

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



ΑΝΑΓΝΩΡΙΣΗ ΠΡΟΤΥΠΩΝ

A simple python script about clustering & classification.

Τελική Εργασία

Όνομα φοιτητή – Αρ. Μητρώου

ΚΩΝΣΤΑΝΤΙΝΟΣ-ΣΠΥΡΙΔΩΝ ΜΩΚΟΣ - Π15098

Ημερομηνία παράδοσης

31/01/2018

Πίνακας περιεχομένων

Πίνακας περιεχομένων	2
Βασικές πληροφορίες	3
1ο Ερώτημα: Κατανόηση του Dataset	3
2ο Ερώτημα: Εκτίμηση αριθμού ομάδων (μέσω BSAS)	3
Προ-επεξεργασία του dataset	4
1. Επιλογή χαρακτηριστικών (feature selection)	4
2. Μέτρο εγγύτητας (proximity measure)	5
3. Κριτήριο ομαδοποίησης (clustering criterion)	5
Εκτίμηση αριθμού ομάδων (μέσω BSAS)	6
Basic sequential algorithmic scheme (BSAS)	6
Χρήση του BSAS για την εκτίμηση των ομάδων	7
Ανάλυση αποτελεσμάτων	8
Τελικό αποτέλεσμα	10
3ο Ερώτημα: Αλγόριθμος k-means και ιεραρχικής ομαδοποίησης	11
K-means	11
Παράμετροι του αλγορίθμου	11
Τελικό αποτέλεσμα	12
Ιεραρχικής ομαδοποίησης	13
Παράμετροι του αλγορίθμου	13
Τελικό αποτέλεσμα	13
Εκτέλεση των αλγορίθμων	14
Προβολή αποτελεσμάτων	15
4ο Ερώτημα: Ταξινομητές νευρωνικού δικτύου & ελαχίστων τετραγώνων	16
Νευρωνικό δίκτυο (Multi Layer Perceptron - MLP)	18
Παράμετροι του αλγορίθμου	18
Ταξινομητής ελαχίστων τετραγώνων	19
Παράμετροι του αλγορίθμου	19
Εκπαίδευση των αλγορίθμων	19
Τελικό αποτέλεσμα	20
Παράρτημα (Appendix)	21
Παράδειγμα ορθής εκτέλεσης	21
core.py	21
classifiers.py	22

Βασικές πληροφορίες

Το κύριο θέμα της εργασίας είναι η ανάλυση δεδομένων που αφορούν τις κριτικές ταινιών και μας ζητείται ανάπτυξη ενός προγράμματος σε python ή Matlab. Η υλοποίηση της εργασίας θα γίνει σε python, μια από τις πιο γνωστές γλώσσες λόγω της απλότητας της και της ευκολίας της. Ο κώδικας έχει σχόλια σε κάθε γραμμή του (όπου εφαρμόζεται κάποια διαδικασία, όχι σε δήλωση μεταβλητών κλπ.).

Στην συνέχεια αναφέρονται κάποιες από τις βασικές πληροφορίες του script:

Βασικές πληροφορίες	Basic information
Python version	2.7.10
Recommended use	Terminal (script is terminal based)
Created in	Sublime Text (text editor)
Main libraries in use	sklearn, numpy, math, itertools, random
Use	single use - analyze MovieLens 100K Dataset
Total files	3 (core.py, appendix.py, classifiers.py)
Need to use in the same folder	u.data, u.item and folder: 5-fold
[5-fold] must contains	u1.base,...,u5.base & u1.test,...,u5.test
Dependence of .py	appendix.py runs through core.py classifiers.py is independent
github repository	https://github.com/diogt52/clustering-and-classifiers

1ο Ερώτημα: Κατανόηση του Dataset

Αφού κατανοήσουμε την χρησιμότητα του κάθε αρχείου, όπως μας ζητείται και στο 1ο ερώτημα, προχωράμε στο 2ο ερώτημα το οποίο μας ζητάει να εκτιμήσουμε τον αριθμό των ομάδων των χρηστών βάσει των προτιμήσεων τους κάνοντας χρήση του βασικού σχήματος ακολουθιακής ομαδοποίησης (BSAS).

2ο Ερώτημα: Εκτίμηση αριθμού ομάδων (μέσω BSAS)

Προκειμένου να εκτιμήσουμε τον αριθμό των ομάδων με την χρήση του BSAS, χρειάζεται να βάλουμε τον αλγόριθμο σε μια επαναληπτική διαδικασία, κατά την οποία θα συλλέξουμε πολλές εκτιμήσεις του BSAS, τις οποίες στην συνέχεια θα πρέπει να αναλύσουμε για να πάρουμε το επιθυμητό αποτέλεσμα.

Προ-επεξεργασία του dataset

Πριν την υλοποίηση του αλγορίθμου απαιτείται προ-επεξεργασία του dataset προκειμένου να είναι όσο πιο αποτελεσματικό γίνεται. Τα βασικά βήματα για την οργάνωση των δεδομένων προς ομαδοποίηση είναι τα ακόλουθα:

Βασικά βήματα οργάνωσης *
1. Επιλογή χαρακτηριστικών (feature selection)
2. Μέτρο εγγύτητας (proximity measure)
3. Κριτήριο ομαδοποίησης (clustering criterion)

*Το 4ο βήμα (η επιλογή αλγορίθμου ομαδοποίησης) δεν είναι αναγκαίο εφόσον ήδη γνωρίζουμε τον αλγόριθμο.

1. Επιλογή χαρακτηριστικών (feature selection)

Αρχικά πρέπει να επιλέξουμε ποια από τα δεδομένα του dataset είναι χρήσιμα, και ποια περιττά, προκειμένου να έχουμε όσο το δυνατόν περισσότερη πληροφορία σχετικά με το υπό εξέταση πρόβλημα, χωρίς πλεονασμό πληροφορίας.

Εφόσον το κριτήριο ομαδοποίησης είναι οι προτιμήσεις των χρηστών, αρχικά θα κρατήσουμε από το βασικό αρχείο u.data τα στοιχεία:

u.data (information to keep)	user_id	movie_id	rating
Example	1	100	3

Εν συνέχεια από το αρχείο u.item:

u.item (information to keep)	movie_id	movie_types (19 entries)
Example	100	[0,1,1,0,...,0]

Και τέλος θα γίνει συγχώνευση (merge) των στοιχείων σε μια τελική λίστα, όπου κάθε υπό-λίστα έχει τον μέσο ορο βαθμολόγησης κάθε χρήστη για την αντίστοιχη κατηγορία, και στην αρχική λίστα κάθε entry αντιπροσωπεύει έναν χρήστη (δηλαδή η πρώτη λίστα έχει τα στοιχεία του χρήστη με id: 1):

Final_List	movie_types avg rating (19 entries)
Example	[3.4,2.4,...,1]

Στα τελικά δεδομένα δεν έχουμε κρατήσει διαφορές πληροφορίες του αρχικού dataset, όπως: [zip_code, timestamp, gender, occupation, age], διότι κανένα από αυτά δεν αφορά την προτίμηση του χρήστη. Σαν τελικά δεδομένα κρατήσαμε μόνο τα απαραίτητα δεδομένα προκειμένου να έχουμε την πλήρη εικόνα για την προτίμηση του κάθε χρήστη (βάση του dataset προς ανάλυση) ως προς το κάθε είδος ταινίας.

2. Μέτρο εγγύτητας (proximity measure)

Σαν μέτρο εγγύτητας θα χρησιμοποιήσουμε την ευκλείδεια απόσταση, όπως αυτή αναφέρεται και στο βιβλίο του μαθήματος με την χρήση του κώδικα:

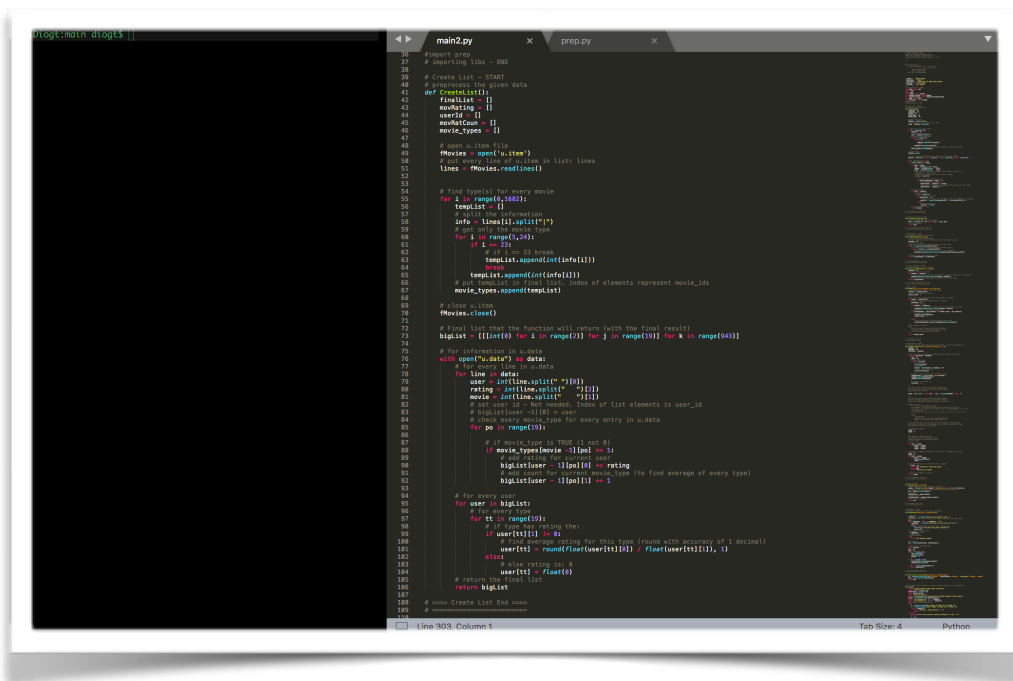
```
def euclidian_distance(a, b):  
    sqrt(sum( (a - b)**2 for a, b in zip(a, b)))
```

Σαν συνάρτηση εγγύτητας χρησιμοποιούμε την συνάρτηση ελαχιστης εγγύτητας, μεταξύ σημείου και συνόλου σημείων (Αναγνώριση προτύπων, σελίδα 523).

3. Κριτήριο ομαδοποίησης (clustering criterion)

Η ομαδοποίηση των χρηστών καλείται να γίνει βάση των προτιμήσεων τους, οπότε το κριτήριο ομαδοποίησης είναι η βαθμολόγηση του κάθε χρήστη ως προς τις κατηγορίες των ταινιών.

Ο κώδικας που χρησιμοποιήθηκε για την επεξεργασία των δεδομένων και την δημιουργία της τελικής λίστας είναι απλή επεξεργασία αρχείων (files) και λιστών (lists) χωρίς την χρήση κάποιου library (πχ pandas).



Εικ.1 - προ-επεξεργασία δεδομένων και δημιουργία τελικής λίστας για τον αλγόριθμο BSAS

Η τελική μας λίστα αποτελείται μόνο από αριθμούς στο διάστημα $[0,5]$ δεν κρίνεται σκόπιμη η κανονικοποίηση των δεδομένων μας. Εφόσον ολοκληρώθηκε η διαδικασία της προ επεξεργασίας των δεδομένων, πλέον μπορούμε να ξεκινήσουμε την εφαρμογή του BSAS για την εκτίμηση των ομάδων.

Εκτίμηση αριθμού ομάδων (μέσω BSAS)

Basic sequential algorithmic scheme (BSAS)

Ο αλγόριθμος BSAS εφαρμόστηκε βάση της αναφοράς του στο βιβλίο του μαθήματος (Αναγνώριση προτύπων, σελίδα 542).

```
def BSAS(vectors, max_distance, max_clusters):
    # Initialize first cluster with first vector
    clusters = [[vectors[0]]]
    # Set current number of clusters
    cluster_count = 1

    # For every vector except the first one (already in cluster)
    for vector in vectors[1:]:
        # keep every distance between vector - clusters
        distances = []
        # for every cluster
        for cluster in clusters:
            # calculate the distance between the current cluster and vector
            distances.append(cluster_distance(vector, cluster))
        # max_distance == 0 book page: 542
        if min(distances) > max_distance and cluster_count < max_clusters:
            # create new cluster
            clusters.append([vector])
            cluster_count += 1
        else:
            # add to the cluster with min distance from vector
            clusters[distances.index(min(distances))].append(vector)

    # return number of clusters
    return cluster_count

# ==== BSAS End ====
# =====
```

Εικ.2 - Basic sequential algorithmic scheme (BSAS) in python.

Για τη εύρεση των αποστάσεων των διανυσμάτων χρησιμοποιήθηκαν τα εξής functions, που κάνουν χρήση του μέτρου εγγύτητας που αναφέραμε παραπάνω.

```
# euclidian_distance - START
def euclidian_distance(a, b):
    # find euclidian distance of 2 vectors
    dist = sqrt(sum( (a - b)**2 for a, b in zip(a, b)))
    # return distance
    return dist

# ==== euclidian_distance End ====
# =====

# cluster_distance - START
# find min distance of vector-cluster
def cluster_distance(vector, cluster):
    distance = []
    # for every vector in cluster
    for cVector in cluster:
        # find distance with function: euc_dist and
        # append the result in list: distance
        distance.append(euclidian_distance(vector, cVector))
    # return min distance vector-vector(in cluster). Book page: 523
    return min(distance)

# ==== cluster_distance End ====
# =====
```

Εικ.3 - proximity measure & proximity function.

Χρήση του BSAS για την εκτίμηση των ομάδων

Για να εκτιμήσουμε τον αριθμό των ομάδων πρέπει να χρησιμοποιήσουμε τον BSAS σύμφωνα με τον αλγόριθμο που αναφέρεται στο βιβλίο του μαθήματος (Σελίδα 545).

Προκειμένου να χρησιμοποιήσουμε τον αλγόριθμο απαιτείται να γνωρίζουμε την μέγιστη και την ελαχιστη απόσταση μεταξύ όλων των διανυσμάτων(vectors) που έχουμε προς ομαδοποίηση. Αυτό το επιτυγχάνουμε με το function: MinMaxVector() του κώδικα μας, στο οποίο βρίσκουμε την απόσταση του κάθε διανύσματος από τα υπόλοιπα και στην συνέχεια επιστρέφουμε την μέγιστη και την ελαχιστη απόσταση σε μια λίστα.

```
# MinMaxVector - START
# Find min/max distance of all vectors
def MinMaxVector(vectorsList):
    # list to hold all the distances.
    # 1 list is better than an if condition with 2 variables to check
    distance = []
    # tqdm for bar effect
    # for every vector find distance with every other vector
    for i in tqdm(range(len(vectorsList))):
        # for vector[i] find every distance with other vectors
        for j in range(i+1, len(vectorsList)):
            # calculate distance and append it in list: distance
            distance.append(euclidian_distance(vectorsList[i], vectorsList[j]))

    # return min and max distance in a list
    return min(distance), max(distance)

# ==== MinMaxVector End ====
# =====
```

Εικ.4 - Εύρεση μέγιστης και ελάχιστης απόστασης μεταξύ των διανυσμάτων.

Στην συνέχεια εφαρμόζουμε τον αλγόριθμο με S και c που μας έχει παρέχει ο χρήστης (σε περίπτωση λάθους εισαγωγής στοιχείων χρησιμοποιούνται αυτόματα οι default τιμές).

```
# FindClusters - START
# Calculate the best number of clusters (with BSAS) as seen on book page: 545
def FindClusters(vectors, minTheta, maxTheta, S, c):
    tempRes = []
    threshold = minTheta

    # Run loop until threshold pass maximum distance between all vectors
    while not threshold > maxTheta:
        res = []
        count = []
        # loop BSAS S times
        for i in range(S):
            # shuffle vectors
            shuffle(vectors)
            # run BSAS
            cluster = BSAS(vectors, threshold, 400)
            # append res in list: res
            res.append(cluster)

        # pick in random if more than 1 elements is max
        numOfClusters = max(set(res), key=res.count)
        tempRes.append(numOfClusters)

        # threshold = threshold + c
        threshold += c
```

Εικ.5 - Χρήση του BSAS για την εκτίμηση των ομάδων (python implementation).

(Η 3η παράμετρος του BSAS πλέον είναι: $\text{len}(\text{vectors})+1$, δηλαδή ο αριθμός των διανυσμάτων αυξημένος κατά ένα)

Εφόσον έχουμε συλλέξει όλα τα δεδομένα μας από τον BSAS, τώρα πρέπει να τα αναλύσουμε για να βρούμε τον καλύτερο δυνατό αριθμό ομάδων. Ο προτιμότερος αριθμός ομάδων είναι αυτός που αντιστοιχεί στην επίπεδη περιοχή με το μεγαλύτερο εύρος στο διάγραμμα: Clusters - Threshold ή Αριθμός ομάδων - Θ .

Ανάλυση αποτελεσμάτων

Έχουμε συλλέξει τον κάθε αριθμό ομάδων από την κάθε επανάληψη του αλγορίθμου σε μια λίστα σειριακά, οπότε το πρώτο στοιχείο αντιστοιχεί στην πρώτη εκτέλεση το δεύτερο στην δεύτερη κοκ. Ουσιαστικά με αυτόν τον τρόπο δεν χρειαζόμαστε τις τιμές του Θ (Threshold) ούτε κάποιο διάγραμμα και αρκεί απλή επεξεργασία λιστών προκειμένου να βρούμε το επιθυμητό αποτέλεσμα.

Πόσες φορές εμφανίζεται συνεχόμενα κάθε αριθμός μέσα στην λίστα, εκτός του αριθμού 1
$\text{group} = [(k, \text{sum}(1 \text{ for } i \text{ in } g)) \text{ for } k, g \text{ in } \text{groupby}(\text{tempRes}) \text{ if } k \neq 1]$
Χρήση του function: $\text{groupby}()$ της βιβλιοθήκης: itertools


```

'''
Find best cluster number by analyzing output data
We will analyze the data by using list comprehension
and groupby from itertools
'''

group = [(k, sum(1 for i in g)) for k,g in groupby(tempRes) if k != 1]

'''

Now on list group we have tuples with every number
and its constant appearance, except number 1 because
there is no meaning in picking as number of clusters: 1

Example of use:
>>> from itertools import groupby
>>> tempRes = [1,1,1,3,4,4,4,4,5,6,6,6,6,7,7,8,9,9,1,4,4,4,7,7]
>>> group = [(k, sum(1 for i in g)) for k,g in groupby(tempRes) if k != 1]
>>> print group
[(3, 1), (4, 4), (5, 1), (6, 4), (7, 2), (8, 1), (9, 2), (4, 3), (7, 2)]

Notes:
    1 is ignored
    numbers: 4 and 7 has two tuples because they
    appear in different places inside the list

As we see this is a very easy and fast way to analyze the given data
with only one library in use (groupby from itertools), without the need
of threshold values for every cluster.
'''

```

Εικ.6 - Ανάλυση δεδομένων (1/2)

Τέλος αρκεί να δούμε ποιος αριθμός έχει την μεγαλύτερη συχνότητα εμφάνισης (σε περίπτωση που είναι πάνω από ένα επιλέγεται τυχαία ένας αριθμός).

```

# temporary variables
temp1 = 0
temp2 = 0

'''
We continue to analyze the data
Now trying to find the number with the
highest frequency of continuous display
in list: group
'''
for item in group:
    if item[1] > temp1:
        temp1 = item[1]
        temp2 = item[0]

# Check if the highest frequency is in more than 1 elements in the list
same = []
tsame = 0
for item in group:
    if item[1] == temp1:
        same.append(item[0])
        tsame += 1

# if: TRUE then pick in random and print a message
if tsame >= 2:
    print "Non succesfull, found with same: "
    print same
    print "\npicking in random"

# return best number of clusters
return temp2

# ===== FinClusters End =====

```

Εικ.7 - Ανάλυση δεδομένων (2/2)

Παρόλο που δεν υπάρχει κάποιο σημείο στον κώδικα για την τυχαία επιλογή, εφόσον ο BSAS τρέχει κάθε φορά δεχόμενος τα διανύσματα με διαφορετική σειρά, το κάθε αποτέλεσμα θεωρείται τυχαίο έτσι και η επιλογή του αριθμού των ομάδων.

Πριν από κάθε επανάληψη του BSAS γίνεται τυχαία αναδιάταξη των διανυσμάτων στην λίστα
<code>shuffle(vectors)</code>
Χρήση του function: <code>shuffle()</code> της βιβλιοθήκης: <code>random</code>

Τελικό αποτέλεσμα

Μετά την ολοκλήρωση των functions μένει η εκτέλεση τους για να λάβουμε την εκτίμηση για τον αριθμό των ομάδων.

Εντολές για την εκτέλεση των functions που αναφέρθηκαν
<code>vectorsList = CreateList()</code>
<code>minmax = MinMaxVector(list(vectorsList))*</code>
<code>FinalClusters = FindClusters(list(vectorsList), minmax[0], minmax[1], S, c)</code>
Το τελικό αποτέλεσμα αποθηκεύεται στην μεταβλητή: <code>FinalClusters</code>

* Η λίστα καλείται με την μορφή `list(vectorsList)` προκειμένου η λίστα στο καλούμενο function να μην συνδέεται άμεσα με την αρχική μας λίστα, στην οποία δεν επιθυμούμε καμία αλλαγή ως προς την σειρά των δεδομένων, διότι θα χαθεί η συσχητική με το ID των χρηστών

Έχοντας εκτελέσει τον αλγόριθμο BSAS σε μια επαναληπτική διαδικασία, προκειμένου να συλλέξουμε αρκετά δεδομένα για το πως ομαδοποιεί τα δεδομένα που του δίνουμε έχουμε πλέον μια εκτίμηση για τον αριθμό των ομάδων, την οποία και θα χρησιμοποιήσουμε στο επόμενο ερώτημα.

3ο Ερώτημα: Αλγόριθμος k-means και ιεραρχικής ομαδοποίησης

Για αυτό το ερώτημα θα χρησιμοποιήσουμε τον αριθμό των ομάδων που εκτιμήσαμε στο προηγούμενο ερώτημα. Επίσης, δεδομένου του ότι μας επιτρέπεται να κάνουμε χρήση οποιασδήποτε βιβλιοθήκης για αυτό το ερώτημα η βασική βιβλιοθήκη που θα χρησιμοποιήσουμε θα είναι η `scikit`, με την οποία θα αυτοματοποιήσουμε την διαδικασία κατασκευής των αλγορίθμων μας

Library: scikit learn, importing:
KMeans
AgglomerativeClustering

K-means

Με την χρήση της βιβλιοθήκης `scikit learn` ξεκινάμε υλοποιώντας τον αλγόριθμο με τις εξής εντολές:

K-means implementation in python with scikit-learn
Init kmeans with scikit lib and fit vectors
kmeans = KMeans().fit(vectors)
kmeansLabels = kmeans.labels_
predictions = kmeans.predict(vectors)

Παράμετροι του αλγορίθμου

Για να ολοκληρωθεί η υλοποίηση του αλγορίθμου πρέπει να θέσουμε τις κατάλληλες παραμέτρους

Ορισμός των παραμέτρων	Λόγος χρήσης παραμέτρου
<code>n_clusters=numofc (numofc == FinalClusters)</code>	Ορίζουμε τον αριθμό των ομάδων που εκτιμήσαμε στο 2ο ερώτημα
<code>init='random'</code>	Επιλέγουμε τυχαία κεντροειδή (centroids)
<code>n_init=15</code>	Αριθμός επαναλήψεων με διαφορετικά κεντροειδή
<code>max_iter=400</code>	Μέγιστος αριθμός επαναλήψεων του αλγορίθμου ανά φορά
<code>precompute_distances=True</code>	Υπολογισμός των αποστάσεων πριν την εκκίνηση του αλγορίθμου
<code>copy_x=True</code>	Αντιγραφή των αρχικών δεδομένων
<code>algorithm='full'</code>	Χρήση του κλασσικού αλγορίθμου (EM-style)

Η επιλογή των παραμέτρων έγινε σύμφωνα με το σχήμα του αλγορίθμου που παρουσιάζεται στο βιβλίο του μαθήματος (σελίδα 659). Πιο συγκεκριμένα σύμφωνα με το σχήμα πρέπει να γίνει αρχικά τυχαία επιλογή για τα κεντροειδή (`init='random'`) και χρήση του βασικού αλγορίθμου (`algorithm='full'`), ο οποίος έχει την ίδια επαναληπτική διαδικασία δύο βημάτων (`two-step procedure`) με το σχήμα EM (Expectation Maximization).

Διαδικασίες	Λειτουργία
Maximization	Αντιστοίχιση των σημείων στα κεντροειδή των ομάδων (υπολογίζοντας την ελαχιστη απόσταση του κάθε σημείου από τα κεντροειδή)
Expectation	Ανανεώνει τα κεντροειδή βάση της κάθε ανάθεσης του προηγούμενου βήματος (υπολογίζοντας τον μέσο ορο των σημείων της κάθε ομάδας μετά την ανάθεση των καινούργιων σημείων)
[repeat]	Η διαδικασία είναι επαναληπτική οποτε πρακτικά υπάρχει και ένα τρίτο στάδιο, η επανάληψη τις διαδικασίας όπως αναφέρθηκε παραπάνω

Τελικό αποτέλεσμα

Τελικός παρατηρούμε ότι παρόλο που η υλοποίηση γίνεται με την χρήση μιας βοηθητικής βιβλιοθήκης η ελευθερία στην επιλογή παραμέτρων μας δίνει την δυνατότητα να προσαρμόσουμε τον αλγόριθμο μας βάση του σχήματος που επιθυμούμε.

```
# Kmeans -- START
def kmeans(vectors, numofc):
    # Init kmeans with scikit lib and fit vectors
    kmeans = KMeans(n_clusters=numofc, init='random', n_init=15, max_iter=400, precompute_distances=True, copy_x=True, algorithm='full').fit(vectors)
    # prediction for our vectors(list ordered by user_id) in list: pre
    pre = kmeans.predict(vectors)
    # labels
    kmeansLabels = kmeans.labels_
    # Cluster Centers
    clusterCenters = kmeans.cluster_centers_
    # Return the predictions
    return kmeansLabels

# === Kmeans End ===
#
```

Εικ.8 - Ο αλγόριθμος Kmeans σε python.

Ιεραρχικής ομαδοποίησης

Εφόσον και για αυτόν τον αλγόριθμο θα κάνουμε χρήση της ίδιας βιβλιοθήκης, θα κινηθούμε με παρόμοια βήματα όπως προηγούμενος. Λόγο της απλότητας του αλγορίθμου και της ομοιότητας του ως προς την υλοποίηση με τον αλγόριθμο Kmeans θα παρουσιάσουμε στην συνέχεια συνοπτικά τον τρόπο υλοποίησής του.

Παράμετροι του αλγορίθμου

Για να ολοκληρωθεί η υλοποίηση του αλγορίθμου πρέπει να θέσουμε τις κατάλληλες παραμέτρους

Ορισμός των παραμέτρων	Λόγος χρήσης παραμέτρου
<code>n_clusters=numofc (numofc == FinalClusters)</code>	Ορίζουμε τον αριθμό των ομάδων που εκτιμήσαμε στο 2ο ερώτημα
<code>affinity = 'euclidean'</code>	Χρήση της ευκλείδειας απόστασης για τον υπολογισμό των αποστάσεων
<code>linkage = 'ward'</code>	Ελαχιστοποιεί την πιθανότητα 2 ομάδες να συγχωνευθούν, με αποτέλεσμα πιο ακριβή δεδομένα.

Τελικό αποτέλεσμα

Τελικός, έχουμε έτοιμο τον αλγόριθμο μας με τις κατάλληλες παραμέτρους, οι οποίες όπως παρατηρούμε είναι μικρότερες σε αριθμό καθώς ο αλγόριθμος kmeans είναι πιο σύνθετος συγκριτικά με τον αλγόριθμο ιεραρχικής ομαδοποίησης

```
# Hierarchical_clustering - START
def Hierarchical_clustering(vectorsList, FinalClusters):
    # init
    hc = AgglomerativeClustering(n_clusters = FinalClusters, affinity = 'euclidean', linkage = 'ward')
    # predict clusters
    y_hc = hc.fit_predict(vectorsList)
    # return results
    return y_hc
# ===== Run H.C. End =====
# =====
```

Εικ.9 - Ο αλγόριθμος ιεραρχικής ομαδοποίησης σε python.

Εκτέλεση των αλγορίθμων

Για την εκτέλεση των αλγορίθμων, η οποία γίνεται μετά εντολή του χρηστη, γίνεται με την χρήση ενός βοηθητικού function. Σκοπός του είναι η εκτέλεση των αλγορίθμων και η επεξεργασία των αποτελεσμάτων τους.

```
def RunKmeansHC(vectorsList, FinalClusters):
    # Ask user
    runKmeans = raw_input("Continue with K-means and H.C.? (y/n): ")
    # keep count and if the user gives 10 times wrong input exit the script
    cnt = 0
    while runKmeans != "n" and runKmeans != "y":
        runKmeans = raw_input("write [y] for yes and [n] for no: ")
        cnt += 1
        if cnt == 10: # break infinite loop of empty input
            print("\n[x] 10 times wrong answer [break]\n")
            sys.exit()
    # if answer is no then exit
    if runKmeans == "n" :
        sys.exit()
    # else run Kmeans
    else:
        print "[*] Running K-means"
    #run K-means
    kmeansLab = Kmeans(list(vectorsList), FinalClusters)
    HcLab = Hierarchical_clustering(list(vectorsList), FinalClusters)
    # Process the output data of Kmeans
    # final list of clusters
    clustersOrdK = []
    clustersOrdH = []
    for i in range(0,943):
        clustersOrdK.append([0])
        clustersOrdH.append([0])
        # +1 so cluster number starts from one(1) and not zero(0)
        kmeansLab[i] += 1
        HcLab[i] += 1
    # create list of clusters
    for i in range(0, 943):
        '''
        For every vector from 1 to 943
        append it to the corresponding cluster
        for every algorithm
        '''
        clustersOrdK[kmeansLab[i]].append(i+1)
        clustersOrdH[HcLab[i]].append(i+1)

    return clustersOrdK, clustersOrdH
# === Run K-meansHC End ===
# =====
```

Εικ.10 - Εκτέλεση των αλγορίθμων και επεξεργασία των αποτελεσμάτων τους.

Βασικές λίστες που χρησιμοποιούνται:

kmeansLab	Label of each point (stating from 0) based on Kmeans
HcLab	Label of each point (stating from 0) based on H.C.
1st (for) loop	change kmeansLab & HcLab to start from 1
2nd (for) loop	append every vector to their cluster*
clustersOrdK	Clusters predicted by kmeans with their vectors
clustersOrdH	Clusters predicted by H.C with their vectors

* Η ταξινόμηση του κάθε διανύσματος στην ομάδα που έχει αντιστοιχηθεί από τους αλγορίθμους είναι δυνατή, διότι στο προηγούμενο ερώτημα έγινε χρήση της βασικής λίστα με τα διανύσματα με την χρήση του list(), έτσι η αρχική μας λίστα είναι ακόμα ταξινομημένη με βάση το ID των χρηστών.

Προβολή αποτελεσμάτων

Μέχρι στιγμής ο κώδικας μας έχει την μορφή ενός script παρα ενός πλήρους προγράμματος, αφού μπορεί να εκτέλεση μια συγκεκριμένη διαδικασία. Επειδή η χρήση του γίνεται μέσω terminal δεν κρίνεται σκοπική η προβολή των αποτελεσμάτων στην οθόνη του τερματικού. Για αυτό τον λόγο στο τέλος του script μας καλείται το function: `appendix.report()`, με το οποίο δημιουργούμε ένα report (υπό την μορφή ενός text file) που αποθηκεύεται στον ίδιο φάκελο με το αρχείο μας. Σε κάθε επανάληψη του αλγορίθμου χρησιμοποιείτε το ίδιο αρχείο.

File name	appendix.py
File's functions	report()
Use of report:	Δημιουργία ενός text αρχείου με τα αποτελέσματα των αλγορίθμων των ερωτημάτων 2 και 3

```
def report(FinalClusters, kmeansInfo, S, c):
    appendix = open('report.txt', 'w')
    appendix.write("Appendix\n")
    appendix.write("-----\n")
    appendix.write("Results of algorithms:\n")
    appendix.write(" - Clusters number found by BSAS: %s\n" % FinalClusters)
    appendix.write(" - Initialization parameters: S=%s, c=%s\n" % (S, c))
    appendix.write("\n - Clusters with vectors as assigned by K-means algorithm: \n")
    for i in range(1,FinalClusters+1):
        appendix.write("\nCLUSTER: %s" % i)
        #print "\nCLUSTER: %s\n" % i
        appendix.write("\n%s\n" % kmeansInfo[0][i][1:])
        #print kmeansInfo[i][1:]
    appendix.write("\n\n - Clusters with vectors as assigned by H.C. algorithm: \n")
    for i in range(1,FinalClusters+1):
        appendix.write("\nCLUSTER: %s" % i)
        #print "\nCLUSTER: %s\n" % i
        appendix.write("\n%s\n" % kmeansInfo[1][i][1:])
        #print kmeansInfo[i][1:]

    appendix.close()
```

Εικ.11- File: appendix.py.

4ο Ερώτημα: Ταξινομητές νευρωνικού δικτύου & ελαχίστων τετραγώνων

Στο τελευταίο ερώτημα της εργασίας καλούμαστε να δημιουργήσουμε δύο ταξινομητές. Ο ένας ταξινομητής θα είναι νευρωνικό δίκτυο και ο άλλος ελαχίστων τετραγώνων. Σκοπός του κάθε ταξινομητή είναι, αφού εκπαιδευτούν με βάση το σχήμα Kfold ($K=5$), να μπορούν να εκτιμήσουν αν ένας χρήστης έχει δει μια συγκεκριμένη ταινία.

Στην βασική βιβλιοθήκη, την οποία έχουμε χρησιμοποιήσει μέχρι τώρα, παρέχεται η δυνατότητα δημιουργίας ενός Kfold σχήματος από το αρχικό dataset.

Function
<code>sklearn.model_selection.KFold</code>

Ωστόσο μας ζητείται να χρησιμοποιήσουμε το 5fold σχήμα που μας παρέχεται μαζί με τα αρχικά δεδομένα, έτσι θα προχωρήσουμε στην επεξεργασία αυτών, βάση του τρόπου λειτουργίας του function που μόλις αναφέραμε.

Training and test files	
Training sets	Test sets
<code>u1.base,...,u5.base</code>	<code>u1.test,...,u5.test</code>

Εφόσον δεν μας διευκρινίζεται από την εκφώνηση της εργασίας ο τρόπος με τον οποίο πρέπει να χρησιμοποιηθούν τα δεδομένα προκειμένου να εκπαιδευτεί το σύστημα και να κάνει ακριβής προβλέψεις καλούμαστε εμείς να βρούμε τον πιο πρακτικό τρόπο.

Αρχικές υποθέσεις:

Θεωρούμε ότι οποιος χρηστης έχει κάνει κάποιο rating σε μια ταινία την έχει δει
Θεωρούμε ότι οποιο rating είναι κάτω από 3, τότε ο χρήστης δεν θα δει την ταινία

1. Θεωρούμε ότι οποιος χρηστης έχει κάνει κάποιο rating σε μια ταινία την έχει δει

Αν και αρχικά θεωρείτε το πιο λογικό, σύντομα παρατηρούμε ότι η έλλειψη στοιχείων που ανήκουν στην 2η κλάση (δεν έχουν δει κάποια ταινία) δεν μας επιτρέπει να εκπαιδεύσουμε κατάλληλα τον αλγόριθμο.

2. Θεωρούμε ότι οποιο rating είναι κάτω από 3, τότε ο χρήστης δεν θα δει την ταινία

Σε αυτό το σενάριο μπορούμε να χωρίσουμε το αρχικό μας dataset στα κατάλληλα μέρη προκειμένου να μπορούμε να εκπαιδεύσουμε τον αλγόριθμο.

Παράδειγμα για το u1.base:

u1.base	
[userID,movieID,rating,timestamp]	
Input in classifier	Expected output
[userID,movieID]	0 if rating <=3 & 1 if rating > 3

Όπως παρατηρούμε και στο παράδειγμα που μόλις αναφέρθηκε..αφαιρούμε άμεσος την καταχώριση: timestamp, καθώς είναι μια μεταβλητή που δεν επηρεάζει κάπως το αποτέλεσμα το οποίο αναζητούμε, δηλαδή το αν ο χρήστης είναι μια ταινία ή όχι.

Σε αυτό το σημείο θα μπορούσαμε να μετατρέψουμε το MovieID σε Movie_Types[ID], όπως στα προηγούμενα ερωτήματα, όμως όπως μπορούμε με μια απλή ανάγνωση του αρχείου u.items περισσότερες από μια ταινίες εμφανίζονται με ακριβώς τα ίδια ήδη ταινιών.

Παράδειγμα ταινιών με ίδιες κατηγορίες	
Movie ID	Movie Type
5	ΙΟΙΟΙΟΙΟΙΟΙΟΙ11ΟΙ11ΟΙΟΙΟΙΟΙΟΙΟΙΟΙ11ΟΙΟ
100	ΙΟΙΟΙΟΙΟΙΟΙΟΙ11ΟΙ11ΟΙΟΙΟΙΟΙΟΙΟΙΟΙ11ΟΙΟ
Αποτέλεσμα σε περίπτωση αντικατάστασης movieID με Movie_types	
Χάνεται η μοναδικότητα της κάθε ταινίας. ένα διάνυσμα αντιπροσωπεύει περισσότερες από μία ταινίες	

Οποτε καταλήγουμε στο τελικό διάνυσμα για εισαγωγή στον αλγόριθμο, που είναι και το ζητούμε της εργασίας: [userID,movieID].

```
def PreProcess(vec):
    bigList = []
    # same loop for all u[1].base files, i E [1,5], i E [R
    for i in tqdm(range(5),desc='Data preprocessing..'):
        # open every file
        with open(vec[i]) as data:
            # temp list too build every line
            temp = []
            # for every line in the file
            for line in data:
                # take first line and split it in: list
                list = line.split(" ")
                # change str to int & remove last entry (timestamp)[:-1]
                # put line in temp list
                temp.append([int(x) for n, x in enumerate(list[:-1])])

            # put temp list in final list (bigList)
            bigList.append(temp)
    # change ratings to: 0 or 1
    for tt in bigList:
        for vector in tt:
            if vector[2] >= 3:
                vector[2] = 1
            else:
                vector[2] = 0
    return bigList

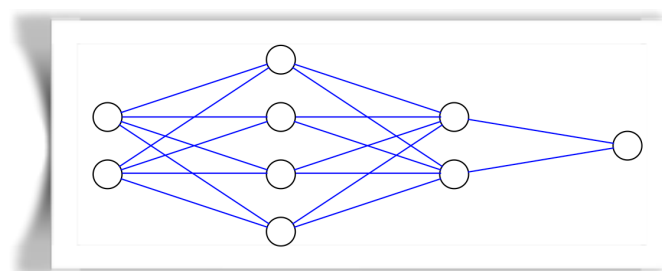
def Process2(ProcessedData):
    userMovieId = []
    res = []
    for lists in list(ProcessedData):
        temp = []
        temp2 = []
        for data in list(lists):
            temp.append(data[:-1])
            temp2.append(data[2:3])
        userMovieId.append(temp)
        res.append(temp2)
    return userMovieId, res
```

Εικ.12- Επεξεργασία αρχικού dataset για διαχωρισμό του σε: input data και target data.

Αφού πλέον ολοκληρώσαμε την προεπεξεργασία των δεδομένων, απομένει η υλοποίηση του κάθε ταξινομητή.

Νευρωνικό δίκτυο (Multi Layer Perceptron - MLP)

Για την υλοποίηση του αλγορίθμου μας θα συμβουλευτούμε το βιβλίο του μαθήματος (σελίδα 171) και θα σχεδιάσουμε έναν αλγόριθμο perceptron τριών επιπέδων.



Εικ.13- Σχέδιο του νευρωνικού μας δικτύου.

Με την βοήθεια της βιβλιοθήκης scikit learn, όπως και στο προηγούμενο ερώτημα έχουμε:

```
Initialize algorithm
MLP = MLPClassifier()
```

Παράμετροι του αλγορίθμου

Για να ολοκληρωθεί η υλοποίηση του αλγορίθμου πρέπει να θέσουμε τις κατάλληλες παραμέτρους

Ορισμός των παραμέτρων	Λόγος χρήσης παραμέτρου
hidden_layer_sizes=(4,2)	Αριθμός νευρώνων σε κάθε μια από τις κρυφές επιφάνειες, όπως φαίνεται και στην Εικ.13
activation='logistic'	Χρήση της λογιστικής συνάρτησης $f(x) = 1 / (1 + \exp(-x))$ (Βιβλίο σελίδα 175)
solver='sgd'	Χρήση του SGD για βελτιστοποίηση των βαρών (SGD = stochastic gradient descent)
alpha=1e-5	
learning_rate='adaptive'	Η ταχύτητα μάθησης είναι προσαρμόσιμη και στην περίπτωση που για δυο συνεχόμενες επαναλήψεις δεν υπάρχει καμία βελτίωση, τότε αλλάζει η τιμή ανάλογος.
max_iter=1000	Μέγιστος αριθμός επαναλήψεων του αλγορίθμου για το κάθε training dataset
shuffle=True	Ανακάτεμα δεδομένων για κάθε επανάληψη, για να εξαλείψουμε την πιθανότητα ο αλγόριθμος να επηρεαστεί από την ταξινόμηση των δεδομένων

Οποτε πλέον έχουμε αρχικοποιήσει τον αλγόριθμο μας, ο οποίος είναι έτοιμος για να εκπαιδευτεί με τα δεδομένα του σχήματος 5fold.

Ταξινομητής ελαχίστων τετραγώνων

Ομοίως για την υλοποίηση του ταξινομητή ελαχίστων τετραγώνων πρέπει να θέσουμε τις παραμέτρους και να αρχικοποιήσουμε τον αλγόριθμο με την χρήση της βιβλιοθήκης scikit learn.

Initialize algorithm
LinReg = LinearRegression()

Παράμετροι του αλγορίθμου

Για να ολοκληρωθεί η υλοποίηση του αλγορίθμου πρέπει να θέσουμε τις κατάλληλες παραμέτρους

Ορισμός των παραμέτρων	Λόγος χρήσης παραμέτρου
copy_X=True	Αντιγραφή των δεδομένων που εισάγονται στον αλγόριθμο προκειμένου να μην υπάρχει αλλοίωση στην αρχική λίστα
fit_intercept=True	Υπολογισμός: intercept
normalize=False	Μη κανονικοποίηση δεδομένων αφού είναι ήδη (from sklearn.preprocessing import StandardScaler)

Οποτε πλέον έχουμε αρχικοποιήσει τον αλγόριθμο μας, ο οποίος είναι έτοιμος για να εκπαιδευτεί με τα δεδομένα του σχήματος 5fold.

Εκπαίδευση των αλγορίθμων

Τέλος πρέπει να εκπαιδεύσουμε τους αλγορίθμους μας με την χρήση του 5fold σχήματος που μας παρέχεται και να κάνουμε τις εκτιμήσεις για την ακρίβεια των αλγορίθμων. Δεν γίνεται να κάνουμε χρήση του function Kfold τις βιβλιοθήκης scikit learn, το οποίο αναφέρθηκε προηγούμενος, αφού έχουμε ήδη έτοιμα τα δεδομένα ας, ωστόσο μπορούμε να επηρεαστούμε από αυτό για το πως θα υλοποιήσουμε το δικό μας σχήμα.

```
# Train classifiers with 5fold cross-validation
for i in tqdm(range(5), desc='Training classifiers'):

    for it in res2[i]:
        for tt in it:
            tt = [1]

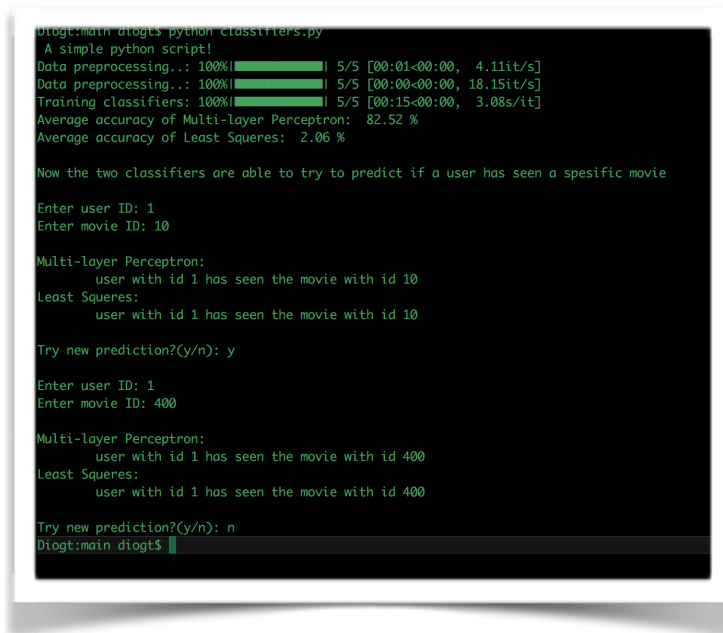
    #scale data
    scaler.fit(userMovieId[i])
    userMovieId[i] = scaler.transform(userMovieId[i])
    uMTest[i] = scaler.transform(uMTest[i])

    MLP.fit(userMovieId[i], np.ravel(res[i]))
    accMLP.append(MLP.score(uMTest[i], np.ravel(res2[i])))
    LinReg.fit(userMovieId[i], np.ravel(res[i]))
    accLinReg.append(LinReg.score(uMTest[i], np.ravel(res2[i])))
print "Average accuracy of Multi-layer Perceptron: ", sum(accMLP)/len(accMLP)*100, "%"
print "Average accuracy of Least Squares: ", round(sum(accLinReg)/len(accLinReg)*100,2), "%"
```

Εικ.14-Εκπαίδευση των αλγορίθμων (5fold cross-validation).

Τελικό αποτέλεσμα

Σαν τελικό αποτέλεσμα έχουμε δυο ταξινομητές, οι οποίοι μπορούν να προβλέψουν, με ένα συγκεκριμένο ποσοστό επιτυχίας, εάν κάποιος χρήστης είδε μια ταινία. Ο χρήστης μπορεί να εισάγει με την σειρά: userID και movieID και να λάβει την πρόβλεψη των 2 αλγορίθμων.



```
diogt:main diogt$ python classifiers.py
A simple python script!
Data preprocessing.: 100% | 5/5 [00:01<00:00, 4.11it/s]
Data preprocessing.: 100% | 5/5 [00:00<00:00, 18.15it/s]
Training classifiers: 100% | 5/5 [00:15<00:00, 3.08s/it]
Average accuracy of Multi-layer Perceptron: 82.52 %
Average accuracy of Least Squares: 2.06 %

Now the two classifiers are able to try to predict if a user has seen a spesific movie

Enter user ID: 1
Enter movie ID: 10

Multi-layer Perceptron:
    user with id 1 has seen the movie with id 10
Least Squares:
    user with id 1 has seen the movie with id 10

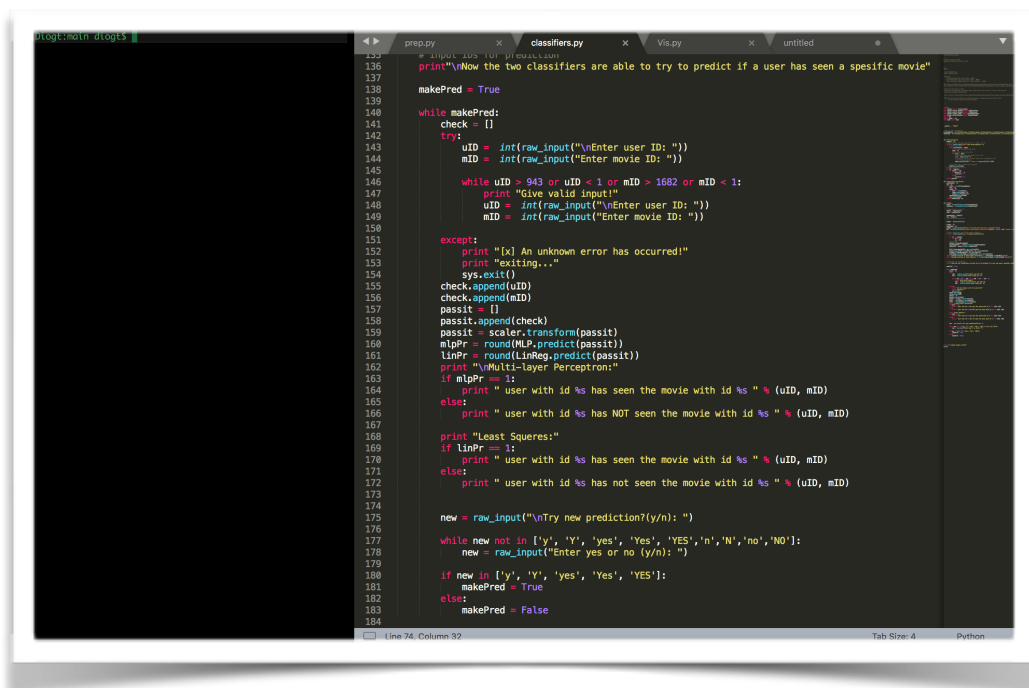
Try new prediction?(y/n): y

Enter user ID: 1
Enter movie ID: 400

Multi-layer Perceptron:
    user with id 1 has seen the movie with id 400
Least Squares:
    user with id 1 has seen the movie with id 400

Try new prediction?(y/n): n
diogt:main diogt$
```

Εικ.15- User terminal UI.



```
132 # Input: userID, movieID
133 print("\nNow the two classifiers are able to try to predict if a user has seen a spesific movie")
134
135 makePred = True
136 while makePred:
137     check = []
138     try:
139         userID = int(raw_input("\nEnter user ID: "))
140         movieID = int(raw_input("\nEnter movie ID: "))
141
142         while userID > 943 or userID < 1 or movieID > 1682 or movieID < 1:
143             print("Give valid input")
144             userID = int(raw_input("\nEnter user ID: "))
145             movieID = int(raw_input("\nEnter movie ID: "))
146
147     except:
148         print "[x] An unknown error has occurred!"
149         print "Exiting..."
150         sys.exit()
151
152     check.append(userID)
153     check.append(movieID)
154
155     passit = []
156     passit.append(check)
157     passit = scaler.transform(passit)
158     mlpPr = round(MLP.predict(passit))
159     linPr = round(linReg.predict(passit))
160     print "\nMulti-layer Perceptron:"
161     if mlpPr == 1:
162         print "user with id %s has seen the movie with id %s " % (userID, movieID)
163     else:
164         print "user with id %s has NOT seen the movie with id %s " % (userID, movieID)
165
166     print "Least Squares:"
167     if linPr == 1:
168         print "user with id %s has seen the movie with id %s " % (userID, movieID)
169     else:
170         print "user with id %s has not seen the movie with id %s " % (userID, movieID)
171
172     new = raw_input("\nTry new prediction?(y/n): ")
173
174     while new not in ['y', 'Y', 'yes', 'Yes', 'YES', 'n', 'N', 'no', 'NO']:
175         new = raw_input("Enter yes or no (y/n): ")
176
177     if new in ['y', 'Y', 'yes', 'Yes', 'YES']:
178         makePred = True
179     else:
180         makePred = False
181
182
183
184
```

Εικ.16- Code for user terminal UI.

Παράρτημα (Appendix)

Παράδειγμα ορθής εκτέλεσης

core.py

The screenshot shows a terminal window on the left and a report.txt file on the right. The terminal window displays the execution of a Python script named main2.py. The script prompts the user for parameters: S (number of loops) and c (steps). The user enters S=15 and c=0.4. The script then calculates the minimum and maximum distance in the vectors, resulting in min distance: 0.721110255093 and max distance: 16.0312195419. The script continues with K-means and H.C. and reports the results in the file: appendix.txt. The report.txt file shows the results of the algorithms, including the number of clusters found by BSAS (5) and the initialization parameters (S=15, c=0.4). It also lists the clusters with vectors as assigned by K-means algorithm, showing three clusters: CLUSTER: 1, CLUSTER: 2, and CLUSTER: 3. Each cluster contains a list of vector indices.

```
Diagt:main diagt$ python main2.py
A simple python script about clustering

[*] Creating List
List is ready.

[!] Continuing on calculation of min/max distance in the vectors.
100% min distance is: 0.721110255093 943/943 [00:01:00:00, 480.86it/s]
max distance is: 16.0312195419

Enter parameters for BSAS (number of loops and step)
Press [Enter] to run with precalculated number of clusters: 5
S = 15, c(Steps)= 0.4

Enter number of loops (S) for BSAS:
Using precalculated data
Number of clusters for S=15 and c=0.4: 5
Number of clusters: 5
Continue with K-means and H.C.? (y/n): y
[*] Running K-means and H.C.
You can see the results in the file: appendix.txt
--- 7.08844304085 seconds elapsed ---
Diagt:main diagt$
```

report.txt

```
Appendix
Results of algorithms:
- Clusters number found by BSAS: 5
- Initialization parameters: S=15, c=0.4
- Clusters with vectors as assigned by K-means algorithm:

CLUSTER: 1
[2, 5, 11, 16, 22, 24, 25, 28, 29, 30, 38, 45, 52, 56, 60, 77, 82, 83, 84, 91, 99,
102, 109, 110, 122, 125, 137, 138, 145, 148, 152, 158, 167, 169, 174, 177, 178,
186, 188, 193, 194, 197, 210, 215, 216, 218, 223, 225, 230, 236, 237, 248, 253,
256, 263, 267, 270, 275, 278, 279, 280, 283, 285, 290, 294, 301, 313, 314, 316,
320, 329, 332, 336, 340, 346, 347, 350, 363, 370, 371, 383, 384, 388, 390, 394,
395, 396, 397, 398, 401, 407, 409, 411, 419, 422, 426, 427, 428, 433, 436, 455,
464, 468, 472, 480, 481, 484, 487, 488, 495, 496, 499, 500, 501, 503, 505, 507,
512, 519, 528, 534, 536, 538, 539, 543, 545, 548, 551, 553, 555, 562, 569, 573,
579, 586, 593, 605, 608, 610, 613, 618, 621, 622, 624, 625, 627, 629, 632, 633,
638, 640, 641, 642, 647, 648, 653, 654, 659, 660, 663, 664, 665, 668, 669, 670,
671, 676, 679, 683, 690, 697, 698, 704, 705, 709, 710, 712, 715, 719, 727, 731,
734, 738, 743, 746, 751, 753, 756, 757, 764, 766, 767, 776, 780, 786, 790, 795,
806, 815, 825, 826, 830, 835, 838, 844, 850, 862, 868, 877, 878, 881, 882, 885,
892, 897, 901, 902, 911, 912, 913, 921, 922, 923, 933, 938, 940, 942]

CLUSTER: 2
[1, 6, 7, 10, 13, 14, 18, 21, 23, 42, 43, 44, 48, 58, 59, 62, 64, 65, 69, 70, 72,
85, 87, 90, 92, 94, 95, 96, 97, 115, 116, 118, 119, 123, 128, 130, 144, 146, 151,
160, 182, 184, 185, 187, 189, 195, 198, 200, 201, 207, 213, 214, 221, 222, 226,
232, 233, 234, 235, 239, 243, 244, 249, 250, 264, 268, 269, 271, 276, 286, 291,
292, 293, 295, 296, 297, 298, 299, 303, 305, 307, 308, 311, 312, 315, 321, 323,
325, 326, 327, 328, 330, 334, 339, 342, 343, 344, 345, 354, 360, 361, 373, 374,
378, 379, 380, 381, 385, 387, 389, 391, 392, 393, 399, 402, 406, 416, 417, 429,
435, 437, 447, 450, 452, 453, 454, 456, 457, 458, 465, 467, 474, 476, 479, 486,
489, 493, 497, 498, 504, 506, 508, 514, 521, 522, 523, 524, 526, 527, 532, 533,
535, 537, 541, 542, 560, 561, 566, 567, 568, 577, 588, 592, 595, 601, 606, 615,
634, 643, 645, 650, 655, 658, 661, 666, 682, 684, 686, 693, 694, 707, 711, 716,
721, 741, 747, 748, 749, 758, 763, 770, 773, 782, 788, 793, 796, 804, 805, 807,
821, 823, 833, 840, 843, 846, 848, 851, 854, 864, 867, 870, 871, 875, 880, 883,
886, 887, 889, 890, 894, 896, 899, 903, 907, 908, 916, 918, 919, 924, 929, 932,
934]

CLUSTER: 3
[4, 19, 27, 32, 33, 35, 36, 51, 53, 61, 68, 78, 80, 86, 98, 103, 127, 133, 143,
153, 155, 156, 163, 166, 181, 183, 196, 205, 208, 212, 217, 220, 224, 228, 255,
257, 258, 260, 266, 273, 288, 289, 300, 302, 304, 306, 310, 316, 333, 335, 353,
355, 356, 366, 376, 377, 382, 386, 400, 405, 413, 418, 424, 439, 440, 443, 444,
445, 462, 476, 477, 482, 502, 513, 529, 530, 546, 558, 559, 563, 565, 572, 575,
578, 583, 585, 590, 591, 594, 596, 604, 607, 609, 631, 657, 662, 667, 672, 675,
685, 687, 724, 725, 728, 729, 732, 739, 750, 754, 761, 765, 774, 778, 797, 810,
816, 824, 832, 834, 855, 856, 857, 858, 859, 861, 866, 873, 876, 884, 888, 891]
```

Εικ.17- Run core.py & see report.txt.

```
Diogt:main diogt$ clear
Diogt:main diogt$ python classifiers.py
A simple python script!
Data preprocessing.: 100%|██████████| 5/5 [00:01<00:00, 3.95it/s]
Data preprocessing.: 100%|██████████| 5/5 [00:00<00:00, 17.70it/s]
Training classifiers: 100%|██████████| 5/5 [00:14<00:00, 3.00s/it]
Average accuracy of Multi-layer Perceptron: 82.52 %
Average accuracy of Least Squeres: 2.06 %

Now the two classifiers are able to try to predict if a user has seen a spesific movie

Enter user ID: 1
Enter movie ID: 10

Multi-layer Perceptron:
    user with id 1 has seen the movie with id 10
Least Squeres:
    user with id 1 has seen the movie with id 10

Try new prediction?(y/n): n
Diogt:main diogt$
```

Εικ.18- Run classifiers.py.