

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής



Εργασία Μαθήματος «Αναγνώριση προτύπων»

	Εργασία Αναγνώρισης Προτύπων
Ονόματα φοιτητών – Αριθμοί Μητρώων	Αθανάσιος Δημήτριος Κανταλάφτ -Π15050
	Παναγιώτης Καφαντάρης – Π15054
	Εμμανουήλ Μανεσιώτης - Π15079
Ημερομηνία παράδοσης	14/02/2019

Ο φοιτητής Παναγιώτης Καφαντάρης δεν έχει δηλώσει το μάθημα στο students λόγω σφάλματος της πλατφόρμας για το αντίστοιχο εξάμηνο, θα πρέπει ο βαθμός να μπει χειροκίνητα.

1) Αφού αποσυμπιέσετε το σχετικό αρχείο, ακολουθήστε τις οδηγίες που βρίσκονται στο README, για να κατανοήσετε το ρόλο κάθε αρχείου που προκύπτει από την αποσυμπίεση.

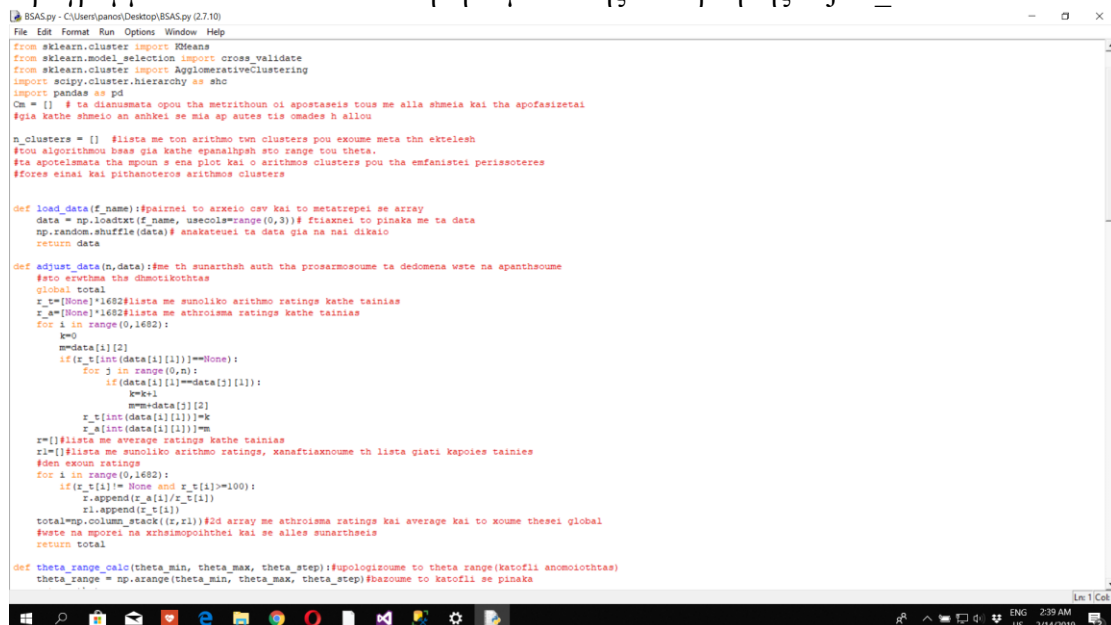
Αφού, αποσυμπιέσαμε το σχετικό αρχείο, ακολουθήσαμε προσεκτικά τις αναλυτικές οδηγίες που βρίσκονταν στο αρχείο README και κατανοήσαμε όσο καλύτερα μπορούσαμε τους ρόλους κάθε αρχείου και τα δεδομένα που μας παρείχαν.

2) Εφαρμόστε το βασικό σχήμα ακολουθιακής ομαδοποίησης για να εκτιμήσετε το πλήθος των ομάδων των χρηστών, ως προς τις προτιμήσεις τους.

Αρχικά τα δεδομένα που χρησιμοποιήσαμε για το συγκεκριμένο ερώτημα ήταν τα δεδομένα του αρχείου u.data.

Για την καλύτερη αποτύπωση των δεδομένων και την εξαγωγή συμπερασμάτων ως προς τις προτιμήσεις των χρηστών που ζητάει το συγκεκριμένο ερώτημα λήφθηκαν τα δεδομένα συνολικά “ratings” και M.O. της κάθε ταινίας. Θεωρήσαμε σωστό να πάρουμε αυτά τα δεδομένα για την αποτύπωση των προτιμήσεων των χρηστών, καθώς μέσω των συγκεκριμένων δεδομένων γίνεται μια δικαιότερη και σωστότερη κατανομή των ταινιών ,με απώτερο σκοπό να μην ομαδοποιηθούν μαζί ταινίες που έχουν μεγάλη διαφορά στο συνολικό αριθμό των “ratings”.

Προγραμματιστικά αυτό υλοποιήθηκε μέσω της συνάρτησης `adjust_data`.



```
from sklearn.cluster import KMeans
from sklearn.model_selection import cross_validate
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as shc
import pandas as pd

n_clusters = 1 # ta diastemata opou tha metritoum oi apotaseis tous me alla shmeia kai tha apofasizetai
Cn = 1 # ta diastemata an anhelei se mia ap autas tis omades h allou

n_clusters = 1 # lista me ton arithmo twon clusters pou exoume meta thn ektelesh
#ton algorithmo hwar gia kathe epantlaph sto range tou theta.
#ta apotelesmata tha mpoun s ena plot kai o arithmos clusters pou tha emfanistei perissoteres
#fores einai kai pithanoteros arithmos clusters

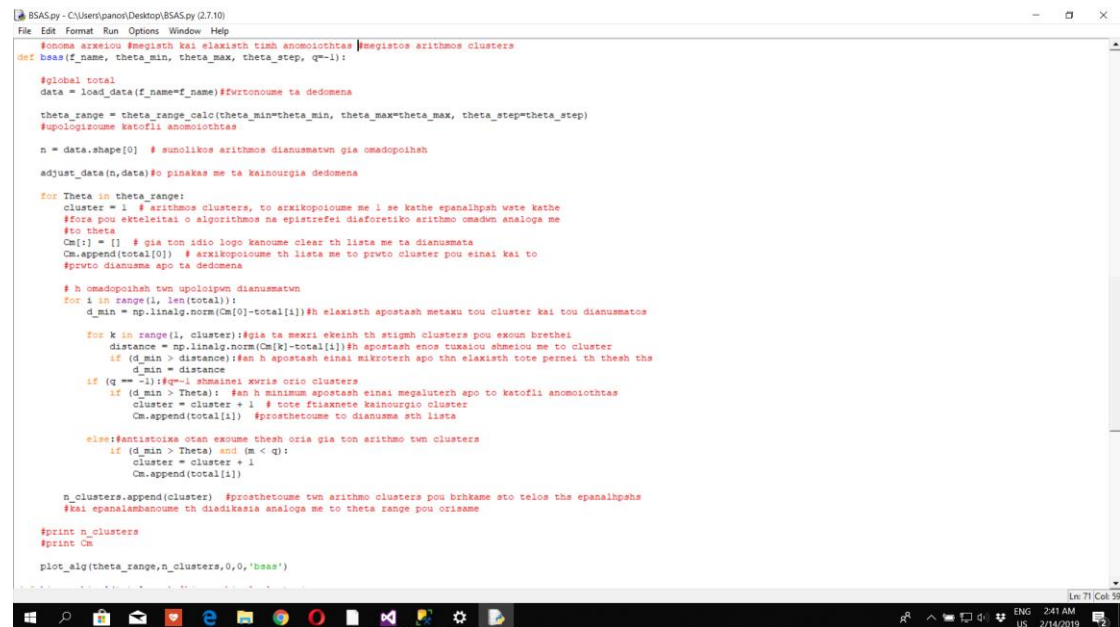
def load_data(f_name):#pairnei to arxio csv kai to metatizei se array
    data = np.loadtxt(f_name, usecols=range(0,3))# ftiaknei to pinaka me ta data
    np.random.shuffle(data)# anakateuei ta data gia na nai dikaiio
    return data

def adjust_data(n,data):#me th sunarthsh auth tha prosarxosume ta dedomena wste na apantusoume
    #sto erwthma tha dhmotikothas
    global total
    r_t=[None]*1682#lista me sunoliko arithmo ratings kathe tainias
    r_s=[None]*1682#lista me athroisma ratings kathe tainias
    for i in range(0,1682):
        k=0
        m=data[i][2]
        if r_t[int(data[i][1])]==None:
            for j in range(0,n):
                if(data[i][1]==data[j][1]):
                    k=k+1
                    m=data[j][2]
                    r_t[int(data[i][1])]=k
                    r_s[int(data[i][1])]=m
    r=[[]]*1682#lista me sunoliko arithmo ratings, xanafiaknουμε th lista giati kapoies tainies
    #den exoun ratings
    for i in range(0,1682):
        if(r_t[i]!=None and r_s[i]>=100):
            r.append(r_s[i]/r_t[i])
            r_s.append(r_s[i])
    total=np.column_stack((r,r_s))#2d array me athroisma ratings kai average kai to xoume thesei global
    #wste na mporei na xrhsimopoihthei kai se alles sunarthseis
    return total

def theta_range_calc(theta_min, theta_max, theta_step):#upologizουμε to theta range(katofli anomolothtas)
    theta_range = np.arange(theta_min, theta_max, theta_step)#bazoume to katofli se pinaka
```

Αρχικά, ορίζουμε το κατώφλι ανομοιότητας θ , θ_{max} , θ_{min} , θ_{step} και τον μέγιστο αριθμό ομάδων q , όπου το q μας δίνει τον μέγιστο αριθμό ομάδων που επιτρέπεται να δημιουργηθούν ενώ σε περίπτωση που είναι -1 σημαίνει ότι δεν θέτουμε κάποιο όριο στον μέγιστο αριθμό ομάδων. Βασική ιδέα του αλγορίθμου ακολουθιακής

ομαδοποίησης (BSAS) είναι ότι ουσιαστικά κάθε νέο διάνυσμα που εξετάζεται από τον αλγόριθμο καταχωρείται είτε σε μία ομάδα που υπάρχει ήδη είτε σε μία νέα ομάδα που θα δημιουργηθεί, ανάλογα με την απόσταση του από τις υπάρχουσες ομάδες. Ο προγραμματιστικός αλγόριθμος που υλοποιήσαμε ακολουθεί την παραπάνω διαδικασία που περιγράφηκε παραπάνω αλλά στον κώδικα του αλγορίθμου υπάρχουν αναλυτικά και επεξηγηματικά σχόλια.



```
BSAS.py - C:\Users\panos\Desktop\BSAS.py (2.7.10)
File Edit Format Run Options Window Help

#ονομα αρχείου #εγκρίστη και ελαχιστη τιμή ανομοιοτητας #εγκρίτος αριθμός clusters
def bsas(f_name, theta_min, theta_max, theta_step, q=1):

    #global total
    data = load_data(f_name=f_name)#φωτονομία τα δεδομένα

    theta_range = theta_range_calc(theta_min=theta_min, theta_max=theta_max, theta_step=theta_step)
    #απολογισμός κατόφλι ανομοιοτητας

    n = data.shape[0] # συνολικός αριθμός διανυσμάτων για ομαδοποίηση
    adjust_data(n,data)#ο πίνακας με τα κείνουργια δεδομένα

    for Theta in theta_range:
        cluster = 1 # αριθμός clusters, το αρχικό ποίση με 1 σε κάθε επανάληψη ώστε κάθε
        #φορά που εκτελείται ο αλγόριθμος να επιστρέφει διαφορετικό αριθμό ομάδων ανάλογα με
        #το theta
        Cm = [] # για τον ίδιο λόγο κάνουμε clear τη lista με τα διανύσματα
        Cm.append(total[0]) # αρχικό ποίση τη lista με το πρώτο cluster που είναι και το
        #πρώτο διάνυσμα από τα δεδομένα

        # h ομαδοποιήση των υπολοίπων διανυσμάτων
        for i in range(1, len(total)):
            d_min = np.linalg.norm(Cm[0]-total[i])#h ελαχιστη απόσταση μεταξύ του cluster και του διανύσματος

            for k in range(1, cluster):#για τα μεξκι εκείνη τη στιγμή clusters που έχουν brethei
                distance = np.linalg.norm(Cm[k]-total[i])#h απόσταση ενός τυχαίου σημείου με το cluster
                if (d_min > distance):#αν h απόσταση είναι μικρότερη από την ελαχιστη τότε περνάει τη thesh the
                    d_min = distance

            if (q == -1):#q=-1 σημαίνει κλείνει οκτιο clusters
                if (d_min > Theta): # αν h minimum απόσταση είναι μεγαλύτερη από το κατόφλι ανομοιοτητας
                    cluster = cluster + 1 # τότε φτιάχνετε κείνουργιο cluster
                    Cm.append(total[i]) #προσθέτουμε το διάνυσμα στη lista

            else:#αντιστοίχια όταν έχουμε thesh οκτιο για τον αριθμό των clusters
                if (d_min > Theta) and (m < q):
                    cluster = cluster + 1
                    Cm.append(total[i])

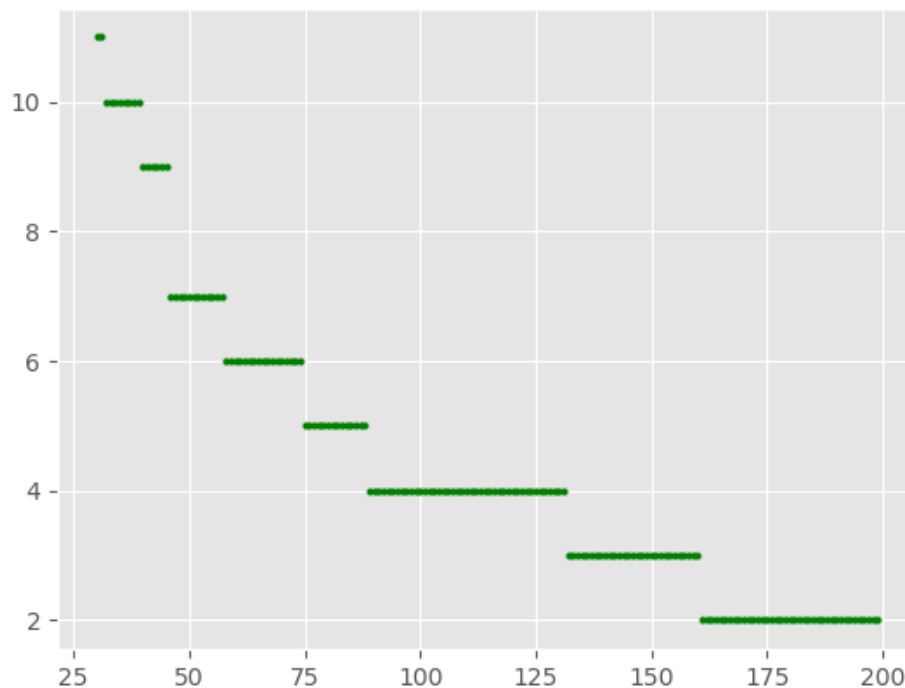
        n_clusters.append(cluster) #προσθέτουμε τον αριθμό clusters που βρήκαμε στο τέλος της επανάληψης
        #και επαναλαμβάνουμε τη διαδικασία ανάλογα με το theta range που ορίσαμε

    #print n_clusters
    #print Cm

    plot_alg(theta_range, n_clusters, 0, 0, 'bsas')

Lev 71 Col 59
```

Τα δεδομένα που προκύπτουν από τον παραπάνω αλγόριθμο είναι ένας πίνακας που αφορά τον αριθμό των ομάδων (clusters) που δημιουργήθηκαν. Πιο αναλυτικά, δημιουργείται μια επαναληπτική διαδικασία με βάση το κατώφλι ανομοιοτητας που έχει δοθεί και δημιουργείται ένα γράφημα ο οποίος αποτυπώνει τον αριθμό των ομάδων (clusters) που προέκυψαν. Ο αριθμός των ομάδων που εμφανίζεται τις περισσότερες φορές, δηλαδή η μεγαλύτερη οριζόντια ευθεία που εμφανίζεται στο γράφημα είναι και κατά πάσα πιθανότητα ο καλύτερος αριθμός ομάδων που θα πρέπει να χρησιμοποιήσουμε αργότερα στους επόμενους αλγορίθμους.



Αξίζει να σημειωθεί πως για την αποφυγή λαθών έχει χρησιμοποιηθεί αντίστοιχη συνάρτηση για την εύρεση του αριθμού αυτού μέσα από τα δεδομένα που προέκυψαν από το γράφημα. Η `cal_nclusters()`.

```

BSAS.py - C:\Users\pano\Desktop\BSAS.py (2.7.10)
File Edit Format Run Options Window Help

#print n_clusters
#print Cm

plot_alg(theta_range,n_clusters,0,0,'bsas')

def hierarchical(total,num):#hierarchical clustering
#επειρωσθ pou de thelete na apothikevtei to grafhma ston antistoixo fakelo
#sto desktop allante ta patha sto savefig
Y=total
plt.figure(figsize=(10, 7))
dend = shc.dendrogram(shc.linkage(Y, method='ward'))
plt.savefig('plots/hierarchical dendrogram ward')
plt.close()
cluster = AgglomerativeClustering(n_clusters=num, affinity='euclidean', linkage='ward')
cluster.fit_predict(Y)
plt.figure(figsize=(10, 7))
plt.scatter(Y[:,0], Y[:,1], c=cluster.labels_, cmap='rainbow')
plt.savefig('plots/hierarchical ward')
plt.close()

def cal_nclusters():#fferei ton arithmo twm clusters, bash tou parapanw theorhmatos oti did o arithmos
#εε th megalutrh suxnothta epilegetai
max=0
max_num=n_clusters[0]
for x in n_clusters:
    i=0
    for y in n_clusters:
        if(x==y):
            i=i+1
        if(i>max):
            max=i
            max_num=x
    return max_num

def kmeans(total,ideal_n_clusters):#algorithmos kmeans
global Y

Y=total
clf = KMeans(n_clusters=ideal_n_clusters)
clf.fit(Y)
centroids = clf.cluster_centers_
labels=clf.labels_

plot_alg(total,centroids,labels,1,'kmeans')

bsas('u.data',30,200,1,-1)
num=cal_nclusters()

```

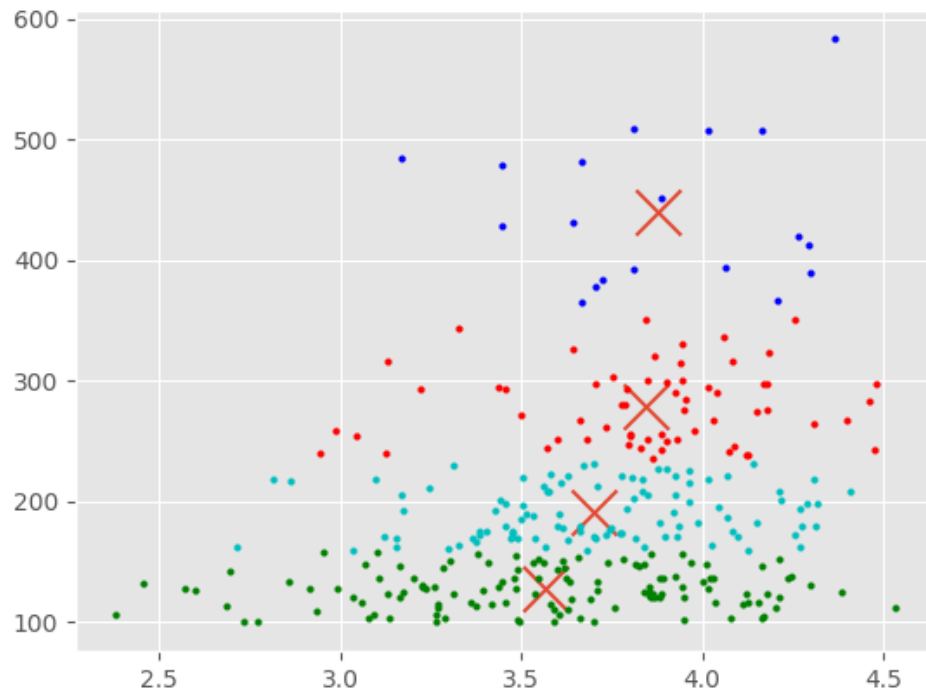
3) Με βάση την εκτίμηση του Βήματος 2, εφαρμόστε τον αλγόριθμο k-means και τον αλγόριθμο ιεραρχικής ομαδοποίησης. Θα χρειαστεί να αποφασίσετε για τις λεπτομέρειες των αλγορίθμων και να τεκμηριώσετε τις υποθέσεις σας. Έχουν κάποια αξία/σημασία οι ομάδες που προέκυψαν;

Υποθέσεις: Χρησιμοποιήσαμε τα ίδια δεδομένα που χρησιμοποιήθηκαν για να αποτυπώσουμε τις προτιμήσεις των χρηστών και επιπλέον από το προηγούμενο ερώτημα βρήκαμε ότι ο ιδανικότερος αριθμός ομάδων που θα πρέπει να δημιουργηθούν είναι τέσσερις.

Λεπτομέρειες για την υλοποίηση των ζητούμενων αλγορίθμων: Ο αλγόριθμος K-means, απαιτεί για την υλοποίηση του να δοθεί ο αριθμός των ομάδων που θα πρέπει να δημιουργηθούν. Ενώ, για την ιεραρχική ομαδοποίηση ο αριθμός των ομάδων (clusters) προκύπτει έμμεσα από τα στιγμιότυπα ομαδοποίησης, τα οποία δίνονται με μορφή δενδρογράμματος. Στα δενδρογράμματα αυτά, επιλέγεται ένα επίπεδο στο οποίο και θα κλαδευτούν, το σημείο που θα κλαδευτεί κάποιο δενδρόγραμμα δείχνει τον αριθμό των ομάδων που θα προκύψουν καθώς και τα σημεία που περιέχει η κάθε ομάδα. Επειδή, όμως η επιλογή του συγκεκριμένου σημείου είναι τελείως υποκειμενική και ουσιαστικά δεν υπάρχει ο σαφής ορισμός του, επιλέγουμε εκ των προτέρων ότι ο σωστός αριθμός των ομάδων θα είναι το τέσσερα.

K-Means

Ο αλγόριθμος εκτελεί επαναληπτικά δύο βήματα. Το πρώτο βήμα αφορά την ανάθεση σε κάποια συστάδα, ενώ το δεύτερο βήμα αφορά τον επαναπροσδιορισμό και τη μετατόπιση του κεντροειδούς κάθε συστάδας. Πιο αναλυτικά, όσον αφορά στο πρώτο βήμα, δηλαδή την ανάθεση σε κάποια ομάδα, ο αλγόριθμος εξετάζει κάθε δείγμα σε σχέση με τα κεντροειδή των ομάδων. Με χρήση κάποιου μέτρου απόστασης, αναθέτει το εξεταζόμενο δείγμα στη συστάδα, της οποίας το κεντροειδές είναι το πλησιέστερο ως προς το συγκεκριμένο δείγμα. Στο δεύτερο βήμα, παίρνοντας τον μέσο όρο των δειγμάτων κάθε ομάδας, επανυπολογίζονται τα κεντροειδή της κάθε συστάδας, ώστε το κεντροειδές να είναι πιο αντιπροσωπευτικό στην πρόσφατα διαμορφωμένη συστάδα. Ο αλγόριθμος εκτελεί επαναληπτικά αυτά τα δύο βήματα, μέχρις ότου τα κεντροειδή των συστάδων να μετατοπίζονται ελάχιστα και σε απόσταση μικρότερη από κάποια δοθείσα τιμή κατωφλίου.



Η διαδικασία αυτή επιτυγχάνεται μέσω των βιβλιοθηκών που προσφέρονται από τη `pyhton` και πιο συγκεκριμένα μέσω του `sklearn`. Η προγραμματιστική υλοποίηση φαίνεται αναλυτικότερα στο κώδικα. Η συνάρτηση που υλοποιεί και αποθηκεύει σε αντίστοιχο γράφημα την ομαδοποίηση (μέσω `kmeans`) ονομάζεται `kmeans` αντίστοιχα.

```

BSASpy - C:\Users\panor\Desktop\BSAS.py (2.7.10)
File Edit Format Run Options Window Help

#print n_clusters
#print CM

plot_alg(theta_range,n_clusters,0,0,'bsas')

def hierarchical(total,num):#hierarchical clustering
    #εεε περιπρωσ pou de thelete na apothikeutei to grafhma ston antistoiko fakelo
    #sto desktop allaste ta patha sto savefig
    Y=total

    plt.figure(figsize=(10, 7))
    dend = sns.dendrogram(sns.linkage(Y, method='ward'))
    plt.savefig('plots/hierarchical dendrogram ward')
    plt.close()
    cluster = AgglomerativeClustering(n_clusters=num, affinity='euclidean', linkage='ward')
    cluster.fit_predict(Y)
    plt.figure(figsize=(10, 7))
    plt.scatter(Y[:,0], Y[:,1], c=cluster.labels_, cmap='rainbow')
    plt.savefig('plots/hierarchical ward')
    plt.close()

def cal_nclusters():#ffernai ton arithmo twm clusters, bash tou parapaw theorhmatos oti did o arithmos
    #me th megalutrh suxnothta epilegetai
    max=0
    max_num=n_clusters[0]
    for k in n_clusters:
        i=0
        for y in n_clusters:
            if (k==y):
                i=i+1
            if (i>max):
                max=i
                max_num=k
    return max_num

def kmeans(total,ideal_n_clusters):#algorithmos kmeans
    global Y

    Y=total
    clf = KMeans(n_clusters=ideal_n_clusters)
    clf.fit(Y)
    centroids = clf.cluster_centers_
    labels=clf.labels_

    plot_alg(total,centroids,labels,1,'kmeans')

bsas('a.data',30,200,1,-1)
num=cal_nclusters()

```

Ιεραρχική ομαδοποίηση

Οι ιεραρχικές τεχνικές χωρίζονται στις Συσσωρευτικές και Διαιρετικές. Οι συσσωρευτικές τεχνικές αρχικά θεωρούν ότι κάθε σημείο είναι από μόνο του μια ξεχωριστή ομάδα και προχωρούν σε συγχωνεύσεις αυτών, μέχρι όλα τα σημεία να τοποθετηθούν σε μία (bottom-up). Στην γενική τους μορφή οι συσσωρευτικές τεχνικές λειτουργούν ως εξής:

1) Αρχικά θεωρούν κάθε σημείο σαν μία ομάδα. Αν δηλαδή υπάρχει ένα σύνολο από N σημεία τα οποία πρέπει να ομαδοποιηθούν, τότε αρχικά υπάρχουν N ομάδες, όπου κάθε ομάδα θα περιέχει ένα σημείο από τα N . Μετρούνται οι μεταξύ τους αποστάσεις.

2) Βρίσκεται το πιο κοντινό ζευγάρι ομάδων. Το ζευγάρι αυτό συγχωνεύεται σε ένα. Πλέον υπάρχει μία ομάδα λιγότερη.

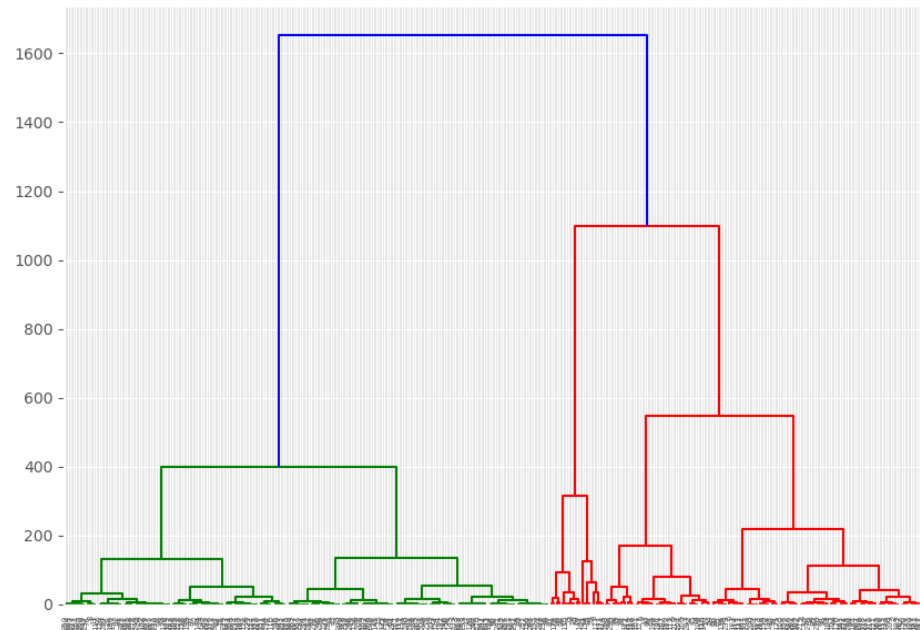
3) Υπολογίζονται εκ νέου οι αποστάσεις των ομάδων μεταξύ τους.

4) Επαναλαμβάνονται τα βήματα 2 και 3 έως ότου και τα N σημεία τοποθετηθούν σε μία και μοναδική ομάδα.

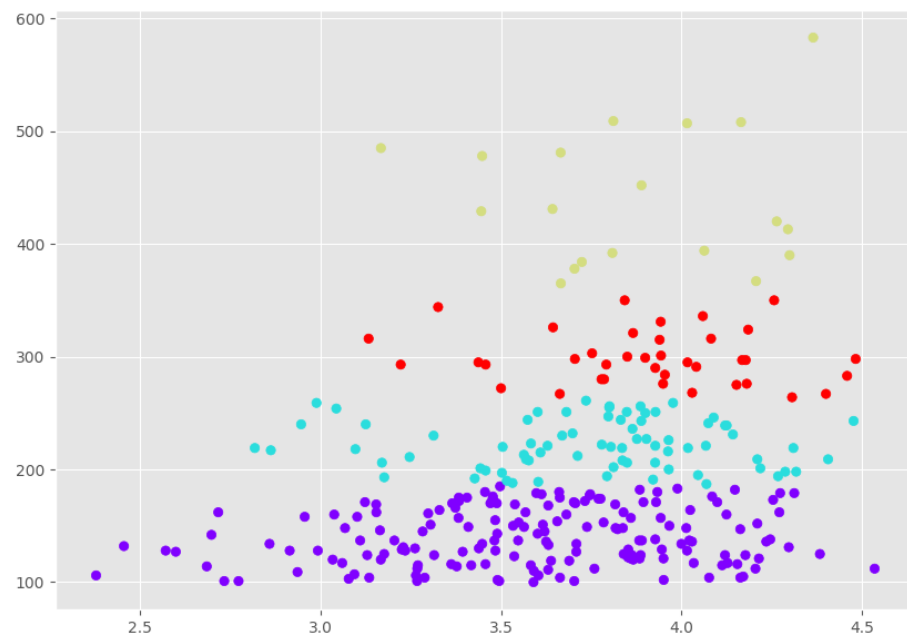
5) Τέλος σχεδιάζεται το αντίστοιχο δενδρόγραμμα και επιλέγεται σε ποιο σημείο θα κλαδευτεί.

Το βήμα 3 μπορεί να πραγματοποιηθεί με διαφορετικούς τρόπους: Στην ιεραρχική ομαδοποίηση απλού συνδέσμου (single-linkage), θεωρείται ως απόσταση μεταξύ δύο ομάδων η μικρότερη απόσταση μεταξύ όλων των ζευγών των προτύπων με στοιχεία και από τις δύο ομάδες. Στην ιεραρχική ομαδοποίηση ολοκληρωμένου συνδέσμου (complete-linkage), θεωρείται ως απόσταση μεταξύ δύο ομάδων η μεγαλύτερη απόσταση μεταξύ όλων των ζευγών των προτύπων με στοιχεία και από τις δύο ομάδες. Στην ιεραρχική ομαδοποίηση μέσου συνδέσμου (average-linkage), θεωρείται ως απόσταση μεταξύ δύο ομάδων η μέση απόσταση μεταξύ όλων των ζευγών των προτύπων με στοιχεία και από τις δύο ομάδες. Στην ιεραρχική ομαδοποίηση με την μέθοδο του Ward's (ward-linkage), η ομοιότητα δύο ομάδων βασίζεται στην αύξηση του τετραγωνικού σφάλματος όταν ενώνονται οι δύο ομάδες, η οποία θα χρησιμοποιηθεί και για την επίλυση του συγκεκριμένου ερωτήματος.

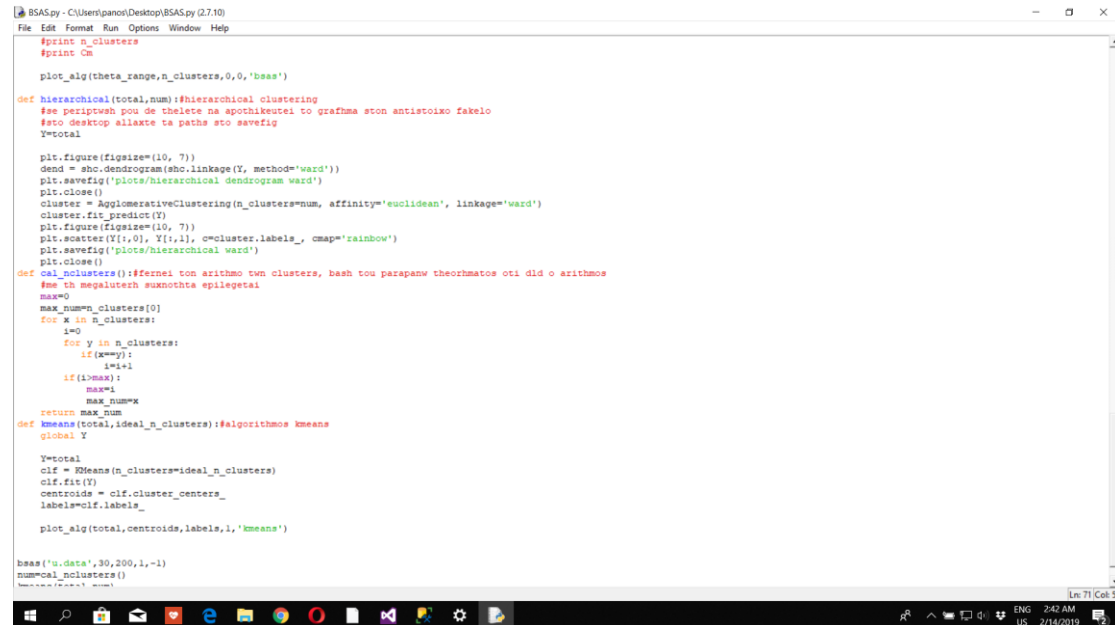
Ιεραρχικό δενδρόγραμμα “Ward”



“Ward” ιεραρχική ομαδοποίηση



Προγραμματιστικά ο συγκεκριμένος αλγόριθμος υλοποιήθηκε με τις έτοιμες συναρτήσεις της python που παρέχονται από την βιβλιοθήκη sklearn και πιο συγκεκριμένα με την έτοιμη συνάρτηση AgglomerativeClustering η οποία θα έχει σαν ορίσματα για τις απαιτούμενες μεταβλητές τα εξής: n_clusters=4, affinity='euclidean' και linkage='ward'.



```
BSAS.py - C:\Users\panos\Desktop\BSAS.py (2.7.10)
File Edit Format Run Options Window Help

#print n_clusters
#print Cn

plot_alg(theta_range,n_clusters,0,0,'bass')

def hierarchical(total,num):#hierarchical clustering
#επειρωσθ pou de thelete na apothikeutei to grafma ston antistoiko fakelo
#ετο desktop allaxte ta paths sto savefig
Y=total

plt.figure(figsize=(10, 7))
dend = sns.dendrogram(sko.linkage(Y, method='ward'))
plt.savefig('plots/hierarchical dendrogram ward')
plt.close()
cluster = AgglomerativeClustering(n_clusters=num, affinity='euclidean', linkage='ward')
cluster.fit_predict(Y)
plt.figure(figsize=(10, 7))
plt.scatter(Y[:,0], Y[:,1], c=cluster.labels_, cmap='rainbow')
plt.savefig('plots/hierarchical ward')
plt.close()

def cal_nclusters():#fernei ton arithmo twv clusters, bash tou parapaw theorhmatos oti did o arithmos
#ne th megalutew suxnothta epilegetai
max=0
max_num=n_clusters[0]
for k in n_clusters:
    for y in n_clusters:
        if (k==y):
            i=i+1
        if (i>max):
            max=i
    max_num=k
return max_num

def kmeans(total,ideal_n_clusters):#algorithmos kmeans
global Y

Y=total
clf = KMeans(n_clusters=ideal_n_clusters)
clf.fit(Y)
centroids = clf.cluster_centers_
labels=clf.labels_

plot_alg(total,centroids,labels,1,'kmeans')

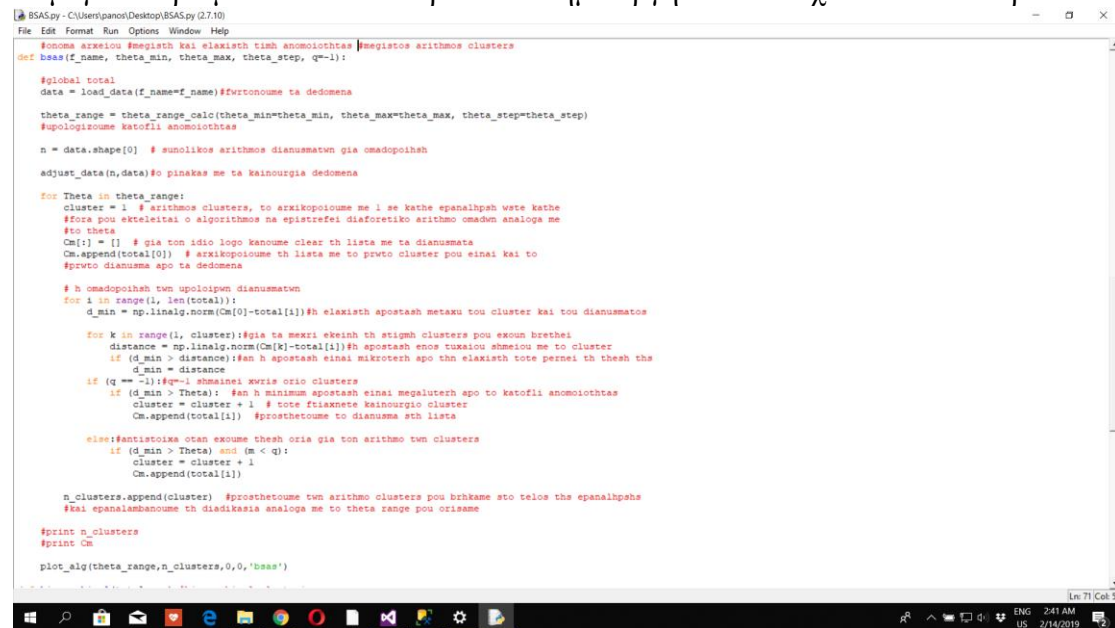
bass(['a.data',30,200,1,-1])
num=cal_nclusters()
```

Συμπεράσματα από τις ομάδες που προέκυψαν

Από τα γραφήματα παρατηρούμε πως προκύπτουν 4 ομάδες και βλέπουμε πως ανάλογα με την αύξηση του πλήθους των ratings της κάθε παραγόμενης ομάδας υπάρχει και μία αντίστοιχη αύξηση στο Μέσο όρο των μέσο όρων των ταινιών που βρίσκονται στην ομάδα. Δηλαδή, με άλλα λόγια οι ταινίες που έχουν βαθμολογηθεί περισσότερες φορές τείνουν να έχουν καλύτερο και υψηλότερο Μ.Ο βαθμολογιών. Συμπερασματικά, μπορούμε να αντιληφθούμε πως στους χρήστες μελλοντικά επρόκειτο να προταθούν ταινίες που έχουν πολλές διαφορετικές βαθμολογήσεις και ουσιαστικά υψηλότερη βαθμολογία και άρα καλύτερες κριτικές.

ΛΙΓΑ ΛΟΓΙΑ ΓΙΑ ΤΟΝ ΚΩΔΙΚΑ

Ο κώδικας καλεί αρχικά τη συνάρτηση `bsas` η οποία παίρνει ορίσματα το `file name(f_name)` όπου βρίσκονται τα δεδομένα, στη δικιά μας περίπτωση το `u.data`. Επίσης τα `theta_min` `theta_max` και `theta_step(theta_min,theta_max,theta_step)` τα οποία ορίσαμε με 30,200 και 1 αντίστοιχα κατόπιν δοκιμών ώστε να φέρνει το καλύτερο δυνατό αποτέλεσμα καθώς πολύ μικρό ή πολύ μεγάλο εύρος τιμών θα έφερναν λάθος αποτελέσματα. Ακόμη στη συνάρτησή μας πρέπει να προσθέσουμε το `q` όπου δηλώνει το μέγιστο αριθμό `clusters` που πρέπει να δημιουργηθούν και έχει αναλυθεί παραπάνω.



```
BSAS.py - C:\Users\panos\Desktop\BSAS.py (2.7.10)
File Edit Format Run Options Window Help
# Ονομα αρχείου #επιλογή και ελαχιστή τιμή ανομοιοτητας #επιλογή αριθμού clusters
def bsas(f_name, theta_min, theta_max, theta_step, q=1):

    #global total
    data = load_data(f_name=f_name)#φωτονομή τα δεδομένα
    theta_range = theta_range_calc(theta_min=theta_min, theta_max=theta_max, theta_step=theta_step)
    #επιλογή και ελαχιστή τιμή ανομοιοτητας
    n = data.shape[0] # συνολικός αριθμός διανυσμάτων για ομαδοποιήση
    adjust_data(n,data)#ο πίνακας με τα κινούμενα δεδομένα

    for Theta in theta_range:
        cluster = 1 # αριθμός clusters, το ερμηνεύουμε με 1 σε κάθε επανάληψη ώστε κάθε
        #force που εκτελείται ο algorithm να επιστρέφει διαφορετικό αριθμό ομαδών ανάλογα με
        #το theta
        Cm[:] = [] # για τον ίδιο λόγο κάνουμε clear τη λίστα με τα διανύσματα
        Cm.append(total[0]) # αρχικοποιούμε τη λίστα με το πρώτο cluster που είναι και το
        #πρώτο διάνυσμα από τα δεδομένα

        # h ομαδοποιήση των υπολοίπων διανυσμάτων
        for i in range(1, len(total)):
            d_min = np.linalg.norm(Cm[0]-total[i])#h ελαχιστή απόσταση μεταξύ του cluster και του διανυσματος

            for k in range(1, cluster):#για τα μενι κείνη τη στιγμή clusters που έχουν βρεθεί
                distance = np.linalg.norm(Cm[k]-total[i])#h απόσταση ενός τυχαίου σημείου με το cluster
                if (d_min > distance):#αν h απόσταση είναι μικρότερη από την ελαχιστή τότε περνεί τη θέση της
                    d_min = distance

            if (q == -1):#q=-1 σημαίνει χωρίς όριο clusters
                if (d_min > Theta): #αν h minimum απόσταση είναι μεγαλύτερη από το κατώφλι ανομοιοτητας
                    cluster = cluster + 1 # τότε φτιάχνετε καινούριο cluster
                    Cm.append(total[i]) #προσθέτουμε το διάνυσμα στη λίστα
            else:#αντιστοίχα όταν έχουμε θέση ορια για τον αριθμό των clusters
                if (d_min > Theta) and (m < q):
                    cluster = cluster + 1
                    Cm.append(total[i])

        n_clusters.append(cluster) #προσθέτουμε τον αριθμό clusters που βρήκαμε στο τέλος της επανάληψης
        #h επανάληψη συνεχίζεται με τη διαδικασία ανάλογα με το theta range που ορίσαμε

    #print n_clusters
    #print Cm
    plot_alg(theta_range,n_clusters,0,0,'bsas')
```

Μέσα στη συνάρτηση καλείται η `load_data` η οποία με τα ορίσματα που αφορούν τα `paths` δημιουργεί τον πίνακα με τις τιμές και κάνει `shuffle` τα δεδομένα για να ναι πιο δίκαιο για το κώδικα.

Μετά καλείται η `theta_range_calc` η οποία φέρνει σε λίστα όλο το εύρος των τιμών μεταξύ των `theta_min` `theta_max` με το αντίστοιχο βήμα.

Κατόπιν καλείται η `adjust_data` η οποία διαμορφώνει τα δεδομένα όπως αναφέραμε και παραπάνω. Τα διαμορφωμένα δεδομένα αποθηκεύονται σε μια λίστα `total` η οποία είναι `global` ώστε να μπορεί να χρησιμοποιηθεί από όλες τις συναρτήσεις.

```
BSAS.py - C:\Users\panor\Desktop\BSAS.py (2.7.10)
File Edit Format Run Options Window Help

Cn = [] # ta dianomata opou ta metritoun oi apostaseis tous me alla shmeia kai ta apofasizetai
#gia kathe shmeio an ankei se mia ap autes tis omades h allou

n_clusters = [] #lista me ton arithmo twm clusters pou exoume meta thn ektelesh
#tou algorithmou bsas gia kathe spanalghsh sto range tou theta.
#ta apotelamata tha mporei n ena plot kai o arithmos clusters pou ta emfanizei perissoteres
#fores einai kai pithanoteros arithmos clusters

def load_data(f_name):#pairnei to arxio csv kai to metatirei se array
    data = np.loadtxt(f_name, usecols=range(0,3))# ftiaxeim to pinaka me ta data
    np.random.shuffle(data)# anakateuei ta data gia na nai dikaio
    return data

def adjust_data(n,data):#me th sunarthsh auth tha prosarxousme ta dedomena wste na apantousme
#sto erwthma ths dhmotikothtas
    global total
    r_t=[None]*1682#lista me sunoliko arithmo ratings kathe tainias
    r_a=[None]*1682#lista me athroisma ratings kathe tainias
    for i in range(0,1682):
        k=0
        m=data[i][2]
        if(r_t[int(data[i][1])]==None):
            for j in range(0,n):
                if(data[i][1]==data[j][1]):
                    k=k+1
                    m=data[j][2]
                    r_t[int(data[i][1])]=m
                    r_a[int(data[i][1])]=m
            r=[None]*1682#lista me average ratings kathe tainias
            r1=[None]*1682#lista me sunoliko arithmo ratings, xanastixanome th lista giati kapoies tainias
            #den exoun ratings
            for i in range(0,1682):
                if(r_t[i]!=None and r_t[i]>=100):
                    r.append(r_t[i]/r_a[i])
                    r1.append(r_t[i])
            total=np.column_stack((r,r1))#2d array me athroisma ratings kai average kai to xoume thesei global
            #wste na mporei na xrhsimopoihthei kai se alies sunarthseis
            return total

def theta_range_calc(theta_min, theta_max, theta_step):#upologizome to theta range(katofii anomiothtas)
    theta_range = np.arange(theta_min, theta_max, theta_step)#bazoume to katofii se pinaka
    return theta_range

def plot_alg(x,y,labels,centroids_exists,file_plot):#h sunarthsh gia thn emfanish tou plot
    #en theloume na deikoume centroids tote bazoume 1, diaporetika 0
    colors=10*["g","r","c","b","k"]
    if(centroids_exists==0):
        for i in range(len(y)):
            plt.plot(x[i], y[i], colors[i], markersize=4)
        else:
            for i in range(len(x)):
                plt.plot(x[i][0], x[i][1], colors[i], markersize=4)
            plt.scatter(y[:,0],y[:,1], marker='x', s=350,linewidths=5)
    plt.savefig('plots/'+file_plot)
    plt.close()
    #plt.show()

    #onoma arxeiou #megisth kai elaxisth timh anomiothtas #megistos arithmos clusters
def bsas(f_name, theta_min, theta_max, theta_step, q=1):
    #global total
    data = load_data(f_name=f_name)#fwtizonome ta dedomena
    theta_range = theta_range_calc(theta_min=theta_min, theta_max=theta_max, theta_step=theta_step)
```

Υστέρα ακολουθεί ο αλγόριθμος βασικής ακολουθιακής ομαδοποίησης όπως περιγράφηκε περιληπτικά παραπάνω. Η αναλυτικότερη περιγραφή βρίσκεται μέσω σχολίων στον κώδικα. Ο αριθμός των clusters που προέκυψε μετά από κάθε επανάληψη κρατιέται σε μια λίστα την `n_clusters`.

Ενώ τέλος καλείται η `plot_alg` για την αποθήκευση των δεδομένων με μορφή γραφήματος. Τα γραφήματα αυτά αποθηκεύονται μέσα σε έναν φάκελο `plot` που στείλαμε μαζί με την εργασία για ευκολότερη χρήση των γραφημάτων(σε περίπτωση που θέλει κάποιος να αλλάξει τη τοποθεσία των γραφημάτων πρέπει να ορίσει ολόκληρο το `path` μέσα στο `savefig` ή αφαιρώντας το, τα γραφήματα θα εμφανιστούν στο φάκελο που είναι και το εκτελέσιμο).

```
BSAS.py - C:\Users\panor\Desktop\BSAS.py (2.7.10)
File Edit Format Run Options Window Help

k=0
m=data[i][2]
if(r_t[int(data[i][1])]==None):
    for j in range(0,n):
        if(data[i][1]==data[j][1]):
            k=k+1
            m=data[j][2]
            r_t[int(data[i][1])]=m
            r_a[int(data[i][1])]=m
    r=[None]*1682#lista me average ratings kathe tainias
    r1=[None]*1682#lista me sunoliko arithmo ratings, xanastixanome th lista giati kapoies tainias
    #den exoun ratings
    for i in range(0,1682):
        if(r_t[i]!=None and r_t[i]>=100):
            r.append(r_t[i]/r_a[i])
            r1.append(r_t[i])
    total=np.column_stack((r,r1))#2d array me athroisma ratings kai average kai to xoume thesei global
    #wste na mporei na xrhsimopoihthei kai se alies sunarthseis
    return total

def theta_range_calc(theta_min, theta_max, theta_step):#upologizome to theta range(katofii anomiothtas)
    theta_range = np.arange(theta_min, theta_max, theta_step)#bazoume to katofii se pinaka
    return theta_range

def plot_alg(x,y,labels,centroids_exists,file_plot):#h sunarthsh gia thn emfanish tou plot
    #en theloume na deikoume centroids tote bazoume 1, diaporetika 0
    colors=10*["g","r","c","b","k"]
    if(centroids_exists==0):
        for i in range(len(y)):
            plt.plot(x[i], y[i], colors[i], markersize=4)
        else:
            for i in range(len(x)):
                plt.plot(x[i][0], x[i][1], colors[i], markersize=4)
            plt.scatter(y[:,0],y[:,1], marker='x', s=350,linewidths=5)
    plt.savefig('plots/'+file_plot)
    plt.close()
    #plt.show()

    #onoma arxeiou #megisth kai elaxisth timh anomiothtas #megistos arithmos clusters
def bsas(f_name, theta_min, theta_max, theta_step, q=1):
    #global total
    data = load_data(f_name=f_name)#fwtizonome ta dedomena
    theta_range = theta_range_calc(theta_min=theta_min, theta_max=theta_max, theta_step=theta_step)
```

Αφού τελειώσει ο αλγόριθμος `bsas` καλείται η συνάρτηση `cal_nclusters()` η οποία κάνει `return` τον αριθμό των clusters που θα χρησιμοποιηθεί στους επόμενους αλγορίθμους βάση του θεωρήματος που αναλύσαμε παραπάνω.

```
BSAS.py - C:\Users\panos\Desktop\BSAS.py (2.7.10)
File Edit Format Run Options Window Help

#print n_clusters
#print Cm

plot_alg(theta_range,n_clusters,0,0,'bsas')

def hierarchical(total,num):#hierarchical clustering
#εε περιπτωσ που δε thelete na apothikeutei to grafma ston antistoiko fakelo
#sto desktop allaxte ta paths sto savefig
Y=total

plt.figure(figsize=(10, 7))
dend = shc.dendrogram(shc.linkage(Y, method='ward'))
plt.savefig('plots/hierarchical dendrogram ward')
plt.close()
cluster = AgglomerativeClustering(n_clusters=num, affinity='euclidean', linkage='ward')
cluster.fit_predict(Y)
plt.figure(figsize=(10, 7))
plt.scatter(Y[:,0], Y[:,1], c=cluster.labels_, cmap='rainbow')
plt.savefig('plots/hierarchical ward')
plt.close()

def cal_nclusters():#fernel ton arithmo twm clusters, bash tou parapaw theorhmatos oti did o arithmos
#εε th megalutere suxnothta epilegetai
max=0
max_num=n_clusters[0]
for x in n_clusters:
    i=0
    for y in n_clusters:
        if (x==y):
            i=i+1
        if (i>max):
            max=i
    max_num=x
return max_num

def kmeans(total,ideal_n_clusters):#algorithmos kmeans
global Y

Y=total
clf = KMeans(n_clusters=ideal_n_clusters)
clf.fit(Y)
centroids = clf.cluster_centers_
labels=clf.labels_

plot_alg(total,centroids,labels,1,'kmeans')

bsas('u.data',30,200,1,-1)
num=cal_nclusters()
```

Τέλος καλούνται οι συναρτήσεις kmeans και hierarchical. Οι συναρτήσεις αυτές περιέχουν τους αλγορίθμους με τα αντίστοιχα ονόματα και υλοποιούνται μέσω των βιβλιοθηκών που παρέχει η pythοn και συγκεκριμένα η sklearn. Ενώ μέσω του plot_alg αποθηκεύονται τα γραφήματα τους.

```
BSAS.py - C:\Users\panos\Desktop\BSAS.py (2.7.10)
File Edit Format Run Options Window Help

#print n_clusters
#print Cm

plot_alg(theta_range,n_clusters,0,0,'bsas')

def hierarchical(total,num):#hierarchical clustering
#εε περιπτωσ που δε thelete na apothikeutei to grafma ston antistoiko fakelo
#sto desktop allaxte ta paths sto savefig
Y=total

plt.figure(figsize=(10, 7))
dend = shc.dendrogram(shc.linkage(Y, method='ward'))
plt.savefig('plots/hierarchical dendrogram ward')
plt.close()
cluster = AgglomerativeClustering(n_clusters=num, affinity='euclidean', linkage='ward')
cluster.fit_predict(Y)
plt.figure(figsize=(10, 7))
plt.scatter(Y[:,0], Y[:,1], c=cluster.labels_, cmap='rainbow')
plt.savefig('plots/hierarchical ward')
plt.close()

def cal_nclusters():#fernel ton arithmo twm clusters, bash tou parapaw theorhmatos oti did o arithmos
#εε th megalutere suxnothta epilegetai
max=0
max_num=n_clusters[0]
for x in n_clusters:
    i=0
    for y in n_clusters:
        if (x==y):
            i=i+1
        if (i>max):
            max=i
    max_num=x
return max_num

def kmeans(total,ideal_n_clusters):#algorithmos kmeans
global Y

Y=total
clf = KMeans(n_clusters=ideal_n_clusters)
clf.fit(Y)
centroids = clf.cluster_centers_
labels=clf.labels_

plot_alg(total,centroids,labels,1,'kmeans')

bsas('u.data',30,200,1,-1)
num=cal_nclusters()
```

Οι κλήσεις όλων των συναρτήσεων για να αρχίσει το πρόγραμμα φαίνεται παρακάτω.

```
BSASpy - C:\Users\panos\Desktop\BSAS.py (2.7.10)
File Edit Format Run Options Window Help

plot_alg(theta_range,n_clusters,0,0,'basas')

def hierarchical(total,num):#hierarchical clustering
#εεε περιπτωσh που δε thelete na apothikeutei to grafhma ston antistoixο fakelo
#στο desktop allante ta patha sto savefig
Y=total

plt.figure(figsize=(10, 7))
dend = shc.dendrogram(shc.linkage(Y, method='ward'))
plt.savefig('plots/hierarchical dendrogram ward')
plt.close()
cluster = AgglomerativeClustering(n_clusters=num, affinity='euclidean', linkage='ward')
cluster.fit_predict(Y)
plt.figure(figsize=(10, 7))
plt.scatter(Y[:,0], Y[:,1], c=cluster.labels_, cmap='rainbow')
plt.savefig('plots/hierarchical ward')
plt.close()

def cal_nclusters():#εfεrnel ton arithmo twm clusters, bash tou parapane theorchmatos oti did o arithmos
#εεε th megalutērη suxnothta epilegetai
max=0
max_num=0
clusters=[0]
for x in n_clusters:
    i=0
    for y in n_clusters:
        if(x==y):
            i=i+1
        if(i>max):
            max=i
            max_num=x
    return max_num
def kmeans(total,ideal_n_clusters):#algorithmos kmeans
global Y

Y=total
clf = KMeans(n_clusters=ideal_n_clusters)
clf.fit(Y)
centroids = clf.cluster_centers_
labels=clf.labels_

plot_alg(total,centroids,labels,1,'kmeans')

basas('u.data',30,200,1,-1)
num=cal_nclusters()
kmeans(total,num)
hierarchical(total,num)
```