SCHOOL OF MEDICINE

UNIVERSITY OF THESSALY

POSTGRADUATE PROGRAMME (MSC)

"RESEARCH METHOLOGY IN BIOMEDICINE, BIOSTATISTICS AND CLINICAL BIOINFORMATICS AT UNIVERSITY OF THESSALY"

Master's Thesis

"Discuss about a software in Python which performs single linkage clustering, furthest neighboring clustering and centroid clustering"

"Ανάλυση ενός προγράμματος σε Python το οποίο εκτελεί συσταδοποιήσεις απλού συνδέσμου, πλήρους συνδέσμου και με βάση το κέντρο βάρους"

by

RENTAS DIMITRIOS

Supervisors: Kowald Axel, Δοξάνη Χρυσούλα, Ζιντζαράς Ηλίας

Larisa, 2017

Contents

Chapter 1: Abstract

(EN) In this master's thesis we will discuss about a software in Python which performs three different clustering models: Single linkage, Furthest neighbor and Centroid. In Chapter 2 (Introduction), we will discuss the meaning of cluster analysis. In Chapter 3 (Methods), we will analyze the three clustering models and also, we will present the algorithm used. At the end of this chapter there is the software's code and the explanation of it, by its creator. In Chapter 4 (Results), I will give an example of how the software operates and we'll discuss the results. Finally, in Chapter 5 (Conclusion) we have a summary of the whole project.

(GR) Στη παρούσα μεταπτυχιακή διπλωματική εργασία θα συζητήσουμε τη λειτουργία ενός προγράμματος, στη γλώσσα προγραμματισμού Python, το οποίο θα εκτελεί τρεις διαφορετικές μεθόδους συσταδοποίησης, οι οποίες αναφέρονται στον τίτλο. Στο Κεφάλαιο 2 (Εισαγωγή) συζητάμε τον ορισμό της Ανάλυσης σε Συστάδες. Στο Κεφάλαιο 3 (Μέθοδοι) γίνεται ανάλυση των τριών μεθόδων και επίσης αναλύεται η διαδικασία που θα βασιστεί ο αλγόριθμος. Στο τέλος αυτού το κεφαλαίου υπάρχει ο κώδικας του προγράμματος και η επεξήγηση της λειτουργίας του, όπως μας έχει δοθεί. Στο Κεφάλαιο 4 (Αποτελέσματα) έχουμε ένα παράδειγμα για το πως λειτουργεί το πρόγραμμα και αναλύουμε τα αποτελέσματα που επιστρέφει. Τέλος, στο Κεφάλαιο 5 (Επίλογος) κάνουμε μία σύνοψη όλων όσων διατυπώθηκαν.

Chapter 2: Cluster Analysis

The purpose of cluster analysis is to partition a set of experimental data into groups in such a way that the data points within the same group, also known as a "cluster", are highly similar while data points in different clusters are very different. There is no simple recipe for choosing one particular approach over another for a particular clustering problem, just as there is no universal notion of what constitutes a "good cluster."

In order to analyze the software, we have to understand what cluster analysis is. So, cluster analysis is a multivariate method which aims to classify a sample of subjects (or objects) on the basis of a set of measured variables into a number of different groups such that similar subjects are placed in the same group. Something important is that cluster analysis has no mechanism for differentiating between relevant and irrelevant variables. Therefore, the choice of variables included in a cluster analysis must be underpinned by conceptual considerations. This is very important because the clusters formed can be very dependent on the variables included.

Chapter 3: Clustering Methods and Software

A)   There are a number of different methods that can be used to carry out a cluster analysis; these methods can be classified as follows:

Hierarchical methods

— **Agglomerative methods**, in which subjects start in their own separate cluster. The two 'closest' (most similar) clusters are then combined and this is done repeatedly until all subjects are in one cluster. At the end, the optimum number of clusters is then chosen out of all cluster solutions.

— **Divisive methods**, in which all subjects start in the same cluster and the above strategy is applied in reverse until every subject is in a separate cluster.

Agglomerative methods are used more often than divisive methods, so this handout will concentrate on the former rather than the latter.

Non-hierarchical methods (often known as k-means clustering methods)

k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

Only agglomerative methods are going to be used in the software.

B)   The definition of 'distance' is what differentiates between the different agglomerative clustering methods. A number of different measures have been proposed to measure 'distance' for binary and categorical data, but for interval data the most common distance measure used is the Euclidean distance.

Euclidean distance

In general, if you have n variables $X_1, X_2, \ldots, X_n$ measured on a sample of p subjects, the observed data for subject i can be denoted by $x_{i1}, x_{i2}, \ldots, x_{in}$ and the observed data for subject j by $x_{j1}, x_{j2}, \ldots, x_{jn}$. The Euclidean distance between these two subjects is given by

$$d_{i,j} = \sqrt{\left(x_{i1} - x_{j1}\right)^2 + \left(x_{i2} - x_{j2}\right)^2 + \cdots + \left(x_{in} - x_{jn}\right)^2}$$

C)   The three models that are going to be executed by the developed software summarized below.

Single linkage method (Nearest neighbor method)

In single-linkage clustering, the distance between two clusters is determined by a single element pair, namely those two elements (one in each

cluster) that are closest to each other. The shortest of these links that remains at any step causes the fusion of the two clusters whose elements are involved. The method is also known as nearest neighbor clustering. The result of the clustering can be visualized as a dendrogram, which shows the sequence of cluster fusion and the distance at which each fusion took place.

Mathematically, the linkage function – the distance D(X,Y) between clusters X and Y – is described by the expression

$$D(X,Y) = \min_{x \in X, y \in Y} d(x,y)$$

where X and Y are any two sets of elements considered as clusters, and d(x,y) denotes the distance between the two elements x and y.

Furthest neighbor method (complete linkage method)

Opposite to the single linkage approach is the complete linkage, also called furthest neighbor sorting. In this method, first proposed by Sørensen (1948), the fusion of two clusters depends on the most distant pair of objects instead of the closest. In this case, the distance between two clusters is defined to be the maximum distance between members — i.e. the distance between the two subjects that are furthest apart.

Mathematically, the complete linkage function – the distance D(X,Y) between the clusters X and Y – is described by the following expression:

$$D(X,Y) = \max_{x \in X, y \in Y} d(x,y)$$

where X and Y are any two sets of elements considered as clusters, and d(x,y) denotes the distance between the two elements x and y.

Centroid method

Here the centroid (mean value for each variable) of each cluster is calculated and the distance between centroids is used. Clusters whose centroids are closest together are merged.

Mathematically, the centroid function – the distance D(X,Y) between the clusters X and Y – is described by the following expression:

$$D(X,Y) = \|c_X - c_Y\| = d(c_X, c_Y)$$

Where c_X and c_Y are the centroids of clusters X and Y, respectively.

Program and description of the code

The module contributed below contains two major classes: Point and Cluster, which allow model points and clusters of points. Once our data have transformed into a list of my generic Point objects, pass it into agglo function, which performs agglomerative clustering. The output is a list of Cluster objects, which we may then process as we see fit.

The agglomerative algorithm can be used even if we have no idea how many clusters we should end up with. It takes two parameters: the linkage type (currently either 's' for single linkage, 'c' for complete linkage, or 't' for centroid linkage) and the cutoff. In each iteration, the algorithm computes the pair of clusters with the "smallest" distance between them and fuses them, either until all the clusters have been fused into one mega-cluster or until the smallest distance is bigger than your "cutoff" input parameter. The distances between clusters are computed by linkage type: single linkage means "the distance between the closest pair of points, one from each cluster"; complete linkage means "the distance between the farthest pair of points, one from each cluster"; and centroid linkage is simply the distance between the cluster centroids.

The main function takes as input our data in a ".csv" file and then passes them through the clustering algorithms and prints the output clusters.

Finally, the python IDE that we used is PyDev and the code editor is Eclipse - Neon (http://www.eclipse.org/neon/), with python 2.7 version. Also, we used the libraries math, sys and pandas for any subsidiary function.

```python
import sys, math, pandas

# -- The Point class represents points in n-dimensional space
class Point:

    # Instance variables
    # self.coords is a list of coordinates for this Point
    # self.n is the number of dimensions this Point lives in
    # (i.e. its   space)
    # self.reference is an object bound to this Point
    # Initialize new Points
    def __init__(self, coords, reference=None):
        self.coords = coords
        self.n = len(coords)
        self.reference = reference

    # Return a string representation of this Point
    def __repr__(self):
        return str(self.coords)
```

4

```python
# -- The Cluster class represents clusters of points in n-dimensional
space
class Cluster:

    # Instance variables
    # self.points is a list of Points associated with this Cluster
    # self.n is the number of dimensions this Cluster's Points live in
    # self.centroid is the sample mean Point of this Cluster
    # Initialize new Clusters

    def __init__(self, points):

        # We forbid empty Clusters (they don't make mathematical
        # sense!)
        if len(points) == 0: raise Exception("ILLEGAL: EMPTY CLUSTER")
        self.points = points
        self.n = points[0].n

        # We also forbid Clusters containing Points in different spaces
        # Ie, no Clusters with 2D Points and 3D Points
        for p in points:
            if p.n != self.n: raise Exception("ILLEGAL: MULTISPACE
                                                        CLUSTER")
        # Figure out what the centroid of this Cluster should be
        self.centroid = self.calculateCentroid()

    # Return a string representation of this Cluster
    def __repr__(self):
        return str(self.points)

    # Calculates the centroid Point - the centroid is
    # the sample mean Point
    # (the average of all the Points in the Cluster)
    def calculateCentroid(self):
        centroid_coords = []

        # For each coordinate:
        for i in range(self.n):

            # Take the average across all Points
            centroid_coords.append(0.0)
            for p in self.points:
                centroid_coords[i] = centroid_coords[i]+p.coords[i]

            centroid_coords[i] = centroid_coords[i]/len(self.points)

        # Return a Point object using the average coordinates
        return Point(centroid_coords)

    # Return the single-linkage distance between this and another
    # Cluster
    def getSingleDistance(self, cluster):
        ret = getDistance(self.points[0], cluster.points[0])
        for p in self.points:
            for q in cluster.points:
                distance = getDistance(p, q)
                if distance < ret: ret = distance
        return ret
```

5

```python
        # Return the complete-linkage distance between this and another
        # Cluster
        def getCompleteDistance(self, cluster):
            ret = getDistance(self.points[0], cluster.points[0])
            for p in self.points:
                for q in cluster.points:
                    distance = getDistance(p, q)
                    if distance > ret: ret = distance
            return ret

        # Return the centroid-linkage distance between this and another
        # Cluster
        def getCentroidDistance(self, cluster):
            return getDistance(self.centroid, cluster.centroid)

        # Return the fusion of this and another Cluster
        def fuse(self, cluster):
            # Forbid fusion of Clusters in different spaces
            if self.n != cluster.n: raise Exception("ILLEGAL FUSION")
            points = self.points
            points.extend(cluster.points)
            return Cluster(points)

# -- Return a distance matrix which captures distances between all
Clusters
def makeDistanceMatrix(clusters, linkage):
    ret = dict()
    for i in range(len(clusters)):
        for j in range(len(clusters)):
            if j == i: break
            if linkage == 's':
                ret[(i,j)] = clusters[i].getSingleDistance(clusters[j])
            elif linkage == 'c':
                ret[(i,j)] =clusters[i].getCompleteDistance(clusters[j])
            elif linkage == 't':
                ret[(i,j)] =clusters[i].getCentroidDistance(clusters[j])
            else: raise Exception("INVALID LINKAGE")
    return ret


# -- Return Clusters of Points formed by agglomerative clustering
def agglo(points, linkage, cutoff):

    # We only allow single, complete, or average linkage
    if not linkage in [ 's', 'c', 't' ]: raise Exception("INVALID
                                                          LINKAGE")

    # Create single Clusters, one for each Point
    clusters = []
    for p in points: clusters.append(Cluster([p]))

    # Set the min_distance between Clusters to zero
    min_distance = 0

    # Loop until the break statement is made
    while (True):
```

```python
        # Compute a distance matrix for all Clusters
        distances = makeDistanceMatrix(clusters, linkage)

        # Find the key for the Clusters which are closest together
        min_key = distances.keys()[0]
        min_distance = distances[min_key]
        for key in distances.keys():
            if distances[key] < min_distance:
                min_key = key
                min_distance = distances[key]

        # If the min_distance is bigger than the cutoff, terminate the
        # loop
        # Otherwise, agglomerate the closest clusters
        if min_distance > cutoff or len(clusters) == 1: break
        else:
            c1, c2 = clusters[min_key[0]], clusters[min_key[1]]
            clusters.remove(c1)
            clusters.remove(c2)
            clusters.append(c1.fuse(c2))

    # Return the list of Clusters
    return clusters

# -- Get the Euclidean distance between two Points
def getDistance(a, b):

    # Forbid measurements between Points in different spaces
    if a.n != b.n: raise Exception("ILLEGAL: NON-COMPARABLE POINTS")

    # Euclidean distance between a and b is sqrt(sum((a[i]-b[i])^2) for
    # all i)
    ret = 0.0
    for i in range(a.n):
        ret = ret+pow((a.coords[i]-b.coords[i]), 2)
    return math.sqrt(ret)

# -- Create a n-dimensional Point from our input
def DataInput(data,n):

    return Point(df[n])

# -- Main function
def main(args):
    # Open and read the data file which is in excel format
    # Header = -1 so the function will not ignore the first row
    df = pandas.read_csv('File_path.csv','rb', header=-1, delimiter=';')

    # Set linkage and cutoff point
    # (s=Single, c=Complete, t=Centroid)
    linkage, agglo_cutoff = 'Our Choice'


    # Convert our data to list and create our data array
    Temp = df.values.tolist()

    points=[]
```

7

```python
    for n in range(len(temp)):
        p=DataInput(temp,n)
        points.append(p)
        #Print the points array
        print "P(",n,"):", p

    # Cluster the points using the agglomerative algorithm, print the
    # results
    print "\nAGGLOMERATIVE"
    if linkage == 's':
        print "SINGLE LINKAGE CLUSTERING\nCLUSTERS:"
    elif linkage == 'c':
        print "COMPLETE LINKAGE CLUSTERING\nCLUSTERS:"
    elif linkage == 't':
        print "CENTROID CLUSTERING\CLUSTERS:"

    clusters = agglo(points, linkage, agglo_cutoff)
    for c in clusters: print "C:", c

# -- The following code executes upon command-line invocation
if __name__ == "__main__": main(sys.argv)
```

(Code source: https://www.daniweb.com/programming/software-development/code/216641/statistical-learning-with-python-clustering)

As we see, the above code creates a distance matrix, in order to store the distances created in each step, and then continues to the merging of the clusters. A distance matrix is a square matrix (two-dimensional array) containing the distances, taken pairwise, between the elements of a set. In this program, because we use the Euclidean metric, we have an Euclidean distance matrix is an $n{\times}n$ matrix representing the spacing of a set of n points in Euclidean space. If A is an Euclidean distance matrix and the points $x_1,x_2,\ldots,x_n$ are defined on m-dimensional space, then the elements of A are given by

$A=(a_{ij})$; $a_{ij}=d_{ij}$, where d_{ij} is the appropriate metric for each method, so:

$$\begin{bmatrix} 0 & d_{12} & d_{13} & \cdots & d_{1n} \\ d_{21} & 0 & d_{23} & \cdots & d_{2n} \\ d_{31} & d_{32} & 0 & \cdots & d_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & d_{n3} & \cdots & 0 \end{bmatrix}$$

Chapter 4: Example using the software

Here I am going to demonstrate, with a simple example, how this software works and we will discuss the results. In this example, we use 4 -dimensional data points, but there isn't any restriction on the number of dimensions.

In a group of 10 patients with cystic fibrosis we recorded several variables Body Mass Percentage (BMP), Forced Expiratory Volume (FEV1), Residual Volume (RV) and Maximum Expiratory Pressure (PEmax). We want to investigate the relationship between them and categorize the patients with similarities. (Source: Elias Zintzaras, 2017, MSc Biomathematics)

The data points are in an excel sheet (csv file) so they are accessible to the software.

The dataset is the following:

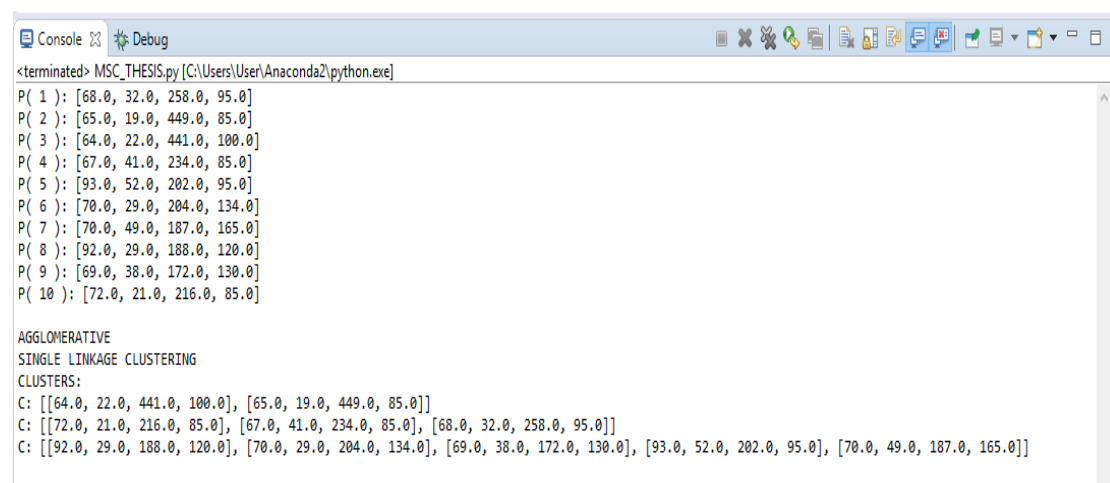| Patient	BMP	FEV1	RV	PEmax
1	68.0	32.0	258.0	95.0
2	65.0	19.0	449.0	85.0
3	64.0	22.0	441.0	100.0
4	67.0	41.0	234.0	85.0
5	93.0	52.0	202.0	95.0
6	70.0	29.0	204.0	134.0
7	70.0	49.0	187.0	165.0
8	92.0	29.0	188.0	120.
9	69.0	38.0	172.0	130.0
10	72.0	21.0	216.0	85.0

At the beginning, we have to pass our data in an excel sheet.
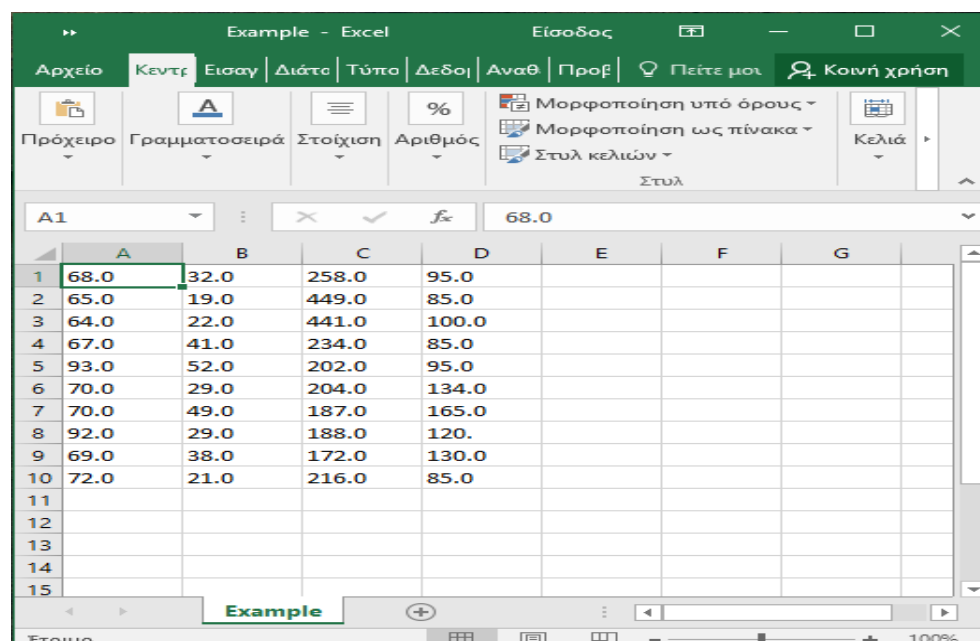
Then, we import the file's path (i.e. *C:\Users\User \Eclipse\Example.csv*), choose the type of linkage (i.e. 's', 'c', 't') and define the cut-off point = 40.0. Here it is necessary to mention that there isn't any correct or wrong method to choose the cut-off point, we just choose based in our experience and by testing it.

We are going to run the software three times, one for each clustering method. We begin with single linkage clustering method, also known as nearest neighbor, so we choose linkage = 's' and press Run.

We see the results at the Console of Eclipse and they are below:

```
Console ⛌  Debug
<terminated> MSC_THESIS.py [C:\Users\User\Anaconda2\python.exe]
P( 1 ): [68.0, 32.0, 258.0, 95.0]
P( 2 ): [65.0, 19.0, 449.0, 85.0]
P( 3 ): [64.0, 22.0, 441.0, 100.0]
P( 4 ): [67.0, 41.0, 234.0, 85.0]
P( 5 ): [93.0, 52.0, 202.0, 95.0]
P( 6 ): [70.0, 29.0, 204.0, 134.0]
P( 7 ): [70.0, 49.0, 187.0, 165.0]
P( 8 ): [92.0, 29.0, 188.0, 120.0]
P( 9 ): [69.0, 38.0, 172.0, 130.0]
P( 10 ): [72.0, 21.0, 216.0, 85.0]

AGGLOMERATIVE
SINGLE LINKAGE CLUSTERING
CLUSTERS:
C: [[64.0, 22.0, 441.0, 100.0], [65.0, 19.0, 449.0, 85.0]]
C: [[72.0, 21.0, 216.0, 85.0], [67.0, 41.0, 234.0, 85.0], [68.0, 32.0, 258.0, 95.0]]
C: [[92.0, 29.0, 188.0, 120.0], [70.0, 29.0, 204.0, 134.0], [69.0, 38.0, 172.0, 130.0], [93.0, 52.0, 202.0, 95.0], [70.0, 49.0, 187.0, 165.0]]
```

Our data are printed for each patient separately and then the single linkage algorithm returns the clusters containing the patients who are more similar, according to the distance used. We see that the first cluster contains Patients #3 and #2, the second cluster contains Patients #10, #4, #1 and the last cluster contains the remaining five patients.

If the clusters aren't satisfying, we can change the cut-off point and see if our new results are satisfying enough. For example, if we set the cut off = 30.0, the results are:

```
AGGLOMERATIVE
SINGLE LINKAGE CLUSTERING
CLUSTERS:
C: [[93.0, 52.0, 202.0, 95.0]]
C: [[70.0, 29.0, 204.0, 134.0]]
C: [[70.0, 49.0, 187.0, 165.0]]
C: [[92.0, 29.0, 188.0, 120.0]]
C: [[69.0, 38.0, 172.0, 130.0]]
C: [[64.0, 22.0, 441.0, 100.0], [65.0, 19.0, 449.0, 85.0]]
C: [[72.0, 21.0, 216.0, 85.0], [67.0, 41.0, 234.0, 85.0], [68.0, 32.0, 258.0, 95.0]]
```

10

As it is obvious, this cut-off point doesn't generate a worthy set of clusters. So, we will keep as cut-off = 40.0

We move to the second method and we choose linkage = 'c' for complete linkage clustering method, also known as farthest neighbor, and press Run.

This time the results are the following:

```
Console ⛶  Debug                                                          ■ ✖ ❎ ◕
<terminated> MSC_THESIS.py [C:\Users\User\Anaconda2\python.exe]
P( 1 ): [68.0, 32.0, 258.0, 95.0]
P( 2 ): [65.0, 19.0, 449.0, 85.0]
P( 3 ): [64.0, 22.0, 441.0, 100.0]
P( 4 ): [67.0, 41.0, 234.0, 85.0]
P( 5 ): [93.0, 52.0, 202.0, 95.0]
P( 6 ): [70.0, 29.0, 204.0, 134.0]
P( 7 ): [70.0, 49.0, 187.0, 165.0]
P( 8 ): [92.0, 29.0, 188.0, 120.0]
P( 9 ): [69.0, 38.0, 172.0, 130.0]
P( 10 ): [72.0, 21.0, 216.0, 85.0]

AGGLOMERATIVE
COMPLETE LINKAGE CLUSTERING
CLUSTERS:
C: [[68.0, 32.0, 258.0, 95.0]]
C: [[93.0, 52.0, 202.0, 95.0]]
C: [[70.0, 49.0, 187.0, 165.0]]
C: [[64.0, 22.0, 441.0, 100.0], [65.0, 19.0, 449.0, 85.0]]
C: [[72.0, 21.0, 216.0, 85.0], [67.0, 41.0, 234.0, 85.0]]
C: [[92.0, 29.0, 188.0, 120.0], [70.0, 29.0, 204.0, 134.0], [69.0, 38.0, 172.0, 130.0]]
```

As we see there are dissimilarities in comparison with the previous method. With the same cut off point, now we have six clusters whereas before we had three. This is something that we expected to happen because the metric we used to group our patients is very different. Thus, here we have three clusters with one patient each, something like cluster of themselves, two clusters with two patients each and one cluster with three patients.

But we observe that this cut-off point isn't satisfying enough for this method because it generates many clusters. So, we can change it to cut-off = 70.0 and now we have a set of three more compact clusters.

```
AGGLOMERATIVE
COMPLETE LINKAGE CLUSTERING
CLUSTERS:
C: [[64.0, 22.0, 441.0, 100.0], [65.0, 19.0, 449.0, 85.0]]
C: [[92.0, 29.0, 188.0, 120.0], [70.0, 29.0, 204.0, 134.0], [69.0, 38.0, 172.0, 130.0], [70.0, 49.0, 187.0, 165.0]]
C: [[72.0, 21.0, 216.0, 85.0], [67.0, 41.0, 234.0, 85.0], [93.0, 52.0, 202.0, 95.0], [68.0, 32.0, 258.0, 95.0]]
```

And finally, in order to run the Centroid clustering method, we set as linkage = 't', cut-off = 40.0 and then press Run.

The results are below:

```
Console 23   Debug
<terminated> MSC_THESIS.py [C:\Users\User\Anaconda2\python.exe]
P( 1 ): [68.0, 32.0, 258.0, 95.0]
P( 2 ): [65.0, 19.0, 449.0, 85.0]
P( 3 ): [64.0, 22.0, 441.0, 100.0]
P( 4 ): [67.0, 41.0, 234.0, 85.0]
P( 5 ): [93.0, 52.0, 202.0, 95.0]
P( 6 ): [70.0, 29.0, 204.0, 134.0]
P( 7 ): [70.0, 49.0, 187.0, 165.0]
P( 8 ): [92.0, 29.0, 188.0, 120.0]
P( 9 ): [69.0, 38.0, 172.0, 130.0]
P( 10 ): [72.0, 21.0, 216.0, 85.0]

AGGLOMERATIVE
CENTROID CLUSTERING
CLUSTERS:
C: [[93.0, 52.0, 202.0, 95.0]]
C: [[70.0, 49.0, 187.0, 165.0]]
C: [[64.0, 22.0, 441.0, 100.0], [65.0, 19.0, 449.0, 85.0]]
C: [[92.0, 29.0, 188.0, 120.0], [70.0, 29.0, 204.0, 134.0], [69.0, 38.0, 172.0, 130.0]]
C: [[72.0, 21.0, 216.0, 85.0], [67.0, 41.0, 234.0, 85.0], [68.0, 32.0, 258.0, 95.0]]
```

With a quick view to the cluster, we observe that two of them contain only one patient each, so this is a good reason to change the cut-off point. By changing it only to cut-off = 50.0 the results change a lot and now we have three clusters, as we observe below:

```
AGGLOMERATIVE
CENTROID CLUSTERING
CLUSTERS:
C: [[64.0, 22.0, 441.0, 100.0], [65.0, 19.0, 449.0, 85.0]]
C: [[92.0, 29.0, 188.0, 120.0], [70.0, 29.0, 204.0, 134.0], [69.0, 38.0, 172.0, 130.0], [70.0, 49.0, 187.0, 165.0]]
C: [[72.0, 21.0, 216.0, 85.0], [67.0, 41.0, 234.0, 85.0], [68.0, 32.0, 258.0, 95.0], [93.0, 52.0, 202.0, 95.0]]
```

Chapter 5: Conclusion

In this thesis, we discussed what cluster analysis is, in an effort to understand the three methods analyzed. Then, an example is given, using the software, where I presented the results that we received and a way to adjust them to our advantage. Through the whole process the most important thing to understand is that there is no better or worse method to use, but everything depends on the problem at hand, because, as it was obvious before, every method generates different results and it is in our consideration how we'll interpret them. From all the above, we can draw the conclusion that we have to find the most suitable clustering method for each problem.

Chapter 6: References

1. Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval*, Cambridge University Press. 2008. (https://nlp.stanford.edu/IR-book/html/htmledition/irbook.html)
2. Daniel Borcard (2006), Multivariate analysis, Chapter 3. Cluster analysis (http://ubio.bioinfo.cnio.es/Cursos/CEU_MDA07_practicals/Further%20reading/Multivariate%20analysis%20Borcard%202006/Chap_3.pdf)
3. Dr. Alexander Schonhuth, Lecture Notes, CMPT441: Algorithms in Bioinformatics, pg. 75-82 (http://www.cs.sfu.ca/CourseCentral/441/asa86/LectureNotes_11_05.pdf)
4. General information about programming in python https://stackoverflow.com/
5. Norusis (2016), IBM SPSS STATISTICS 19 Guide to Data analysis (http://www.norusis.com/pdf/SPC_v13.pdf)
6. Pavel A. Pevzner, Neil C. Jones, An Introduction to Bioinformatics Algorithms (2004), Chapter 10. Clustering and Trees, pg.339-386 (http://bioinformaticsinstitute.ru/sites/default/files/an_introduction_to_bioinformatics_algorithms_-_jones_pevzner.pdf)
7. Pierre and Louis Legendre, Numerical Ecology, Elsevier Scientific Publishing Company, Amsterdam, 1983, pg.303-386
8. Rosie Cornish (2007), Learning Support Center, Chapter 3 Cluster Analysis (http://www.statstutor.ac.uk/resources/uploaded/clusteranalysis.pdf)
9. https://www.daniweb.com/programming/software-development/code/216641/statistical-learning-with-python-clustering