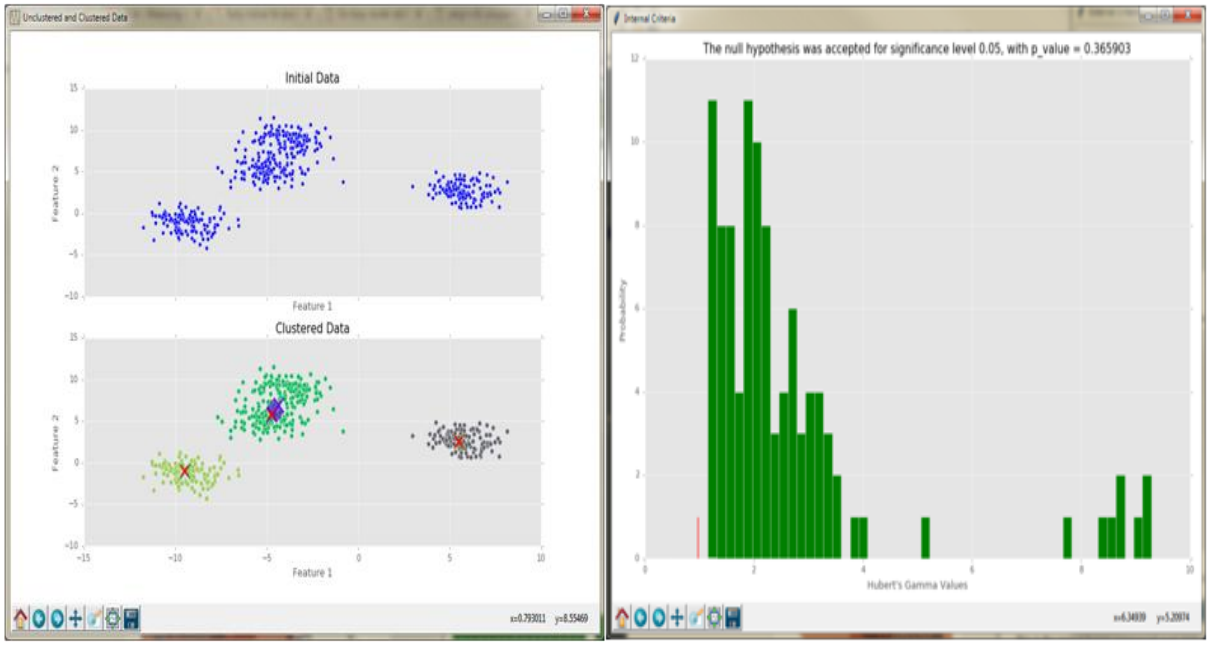

Figure 11 – a) Execution for PCA with m = 3, q = 1.25, 4 blobs, seed = 46, b) Internal Criteria Gamma index for dataset of 4 blobs, 500 nodes, seed = 46

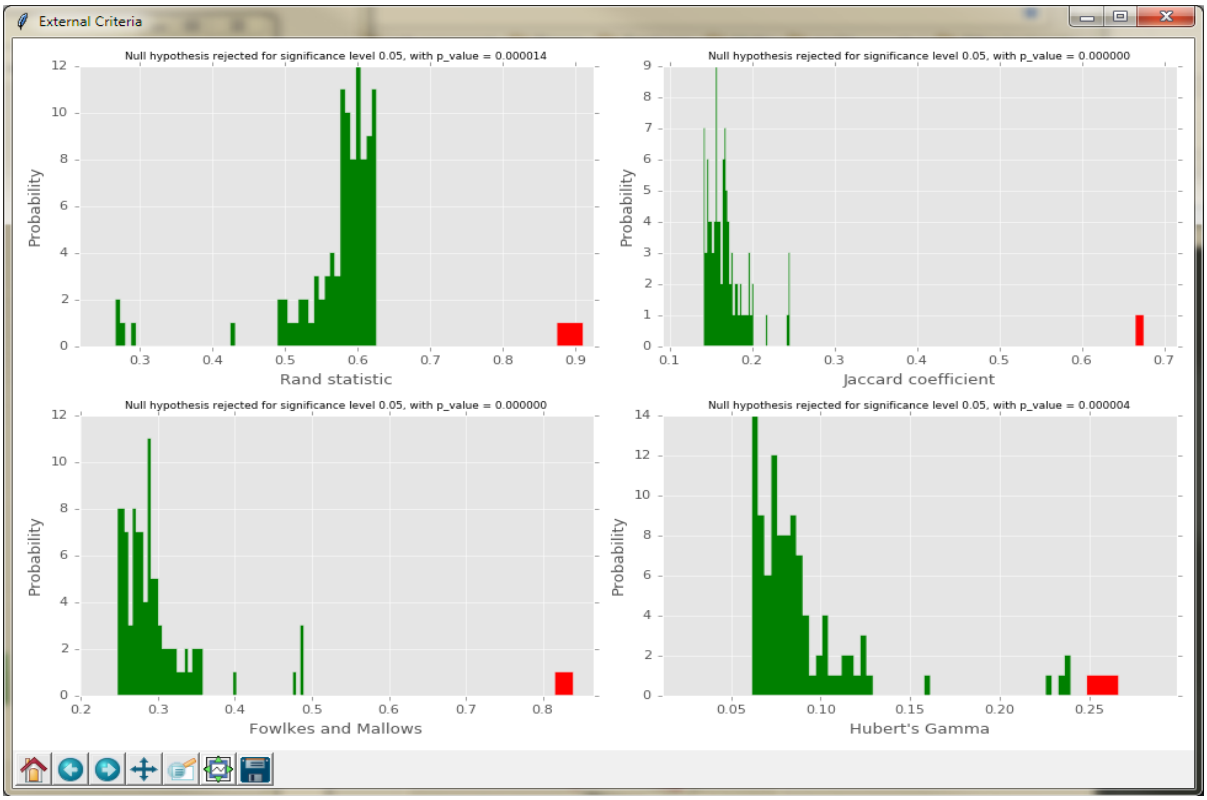

Figure 12 - External Criteria for PCA on dataset of 4 blobs, 500 nodes, seed = 46

In the second scatter plot in figure 11 a) it can be easily observer, in the cluster depicted with the green color, how the PCA behaves as a refinement procedure by pulling the centroids towards the most high density areas of the clusters.

The external criteria verify that our clustering is a good partition of the dataset. On the other hand, the internal criteria do not agree with this result if we consider the distribution of the gamma index a normal distribution. However, we can see that the gamma index for our clustering is lower than any other index run on the random datasets, so this is an indication of a good clustering.

- **Concentric Circles**

The relative indices can be found below:

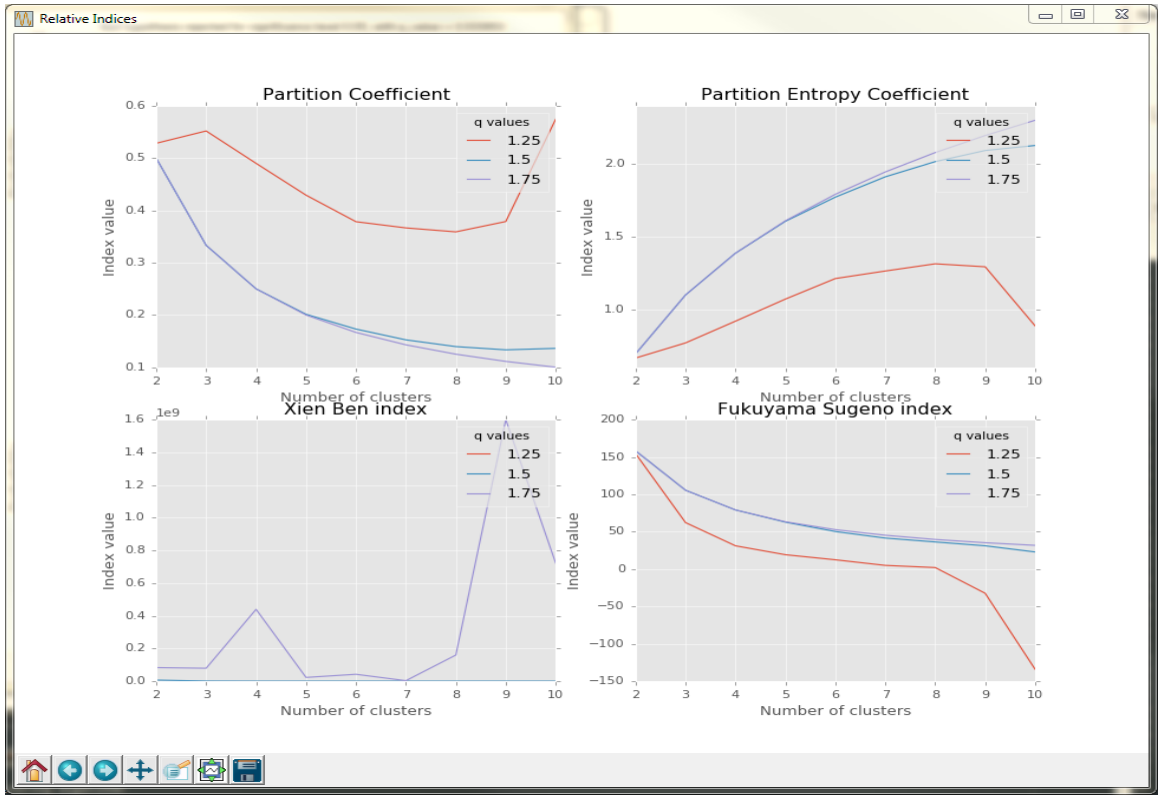

Figure 13 - Execution for PCA with m = 3, 2 concentric circles, seed = 10

As in the case of FCA executed on the same dataset, we note that the indices are not able to give a correct indication on the number of clusters and the value of the fuzzifier parameter.

Finally we can see below that most of the external indices reject the null hypothesis, however the result is not trustworthy. First because the distribution of the values of these indices cannot be considered a normal one and second because the values of the indices is small, something which indicates a low grade of matching between our clustering result and the external criteria.

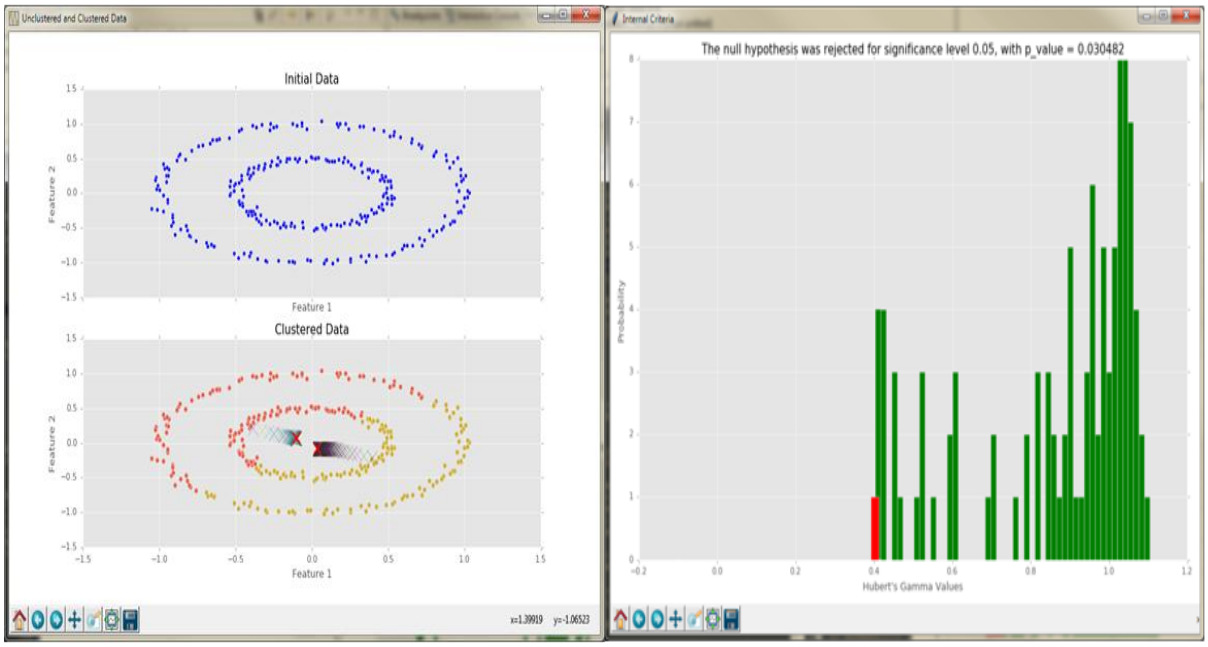

Figure 14 – a) Execution for PCA with m = 3, 2 concentric circles, seed = 10, b) Internal criteria for PCA with m = 3, 2 concentric circles, seed = 10

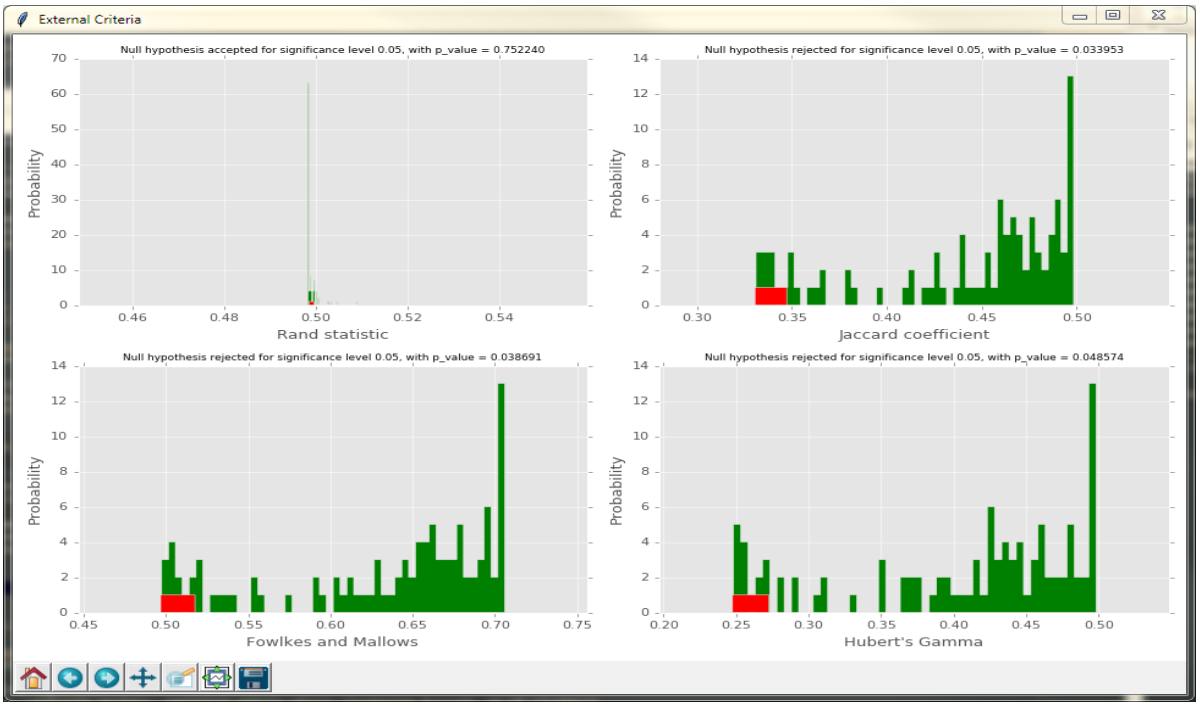

Figure 15 - External criteria for PCA with m = 3, 2 concentric circles, seed = 10

- **Moons**

Since the comments are the same with the concentric circles dataset examined before, we just provide the results of the execution without further comments:

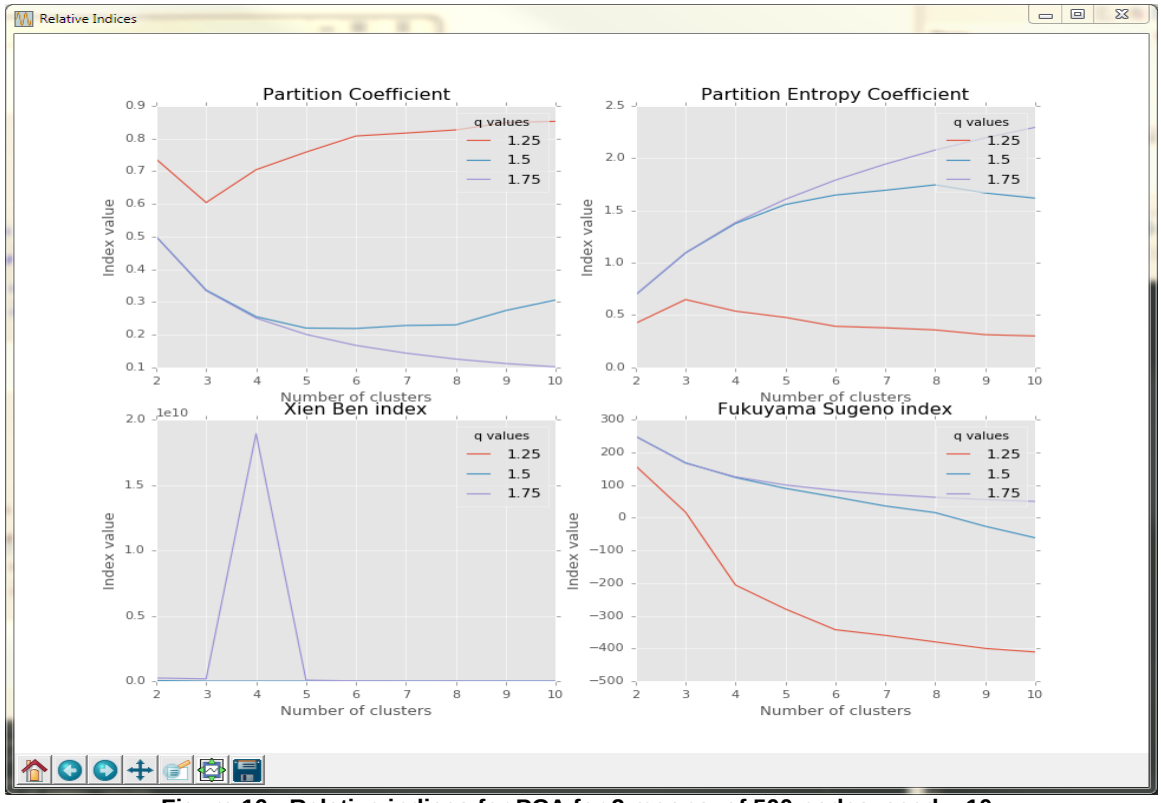

Figure 16 - Relative indices for PCA for 2 moons, of 500 nodes, seed = 10

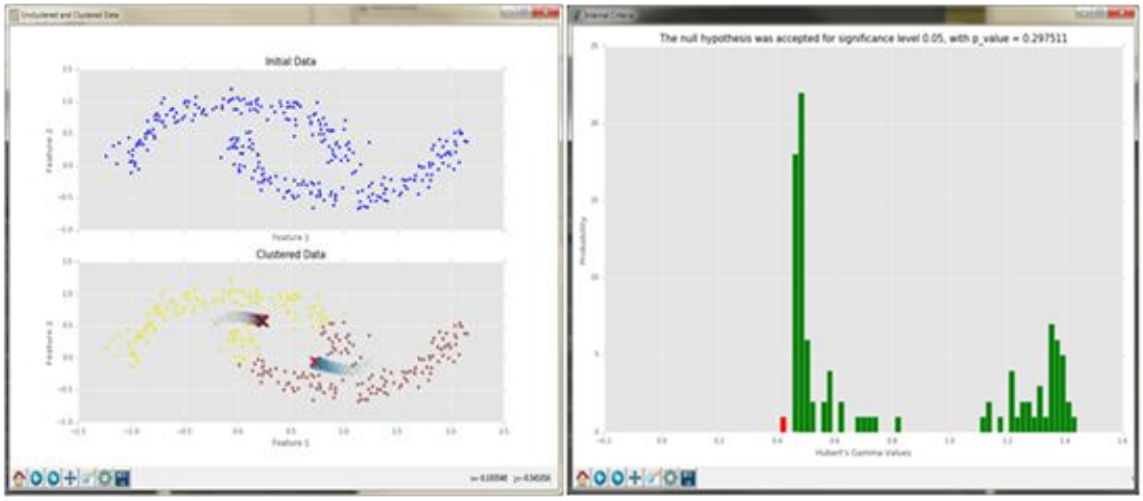

Figure 17- a) Execution for PCA with m = 3, 2 moons, seed = 10, b) Internal Criteria Gamma index for dataset of 4 blobs, 500 nodes, seed = 46

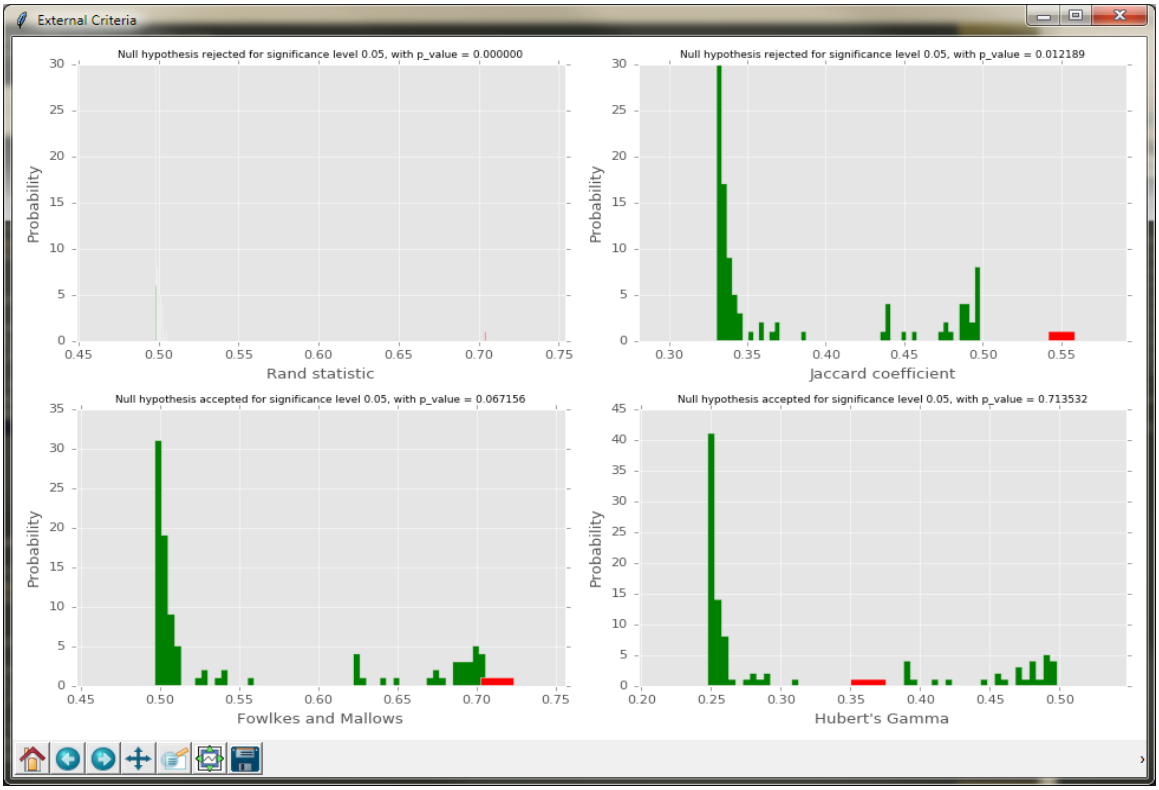

Figure 18 - External criteria for PCA with m = 3, 2 moons, seed = 10

3.3 Hard Clustering Algorithms

All the algorithms of this category are based on the minimization of the cost function 2.1 where $u_{ij} \in \{0,1\}$. The value 0 is assigned when the vector does not belong to a cluster whereas 1 is assigned when the vector belongs to a cluster. This is the reason why this type of clustering is called "hard" as opposed to "fuzzy". Every vector either belongs to one specific cluster or not.

In order to easily minimize the cost function, the following argument is used: if we assign each vector to its closest cluster, then the cost function will be minimized. This can be expressed in notation form:

$$u_{ij} = \begin{cases} 1, & if\ d(x_i, \theta_i) = min_{k=1,\dots m} d(x_i, \theta_k) \\ 0, & otherwise \end{cases}$$

The most known representatives of this category are k-means(Lloyd, 1982) and isodata algorithms (Ball & Hall, 1965). They are based on the same notion of minimization of the cost function, with their only difference being that the isodata allows for different number of clusters and does not require a predefined number. Their success is due to the fact that they are the simpler, the least sophisticated and therefore the fastest clustering algorithms.

We will now examine the k-means algorithm as a representative of the hard clustering algorithms. The pseudocode can be found below:

Input values: data, number of clusters

- initialize $\theta_{j,}, \eta_j$ for every j randomly, or by running BSAS
- while termination condition:
  - E-step: for i = 1 to N
    - for j = 1 to m
      - $u_{ij} = \begin{cases} 1, & \text{if } d(x_i, \theta_i) = \min_{k=1,\dots m} d(x_i, \theta_k) \\ 0, & \text{otherwise} \end{cases}$
      - set b(i) = j
  - M-step: *Set θ_j as the mean of the vectors $x_i \in X$ with $b(i) = j$*

Returns: clustered data

3.3.1 Disadvantages of the algorithm

K-Means algorithm is very sensitive to the initial positions of the centroids. The naïve approach to their initialization is to pick up random positions in the space defined by the clustering task. Most of the times this leads the algorithm to a convergence to a local minimum of the cost function, even to simple clustering tasks. In order to overcome this major drawback we first execute a sequential algorithm on the dataset, such as BSAS that we will see in the next chapter, and we initialize the centroids of the k – means by setting them equal to the centroid values returned by the sequential algorithm.

We should remember however that sequential algorithms are not being passed the number of clusters as an argument. This means that the number of clusters returned cannot be defined a priori. In cases where the sequential algorithm

returns a larger number of centroids than the number we want to feed to our k – means algorithm, then we pass to the k-means only the number of the centroids demanded. In the opposite case, we initialize randomly the excessive centroids. It is a technique that in practice displayed very good results.

3.3.2 Algorithm's testing on synthetic data

- **Blobs**

We will first execute the k-means algorithm to a dataset of 4 blobs that can be characterized as a "perfect" one, since all the relative indices give indications of 4 as the perfect number of clusters. We can see the plots below:

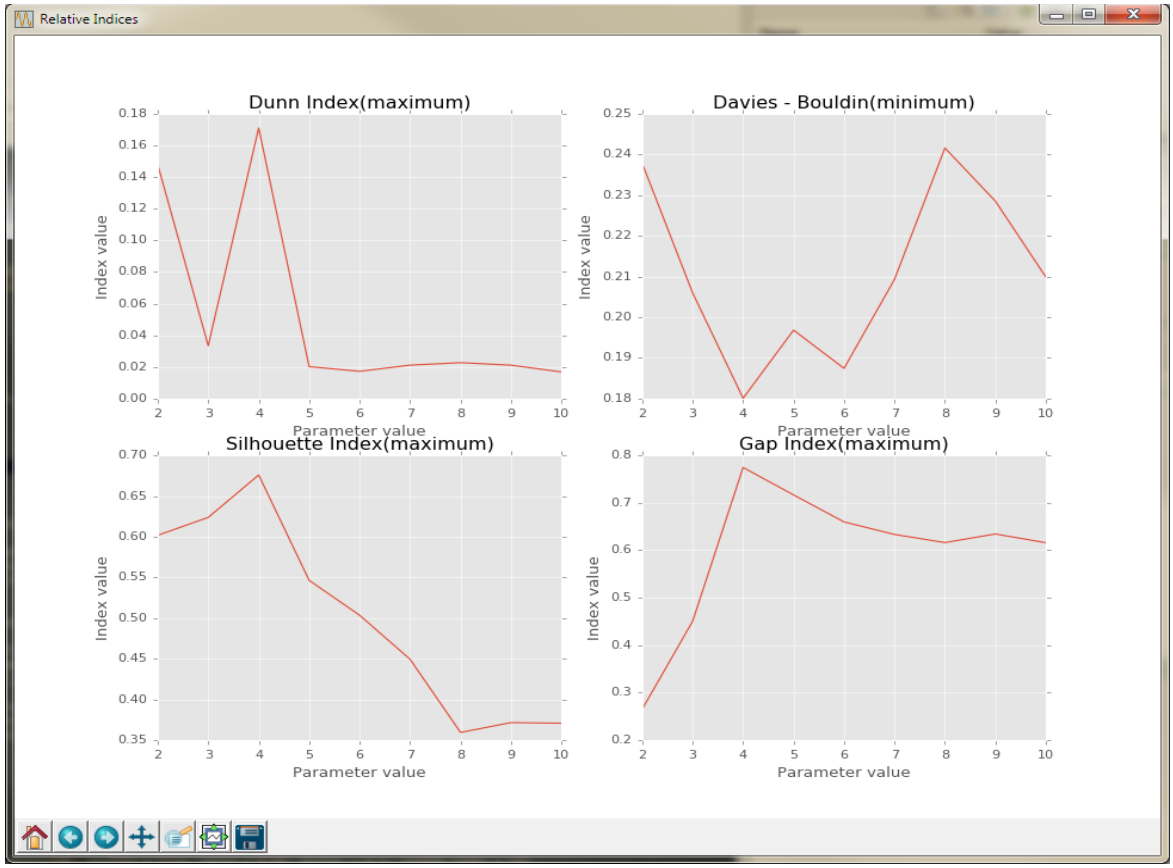

Figure 19 - Relative indices for k-means for 4 blobs, of 500 nodes, seed = 199

The actual clustering result provides us with a visual evaluation that the clustering was the best one possible. So do the external and internal criteria:

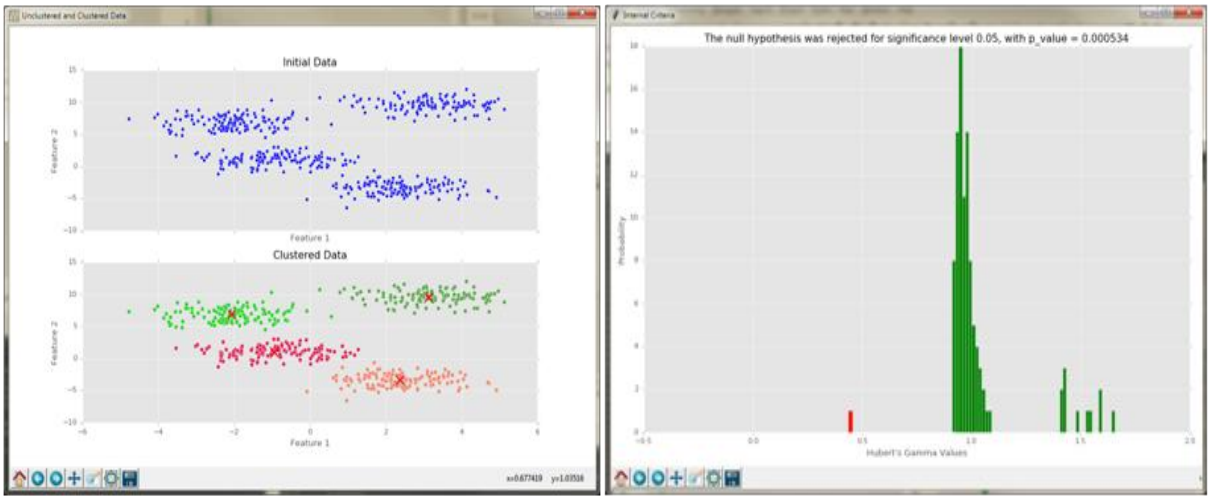

Figure 20 – a) Execution for k-means for 4 blobs with m = 4, seed = 199, b) Internal Criteria for k-means for 4 blobs with m = 4, seed = 199

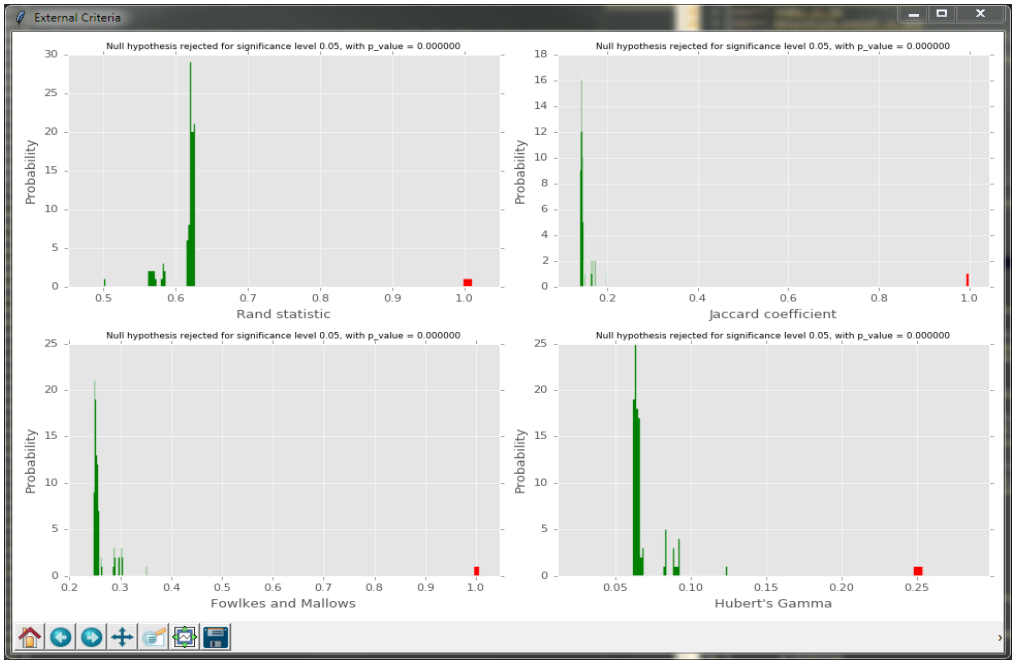

Figure 21 - External Criteria for k-means for 4 blobs with m = 4, seed = 199

Let us see however another execution on a different dataset.

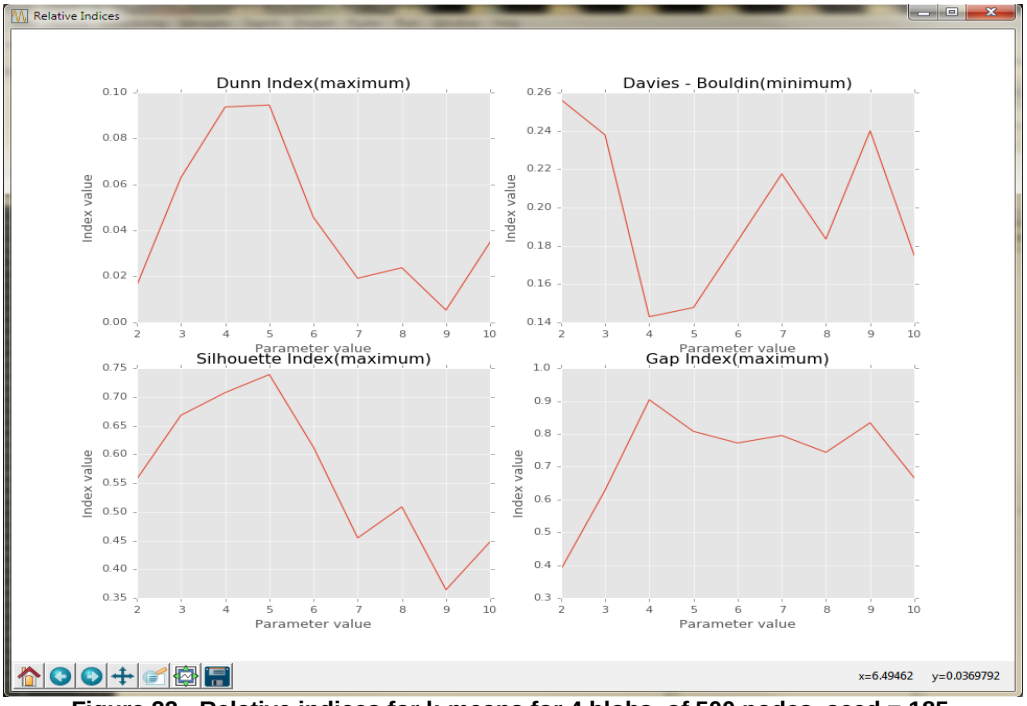

Figure 22 - Relative indices for k-means for 4 blobs, of 500 nodes, seed = 185

In this case, Dunn and Silhouette indices take their optimal values for m = 5, whereas Davies – Bouldin and Gap indices take their optimal values for m = 4. If we execute k-means for the two different values of m we get the following results:

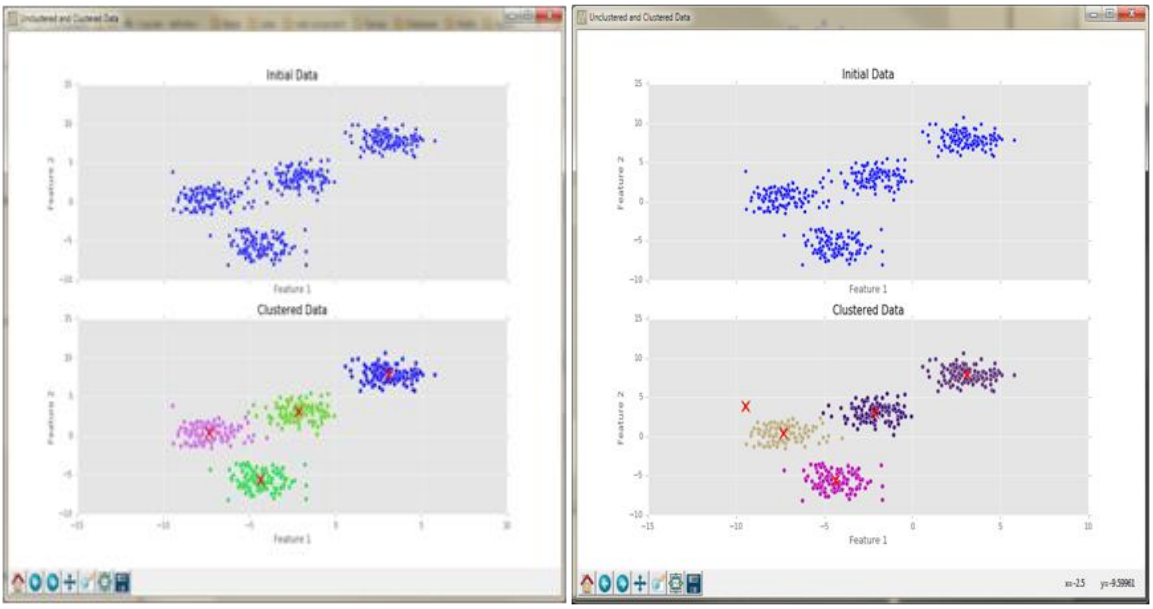

Figure 23 - a) Execution for k-means for 4 blobs with m = 4, seed = 185, b) Execution for k-means for 4 blobs with m = 5, seed = 185

It is obvious that the values of Dunn and Silhouette indices are misleading due to the existence of one vector that could be characterized as

"noisy" since it is placed in a quite big distance from its centroids in the case of m = 4. When this vector is detached from its cluster and consists of one new cluster itself in the case of m = 5, then the results of the two indices are better.

The conclusion is that the Dunn index and the silhouette index are not able to give us good indications of the correct number of clusters if they are executed in noisy datasets. Calculating indices by using extreme values, minimum or maximum, is not such a good indicator of a good clustering such as indices that are based on average values.

- **Concentric Circles - Moons**

K-means, executed by using centroids as cluster representatives cannot give satisfying results in datasets which form clusters with non compact shape. For reasons of completeness we present the result of the executions below.

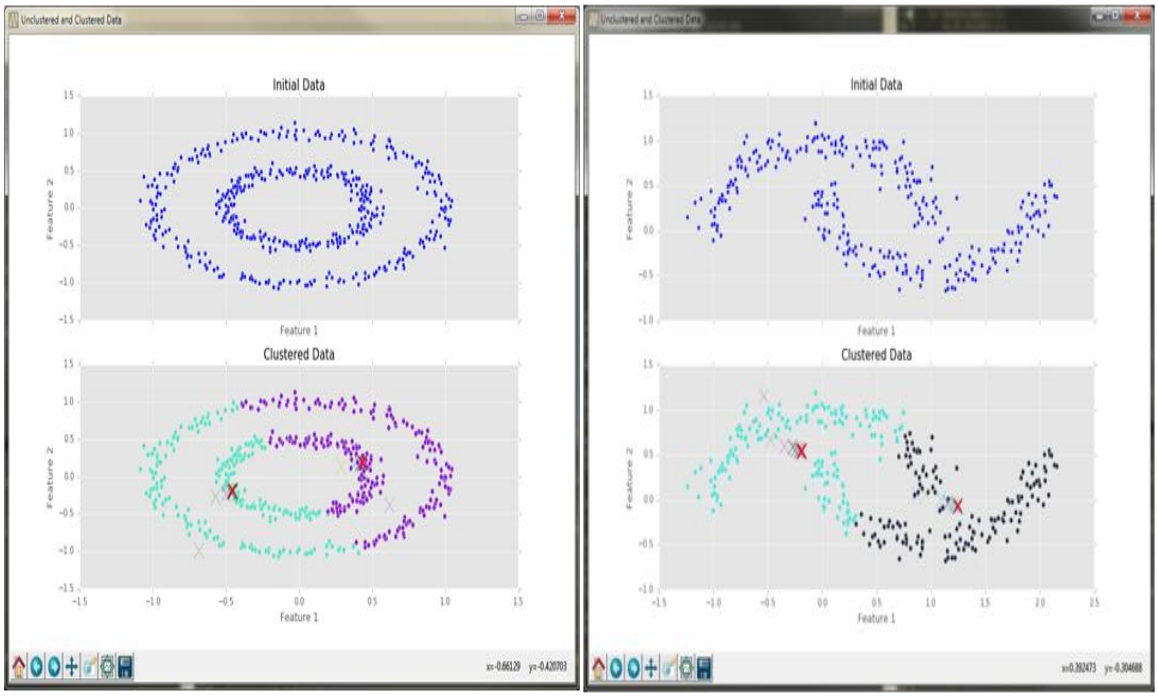

Figure 24 – a) Execution for PCA with m = 2, 2 concentric circles, seed = 10, b) Execution for k-means with m = 2, 2 moons , seed = 10

Chapter 4
Sequential algorithms

The term "sequential" comes from the field of statistics call "Sequential analysis"(Ghosh & Sen, 1991). It is used to describe the unique feature of this field, to analyze the available data not all in once but partially. This technique usually applies to cases where the whole data are not available from the beginning but are collected gradually.

Sequential clustering algorithms follow this exact technique. The main loop of the algorithm does not handle all the dataset as a group, as in the case of other non-sequential clustering algorithms, but in each step it confines itself to handling just one vector and immediately proceeds to the updating of the parameters of the algorithm (such as for example the centroids).

4.1 Basic Sequential Algorithmic Scheme (BSAS)

The BSAS is described in (Theodoridis & Koutroumbas, 2009) as a generalization of a scheme provided in (Hall, 1967). As the writers suggest the basic idea of the algorithm is to sequentially feed it the vectors of the dataset, one at a time, and either assign the vector to an already existing cluster, if its distance from this cluster is less than the value of a threshold θ, or use it to initialize a new cluster if this distance is larger than the value of θ.

BSAS is a very good choice of clustering algorithm for big data, since it is very fast, its time complexity is only O(N). It also applies to cases where the data are not available all at once, but are gradually fed to the algorithm.(Trahanias & Skordalakis, 1989) mention as an example of such scenario the monitoring of a patient's electrocardiogram for several days. The patterns in the electrocardiogram *"have to be clustered as soon as possible in order to detect the presence or absence of certain abnormalities"*.

The pseudocode can be found below:

Input values: data, threshold, (optional)maximum number of clusters
- Use the first vector of the data to create a new cluster
- Update the centroids by adding this new cluster
- For each other vector x_i in data:
    - count the distance from x_i to the clusters' centroids
    - take the closest cluster
    - if the closest distance is smaller than θ:
        - add it to the closest cluster
    - else:
        - create a new cluster
        - Update the centroids

Reassignment Procedure
- For every vector in data:
    - find the closest cluster
    - assign the vector to the closest cluster
- For every cluster:
    - update the representatives

Outputs: The labeled data, the centroids

4.1.1 Implementation Notes

As in all algorithms implemented in this thesis, we express programmatically the final clustering by adding a column at the end of the data matrix which holds the number assigned to each cluster. In this implementation we keep two copies of the data matrix, one without this column, the **data** matrix, and another including this column, the **clustered_data** matrix. This has been decided mainly for performance –and convenience- reasons, so that we do not resort all the time to slicing the data matrix, in order to leave the cluster column out.

The most important point in the BSAS implantation is the calculation of the threshold value out of the histogram of the vector distances we are describing in the next section. In order to calculate the histogram without plotting it we have used the function **numpy.histogram(X, bins)** instead of the **hist** function of the matplotlib.pyplot. We also used the **argrelextrema(X, comparator)** function of the scipy.signal package in order to calculate the peaks and valleys of the histogram.

4.1.2 Disadvantages of the algorithm

The main disadvantages of the BSAS algorithm are the following:

- The order the vectors are fed into the algorithm plays an important role to the final clustering results. The proposed remedy is to execute the algorithm many times and each time to present the data in different order (Theodoridis & Koutroumbas, 2009). After that, the choice of the number of the clusters should be the most frequent number of clusters resulting from all the different executions. Although this technique does not completely addresses the problem, it still is a good way to approach the best clustering partition.
- The user is responsible to provide a value for the θ threshold. This can be difficult if there exists no indication at all about the magnitude of the distances between vectors. In order to overcome this we have developed an automatic technique in order to automatically provide a suggested value for θ to the algorithm.

In order to choose the proper value for the threshold θ, we resort to the image segmentation processes of the computer vision field. One of the methods used there is called thresholding and consists of defining a fixed constant T, the threshold. Every pixel in a grayscale image that has intensity greater than t is replaced by a white pixel and on the other hand, every pixel that has intensity less than T is replaced by a black pixel. A description of the several techniques used in order to define the threshold value can be found in (Mehmet & Bulent, 2004). The technique we are choosing to adapt to our clustering algorithm is called “Peak and valley thresholding”.

The first stage of this technique is the same for BSAS and for the sequential algorithm that we are going to see in the following section, TTSS, and consists of constructing the histogram of the different distances between the vectors of our dataset. The second stage includes “reading” the histogram and extracting the threshold directly out of it.

In the case of BSAS we should think that the distances between the vectors of the dataset should roughly be divided into two categories. One is distances between vectors belonging to the same cluster which would be small and the

other, distances between vectors belonging to different clusters which would be large. The way this is depicted in the histogram is by two peaks separated by a valley. Obviously our next step is to find the peaks of the histogram and define which of them would represent the small distances between each cluster and which one the large distances between vectors of different clusters.

If the histogram has only two peaks, then our job is easy. However, many times this is not the case. In such cases, where the histogram has more than two peaks, we are choosing the two highest. After the peak determination, we search for the valley between the peaks. Again, if the number of valleys is only one, then we choose it. If the number of valleys is greater than one, we choose the deepest valley. The value of the threshold is the value of the distance at the point where the histogram takes its lowest value.

As an example, let's see the histogram below that corresponds to a data set of 10 blobs created from Gaussian pdfs of a total number of 1000 vectors.

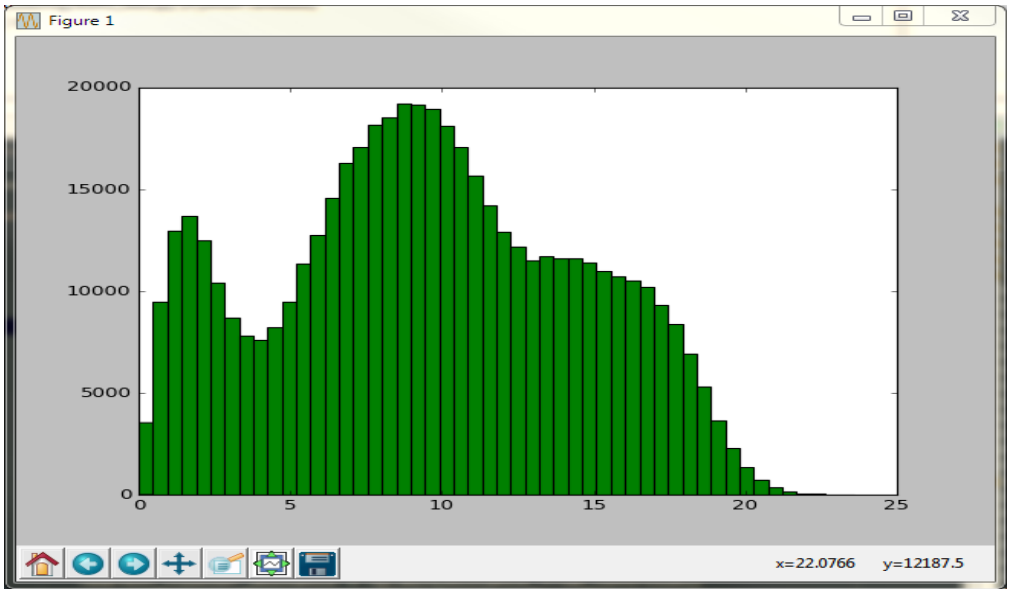

Figure 25 Histogram of the distances of 1000 vectors of 10 Gaussian pdfs

In this histogram the two peaks are very obvious at x = 1.5 and x = 8.4. The valley between them is at x = 3.778, which is defined as the threshold for our algorithm.

In the next section's algorithm, the Two Threshold Sequential Scheme (TTSS), which is a variation of the BSAS, the technique is the same except from the fact that since we are seeking for two threshold values, one that would separate the smaller distances from the middle distances and one between the middle and the large distances, we are now searching for three peaks. The values of the thresholds would be the distance at the two deepest valleys between the three peaks. In our example below we can see the third peak at x = 13.4 and the second valley at 12.73.

As every technique that has to do with cluster analysis, automatic thresholding with the proposed technique does not give good results in a very noisy dataset. This is why in many cases, manual inspection and several tests need to be applied, as every different dataset needs different manipulation.

The automatic procedure should be regarded more as an indication of the value of the threshold rather than a method that always gives correct results.

- After a vector is assigned to a cluster it cannot be reassigned to another one. In cases where, at the final results of the algorithm, a centroid has moved close to a vector that does not belong to the cluster it represents, this results in bad clustering. This can be easily seen in the example below.

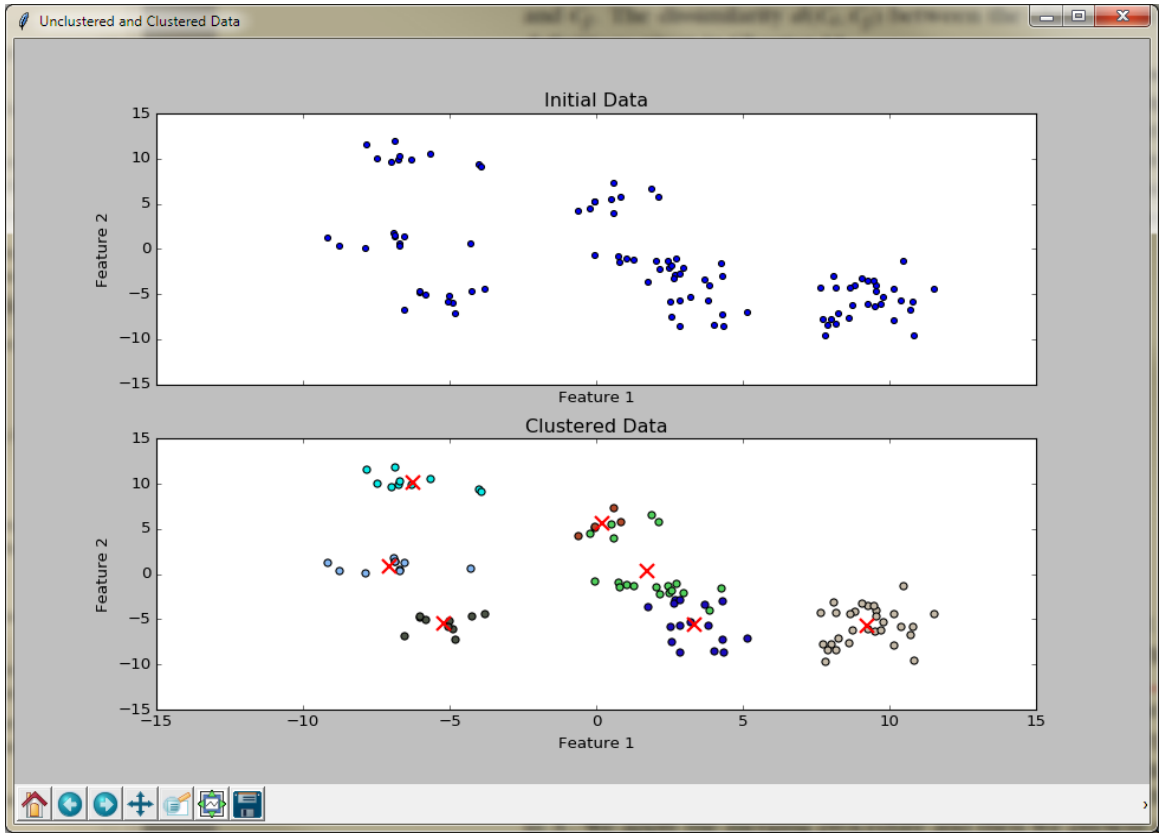

Figure 26 BSAS final results without Reassignment

We can see from the first scatter plot above that some of the vectors belong to the cluster with green color, although they are closer to the centroid of the cluster with the red color. As noted before this happens because these green colored vectors were initially assigned to the green cluster, however at the end of the execution of the algorithm, the red cluster centroid approached them very closely.

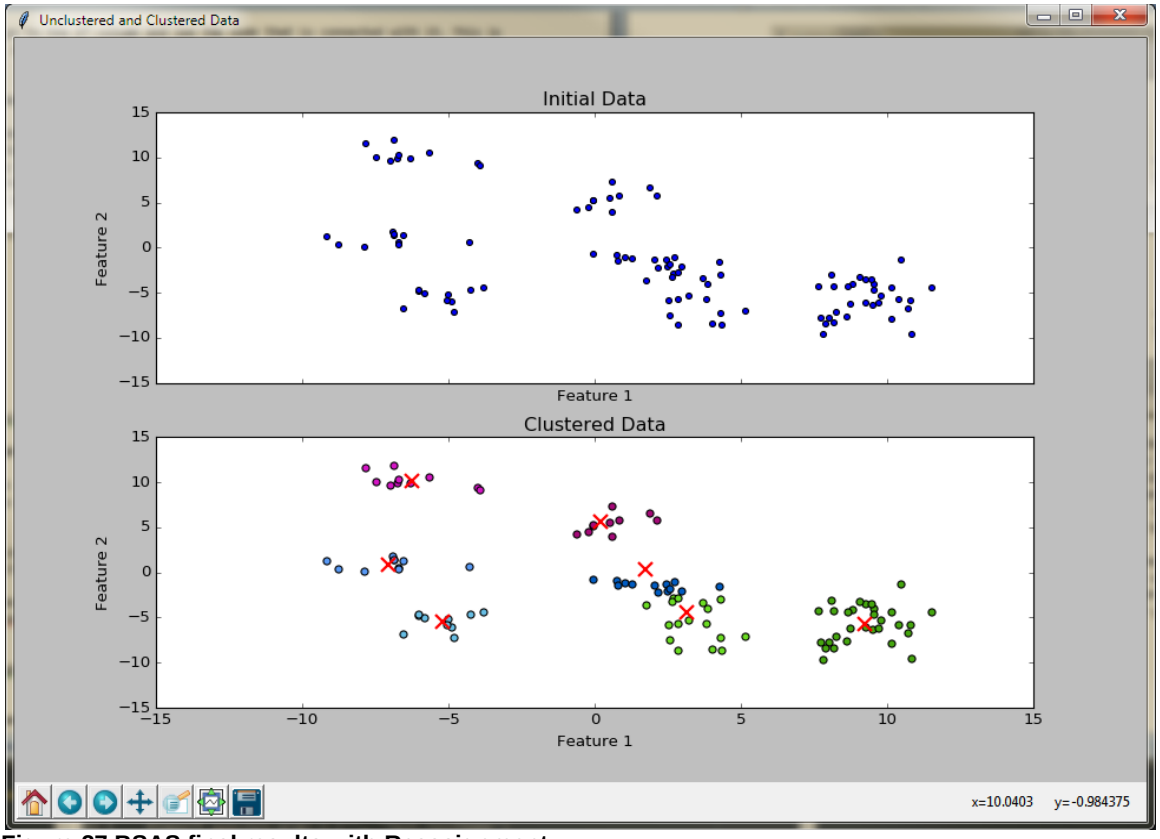

Figure 27 BSAS final results with Reassignment

The way we are dealing with this inconsistence is by performing a reassignment procedure at the end of the BSAS algorithm, essentially a second pass over the dataset, during which we assign each vector to the cluster that is closer to it. The result for the same dataset appears at the second scatter plot.

4.1.3 Testing on synthetic data

- **Blobs**

In order to calculate the indices for the crisp clustering results of the BSAS algorithm we have proceeded in a basic adjustment in the relative criteria script. In this case, the indices are calculated with regards to the different clustering results that take place after amending the value of the threshold parameter of the BSAS. This is contrary to the relative indices of the cost minimization algorithms where we were amending the number of clusters throughout the different executions. As a general rule, the relative indices for every algorithm are calculated with regards to the different parameters this algorithm is provided with. After running the relative criteria we get the following results:

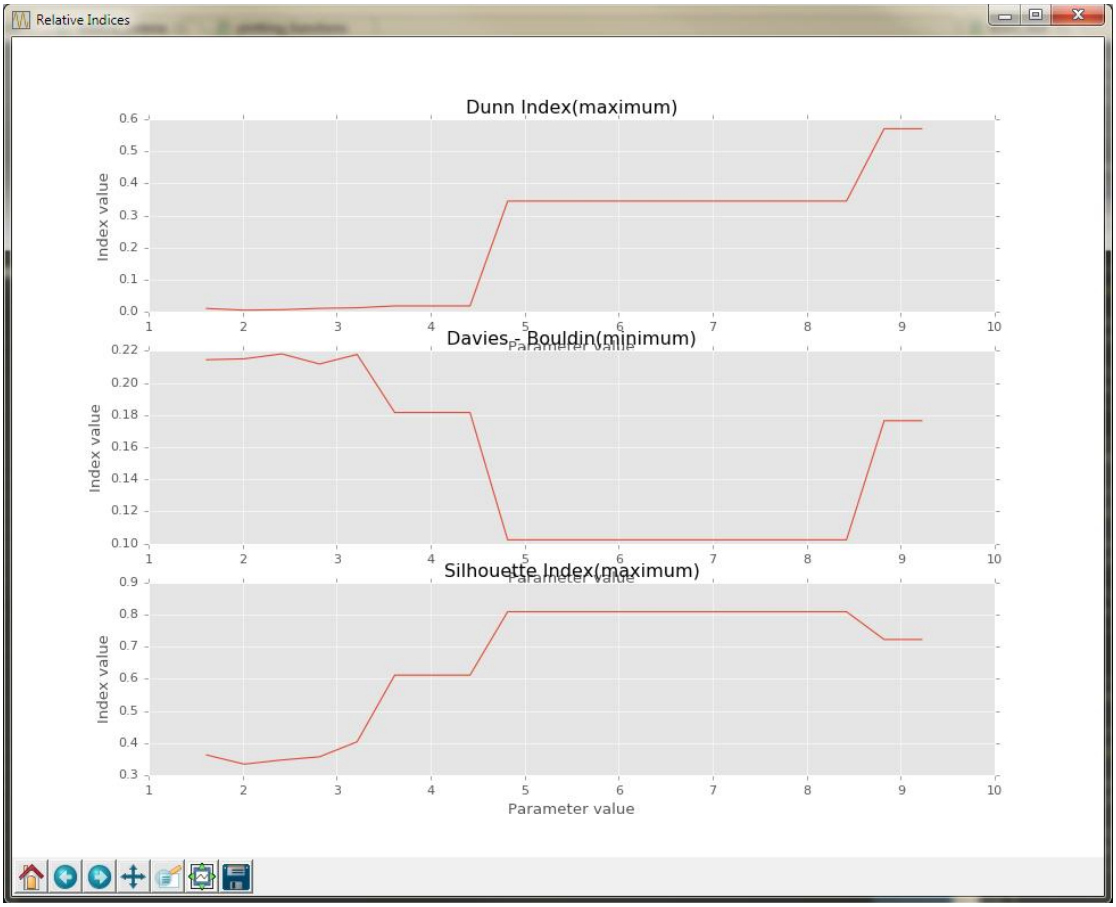

Figure 28 - Relative Criteria Indices for 4 blobs of 500 nodes, seed = 121

We can see from the plots above that the last two out of the three indices agree that the best clustering is taking place when the value of the threshold is in the range [4.8, 8.5]. On the other hand, Dunn Index suggests that the best clustering is obtained for values of t > 8.4.

Let us execute the BSAS algorithm for t = 5 and t = 9 in order to include both cases:

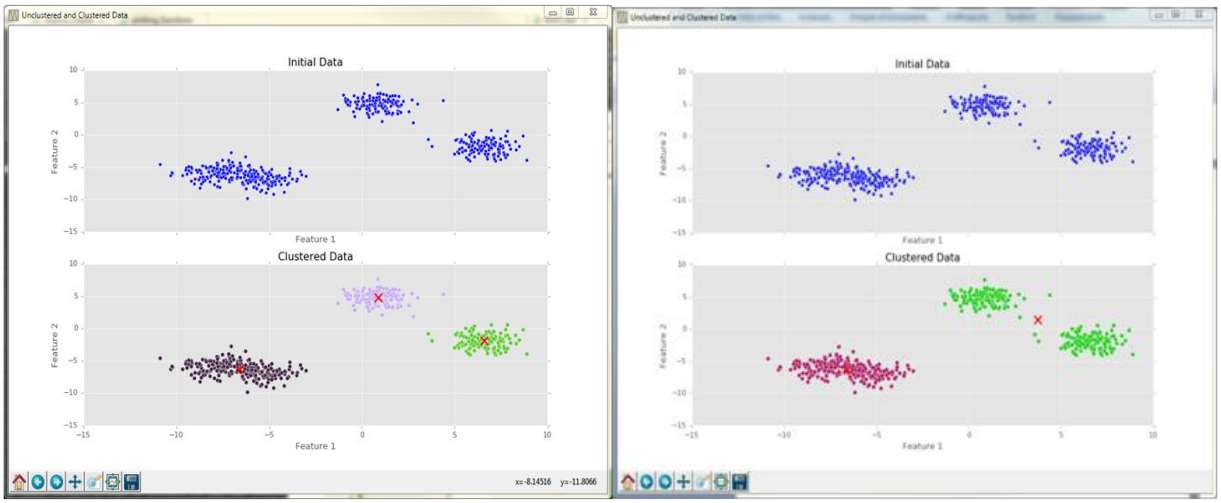

Figure 29 – a) Execution of BSAS for t = 5 for 4 blobs of 500 nodes, seed = 121, b) Execution of BSAS for t = 9 for 4 blobs of 500 nodes, seed = 121

What we can visually confirm is that the correct value for the threshold θ is 5. This value partitions the dataset into three clusters which can be considered the correct number.

The question raised here is why the Dunn Index gives us a clearly wrong indication about the partition. A general answer is that this index is dependent on extreme, maximum and minimum, values rather than average ones. A more specific answer is that in the partition of figure 29 a), the minimum distance between the clusters is 2.79 and the maximum diameter of a cluster is 8.087. This raises the value of the index at 2.79/ 8.087 = 0.35. In figure 17 b) where two of the clusters are merged into one, the minimum distance between clusters becomes 8.07 and the maximum diameter 14.15. This means that the minimum distance is raised by 2.89, whereas the maximum diameter only by 1.74. It is this relatively larger increase in the nominator of the index that gives us a larger value which leads to wrong indications about the value of the threshold.

Having figured out the correct value for the threshold t we can now run the external and internal criteria:

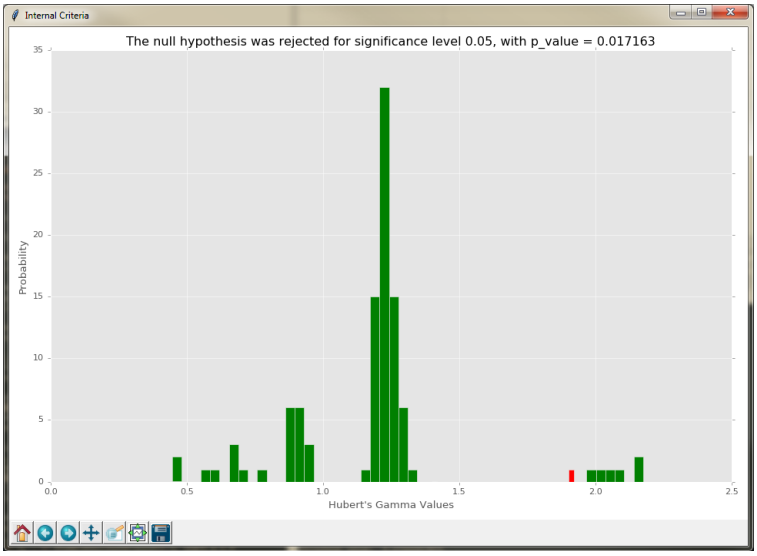

Figure 30 - Internal Criteria Gamma index for dataset of 4 blobs, 500 nodes, seed = 121

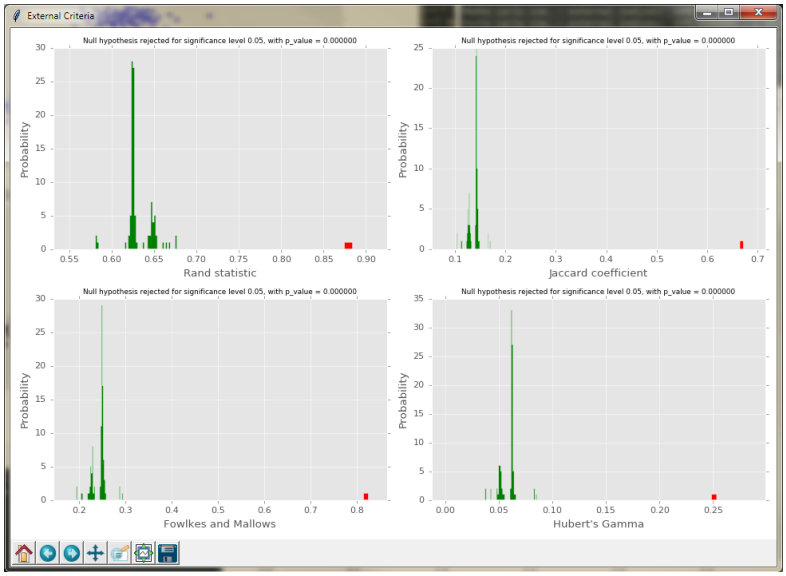

Figure 31 - External Criteria for dataset of 4 blobs, 500 nodes, seed = 121

The internal criteria reject the null hypothesis as expected. The external criteria also reject the null hypothesis however the absolute values of the first three are not close to one. The reason is that according to the external partition the number of the clusters should be 4. The two clusters however are so close that it is inevitable to consider them as one.

- **Concentric Circles**

We do not expect the BSAS algorithm to be able to recognize clusters of other shapes, except from compact ones. In this section and the next one we are interested to see how this failure is expressed in the values of the indices and in the actual clustering. The plotting of the relative indices can be seen below:

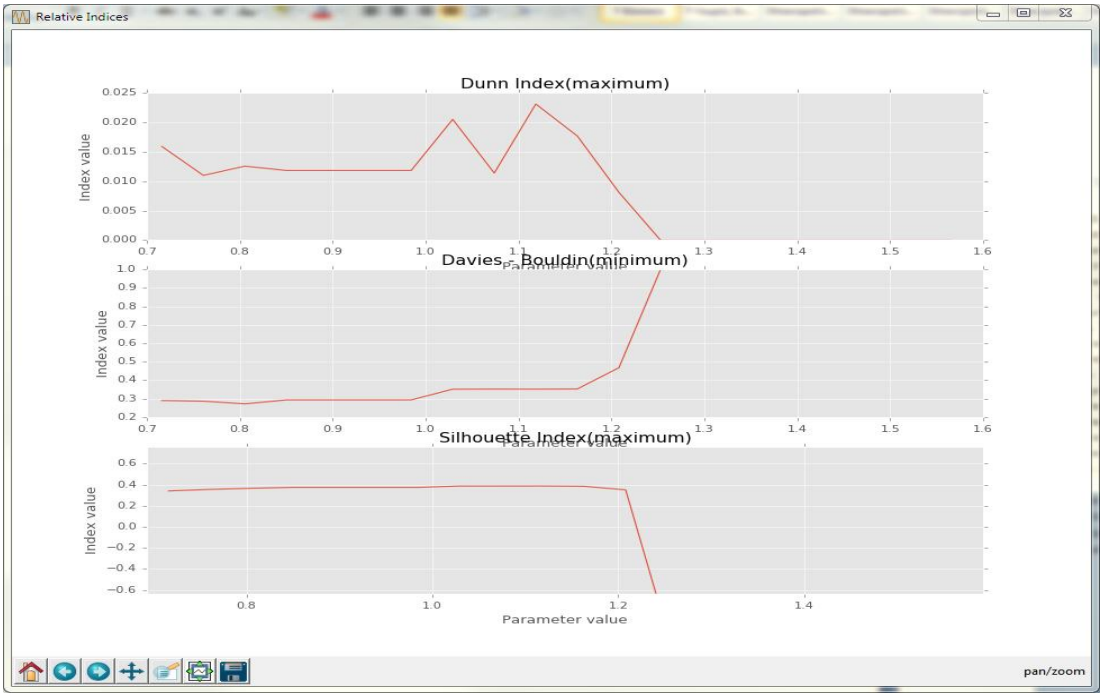

Figure 32 - Relative Criteria Indices for BSAS for 2 concentric circles of 500 nodes, seed = 121

Compared to the plot of the previous test case, where we examined compact clusters, we can see that the values of the indices do not generally agree to some specific values for the threshold to give a good clustering. If we execute the BSAS with t = 1.1 which is the maximum value of the Dunn Index the result is the following:

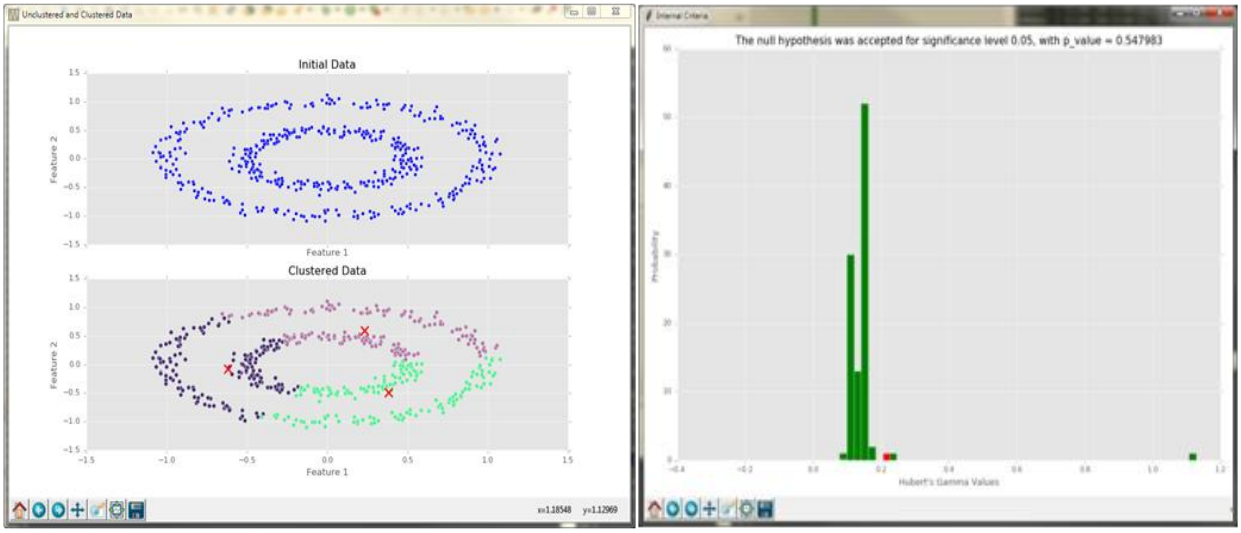

Figure 33 – a) Execution of BSAS for t = 1.1, concentric circles of 500 nodes, seed = 121, b) Internal Criteria for dataset of 2 concentric circles, 500 nodes, seed = 121

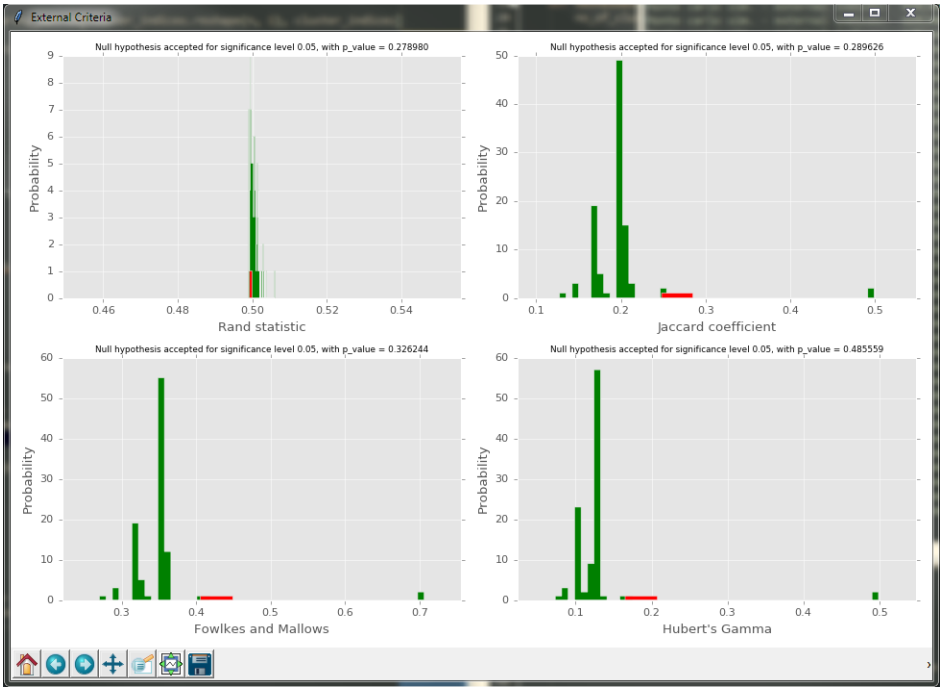

Figure 34 - External Criteria for dataset of 2 concentric circles, 500 nodes, seed = 121

As expected, the values of all the indices, internal and external, make us accept the null hypothesis of randomness which means that our clustering did not produce good results.

- **Moons**

The same comments made for the test on the concentric circles also apply in this test scenario. The null hypotheses of the external criteria are rejected as our dataset presents a structure, however the values of the indices are very low.

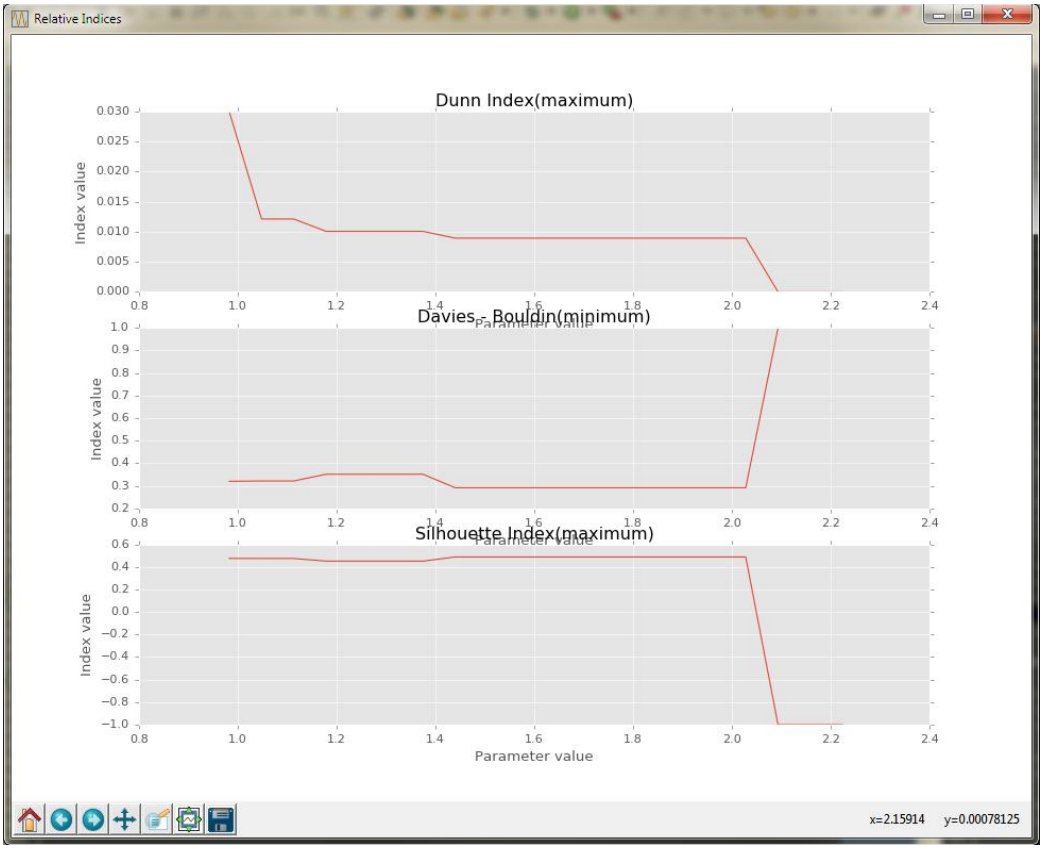

Figure 35 - Relative Criteria Indices for 2 moons of 500 nodes, seed = 121

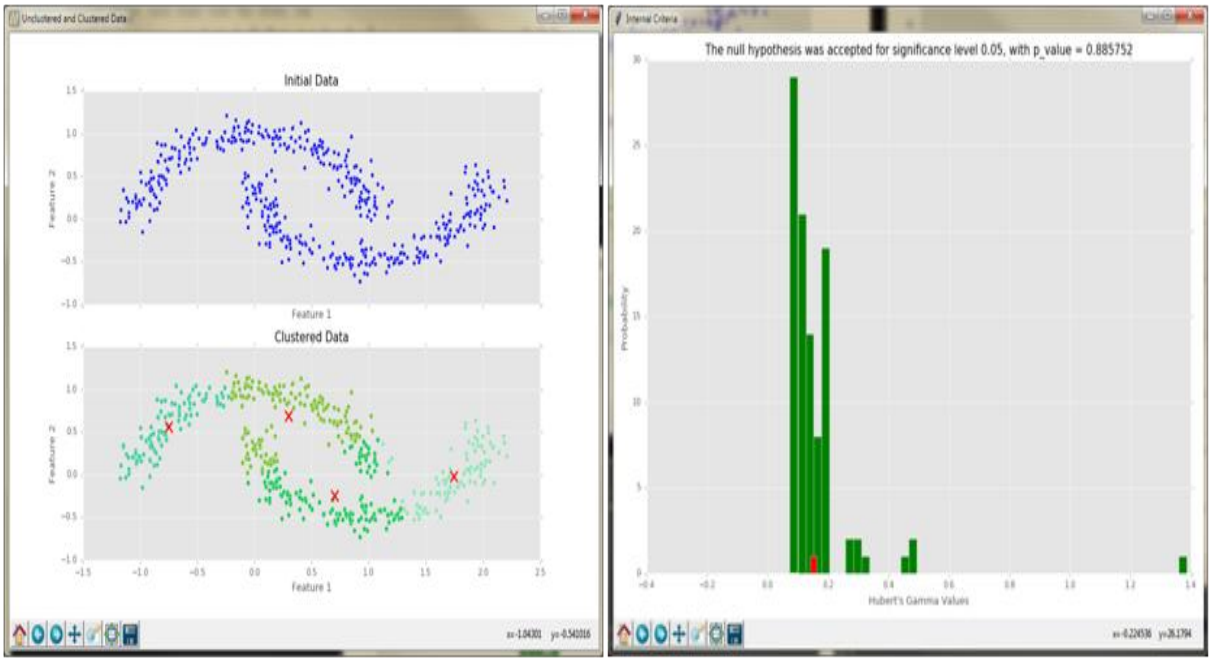

Figure 36 – a) Execution of BSAS for t = 1.1, moons of 500 nodes, seed = 121, b) Internal Criteria for dataset of 2 moons, 500 nodes, seed = 121

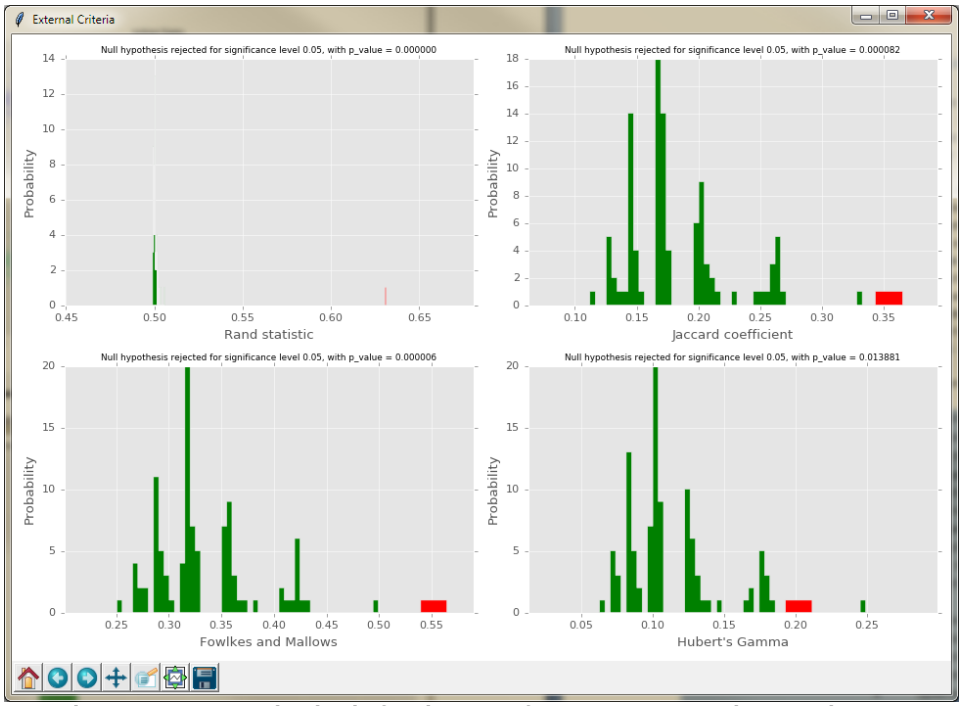

Figure 37 - External Criteria for dataset of 2 moons, 500 nodes, seed = 121

4.2 Two Threshold Sequential Scheme (TTSS)

TTSS was proposed by (Trahanias & Skordalakis, 1989) as a slightly alternative approach to sequential clustering that would deal with the disadvantage of BSAS of not being able to reassign a vector to another cluster after it has been assigned to one, as we described at the relative section of this thesis.

In the BSAS the distances are divided into two categories, the ones "smaller" than the value of the threshold θ and the ones "larger" than it. The writers introduced another distance between these two categories, the "middle" distance. In order to determine this extra kind of distance we need to define two thresholds instead of one let them be T_1 and T_2, $T_1 < T_2$. If the distance d between a vector and the nearest cluster is less than T_1 or more than T_2 then the algorithm works exactly the same way the BSAS does. If however $T_1 < d < T_2$, then the TTSS puts these vectors aside and tries to assign them to a cluster after having assigned all other vectors. If they still remain at this "middle" distance area, then one of them forms a new cluster and the procedure goes on until no vector remains unassigned.

The pseudocode of the algorithm can be seen below:

Input values: data, threshold1, threshold2

- Use the first vector of the data to create a new cluster
- Update the centroids by adding this new cluster
- while there are still unprocessed vectors:
    - For each other vector x_i in data:
        - if the x_i is not processed:
            - count the distance from x_i to the clusters' centroids

- take the closest cluster
  - if the closest distance is smaller than 0:
    - add it to the closest cluster
  - else:
    - create a new cluster
    - Update the centroids
- If no vector was assigned to a new or preexisting cluster:
  - create a new cluster
  - Update the centroids

Outputs: The labeled data, the centroids

4.2.1 Disadvantages of the algorithm

The TTSS algorithm exchanges a less sensitivity to the order of data presentation with more complexity, but most of all with making the user responsible to define a second threshold value, along with the first one.

4.2.2 Testing on synthetic data

- **Blobs**

After running the relative criteria we get the following results:

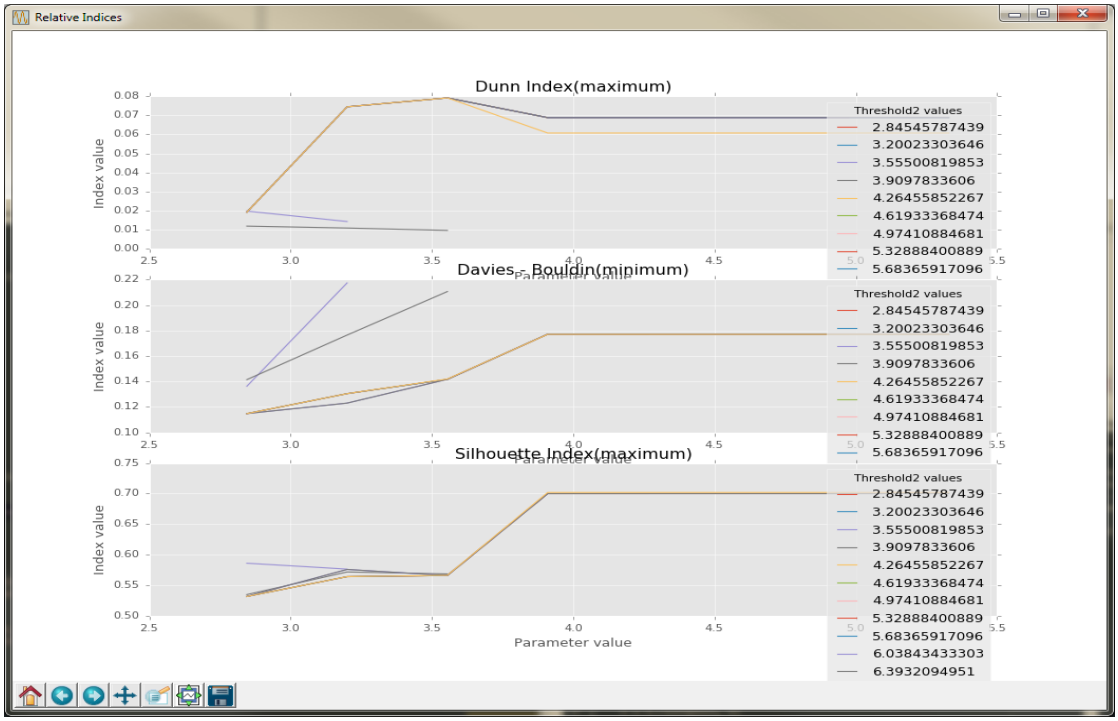

Figure 38 - Relative Criteria Indices for TTSS for 4 blobs of 500 nodes, seed = 124

What is noteworthy is that each index takes the same values for many of the different values of the threshold2. This is why there are so many values for threshold2 but so few line plots.

Every index indicates a different set of values for the two thresholds as the optimal one. For Dunn index this is t1 = 3.55, t2 = 4.26, for Davies-Bouldin t1 = 3.20, t2 = 3.55 and for silhouette index is t1 = 4, t2 = 4.26. Let us execute all the above combinations of thresholds. The results are:

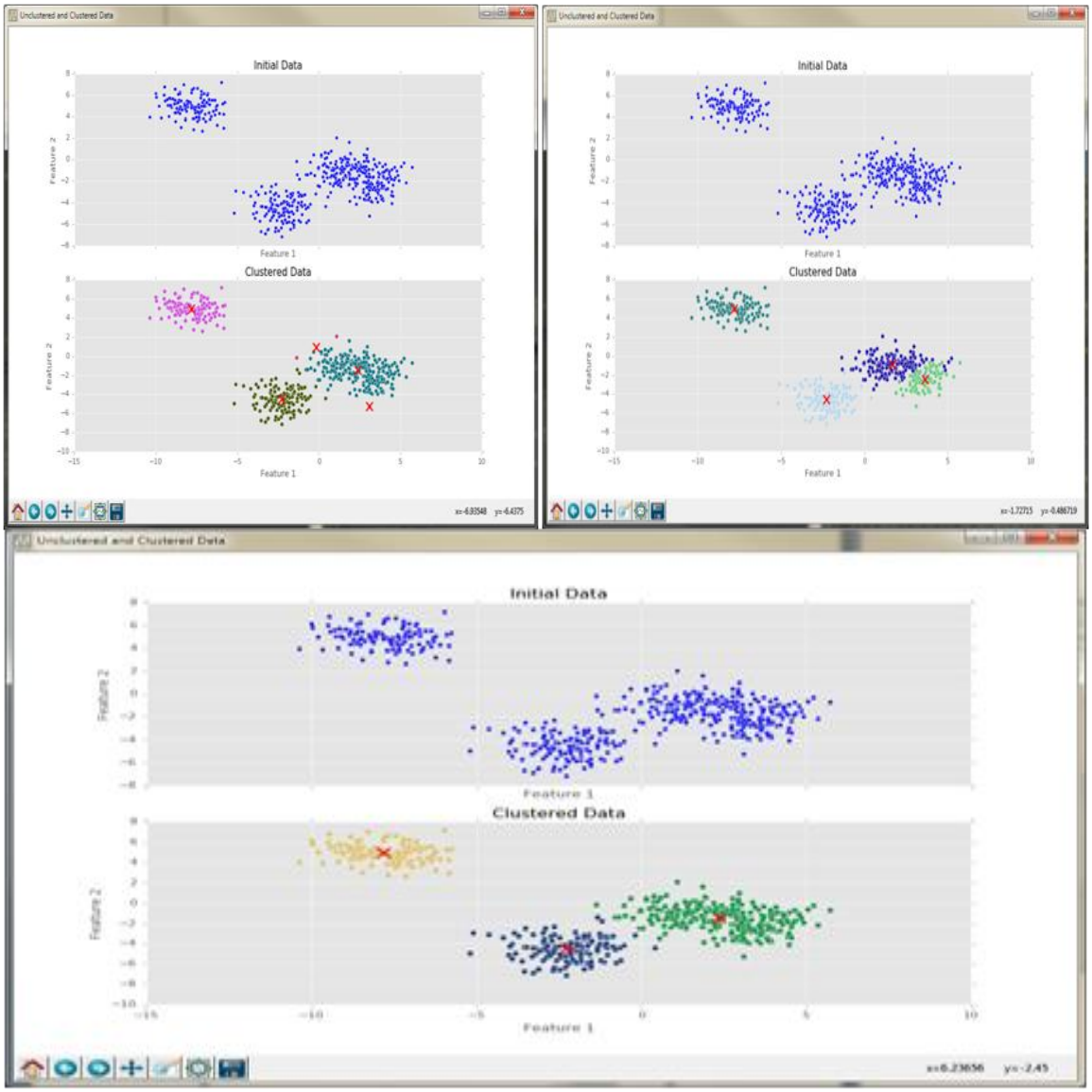

Figure 39 – a) Execution of BSAS for t1 = 3.55, t2 = 4.26, 4 blobs of 500 nodes, seed = 124, b) Execution of BSAS for t1 = 3.20 t2 = 3.55, 4 blobs of 500 nodes, seed = 124, c) Execution of BSAS for t1 = 4, t2 = 4.26, 4 blobs of 500 nodes, seed = 124

The first execution in figure 27 a) partitions the dataset into 3 large clusters and assigns a few noisy vectors at their own separate cluster. The second execution at b) manages to partition the large cluster that is consisted of two separate blobs placed closely to each other into two smaller ones. Finally the third execution at c) partitions the dataset into three clusters. All executions make sense. The first because it isolates noisy

vectors, the second because it follows the external partition of 4 blobs and the third because it merges two blobs that are very close to each other into one cluster. In this case, all indices provide good indications.

We choose to execute the internal and external criteria for the second execution at b). All the results are the expected ones.

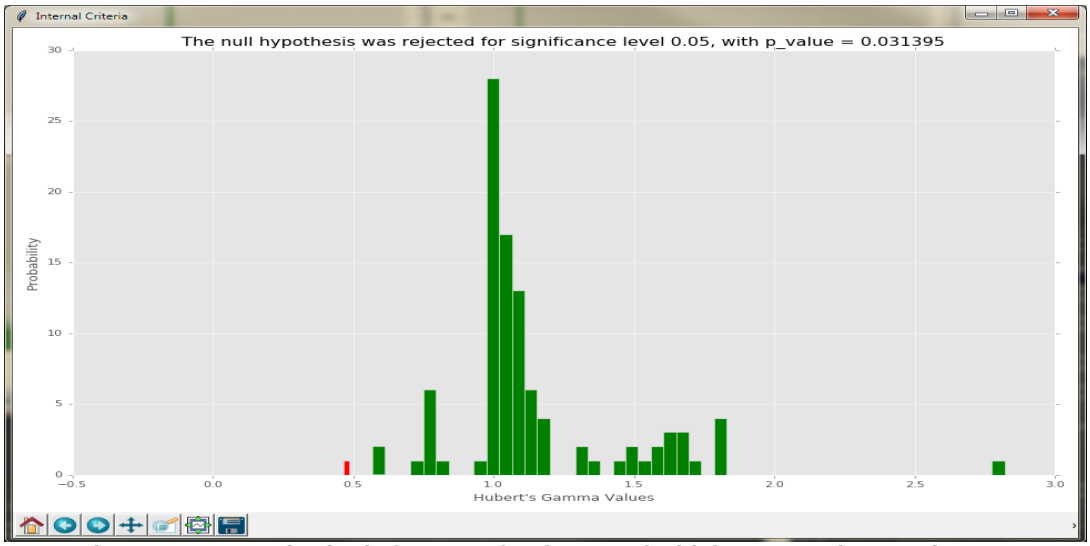

Figure 40 - Internal Criteria for TTSS for dataset of 4 blobs, 500 nodes, seed = 124

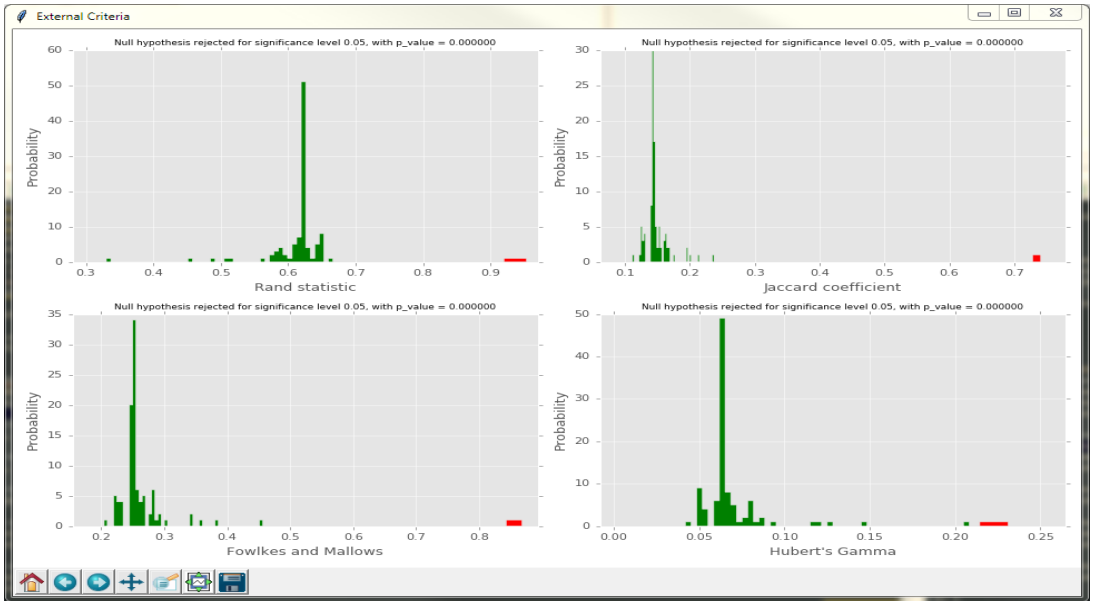

Figure 41 External Criteria for dataset of 4 blobs, 500 nodes, seed = 124

- **Concentric Circles - Moons**

As in BSAS the TTSS fails to reveal the structure of datasets having the above shapes. This is why we will not provide the results of the execution for such cases.

Chapter 5
Clustering algorithms based on graph theory

The algorithms of this category derive the main methods they use to approach the clustering problem from the theory of graphs. As opposed to the rest of the algorithms described in this thesis, in graph clustering we do not view the dataset as a set of vectors but rather as the nodes of a graph and the distances between them as the edges of the graph. (Marinus van Dongen, 2000) makes this distinction explicit by referring to graph clustering instead of vector clustering. According to the writer the two models do not exclude each other but one model may inspire methods which are hard to conceive in the other model.

The steps required to divide a group of vectors into clusters according graph clustering algorithms can be summarized to just two:

- Step 1: Take a random set of data vectors $\mathbf{X} \in R^N$ and consider them the nodes of a graph. After that take the set of all possible edges between the nodes $\mathbf{E} = \{Edge(x_i, x_j) | \ 0 \leq i,j \leq N, i \neq j\}$ and apply some criteria in order to choose a subset of **E** that will make the graph a connected one. There are several ways this can happen and several kinds of graphs we can come up with. In this thesis we examine the minimum spanning tree graph and the Delaunay triangulation graph.
- Step 2: Partition the graph into subgraphs by applying some partitioning rules, usually rules that result in cutting off the longest edges. In this way, each subgraph consists of a cluster.

5.1 Minimum Spanning Tree Algorithm

Although similar attempts to cluster data based on the notion of the minimum spanning tree had been made in the past (Arkadev & Braverman, 1967) (Johnson, 1967), this specific algorithm presented here is proposed by (Zahn, 1971) and is used in order to find clusters in the 2-D space. The definition of the cluster is based in the Gestalt psychology.

The algorithm considers the data vectors as the nodes of a complete graph and at a second stage it constructs the minimum spanning tree of this graph. The third step is the most important, as it introduces the basic concept of the algorithm, the term “inconsistent” which is used to characterize the edges of the minimum spanning tree which are “significantly” larger than the average edges.

The quantification of the term “significantly” and through this, the mathematical modeling of the “inconsistency” of the edges can be achieved in two ways. However, before we describe them, we should define the notion of the “neighborhood” used in both methods. For every edge e_i, its neighborhood is the set of edges on paths from e_i having a certain length that in our case is defined as an integer amount and provided to the algorithm as an argument. Since every edge is attached by default to two nodes, we can divide its total neighborhood in two parts, which can be denoted as N_1 and N_2, for each of the edges’ nodes.

The first method of characterizing an edge as inconsistent is by setting a threshold to how many times the number of standard deviations of the weights

of its total neighborhood can go above the mean weights of the total neighborhood. Mathematically this can be written as:

$$w > \max(\bar{w}_{N1} + q * \sigma_{N1}, \bar{w}_{N2} + q * \sigma_{N2}) \quad (5.1)$$

where q is the user defined integer number of standard deviations.

The second is by calculating the ratio between the weight of the edge under investigation and the neighborhood average weights and also setting a threshold to it, provided to the algorithm as user defined parameter. This can be written as:

$$\frac{w}{\max(\bar{w}_{N1}, \bar{w}_{N2})} > t \quad (5.2)$$

It is obvious that, different approaches applied in order to define the "inconsistency" of an edge can lead to different results.

The algorithm is described as:

Inputs: The data matrix, number of steps defining neighbor edges, number of standard deviations, threshold for the ratio of the weights

- Construct a weighted complete graph G, by setting the data vectors as its vertices and the distance between two vertices as the weight of the edge that connects them.
- Define the minimum spanning tree of G.
- Define the inconsistent edges of the minimum spanning tree.
- Define the clusters by defining the connected components of the minimum spanning tree after the removal of the inconsistent edges.

Outputs: The labeled data

5.1.1 Implementation Notes

The easiest way to represent the two graphs that are manipulated in the algorithms, the complete graph and the minimum spanning tree (which is a special case of graph) is with the use of two-dimensional arrays, where the indices represent the nodes of the graph and the values of the arrays the weights of the edges.

Constructing the complete graph by setting the euclidean distances as weights is an easy task and is accomplished by calculating the euclidean distance between all nodes of the graph. After that we build the MST matrix by executing the Kruskal algorithm. According to it, we iterate through the final number of the edges of the MST graph, which is known on advance and it is equal to N – 1, where N is the number of the nodes. In each iteration we add in the MST the edge with the minimum weight out of the set of the unvisited edges.

The next step includes characterizing some of the edges as inconsistent. We already mentioned the two inconsistency criteria we can use. After many executions of the algorithm the best way is to use both the criteria in a supplementary way. So there may be cases where one criterion may fail by an edge still be characterized as inconsistent because of the other criterion.

However, what is more difficult from the implementation point of view, at this step is not applying the inconsistency criterion, but gathering all the weights of each edge's neighborhood edges. As neighborhood edges we define the edges that lie k steps far from the edge under consideration. The value of k is provided as an argument to the algorithm.

The solution that was chosen was to build a recursive utility function, the **_recursion_util** that would be called twice from the algorithm's main function, one time for each of the nodes defining the edge under consideration. This function is then called min(k, depth of N_1) more times and returns the list of weights of all the neighborhood edges. The same procedure is also applied for the N_2 neighborhood.

Finally, after obtaining the MST matrix and the inconsistent edges, we need to cut the MST at the inconsistent edges. The cut would create a forest of threes, each of which will consist of a separate cluster. We know that the graph traversing algorithm that creates a forest of trees is depth first search, so we are implementing a dfs recursive function that will assign to each node a cluster id depending on which tree it belongs to.

5.1.2 Disadvantages of the algorithm

Although simple as a concept, the MST algorithm has a more complex implementation than other clustering algorithms so it should not be considered for a speed execution.

The most important disadvantage however is the case when different clusters are separated by two or more adjacent edges, rather than one, which have greater weight than the average. In such case a formation like a "bridge" is created that leads to the non separation of the different clusters. (Zahn, 1971) describes an aspect of this disadvantage as the case of "touching clusters".

We can observe an example of such a case to the figures below.

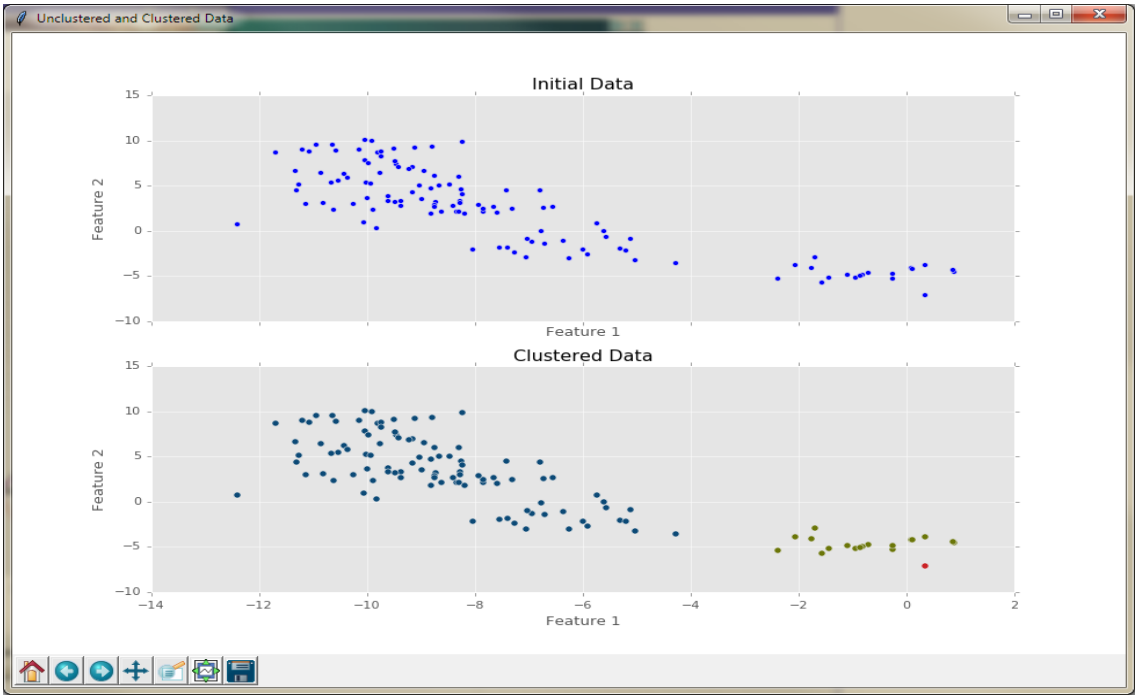

Figure 42 - MST execution on three clusters with 120 nodes, seed = 34

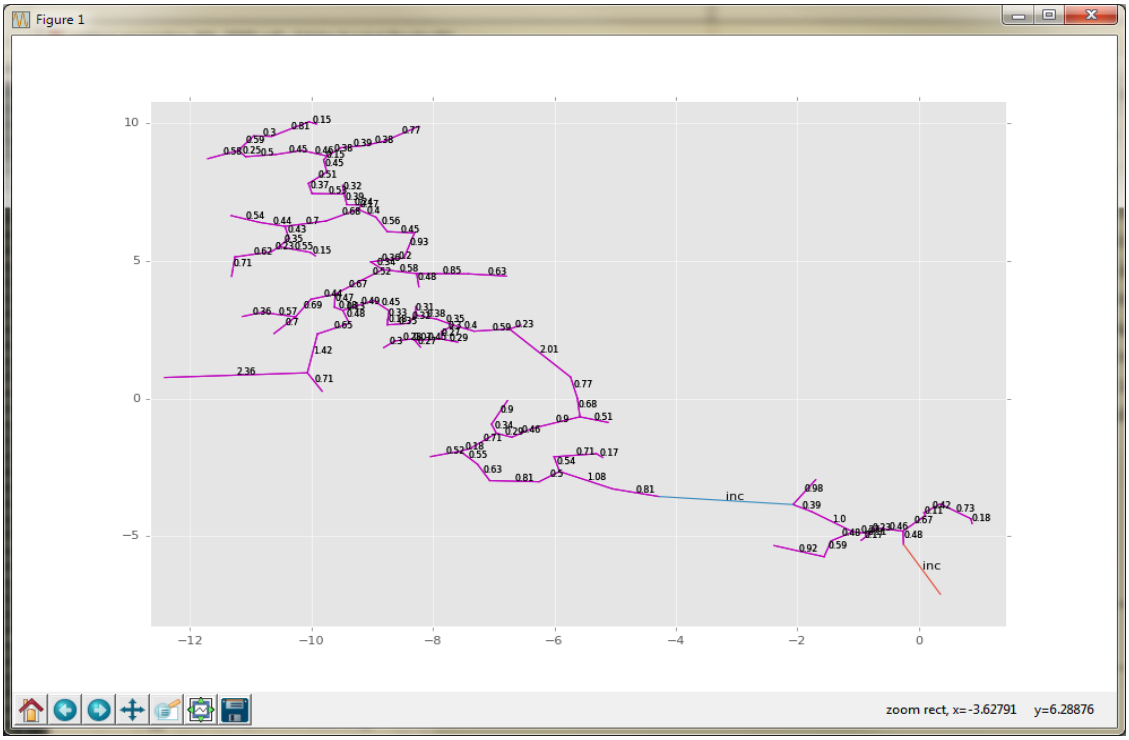

Figure 43 - The MST of three clusters with 120 nodes along with edge weights, seed = 34

In Figure 49 we can see that although we would expect three clusters, the cluster in the middle of the feature space is considered the same with the

larger one on the upper left side. The reason can be seen on Figure 50 below. The two clusters are connected with an edge that has a weight of 2.01. The edges that consist of the neighborhood of this edge are large enough for this edge to be characterized as inconsistent, given of course the specific parameterization we used to run the algorithm.

One final disadvantage is that it creates separate clusters for the noisy nodes. We can see this in the execution below.

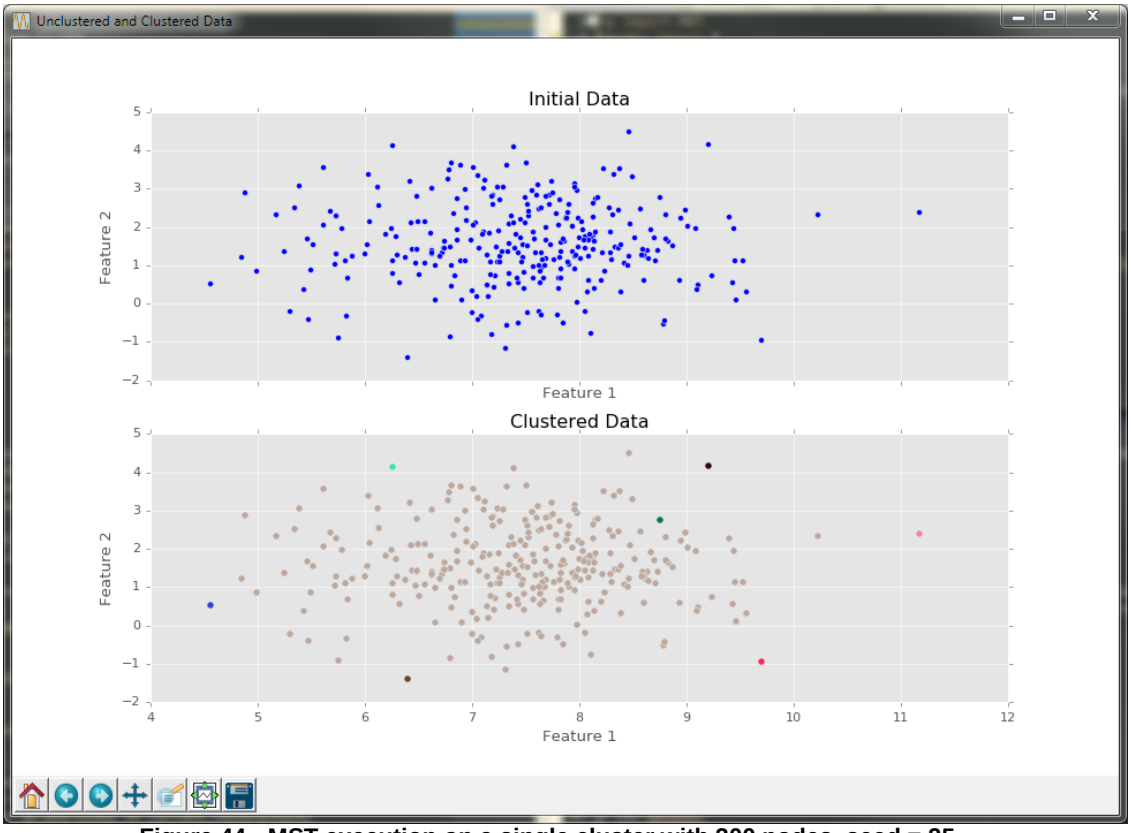

Figure 44 - MST execution on a single cluster with 300 nodes, seed = 25

All around the single cluster we can spot nodes having colors other than blue which means that they belong to different clusters. One remedy would be to run a merge procedure after the execution of the MST, between such clusters with a small number of nodes, for example 1 – 5, and assign them to the nearest larger cluster.

We have implemented such a procedure in our code for clusters that have up to only two nodes.

5.1.3 Testing on synthetic data

- **Blobs**

We first execute the relative criteria indices on sequential executions on the dataset during which we are varying the parameters of the algorithm. Since MST algorithm is a hard clustering algorithm, the relative criteria indices we have chosen to implement are indices defined for this type of clustering,

namely Dunn and Silhouette index. However, instead of varying the number of clusters as in the case of the k-means algorithm, we have adjusted our code so that to calculate the indices values to variations of 2 of the parameters used in MST algorithm, the k number of neighbor nodes and the t threshold of the second inconsistency criterion. We left q unchanged to 1.5 since it is the parameter which affects the least the clustering result among the two others.

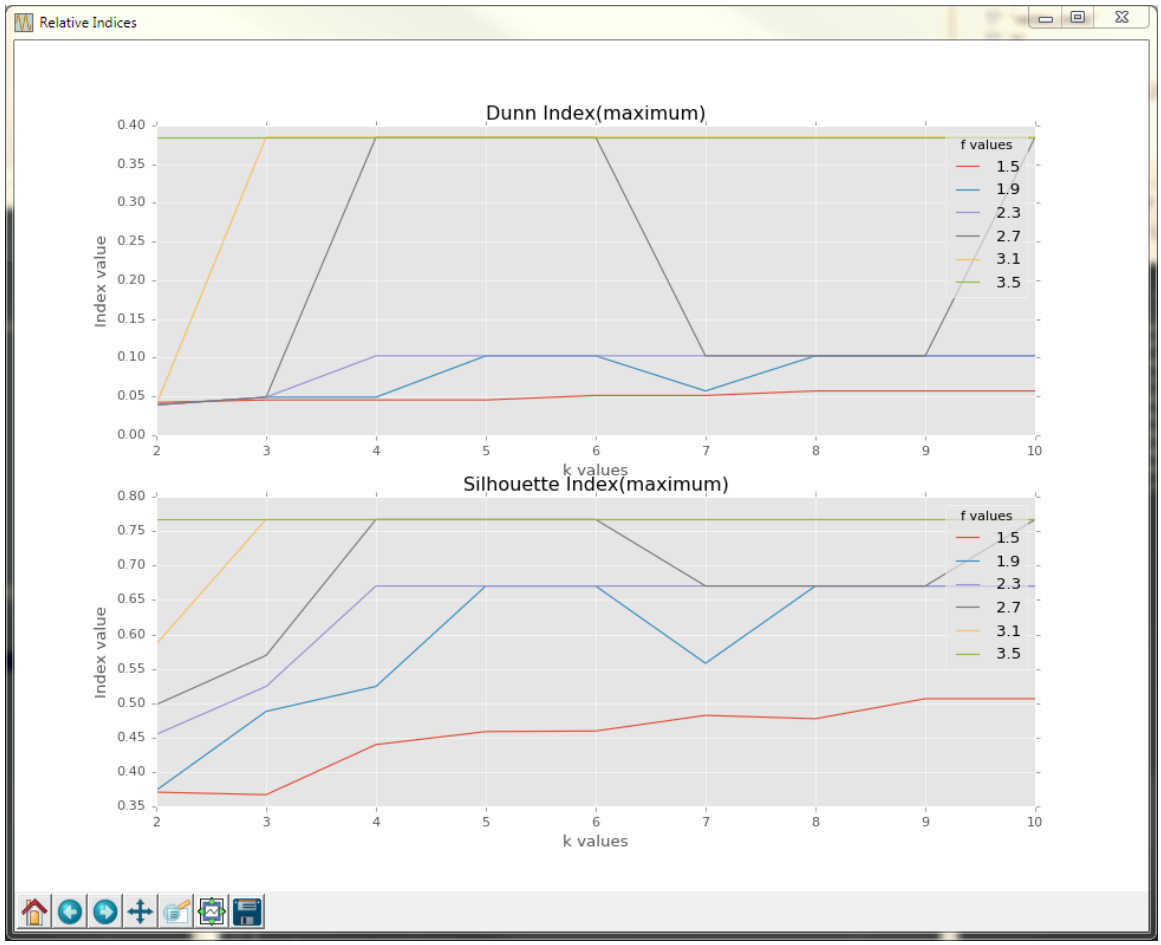

Figure 45 MST - Relative Criteria Indices for 4 blobs of 500 nodes, seed = 118

We can notice that the plots of both the indices look alike. They both suggest that the best clustering results are obtained for values of f above 2.7 and for values of k roughly between 3 and 6. We tested all the possible combinations of k and f that maximize the indices and we have confirmed that the clustering in all of them is the same and the best that could be made.

It can be seen below:

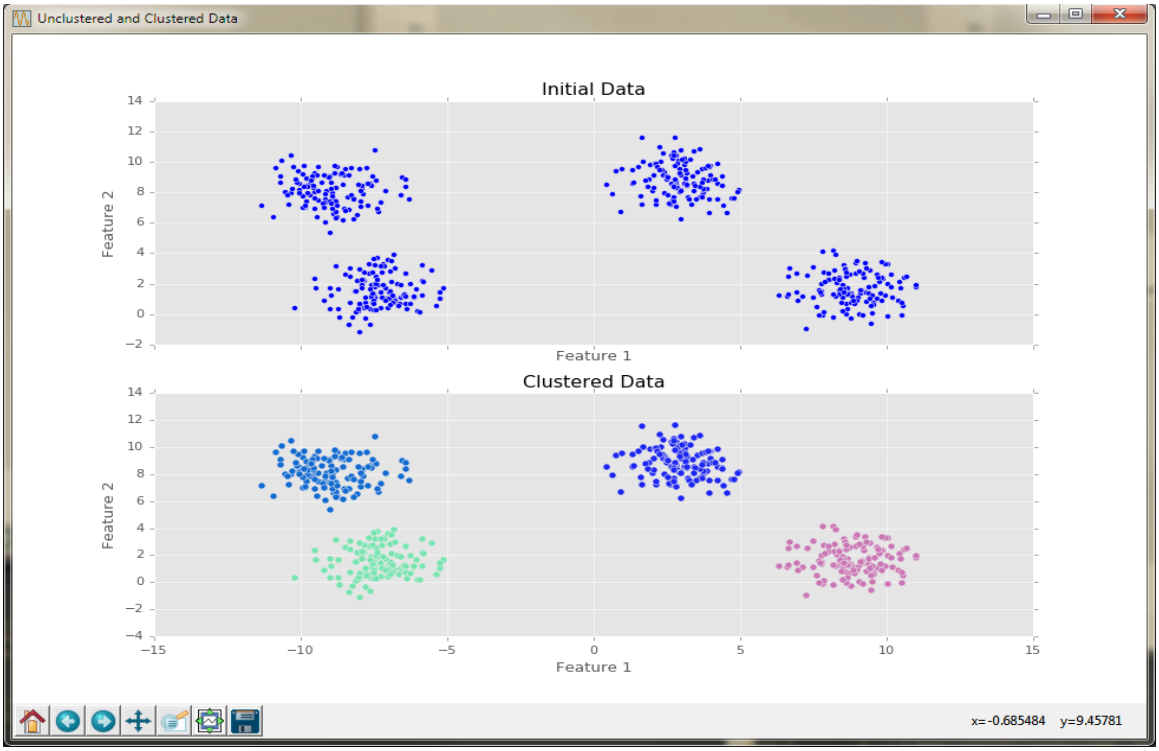

Figure 46 - Execution of MST for k = 4, f = 2.7 for 4 blobs of 500 nodes, seed = 118

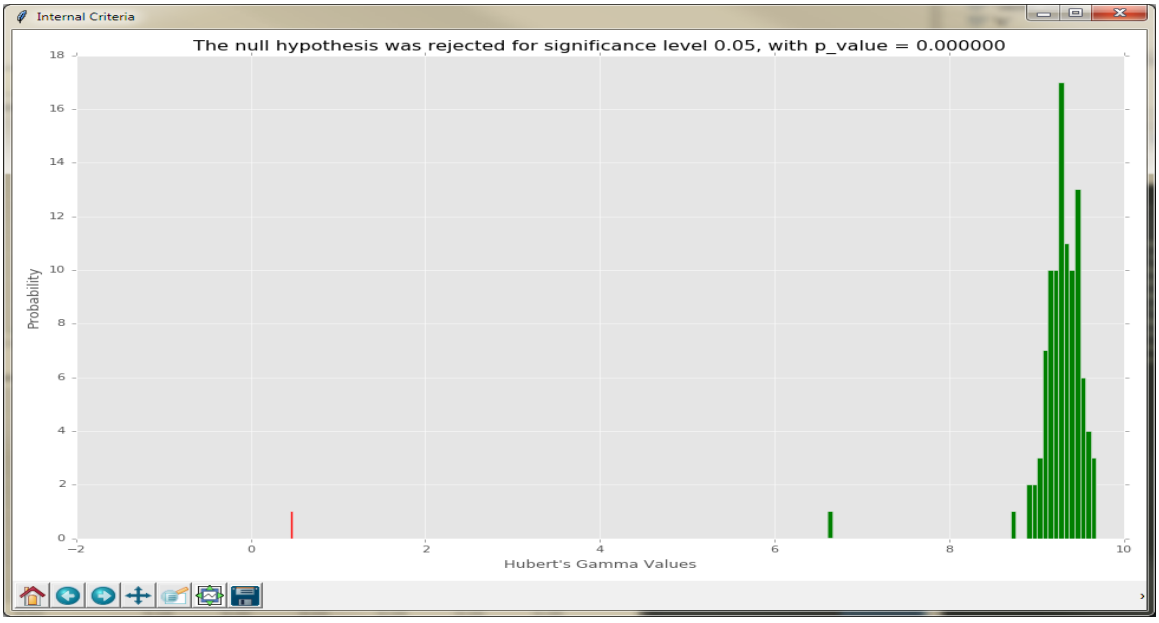

Figure 47 - Internal Criteria Gamma index for dataset of 4 blobs, 500 nodes, seed = 118

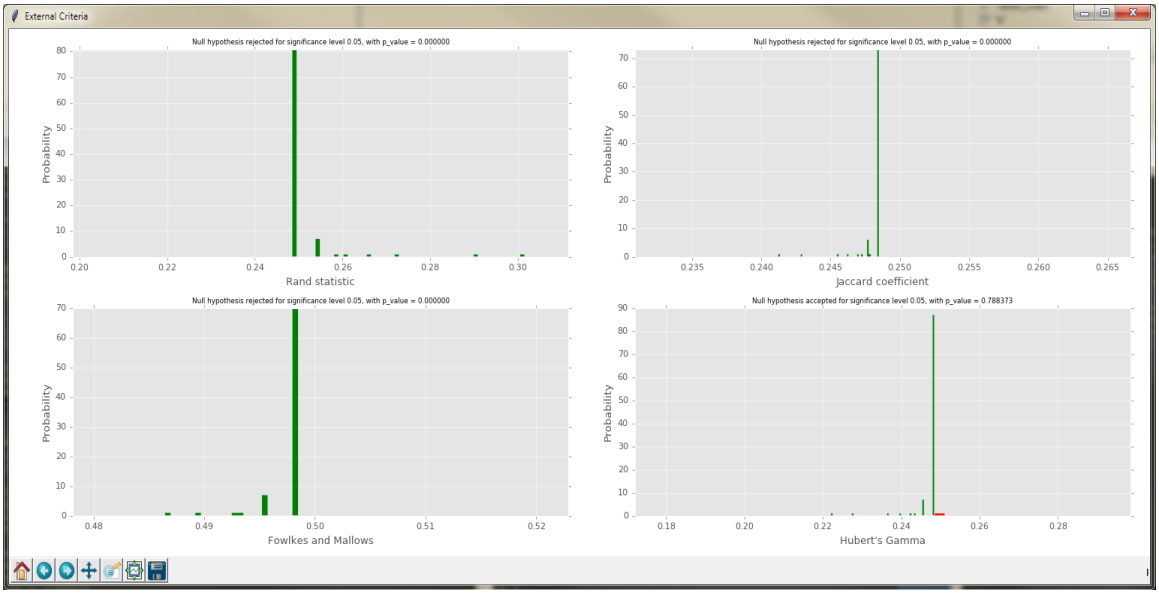

Figure 48 - External Criteria for dataset of 4 blobs, 500 nodes, seed = 118

The only index for which the null hypothesis is accepted is the Hubert's Gamma index. However, and this is valid for all external criteria indices, the values of this index are almost all concentrated into one bin (roughly 90% of them) so we cannot make easily the assumption that they are normally distributed. After all the index's value for our dataset is larger than all of the values of the same index for the random datasets created with the monte carlo simulation.

The second example we will see on this category of datasets is one where the MST algorithm fails to provide the best clustering result due to the "touching clusters" disadvantage. We execute the relative indices and the MST for the dataset with seed 121:

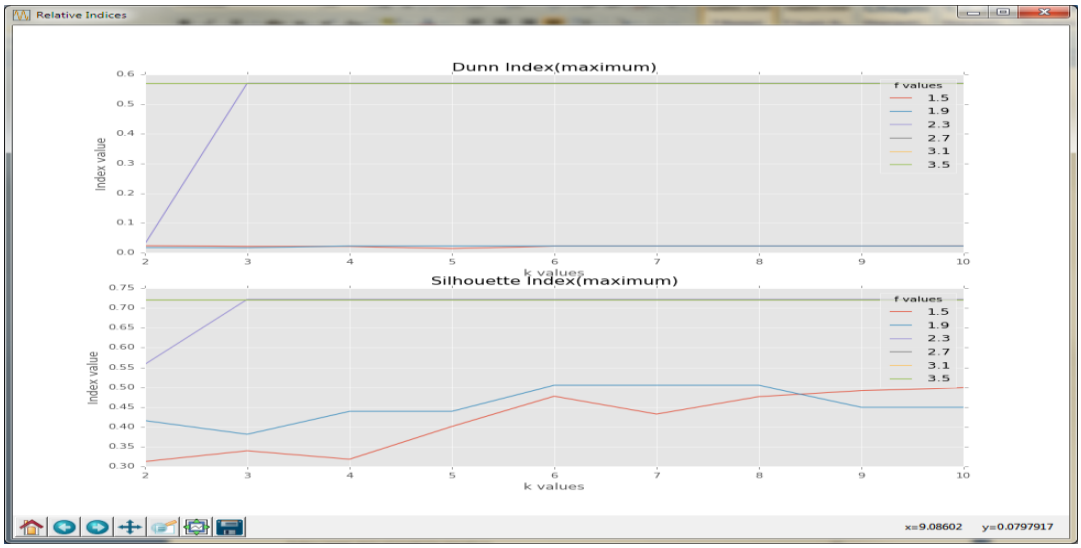

Figure 49 - MST - Relative Criteria Indices for 4 blobs of 500 nodes, seed = 121

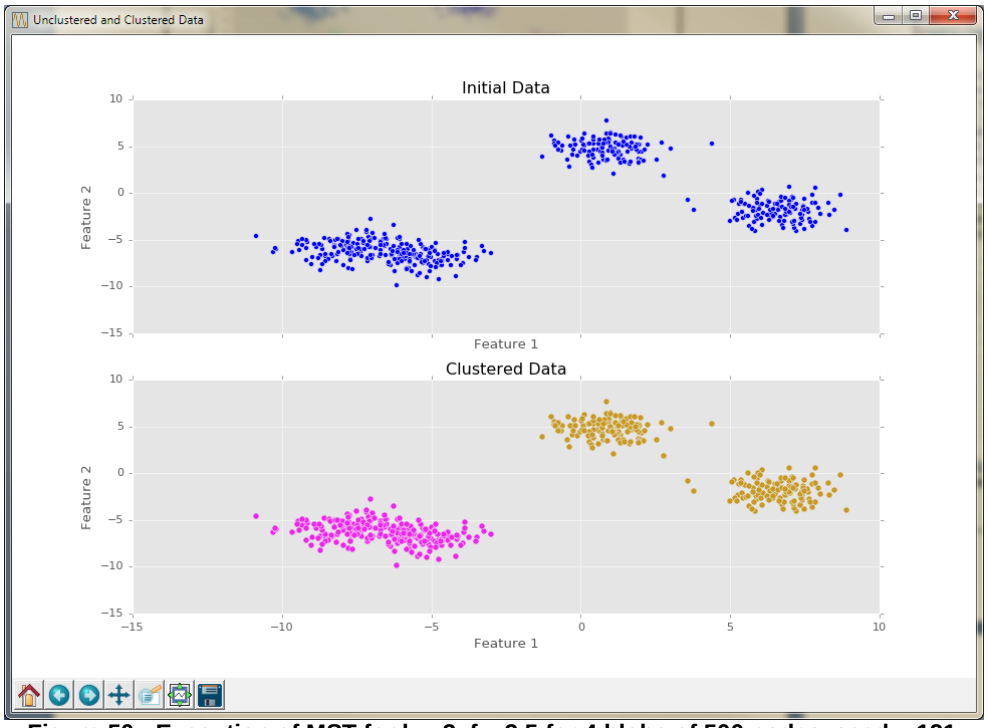

Figure 50 - Execution of MST for k = 3, f = 3.5 for 4 blobs of 500 nodes, seed = 121

The clustering result of figure 38 is the best that can be accomplished by MST. No other combination of parameters can lead the indices at better values due to the fact that between the two clusters on the right there are some nodes that act as a "bridge" between them, making it impossible for the algorithm to separate them without disturbing the rest of the partition.

- **Concentric circles**

In the case of non spherical clusters, the silhouette index does not perform correct, as it was not designed for them. Consequently, we should ignore the results taken from it. However, the Dunn Index performs pretty well in the case of concentric circles by giving correct indications of the values of the parameters that should be used in the MST algorithm executions. We should not consider though using it as an general index designated to be applied in cases where the clusters are not spherical. It only works here because of the morphology of the dataset. Specifically, it is the nominator of the index, which is the minimum distance between nodes on different clusters that takes its maximum value when the clustering result returned is the correct one, meaning that every circle has been assigned to one cluster exactly. This is where the index also takes its largest value. The fact that the denominator also takes its maximum value at the same result, and therefore pushes the value of the index to a lower level, does not affect the index that much.

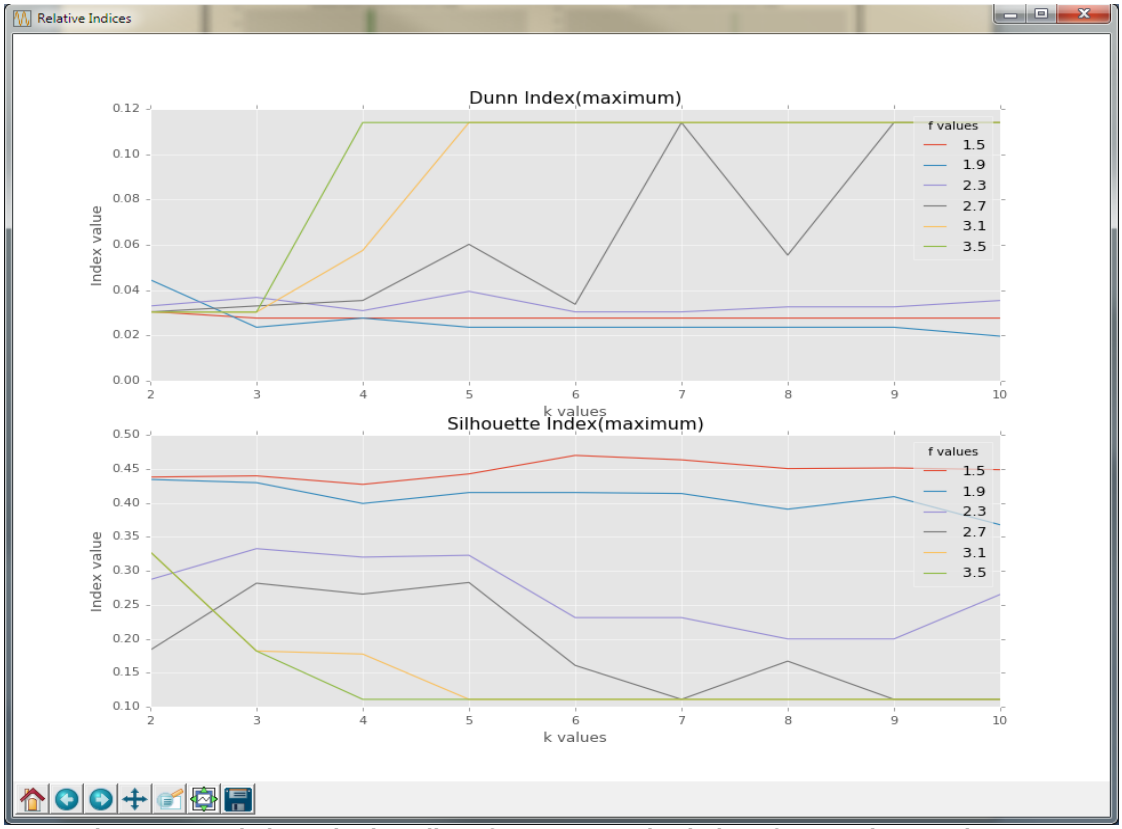

Figure 51 - Relative Criteria Indices for 2 concentric circles of 500 nodes, seed = 118

After inspection of Figure 46 we run the MST algorithm with the parameters k = 4, f = 3.5. The result we take can be seen in the following scatter plots and histograms for the internal and external indices:

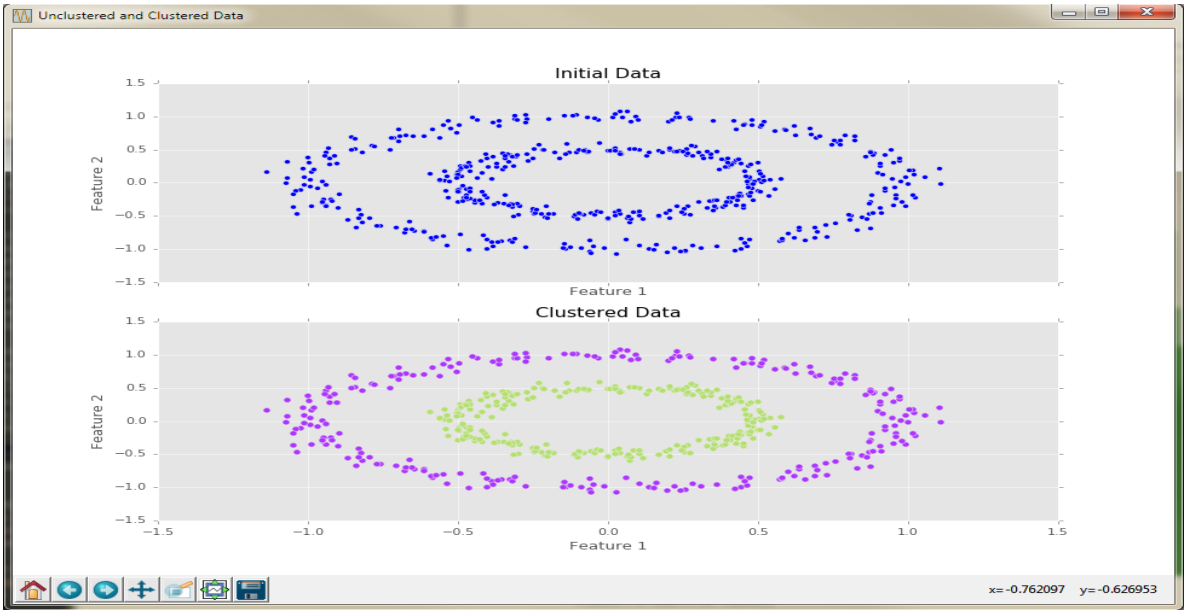

Figure 52 - Execution of MST for k = 4, f = 2.7 for 2 concentric circles of 500 nodes, seed = 118

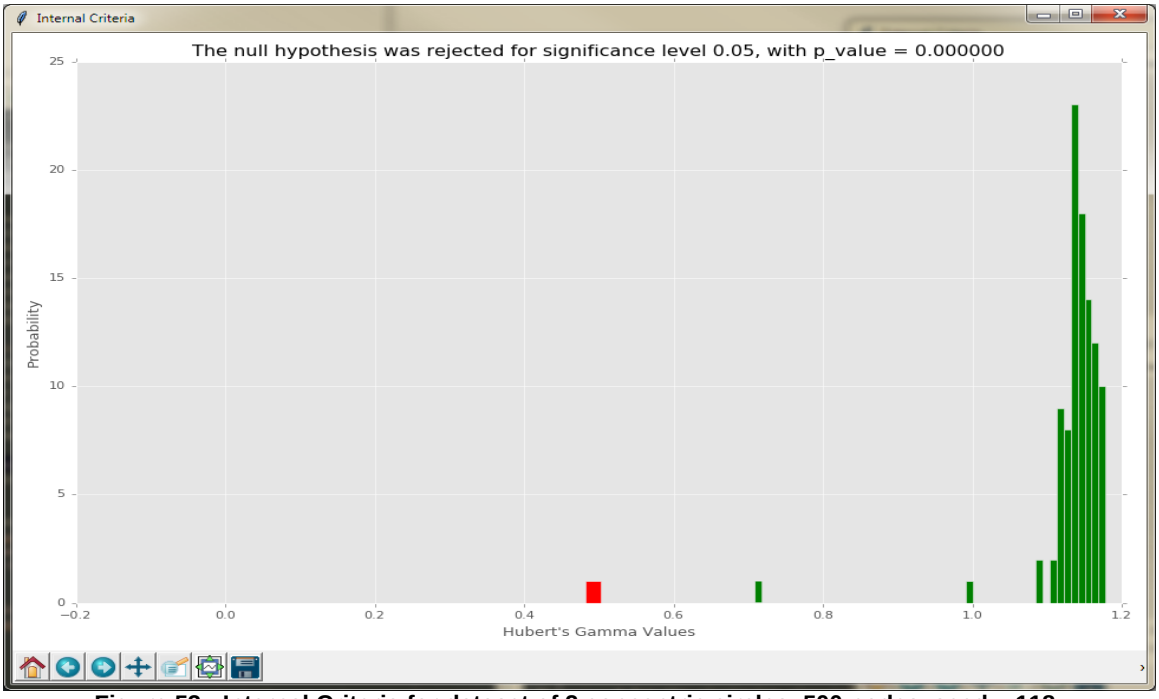

Figure 53 - Internal Criteria for dataset of 2 concentric circles, 500 nodes, seed = 118

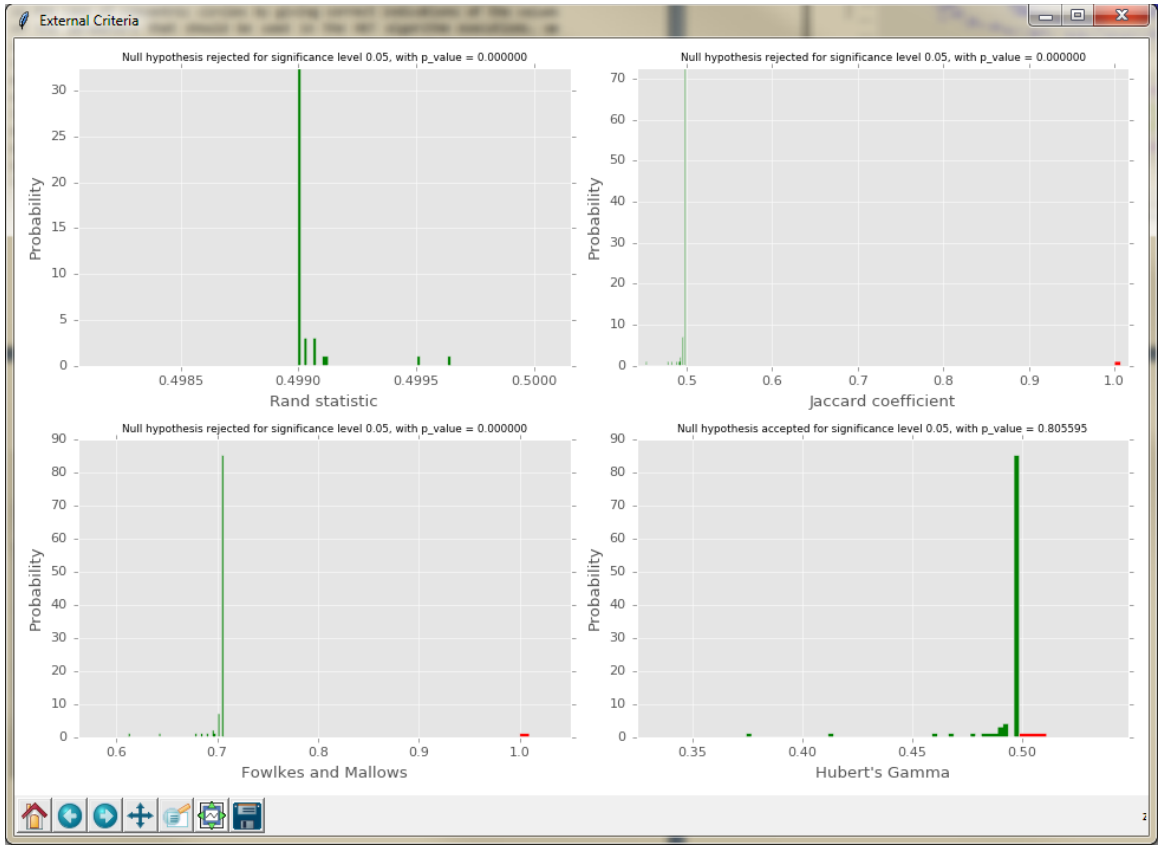

Figure 54 - External Criteria for dataset of 2 concentric circles, 500 nodes, seed = 118

As in the previous example for MST algorithm, the only index that does not agree with the correct clustering is the gamma index run for on the external criteria frame. We explained why we can overlook this result on the previous execution of the algorithm.

- **Moons**

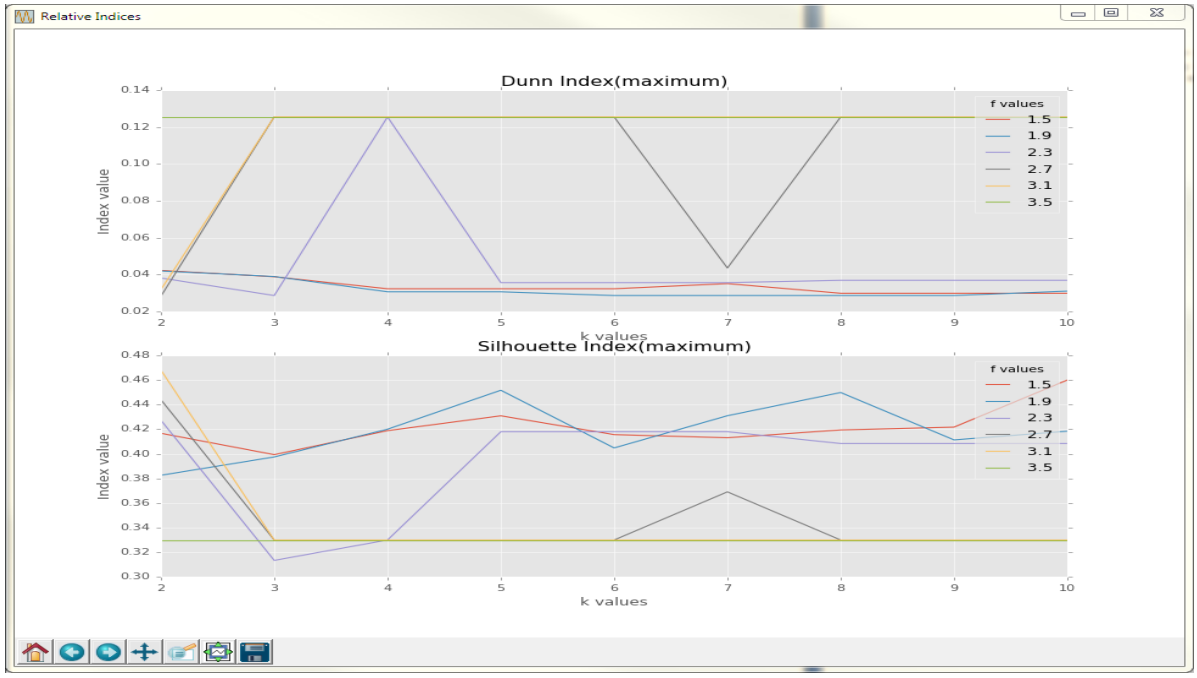

Figure 55 - Relative Criteria for dataset of two moons, 500 vectors, seed = 118

The same conclusion we made for the indices in the previous section is also valid here. The Dunn index, based on its nominator manages to perform well. On the other hand, the Silhouette index is useless in a non spherical dataset. Running the MST with the one of the parameter pairs suggested by the Dunn Index, k = 3, f = 2.7, we obtain:

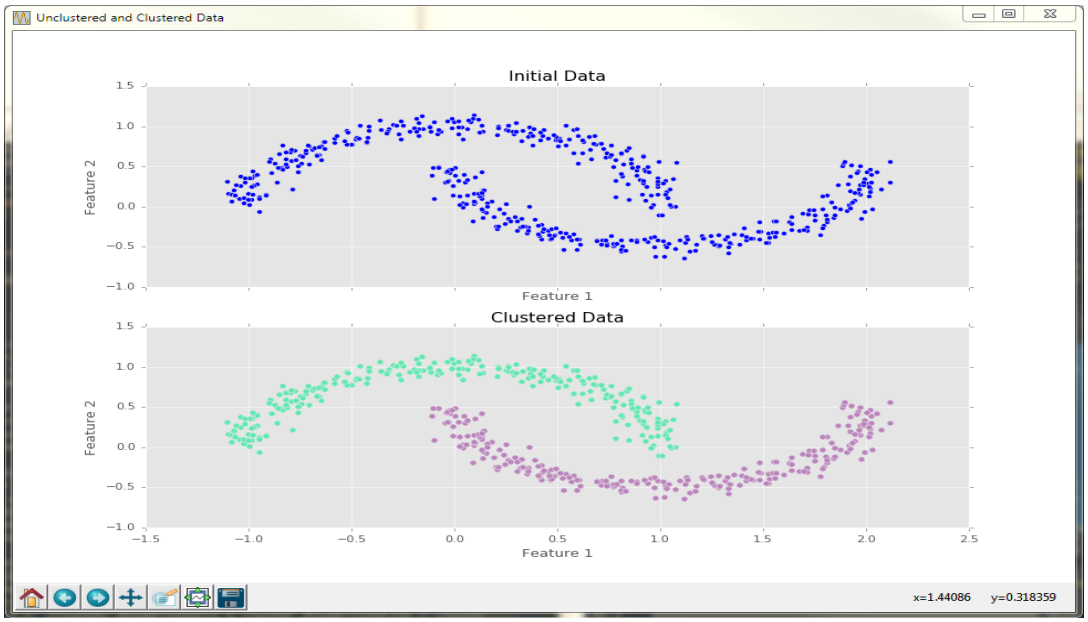

Figure 56 - Execution of MST for k = 4, f = 2.7 for 2 moons of 500 nodes, seed = 118

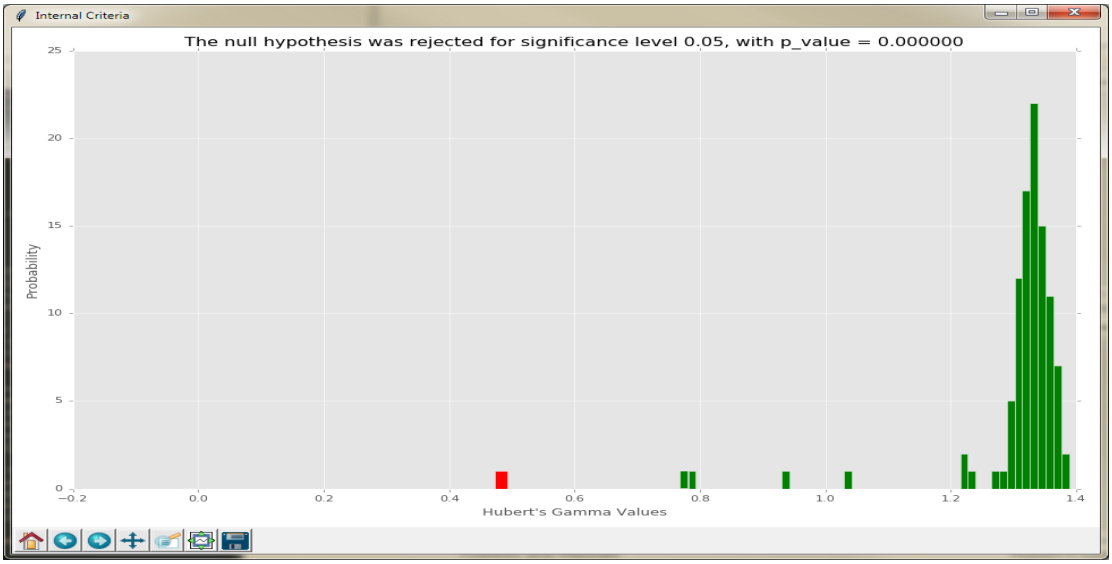

Figure 57 - Internal Criteria for dataset of 2 moons, 500 nodes, seed = 118

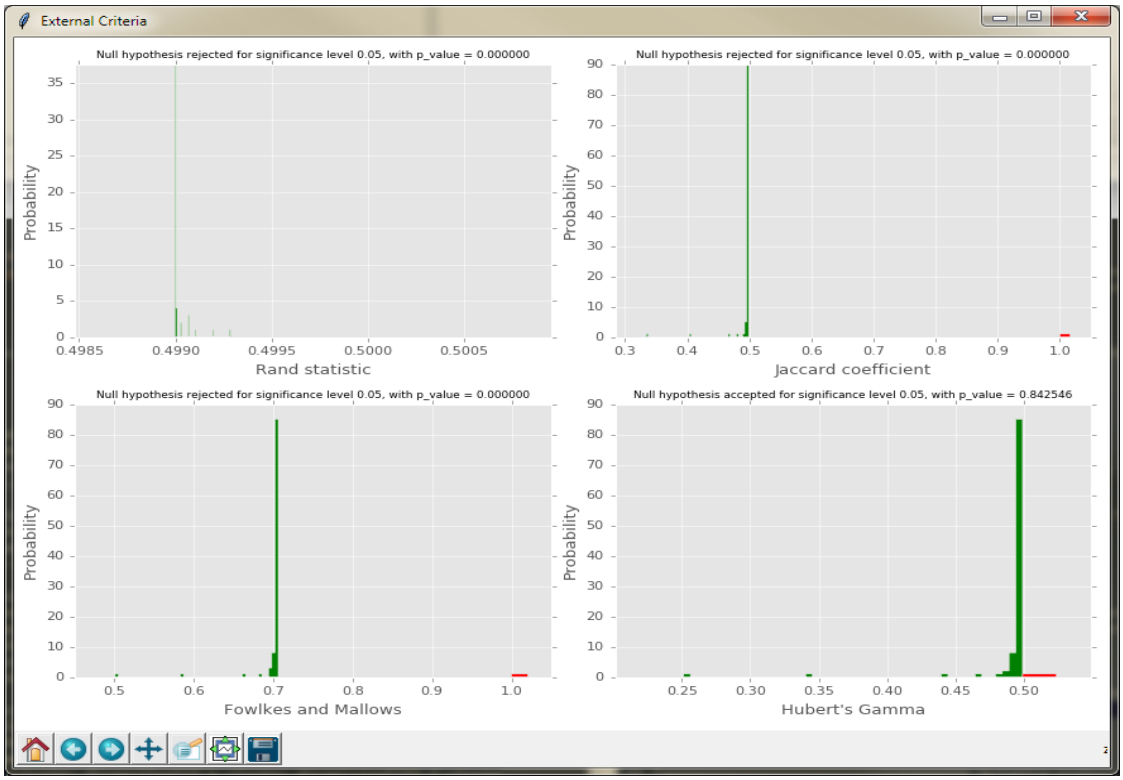

Figure 58 - External Criteria for dataset of 2 moons, 500 nodes, seed = 118

The clustering is the best possible. All remarks for internal and external criteria made in the previous section of concentric circles are valid here too.

5.2 Delaunay Triangulation Algorithm

(Eldershaw & Hegland) proposed an algorithm in order to overcome the limitation of many preexistent algorithms, mainly those which belong to the category of the cost minimizing function, to presume that the clusters under consideration are spherical.

Following the general method described in **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**, the proposed algorithm constructs the initial graph of the dataset by using the Delaunay triangulation technique(Delaunay, 1934), a technique heavily used in the computer graphics field. A visual example of the result of triangulating a dataset in this way is the following:

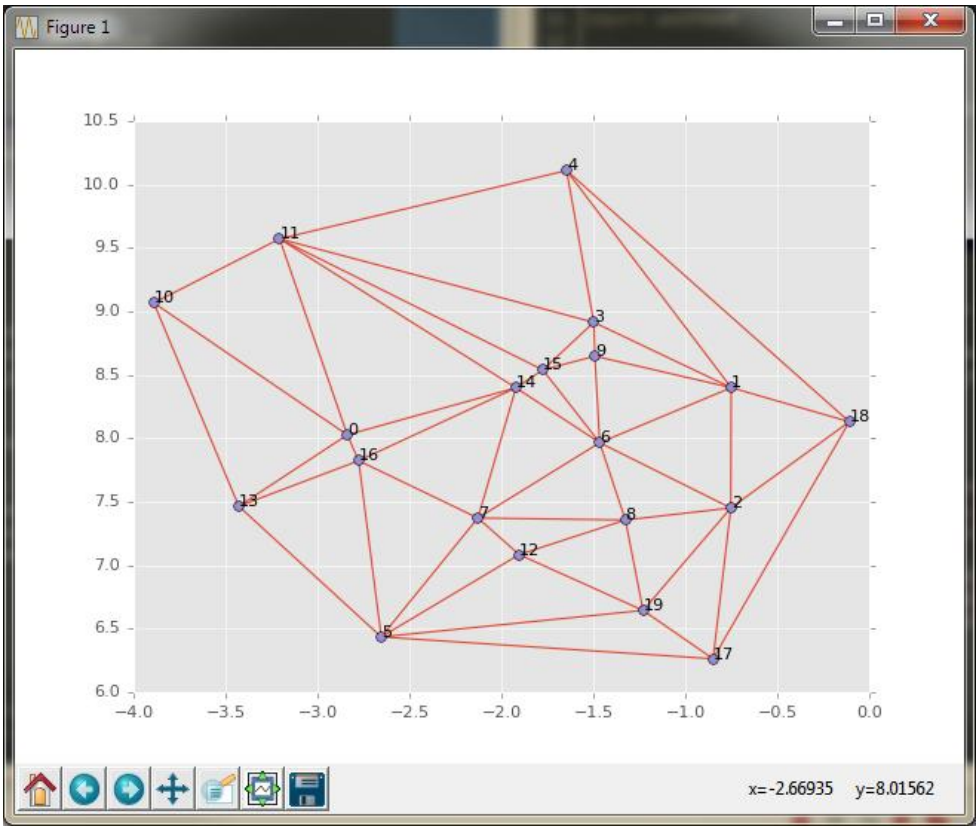

Figure 59 - Example of Delaunay triangulation of a dataset of 1 blob of 20 nodes

After the construction of the initial graph, a graph partitioning algorithm has to be defined in order to choose a cut-off point p that will partition the graph into clusters. The writers tested the technique used in the MST algorithm and we described in 5.1 with little grade of success, as it resulted in preserving all the edges due to the large number of edges coming from the same vertex, contrary to the small number that exists in minimum spanning tree graph. Another reason they suggest is that the MST technique is not effective on the presence of noisy data.

The solution they propose is to divide the set of edges in two subsets. One subset will contain the inter-cluster edges and the other the intra-cluster ones. Then, the following function is defined:

$$T(p) = \frac{\sum_{i=1}^{n_x}(x_i - \bar{x})^2}{n_x} + \frac{\sum_{i=1}^{n_y}(y_i - \bar{y})^2}{n_y} \quad (3.15)$$

where each of the two terms is a measure of the variance of each subset. Obviously, the more homogenous the two subsets are, the less the value of T. So the purpose is the minimization of T with respect to the cut p. This is performed in an empirical way. The edges are sorted and then a number of twenty evenly spaced values of p are chosen. Twenty values of T are calculated and the p for which the T value is minimum is selected.

Unfortunately, we could not reproduce the good results mentioned by the writers by using their suggested method. So we created a small and simple variation of it. Instead of summing up the variations of the two subsets and picking up the minimum summation, we calculated the differences between the means of the subsets and selected the cut p for which this difference was maximum.

$$\mathrm{T} = \max(\bar{x} - \bar{y}) \quad (3.16)$$

The phrase "the simpler is better" matches perfectly here, since this simple technique led to pretty good results.

5.2.1 Disadvantages of the algorithm

Although the Delaunay triangulation algorithm does not require any parameter at all, which is a huge advantage, it has the same disadvantages that the MST algorithm has. The first is that it cannot avoid the curse of the "touching clusters", which of course can be an advantage in cases where these clusters need to be recognized as one. An example can be found bellow:

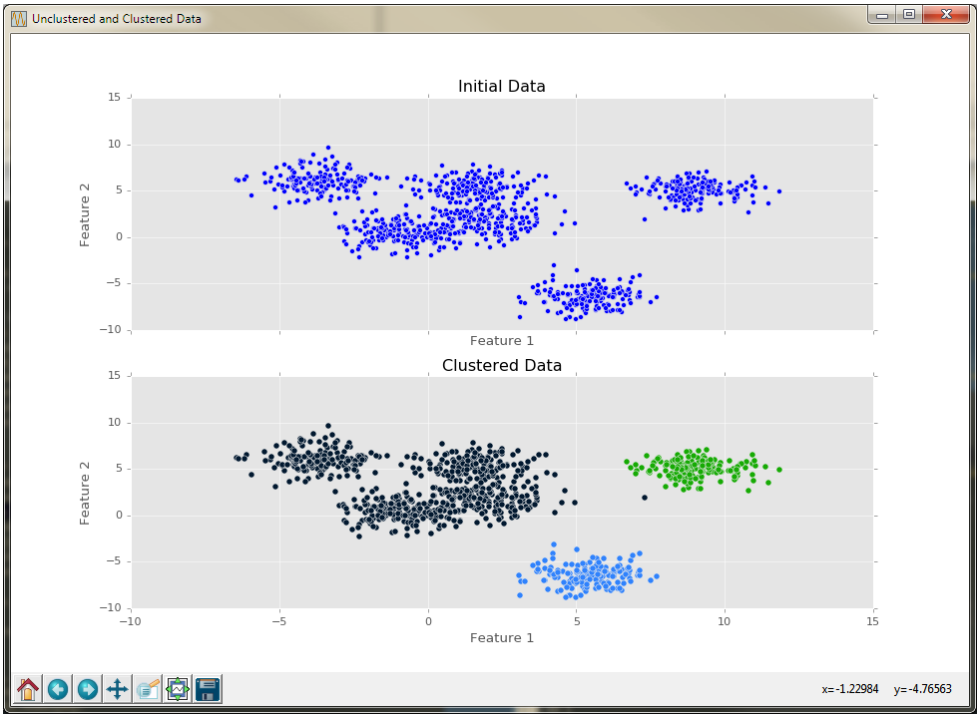

Figure 60 - Example of Delaunay clustering algorithm for 6 blobs of 1000 nodes, seed = 151

Here, the "bridges" of nodes between the different clusters result in the failure of the algorithm, as it cannot distinguish in the cluster with black color the two(at least) separate clusters that we can easily see by mere visual observation.

Finally, the algorithm assigns noisy isolated nodes to single clusters which can be easily handled with a merging procedure as mentioned in 5.1.2.

5.2.2 Testing on synthetic data

- **Blobs**

Since the Delaunay triangulation algorithm (DTA) does not take any parameters we will not be running relative criteria for it. The execution on spherical clusters can be found below:

Figure 61 - Execution of DTA for 4 blobs of 500 nodes, seed = 352

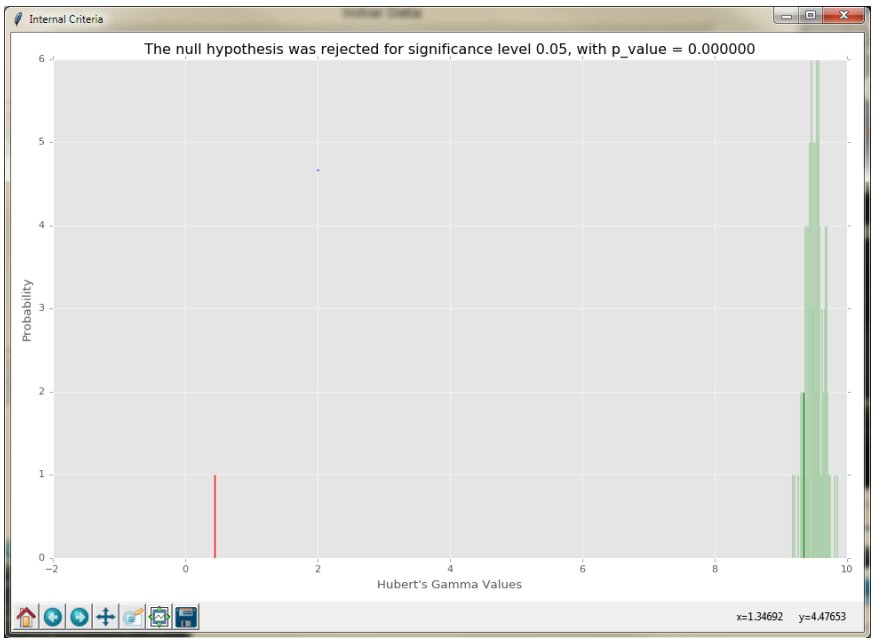

Figure 62 - Internal Criteria Gamma index for dataset of 4 blobs, 500 nodes, seed = 352

When executing the monte carlo simulation all the uniformingly distributed vectors belong to one cluster as seen below:

Figure 63 – DTA clustering of a uniformingly random dataset

This means that the external indices calculated with reference to the random data produced by the monte carlo simulations have always the same value, as can be seen below.

- **Concentric circles – Moons**

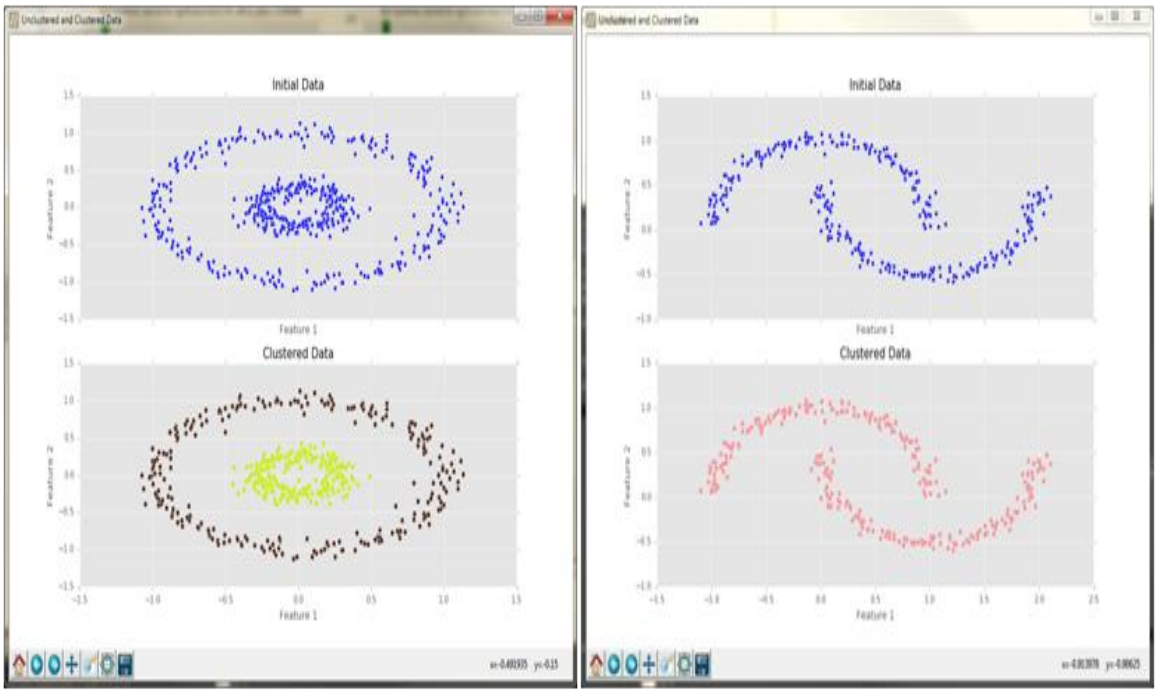

Figure 64 – a) Execution of DTA for 2 concentric circles of 500 nodes, seed = 107, b) Execution of DTA for 2 moons of 500 nodes, seed = 118

In the cases of non spherical clusters, the DTA will be able to distinguish clusters that have a certain distance between them. It fails to divide a dataset into clusters when the clusters are too close to each other, something obvious in Figure 56 b).

Chapter 6
Application of Cluster Analysis to Image Segmentation

One of the most promising fields of research today is Computer Vision. Although Computer Vision has made great advances during the last decades it still remains very far away of its final, its ultimate goal. This is making computers able to interpret an image in the same effortless way humans and animals do (Szeliski, 2010). This definition of the scope of the field is usually offered as a meaningful definition of the field itself. One of the subfields of Computer Vision which is going to employ our efforts in the current chapter is image segmentation. This is because we can apply in this particular subfield the clustering algorithms of the previous chapters.

According to (Shapiro & Stockman, 2000) *image segmentation refers to the partition of an image into a set of regions that cover it.* Another, more informal way to express the same meaning is found in (Szeliski, 2010) where image segmentation is described as *the task of finding groups of pixels that "go together".* (Gonzalez & Woods, 2007) provide a mathematical definition of the process of image segmentation based on set theory. By reading these definitions, we can easily understand why image segmentation fits so well into the frame of cluster analysis as described in 1.1. In fact cluster analysis can provide a general model for solving image segmentation tasks. This is because the aim of both fields is to search into datasets for groups that present certain uniformity, compared to other groups.

Although we will examine only one type of methods of image segmentation in the current thesis, the ones based on clustering techniques, there are several more techniques used in practice. We have already described threshold based techniques for example in 4.1.2, as a way to choose a value for the threshold in BSAS. Other ones include Edge based techniques, region based techniques etc.

6.1 Testing on real data

In order to test our algorithms on real data we use images that can be found on the site (Berkeley Segmentation Dataset and Benchmark). This is a very commonly used library of images for testing segmentation techniques. What is so special about these images is that they have been processed by 30 human subjects who gave their clustering results. We accept these human-processed results as a benchmark for the validation of our algorithms (remember the external criteria on section 2.1.2), or as the site describes it *the human segmented images provide our ground truth boundaries.*

We have chosen some of the algorithms that we implemented in this thesis or a combination of them in order to test them on the images. The choice was not a free one, as the algorithms based on graph theory could not practically be used because of the time demanded to process the data. We have chosen instead to run

- k-means after initialization of the centroids by using BSAS,
- FCA after initialization of the centroids by using BSAS,
- FCA and PCA after initialization of the centroids by using BSAS.

Regarding the validation, as we indicated in the previous paragraph, we use the external criteria and specifically a modification of the Rand index to measure the similarity between our clustering result and one human processed clustering result. The modification consists of the fact that the volume of the data is so large that it is impossible to calculate the rand index for all the pixels of the images. The workaround chosen was to uniformingly take 5000 pixels, the same ones from each dataset, the clustered and the human segmented, and calculate the index based on them.

Each of the algorithms was executed on three different images randomly picked out of the image library used. We have proceeded to sequential executions for m = 2…8, k, where m is the number of clusters and k the number of clusters used by the human object in order to segment the image. In order to visually observe the results of our clustering efforts, we assign a random color to each of the clusters and depict all the pixels of the image by using the corresponding color of the cluster they belong to. The clustered images we result in by using the clustering methods are placed into tables. The first image is the original one that can be seen as the starting point, whereas the final image is the human segmented image that can be seen as the final point, the point we want to reach to.

Based on the notes above, the results can be found in the following pages:

K-means – initialization with BSAS

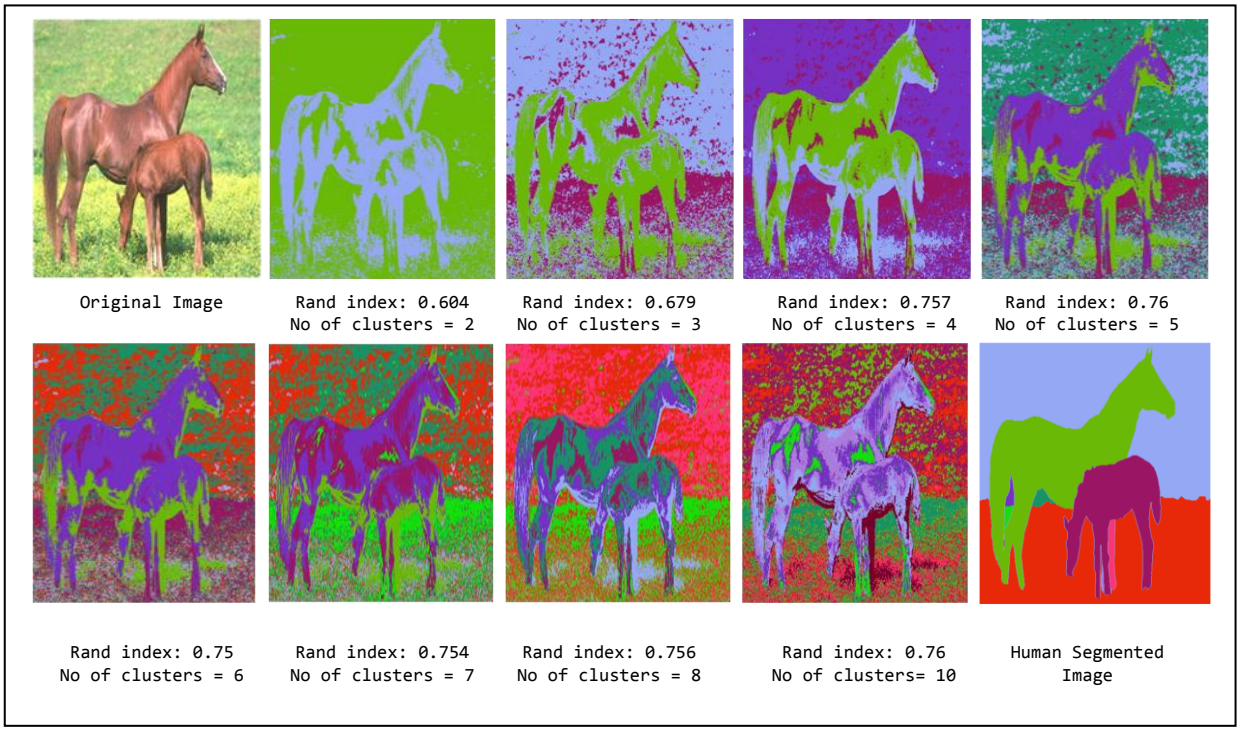

Figure 65 – Image number 113044, k-means algorithm, BSAS for centroid initialization

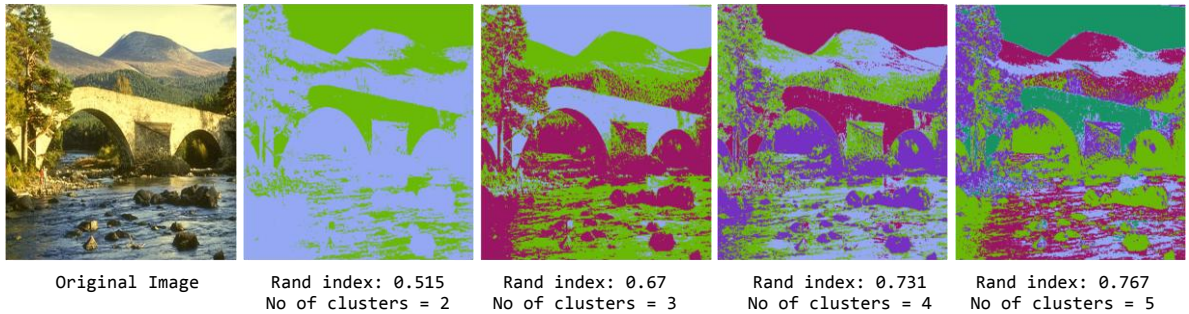

Figure 66 -Image number 231015, k-means algorithm BSAS for centroid initialization

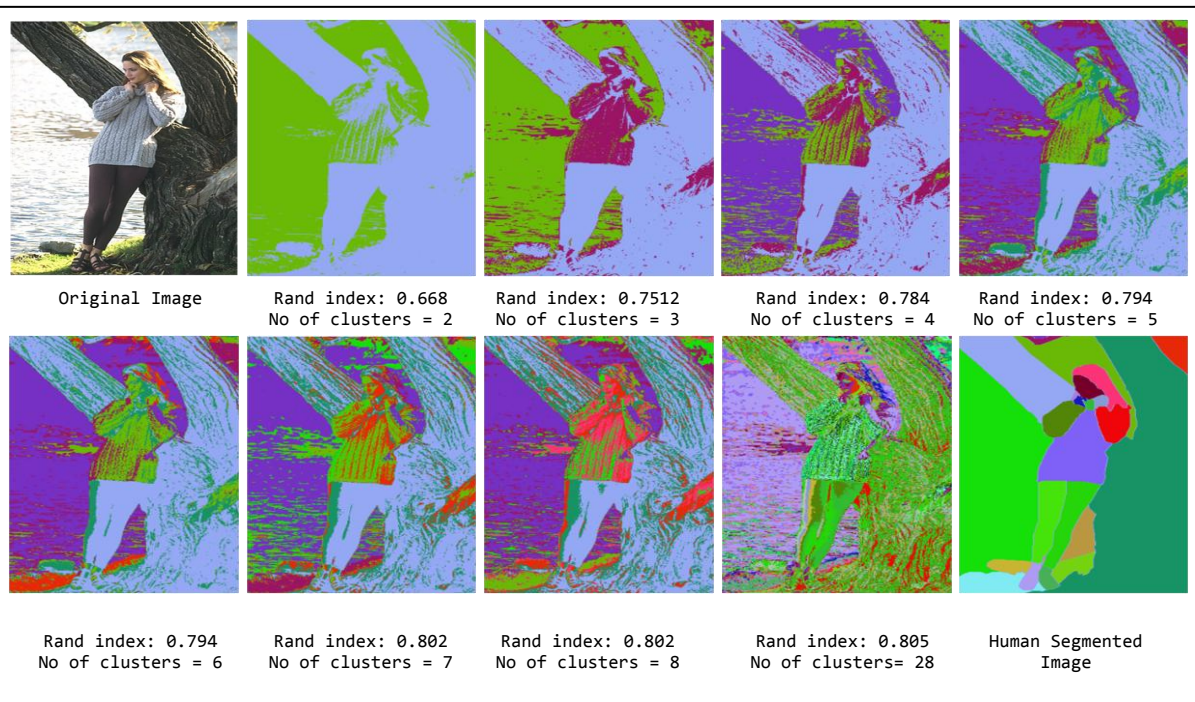

Figure 67 - Image number 181091, k-means algorithm, BSAS for centroid initialization

FCA – initialization with BSAS

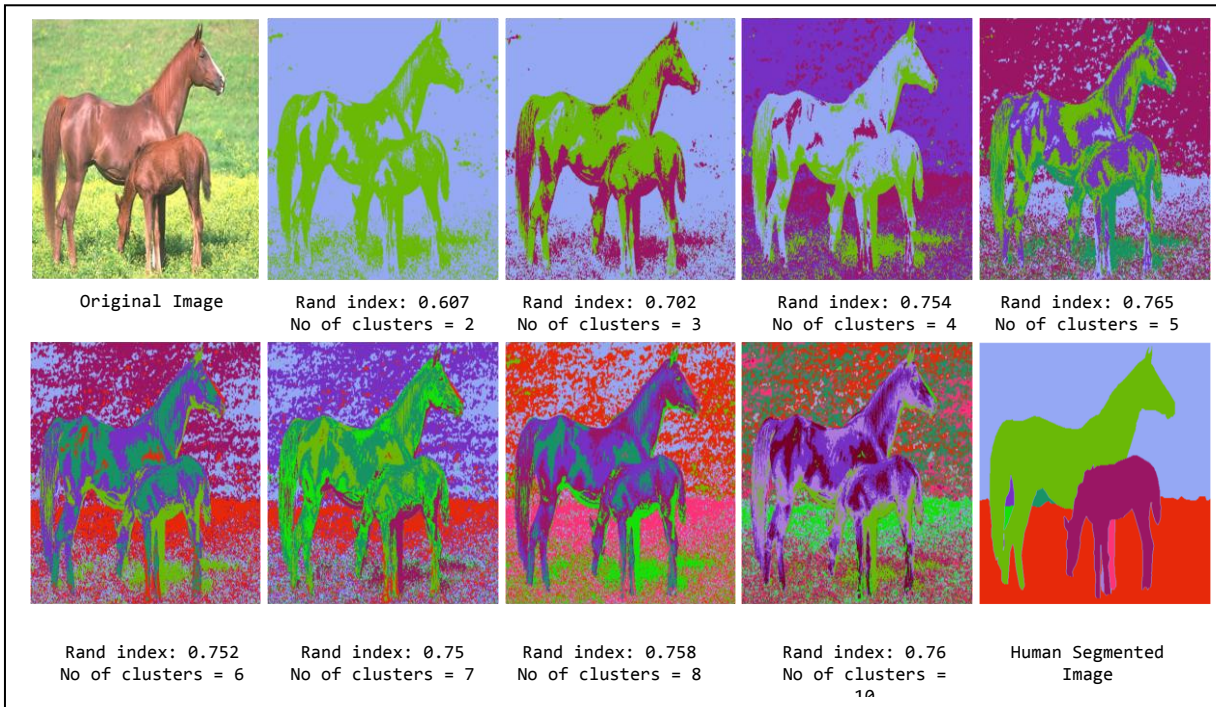

Figure 68 - Image number 113044 FCA, BSAS for centroid initialization

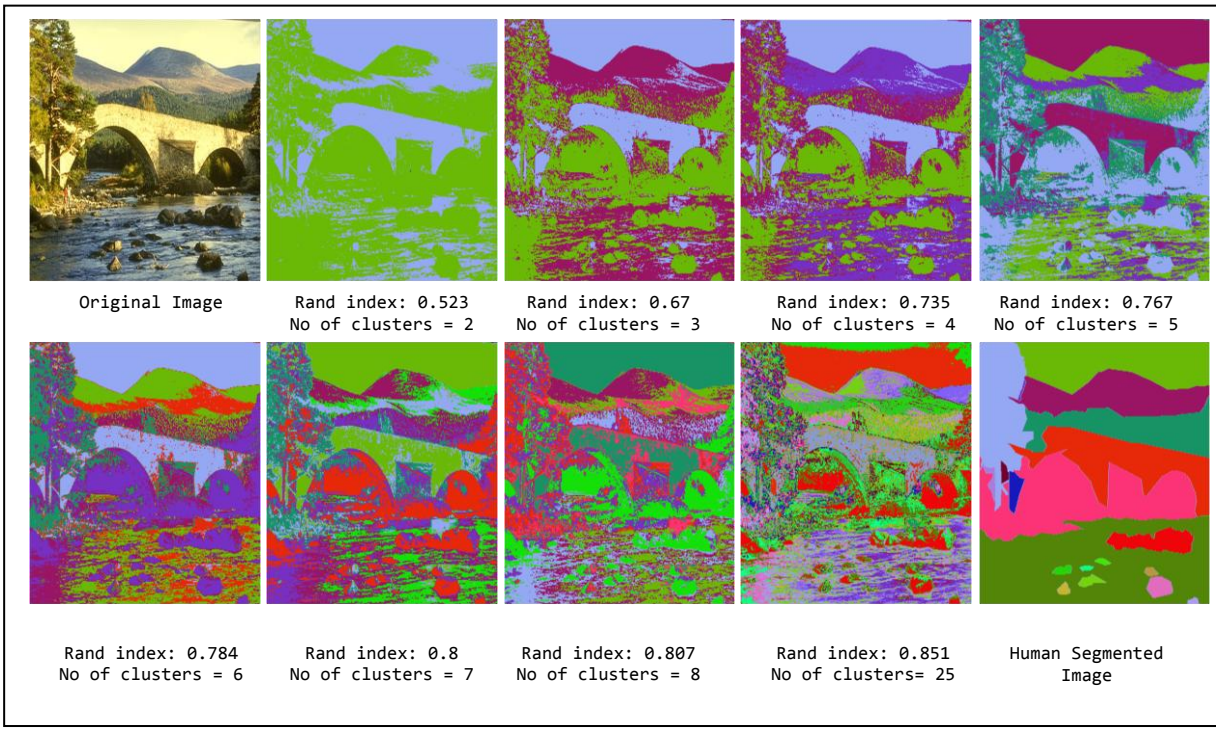

Figure 69 - Image number 231015, FCA, BSAS for centroid initialization

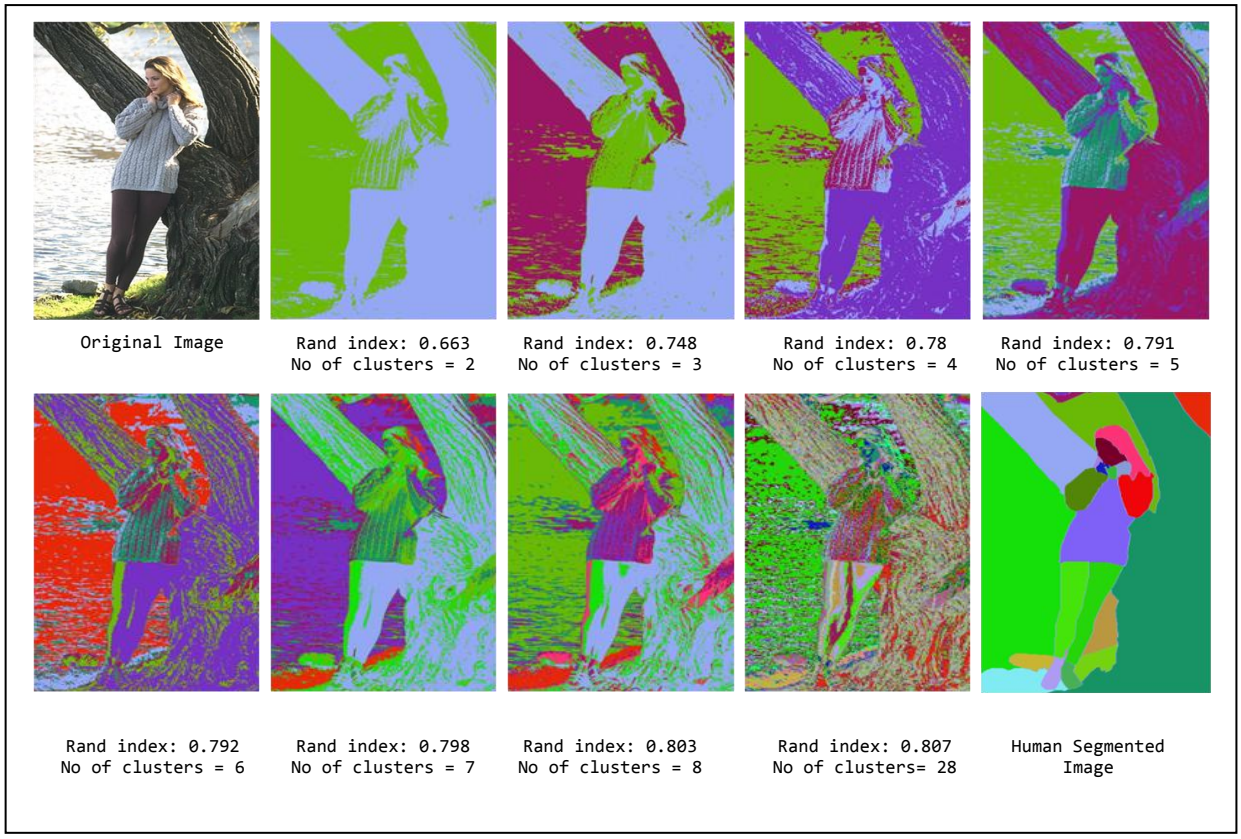

Figure 70 - Image number 181091, FCA, BSAS for centroid initialization

FCA and PCA - initialization with BSAS

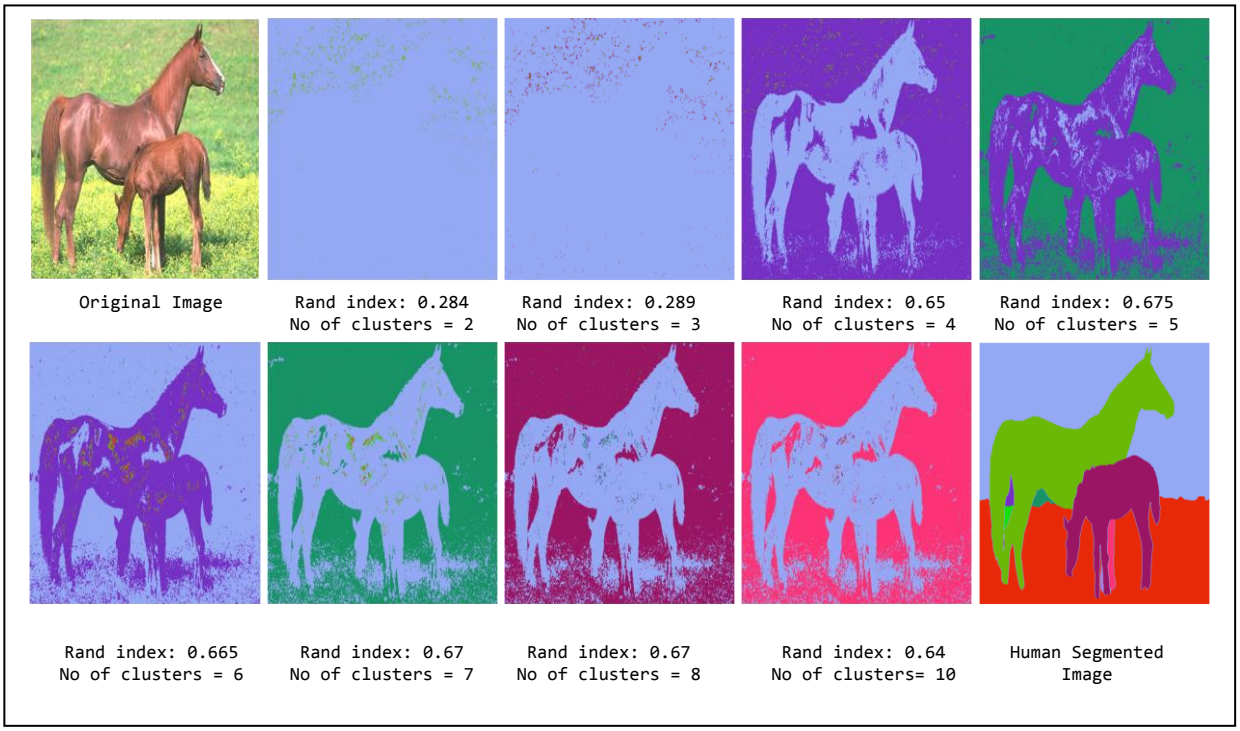

Figure 71 - Image number 113044 FCA, PCA, BSAS for centroid initialization

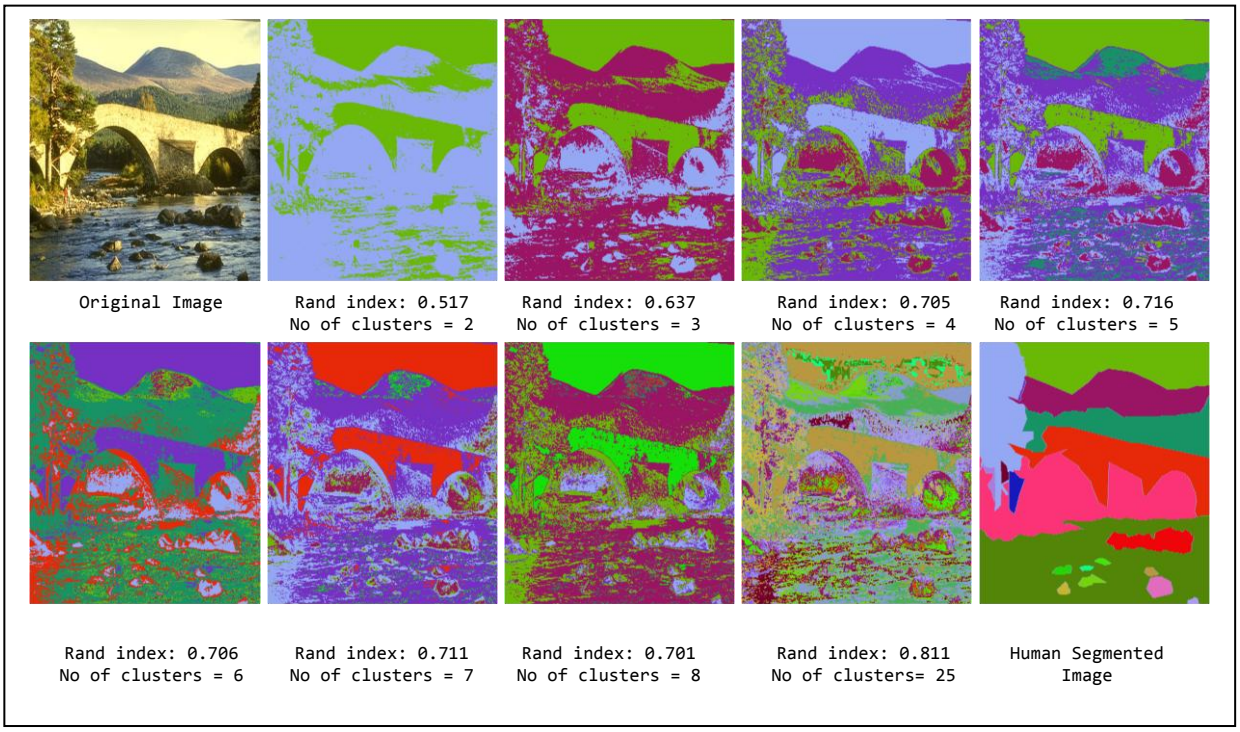

Figure 72 - Image number 231015, FCA, PCA, BSAS for centroid initialization

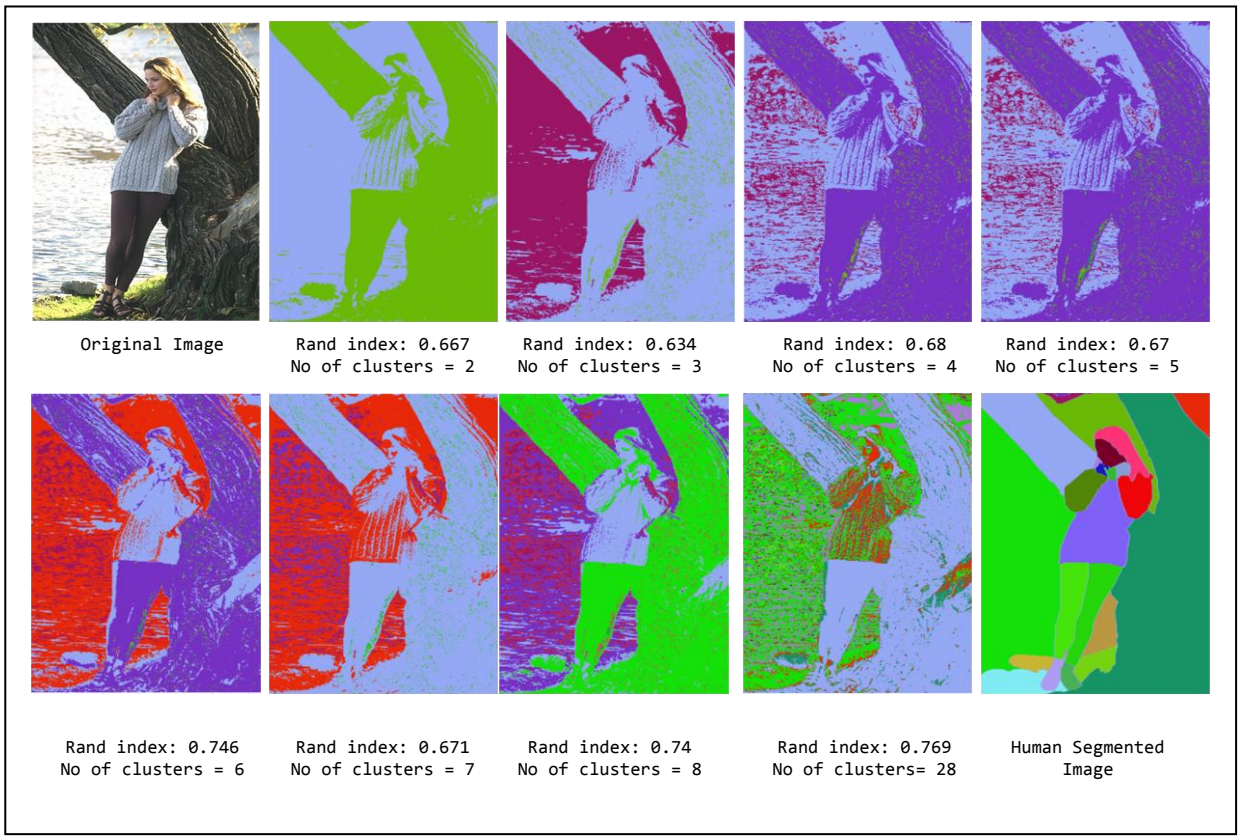

Figure 73 - Image number 181091, FCA, PCA, BSAS for centroid initialization

We notice that the rand index calculated after the execution of the algorithms manages to reach up to the level of 85% in the case of the image 231015. The highest values of the index are achieved when the number of clusters is the same as the number of the human defined segments. K-means and FCA give pretty much the same results in all the executions so we can safely consider that these two algorithms will achieve the same performance when executed on an image. On the other hand, the combination of FCA and PCA, does not reach similarly high values for rand index. What it actually does though is that it makes the objects of the image more compact. In a task where we would have to separate the foreground and the background of an image, this would probably be an algorithm of choice.

We also notice that even in the cases where the Rand index comes to report pretty good values, the resulting images are over-segmented. This is of course expected since each object in the image is consisted of pixels that have different values of color for several reason, as we will see. In the following section we are applying a merging procedure on the clustered image that incorporates the small segments that are scattered throughout the image into larger segments.

6.2 Merging Procedure

As we have already noted, we are making the assumption that the images under investigation are consisted of concrete objects which, for some reason, have different colors in different parts of their surface. This happens because of the reflection of the light, or because they are made of different materials. The assumption however considers that they present a basic structure, which is expressed by a basic color, which in turn is expressed by a basic cluster, into which we can integrate all the other secondary clusters that consist of the object under investigation.

For example, if we represent an image array with the cluster id of each of its pixels we would have:

5	5	5	5	5	5	5
5	5	1	1	1	5	5
5	5	5	1	1	5	5
5	5	1	1	1	5	5
5	5	5	5	1	5	5
5	5	5	5	5	5	5

Then the cluster with id = 1, which forms an “island” into the cluster with id = 5 can be integrated into the latter under our assumption.

A way to find these islands into our clustered images is by using depth first search. We are defining a threshold for the largest number of pixels an “island” can have and we are then traversing the whole image. After finding one such “island” we keep track of which is the cluster of each of its neighborhood pixels. The cluster appearing the most is the prevailing, the dominant one, so the “island” is merged into this.

For each algorithm we are executing the merging procedure on the clustering results of the previous section. The threshold value we experimentally found that it gives us good clustering values is 500.

- **K-means – initialization with BSAS – after merging operation**

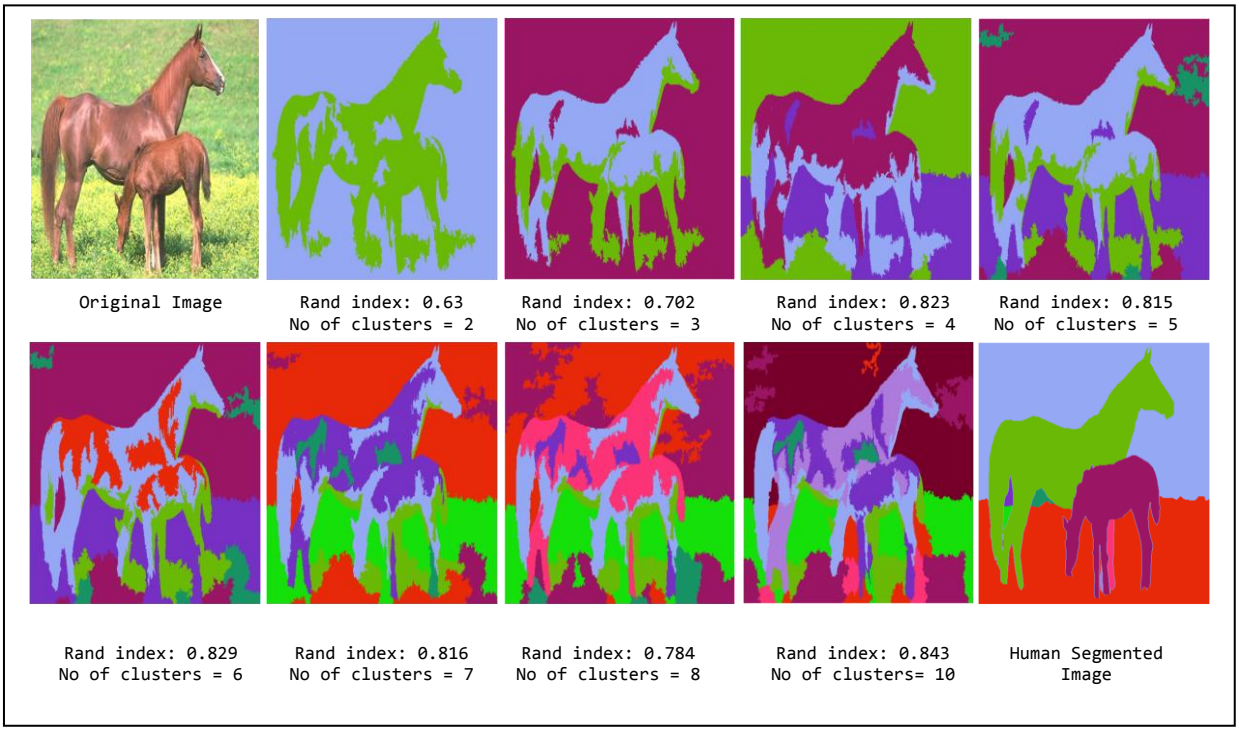

Figure 74 - Image number 113044, k-means algorithm, BSAS for centroid initialization, after merging

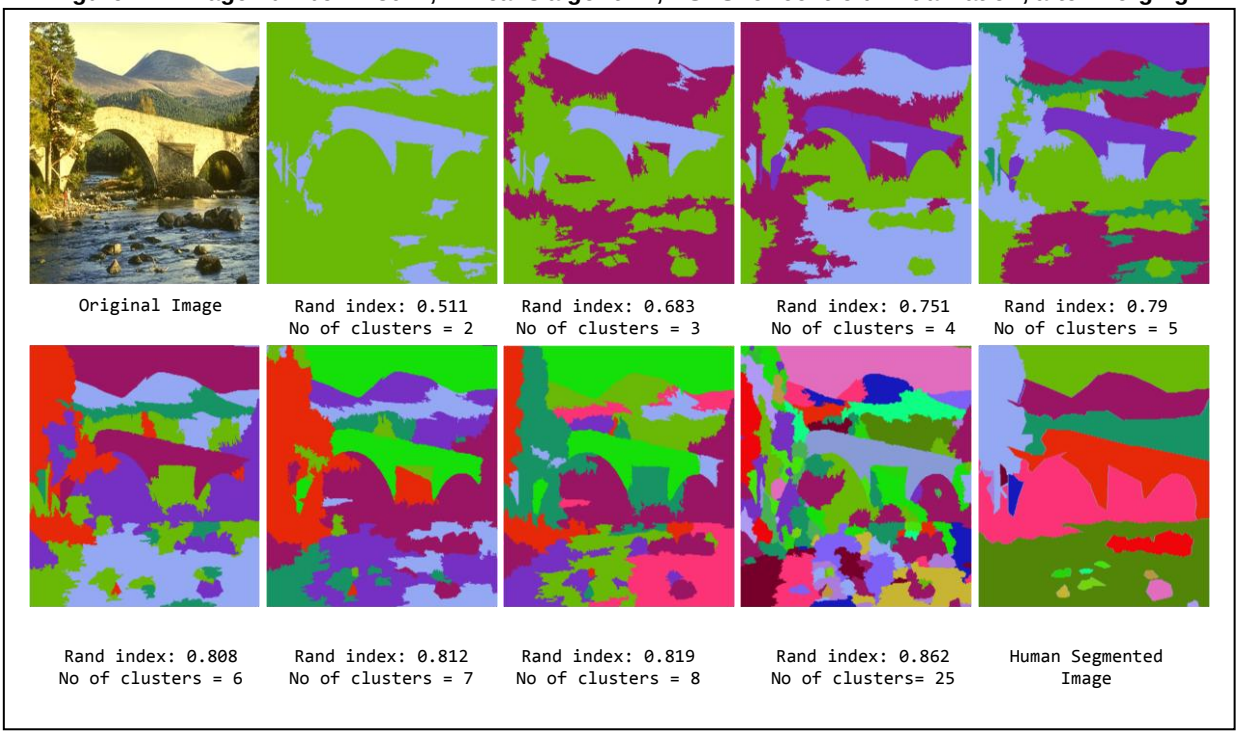

Figure 75 - Image number 231015, k-means algorithm, BSAS for centroid initialization, after merging

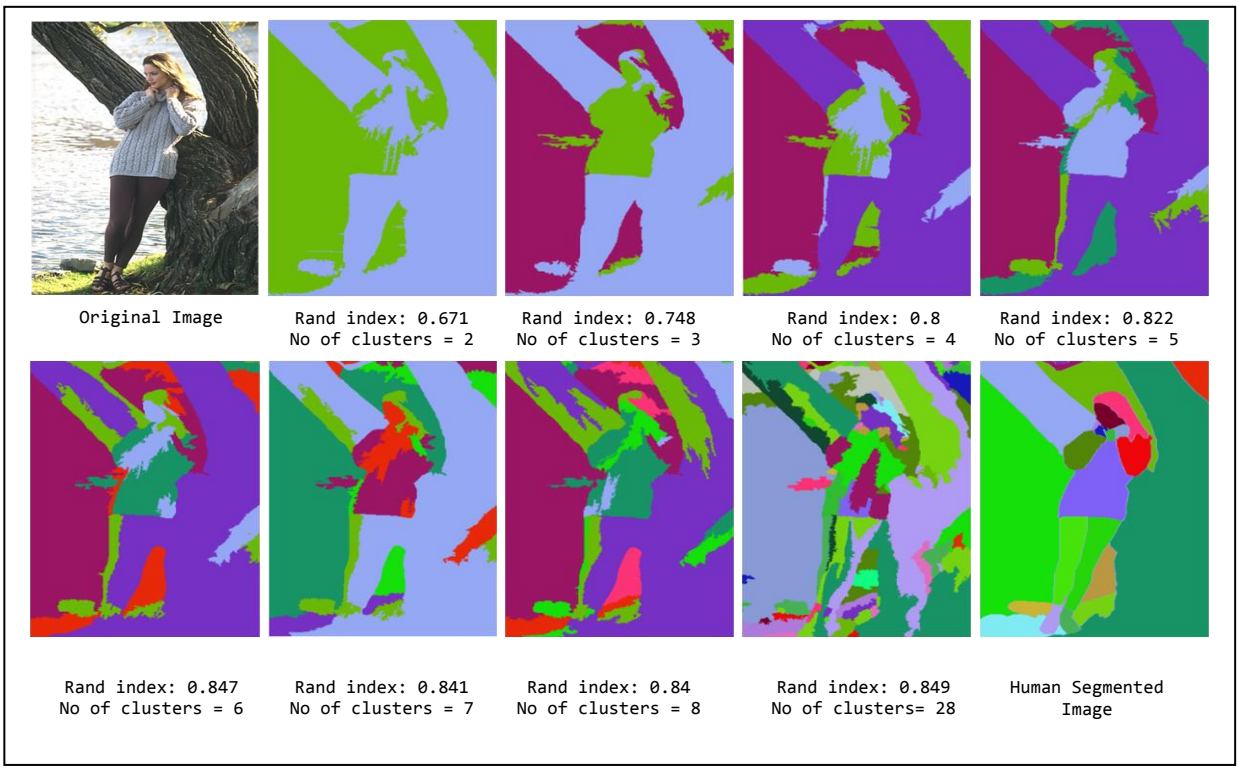

Figure 76 - Image number 181091, k-means algorithm, BSAS for centroid initialization, after merging

FCA initialization with BSAS – after merging operation

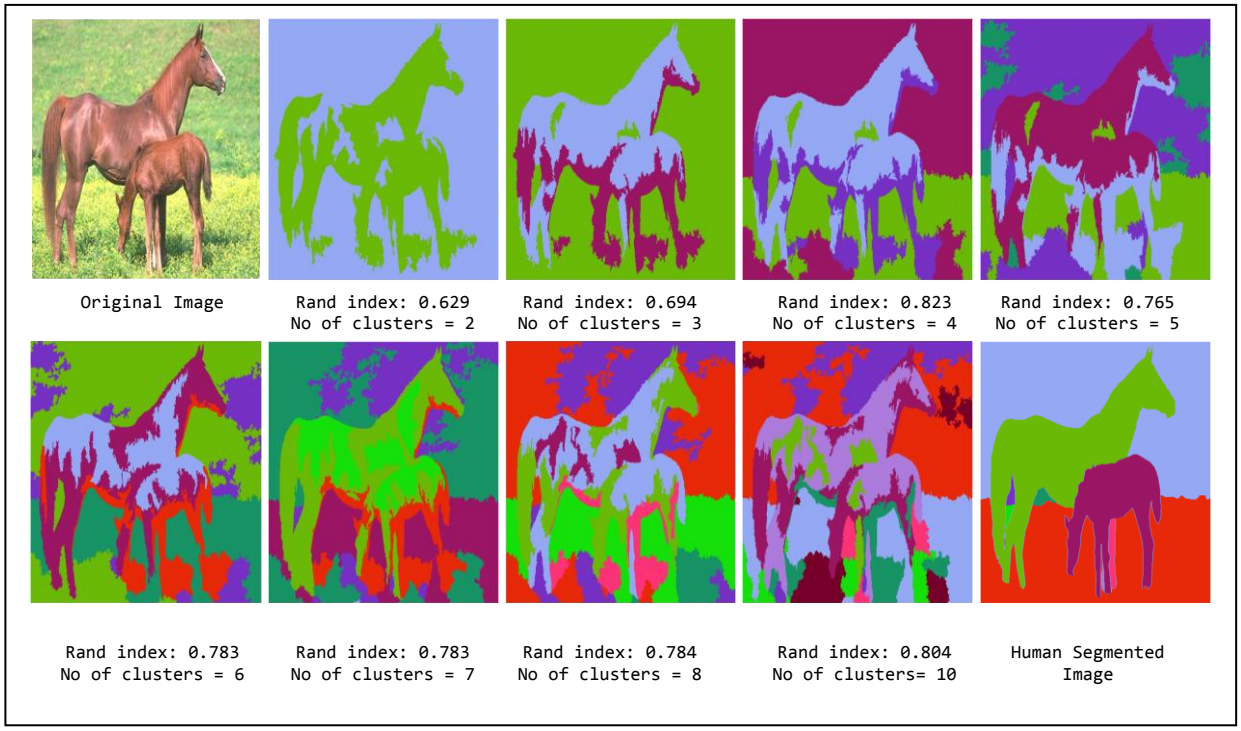

Figure 77 Image number 113044 FCA, BSAS for centroid initialization after merging

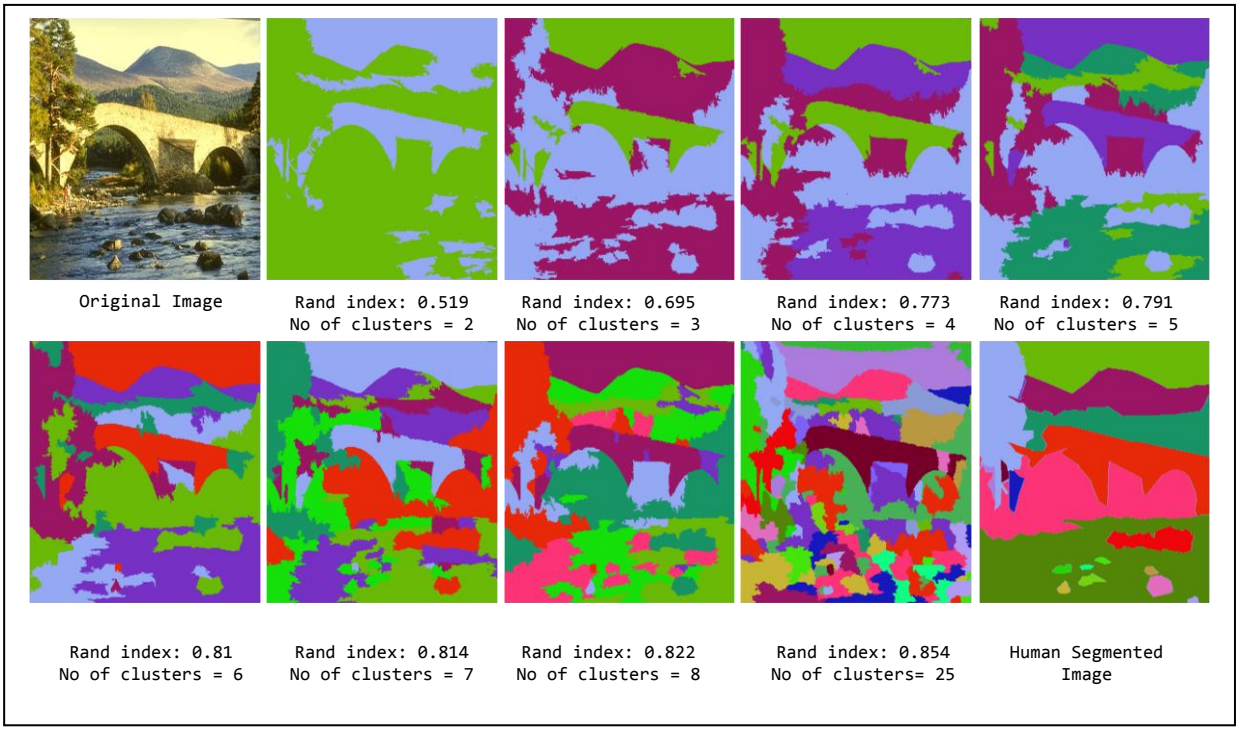

Figure 78 - Image number 231015, FCA, BSAS for centroid initialization after merging

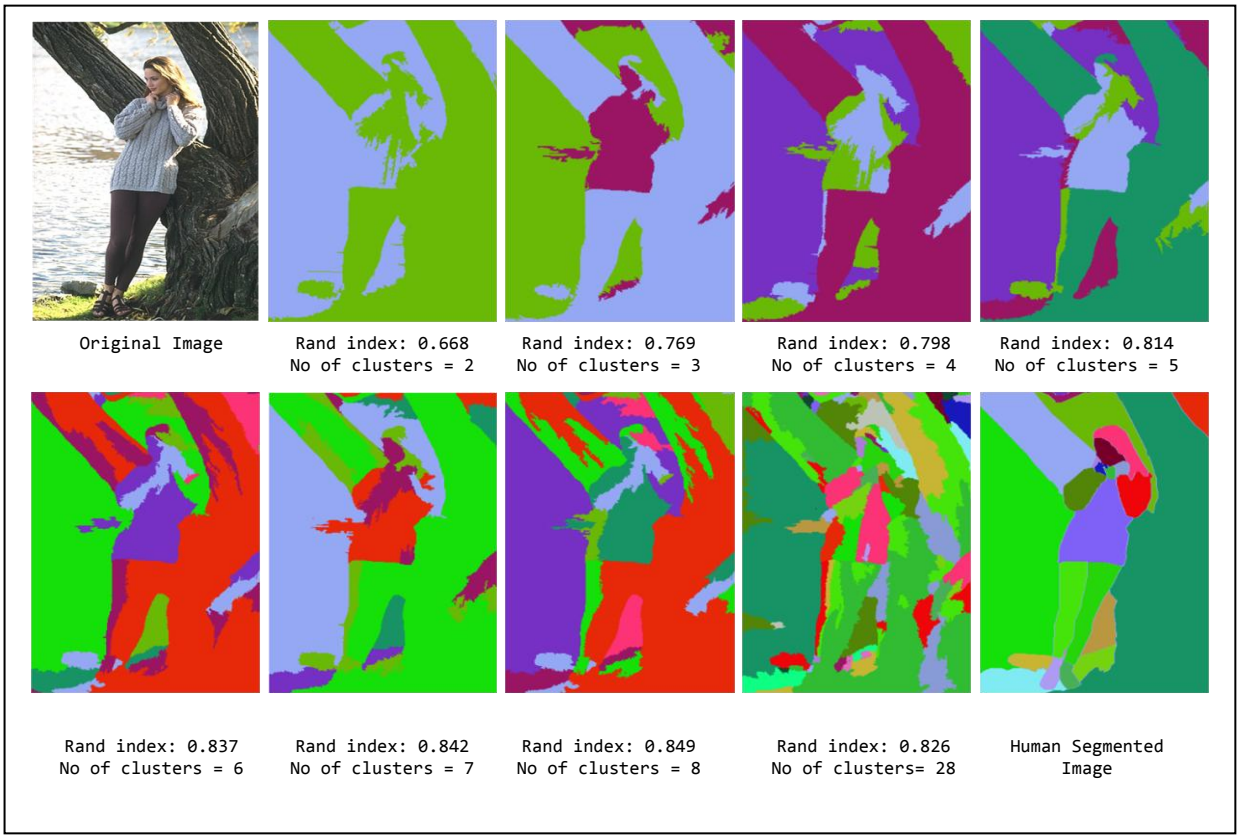

Figure 79 - Image number 181091, FCA, BSAS for centroid initialization after merging

FCA and PCA - initialization with BSAS after merging operation

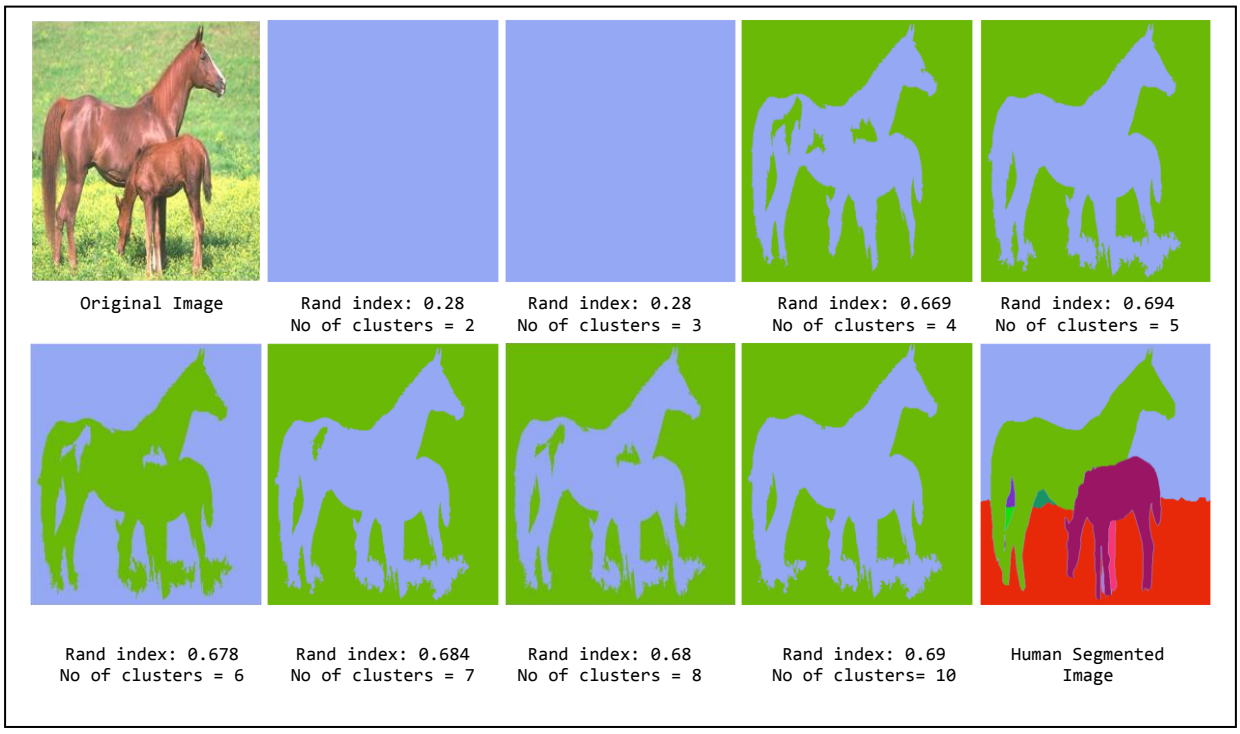

Figure 80 - Image number 113044 FCA, PCA, BSAS for centroid initialization after merging

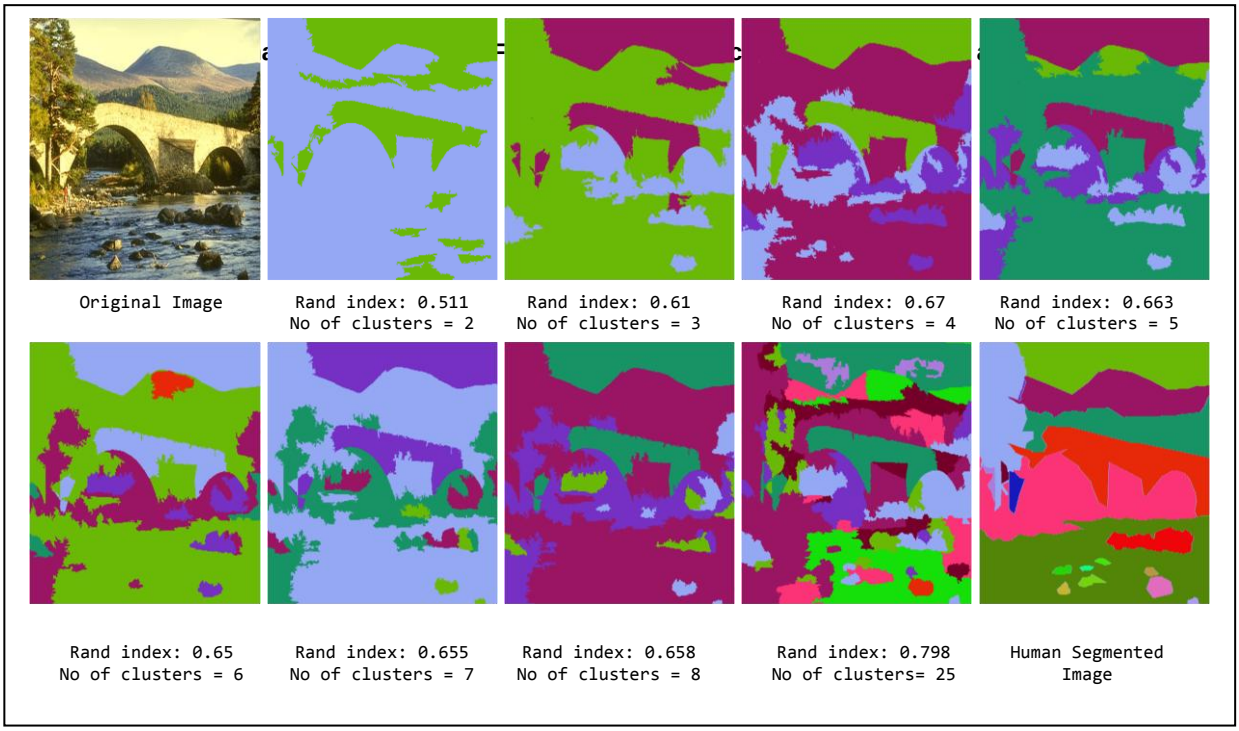

Figure 81 - Image number 231015, FCA, PCA, BSAS for centroid initialization after merging

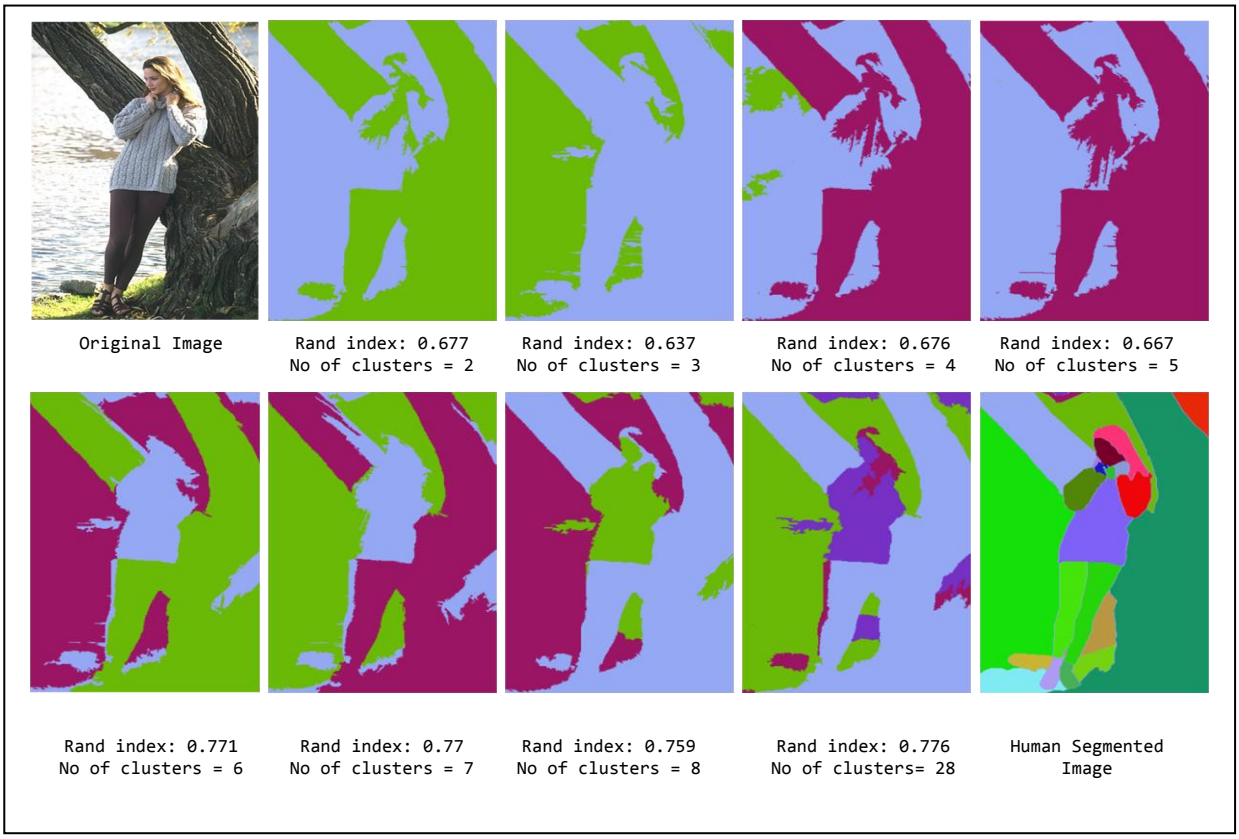

Figure 82 - Image number 181091, FCA, PCA, BSAS for centroid initialization after merging

The clustering results after the execution of the merging operation in some cases present significant improvement. For example, the rand index for the k-means best execution increases for image 113044 from 76% to 84%. The results for the same algorithm on image 231015 increase from 81% to 85%. Similar increases are noted for the majority of the executions of the previous sections.

What is more important though is the fact this merging operation accomplishes to make the objects on the image appear far more compact, something that cannot be achieved only by the execution of the clustering algorithms on the image.

These results force us to consider clustering procedures not as independent procedures for image segmentation but better as the first stage to segmenting an image followed by refinement procedures one of which is the merging procedure just described.

Conclusions

Cluster analysis is the field of unsupervised learning that includes processes that divide data into groups according to some proximity measure. Cluster validity is the term used to refer to all procedures used to evaluate the final result of a clustering algorithm. We examined, implemented and tested on synthetic datasets seven different clustering algorithms along with all the appropriate cluster validity criteria. Finally we applied selected algorithms to the task of segmenting images and proposed an effective technique to make objects on clustered images more compact.

Appendix

The current section serves as a user guide and the documentation to the scripts provided along with the thesis at the location https://github.com/thanSkourtan/Cluster-Analysis-Algorithms.

User Guide

Every algorithm category can be located in the package named after it. For every algorithm there are two basic scripts. One of them is the basic module which is named after the algorithm and contains the implementation of the algorithm and the other is included in the folder test (this is not a module, so it cannot be imported anywhere) in each algorithm's package and is named after the algorithms plus the string "_test" at the end of the script. The second file contains a class of type TestCase and each of the class' functions can be regarded as a unit test. The important thing to note here is that although we are using the logic of unit tests, we are not exactly testing some functions but rather use the test functions as entry points for our scripts. Let us see an example that will make things clearer.

If we want to execute the k-means algorithm in a dataset of 4 blobs, 2 features and 500 samples then we should go to the path './cost_function_optimization/tests/kmeans_test.py' and choose the first function 'testBlobs' to run. This can be accomplished by commenting out the line '@unittest.skip('no') that can be located in the line right above each of the test functions. From inside the test function we can change the parameters of the synthetic data as we wish. In the same way we can test the algorithms on concentric circles by running the function testCircles or moon-like data shapes by running the function testMoons. Same procedure applies in the test functions that execute the relative criteria and the image segmentation scripts. Regarding the last category of test function, they can be found only in kmeans_test, fuzzy_test and possibilistic_test scripts.

Of course somebody could dispose the test scripts and directly call the algorithm functions with his own data. The test scripts in other words are not part of the library, but rather left there for convenience in order the user to find an already set up environment to execute the algorithms and reproduce the results included in this thesis.

Documentation

Module: internal_criteria.py

A module containing all the necessary functions in order to run the internal criteria validation indices. The main reference for this module is (Theodoridis & Koutroumbas, 2009).

Functions:

internal_validity

A function that wraps the rest of the functions of this module and calls them in the appropriate order. It could be defined as the only public function of the module.

Parameters:

- data((N x m) 2-d numpy array): a data set of N instances and m features
- no_of_clusters(integer): the number of clusters

Returns:

- initial_gamma(float): the Gamma statistic of the clustering under consideration
- list_of_gammas(list): the Gamma statistics of all the monte carlo sample distributions
- result(string): a string containing the result of the function's computations

significance_calc

Calculates z-statistic for initial_gamma with regards to the normal distribution of list_of_gammas the p_value of the z-statistic and based on the results accepts or rejects the null hypothesis of randomness.

Parameters:

- initial_gamma(float): the Gamma statistic of the clustering under consideration
- list_of_gammas(list): the Gamma statistics of all the monte carlo sample distributions

Returns:

- result(string): a string containing the result of the function's computations

monte_carlo

Creates 100 (could be set as argument) sampling distributions of uniformingly distributed data and calls the algorithm passed as argument in order to cluster each distribution and calculate its Gamma statistic.

Parameters:

- data((N x m) 2-d numpy array): a data set of N instances and m features
- no_of_clusters(integer): the number of clusters

- algorithm: the algorithm function to be used to cluster the data

Returns:

- list_of_gammas(list): the Gamma statistics of all the monte carlo sample distributions

gamma

Calculates the Hubert's Gamma Statistic for the proximity matrix P and matrix Y, where Y (i,j) = 1 if i, j are in the same cluster, 0 otherwise. These matrices are fixed for the internal criteria case so they are integrated into this function, rather than been provided as arguments.

Parameters:

- data((N x m) 2-d numpy array): a data set of N instances and m features

Returns:

- g(float): the gamma index for P, Y

Module: external_criteria.py

A module containing all the necessary functions in order to run the external criteria validation indices. The main reference for this module is (Theodoridis & Koutroumbas, 2009).

Functions:

external_validity

A function that wraps the rest of the functions of this module and calls them in the appropriate order. It could be defined as the only public function of the module.

Parameters:

- data((N x m) 2-d numpy array): a data set of N instances and m features
- no_of_clusters(integer): the number of clusters

Returns:

- initial_indices(float): the initial indices of the clustering under consideration
- list_of_indices(list): the list of calculated indices of all the monte carlo sample distributions
- result_list(list): a list of strings containing the results of the function's computations

significance_calc

Calculates z-statistic for initial_indices with regards to the normal distribution of the values contained in the list_of_indices, the p_value of the z-statistic and finally, based on the results, accepts or rejects the null hypothesis of randomness.

Parameters:

- initial_indices (float): the initial indices of the clustering under consideration

- list_of_indices (list): the list of calculated indices of all the monte carlo sample distributions

Returns:

- result(string): a string containing the result of the function's computations

monte_carlo

Creates 100 (could be set as argument) sampling distributions of uniformingly distributed data and calls the algorithm passed as argument in order to cluster each distribution and calculate the external indices.

Parameters:

- data((N x m) 2-d numpy array): a data set of N instances and m features
- no_of_clusters(integer): the number of clusters
- external_data_info(list): a list containing the cluster id for each of the vector of the dataset
- algorithm(function object): the algorithm function to be used to cluster the data

Returns:

- list_of_gammas(list): the Gamma statistics of all the monte carlo sample distributions

external_indices

Calculates three indices (rand statistic, jaccard coefficient, Fowlkes and Mallows) based on a matrix P that shows the similarity between the clustering under consideration and an external clustering. Also calculates the Hubert's Gamma Statistic for matrices X and Y, where X (i,j) = 1 if i, j are in the same cluster in the clustering under consideration, 0 otherwise and Y(i,j) = 1 if i, j are in the same cluster in the external clustering, 0 otherwise.

Parameters:

- data((N x m) 2-d numpy array): a data set of N instances and m features
- external_data_info(list): the external clustering results

Returns:

- rand_statistic(float): the rand statistic
- jaccard_coefficient(float): the jaccard coefficient statistic
- fowlkes_and_mallows(float): the Fowlkes and Mallows index
- gamma(float): the gamma index for X, Y

Module: relative_criteria.py

A module containing the implementation of the all relative criteria indices.

Functions:

relative_validity_hard_sequential

Defines the several values of the BSAS parameter. Then conducts successive executions of the algorithm by passing to it those values and calculates all the proper relative indices.

Parameters:

- X((N x m) numpy array): a data set of N instances and m features

Returns:

- no_of_threshold_values: the different values of the threshold parameter
- DI, DB, SI: the arrays holding the values of the relative indices

relative_validity_TTSS

Defines the several values of the TTSS parameters. Then conducts successive executions of the algorithm by passing to it those values and calculates all the proper relative indices.

Parameters:

- X((N x m) numpy array): a data set of N instances and m features

Returns:

- no_of_threshold_values1: the different values of the threshold1 parameter
- no_of_threshold_values2: the different values of the threshold2 parameter
- DI, DB, SI: the arrays holding the values of the relative indices

relative_validity_hard_graph

Defines the several values of the MST parameters. Then conducts successive executions of the algorithm by passing to it those values and calculates all the proper relative indices.

Parameters:

- X((N x m) numpy array): a data set of N instances and m features

Returns:

- no_of_k_list: the different values of the k parameter
- no_of_f_list: the different values of the f parameter
- DI, SI: the arrays holding the values of the relative indices

relative_validity_hard

Defines the several values of the kmeans parameter. Then conducts successive executions of the algorithm by passing to it those values and calculates all the proper relative indices.

Parameters:

- X((N x m) numpy array): a data set of N instances and m features

Returns:

- no_of_clusters_list: the different values of the clusters number
- DI, DB, SI, GI: the arrays holding the values of the relative indices

relative_validity_fuzzy

Defines the several values of the fuzzy parameter. Then conducts successive executions of the algorithm by passing to it those values and calculates all the proper relative indices.

Parameters:

- X((N x m) numpy array): a data set of N instances and m features

Returns:

- no_of_clusters_list: the different values of the clusters number
- values_of_q: the different values of the q parameter
- PC, PE, XB, FS: the arrays holding the values of the relative indices

relative_validity_possibilistic

Defines the several values of the possibilistic parameter. Then conducts successive executions of the algorithm by passing to it those values and calculates all the proper relative indices.

Parameters:

- X((N x m) numpy array): a data set of N instances and m features

Returns:

- no_of_clusters_list: the different values of the clusters number
- values_of_q: the different values of the q parameter
- PC, PE, XB, FS: the arrays holding the values of the relative indices

Dunn_index

Calculates the Dunn index of a clustered dataset.

Parameters:

- X((N x m + 1) numpy array): a clustered data set of N instances, m features and the cluster id at the last column of each vector

Returns:

- The Dunn index

Davies_Bouldin

Calculates the Davies Bouldin index of a clustered dataset. Whereas in Dunn index the distance between clusters is the distance between the closest vectors of the clusters, in Davies Bouldin the same distance is the distance between the centroids.

Parameters:

- X((N x m + 1) numpy array): a clustered data set of N instances, m features and the cluster id at the last column of each vector
- centroids: The centroids returned from the clustering algorithm

Returns:

- The Davies Bouldin index

silhouette_index

Calculates the silhouette index of a clustered dataset.

Parameters:

- X((N x m + 1) numpy array): a clustered data set of N instances, m features and the cluster id at the last column of each vector

Returns:

- The silhouette index

_gap_index_calculation

Calculates the log(W) for the provided dataset.

Parameters:

- X((N x m + 1) numpy array): a clustered data set of N instances, m features and the cluster id at the last column of each vector

Returns:

- The log(W)

gap_index

Calculates the Gap index of a clustered dataset.

Parameters:

- X((N x m + 1) numpy array): a clustered data set of N instances, m features and the cluster id at the last column of each vector
- no_of_clusters: the number of clusters
- algorithm: the function object representing the algorithm that called the function

Returns:

- The Gap index

Xie_Beni

Calculates the Xie Beni index.

Parameters:

- X((N x m + 1) numpy array): a clustered data set of N instances, m features and the cluster id at the last column of each vector
- centroids: the value of the centroids after running a clustering algorihtm on the data set
- partition_matrix: the partition matrix

Returns:

- Xie_Beni(float): the value of the Xie Beni index

fukuyama_sugeno

Calculates the fukuyama sugeno index.

Parameters:

- X((N x m + 1) numpy array): a clustered data set of N instances, m features and the cluster id at the last column of each vector
- centroids: the value of the centroids after running a clustering algorihtm on the data set
- partition_matrix: the partition matrix

Returns:

- total_sum(float): the value of the fukuyama sugeno index

Module: fuzzy_clustering.py

A module containing the implementation of the fuzzy clustering algorithm.

Functions:

fuzzy

An implementation of the fuzzy clustering algorithm.

Parameters:

- data((N x m) 2-d numpy array): a data set of N instances and m features
- no_of_clusters(integer): the number of clusters
- centroids_initial: the optional initial values for the centroids
- q(integer): the fuzzifier parameter

Returns:

- data((N x (m + 1)) 2-d numpy array): the data set with one more column that contains the vector's cluster
- centroids_new((k x n) 2-d numpy array): contains the k = no_of_clusters centroids with n features
- ita(float): a parameter used in possibilistic clustering.
- centroids_history((l x 2) 2-d numpy array): an array to keep the previous positions of the centroids for better visualisation of the result.
- partition_matrix ((n x 2) 2-d numpy array): the matrix containing the weights which depict the grade of membership of a vector i to the cluster j

Module: possibilistic_clustering.py

A module containing the implementation of the fuzzy clustering algorithm

Functions:

possibilistic

An implementation of the possibilistic clustering algorithm.

Parameters:

- data((N x m) 2-d numpy array): a data set of N instances and m features
- no_of_clusters(integer): the number of clusters
- ita(list): contains the values of the ita parameter for every cluster
- centroids_initial(numpy array): the optional initial values for the centroids
- q(float): fuzzifier parameter

Returns:

- data((N x (m + 1)) 2-d numpy array): the data set with one more column that contains the vector's cluster

- centroids_new((k x m)2-d numpy array): contains the k = no_of_clusters centroids with m features
- centroids_history((1 x 2) 2-d numpy array): an array to keep the previous positions of the centroids for better visualization of the result.
- typicality_matrix ((n x 2) 2-d numpy array): the matrix containing the weights which depict the typicality of a vector i to the cluster j

Module: kmeans_clustering.py

A module containing the implementation of the fuzzy clustering algorithm

Functions:

kmeans

An implementation of the kmeans clustering algorithm.

Parameters:

- data((N x m) 2-d numpy array): a data set of N instances and m features
- no_of_clusters(integer): the number of clusters
- centroids_initial(): the optional initial values for the centroids

Returns:

- data((N x (m + 1)) 2-d numpy array): the data set with one more column that contains the vector's cluster
- centroids_new((k x n)2-d numpy array): contains the k = no_of_clusters centroids with n features
- centroids_history((1 x 2) 2-d numpy array): an array to keep the previous positions of the centroids for better visualisation of the result.

Module: image_segm_utility.py

Module of utility functions set to work along with the image repository of "The Berkeley Segmentation Dataset and Benchmark".

Functions:

insert_clusters

A function that takes the .seg format files along with the original image and returns the image as a numpy array, ALONG with the externally provided clusters (called segments in the seg file).

Parameters:

- original_image(string): the name of the original image. It must be placed into the "Images" folder.

- seg_file(string): the name of the .seg file with the human segmented results. It must be placed into the images folder.

Returns:

- clustered_image(numpy 3-d array): the .seg file in the form of a 3-d numpy array which contains the segment id for each pixel.

rand_index_calculation

Calculates the rand index of two different clustering results. The matrices provided as arguments must be contain at least 5000 elements. Instead of comparing all elements, the function chooses 5000 element uniformingly distributed in the input matrices and perform its calculations solely on them.

Parameters:

- X_(numpy array): the first clustering result as a numpy array
- external_info(numpy array): the second clustering result as a numpy array

Returns:

- rand index(float): the value of the rand index

marging_procedure

Takes a clustered image as a numpy 3-D array, containing the cluster id for each pixel, and transforms it in such a way that small clusters are merged into their neighborhood ones.

Parameters:

- image(3-D numpy array): array containing the cluster ids before the merge procedure
- threshold(integer): the user defined threshold for the maximum number of pixels allowed in a recursion

Returns:

- image(3-D numpy array): array containing the cluster ids after the merge procedure

_moves

Private function that takes the coordinates of the current position as arguments and calculates all the possible next positions.

Parameters:

- y(integer): the 'vertical coordinate' of the current pixel of the image
- x(integer): the 'horizontal coordinate' of the current pixel of the image

Returns:

- list_of_new_positions(list): a list of tuples of length 2 containing all the next possible pixels on the image, either eligible or not

_constraints

Private function that takes the coordinates of a position as arguments and calculates whether it is eligible or not. Please note that in order to process an image we reshape it to 2 dimensions

Parameters:

- y(integer): the 'vertical coordinate' of the position of the image
- x(integer): the 'horizontal coordinate' of the position of the image
- N(integer): the length of the second dimension of the image
- m(integer): the length of the first dimension of the image

Returns:

- list_of_new_positions(list): a list of tuples of length 2 containing all the next possible pixels on the image, either eligible or not

_dfs_util

Private function that implements the depth first search algorithms on the image by visiting pixels that belong to the same cluster. It also returns the cluster that appears most often in the neighborhood pixels.

Parameters:

- image(numpy array): the 2-D image array
- y(integer): the 'vertical coordinate' of the current position of the image
- x(integer): the 'horizontal coordinate' of the current position of the image
- N(integer): the length of the second dimension of the image
- m(integer): the length of the first dimension of the image
- visited(numpy array): a 2-D array to hold the several stages of a pixel
- pixels_cluster(integer): the cluster id of the current pixel
- counter(integer): a counter to measure the recursion depth
- dominant_cluster_list(numpy array): a list to count the prevailing cluster of the neighbourhood pixels
- threshold(integer): the user defined threshold for the maximum number of pixels allowed in a recursion

Returns:

- list_of_new_positions(list): a list of tuples of length 2 containing all the next possible pixels on the image, either eligible or not

Module: MST.py

A module containing the implementation of the Minimum Spanning Tree clustering algorithm.

Functions:

minimum_spanning_tree

An implementation of the Minimum Spanning tree clustering algorithm.

Parameters:

- data((N x m) 2-d numpy array): a data set of N instances and m features
- k(integer): the user defined step to define the depth of the neighborhood of a function
- q(float): the user defined number of standard deviations of the weights mean above which an edge is considered inconsistent
- f(float): the user defined threshold of the ratio of the weight of the edge under investigation and the neighborhood average weights

Returns:

- clustered_data((N x (m + 1)) 2-d numpy array): the data set with one more column that contains the vector's cluster
- no_of_clusters(integer): the number of clusters

_dfs_util

A private utility depth first search algorithm used in order to find the forest of trees the dataset is consisted of.

Parameters:

- MST: the minimum spanning tree matrix
- s: the current node index
- visited_nodes: the list of nodes that have been visited
- cluster_id: the id of the cluster to be assigned to a node
- data: the data matrix

_recursion _util

A utility recursive method used in order to gather the weights of an edge's neighborhood edges.

Parameters:

- nodes: the two nodes of the edge
- k: the step defining the depth of the neighbourhoud edges
- list_of_weights: a list to fill in the weights of the neighborhoud edges
- MST: the minimum spanning tree matrix

Returns:

- list_of_weights: a list to fill in the weights of the neighborhoud edges

Module: DTA.py

A module containing the implementation of the Delaunay Triangulation clustering algorithm.

minimum_spanning_tree_variation

An implementation of a graph algorithm based on Delaunay triangulation graph.

Parameters:

- data((m x n) 2-d numpy array): a data set of m instances and n features

Returns:

- clustered_data((N x (m + 1)) 2-d numpy array): the data set with one more column that contains the vector's cluster

_dfs_util

A private utility depth first search algorithm used in order to find the forest of trees the dataset is consisted of.

Parameters:

- MST: the minimum spanning tree matrix
- s: the current node index
- visited_nodes: the list of nodes that have been visited
- cluster_id: the id of the cluster to be assigned to a node
- data: the data matrix

Module: BSAS.py

A module containing the implementation of the basic sequential scheme clustering algorithm.

Functions:

basic_sequential_scheme

An implementation of the basic sequential scheme clustering algorithm.

Parameters:

- data((m x n) 2-d numpy array): a data set of m instances and n features
- max_number_of_clusters(integer): the maximum allowable number of clusters

Returns:

- clustered_data((m x (n + 1)) 2-d numpy array): the data set with one more column that contains the vector's cluster
- centroids((k x n)2-d numpy array): contains the k = no_of_clusters centroids with n features
- no_of_clusters(integer): the final number of clusters created

thresholding_BSAS

A function to calculate the value of the threshold by using the peaks and valleys technique

Parameters:

- data((m x n) 2-d numpy array): a data set of m instances and n features

Returns:

- deepest_valley(float): the height of the histogram at the point of the deepest valley between the two highest peaks. It is actually the threshold value

Module: TTSS.py

A module containing the implementation of the Two Threshold Sequential Scheme.

Functions:

two_threshold_sequential_scheme

An implementation of the two threshold sequential scheme clustering algorithm.

Parameters:

- data((m x n) 2-d numpy array): a data set of m instances and n features

Returns:

- clustered_data((m x (n + 1)) 2-d numpy array): the data set with one more column that contains the vector's cluster
- centroids((k x n)2-d numpy array): contains the k = no_of_clusters centroids with n features
- no_of_clusters(integer): the final number of clusters created

thresholding_TTSS

A function to calculate the values of the thresholds

Parameters:

- data((m x n) 2-d numpy array): a data set of m instances and n features

Returns:

- deepest_valley1(float): the height of the histogram at the point of the first deepest valley between the three highest peaks. It is actually the threshold 1 value
- deepest_valley2(float): the height of the histogram at the point of the second deepest valley between the three highest peaks. It is actually the threshold 2 value

Bibliography

Anderberg, M. R. (1973). *Cluster Analysis for Applications*.

Arkadev, A., & Braverman, E. (1967). *Computers and Pattern Recognition*. Washington D.C.: Thompson Book Co.

Ball, G., & Hall, D. (1965). Isodata, a novel method of data analysis and pattern classification. *Technical Report*.

Bellman, R., Kalaba, R., & Zadeh, L. (n.d.). Abstraction and pattern classification. *Journal of the American Society for Information Science and Technology, 2*.

Berkeley Segmentation Dataset and Benchmark. (n.d.). Retrieved from https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/

Bezdek, J. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press.

Bezdek, J. C. (1975). *Mathematical Models for Systematics and Taxonomy*.

Bezdek, J. C. (1993). Fuzzy models - What are they, and why? *IEEE Transactions on Fuzzy Systems, 1*(1).

Bezdek, J. C., Keller, J., Krisnapuram, R., & Pal, N. R. (2005). *Fuzzy Models Algorithms for Pattern Recognition and Image Processing*. Springer Science and Business Media, Inc.

Bezdek, J., & C. (1973). *Cluster Validity with Fuzzy Sets*.

Bonner, R. E. (1964). On some clustering techniques. *International Business Machines Journal of Research and Development, 8*(22-32).

Bouldin, D. W., & Davies, D. L. (1979). A Cluster Separation Measure. *IEEE Transactions on pattern analysis and machine intelligence, PAMI - 1*(2).

Buhmann, J. M. (2002). *The Handbook of Brain Theory and Neural Networks*.

Cormack, R. M. (1971). A Review of Classification. *Journal of the Royal Statistical Society, 134*(3).

Delaunay, B. (1934). Sur la sphere vide. *Bulletin de l'Academie des Sciences de l' URSS*.

Deluca, A., & Termini, S. (1972). A definition of nonprobabilistic entropy in the setting of fuzzy sets theory. *Inf and Control, 20*(301 - 312).

Dunn, J. (1973). A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *J. Cybernet, 3*.

Dunn, J. C. (1974). Well separated clusters and optimal fuzzy partitions. *Journal of Cybernetics, 4*.

Eldershaw, C., & Hegland, M. (n.d.). Cluster Analysis using Triangulation. *Computational Techniques and Applications*.

Feldman, J. (1995). Perceptual models of small dot clusters. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*.

Fowlkes, E. B., & Mallows, C. (1983). A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*.

Ghosh, B. K., & Sen, P. K. (1991). *Handbook of sequential analysis*. Marcel Dekker.

Gonzalez, R. C., & Woods, R. E. (2007). *Digital Image Processing*. Pearson International Edition.

Guojun Gan, C. M. (2007). *Data Clustering, Theory, Algorithms and Applications*. American Statistical Association.

Hall, A. V. (1967). Methods for demonstrating resemblance in taxonomy and ecology. *Nature, 214*.

Irvin Rock, S. P. (1990). The Legacy of Gestalt Psychology.

Iverson, K. E. (1962). *Notation as a Tool of Thought*. Wiley.

Jain, A. K. (2008). *Data Clustering, 50 Years beyond K-means*.

Jain, A. K., & Dubes, R. C. (1988). *Algorithms for Clustering Data*. Prentice Hall Advanced Reference Series.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data Clustering: A Review. *ACM Computing Surveys*.

Johnson, S. (1967). Hierarchical clustering schemes. *Psychometrika, 32*.

Kaufman, L., & Rousseeuw, P. J. (2005). *Finding groups in data*. Wiley Interscience.

Krishnapuram, R., & Keller, J. (1993). A possibilistic approach to clustering. *IEEE Transactions on Fuzzy Systems, 1*.

Lloyd, S. P. (1982). Least Squares Quantization in PCM. *IEEE Transactions on information theory*.

Marinus van Dongen, S. (2000). *Graph clustering by flow simulation*.

Mehmet, S., & Bulent, S. (2004). Survey over image thresholding techniques and quantitative performance evalution. *Journal of Electronic Imaging, 13*.

Miin-Shen, Y., & Kuo-Lung, W. (2005). Unsupervised possibilistic clustering. *Pattern Recognition*.

Nikhil, P. R., Kuhu, P., Keller, J., & Bezdek, C. J. (2005). A Possibilistic Fuzzy c-Means Clustering Algorithm. *IEEE TRANSACTIONS ON FUZZY SYSTEMS, 13*(4).

Rand, W. M. (1971). Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association*.

Rousseeuw, P. J. (1986). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics, 20*.

Ruspini, E. (1969). A new approach to clustering. *Information and Control, 15*.

Shapiro, L., & Stockman, G. (2000). *Computer Vision*.

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer.

Theodoridis, S., & Koutroumbas, K. (2009). *Pattern Recognition*. Elsevier.

Tibshirani, R., Walther, G., & Hastie, T. (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology),, 63*.

Trahanias, P., & Skordalakis, E. (1989). An efficient sequential clustering method. *Pattern Recognition, 22*(4).

Wertheimer, M. (1923). Untersuchungen zur Lehre von der Gestalt II. *Psycologische Forschung, 4*.

Xie, X. L., & Beni, G. (1991). A Validity Measure for Fuzzy Clustering. *IEEE Transactions on pattern analysis and machine intelligence, 13*(8).

Zadeh, L. A. (1965). Fuzzy Sets. *Information and Control, 8*(3).

Zadeh, L. A. (1978). Fuzzy sets as a basis for a theory of possibility. *Fuzzy sets and systems, 1*(3).

Zahn, C. T. (1971). Graph - Theoretical Methods for Detecting and Describing Gestalt Clusters. *IEEE Transactions on Computers, C-20*(1).

Zimmermann, H. (2001). *Fuzzy Set Theory and Its Applications*. New York: Springer Science and Business Media, LLC.