Αναγνώριση Προτύπων

Χειμερινό Εξάμηνο, 2017-2018

Παναγιώτα Θωμοπούλου | Π14053

Γιωργός Κονσουλάκης | Π14078

Γεράσιμος Δεληβοριάς | Π14029

# Εισαγωγή

Για την επίλυση της εργασίας χρησιμοποιήθηκε η γλώσσα python 3.6 και οι βιβλιοθήκες:

* Numpy
* Matplotlib
* Scikit-learn
* Scipy

# Προετοιμασία dataset – saveasnumpy.py

Για την χρήση των δεδομένων, χρειάστηκε πρώτα η κατάλληλη επεξεργασία τους. Οι προτιμήσεις των χρηστών θεωρήθηκαν οι πρώτες 18 στήλες με τα είδη των ταινιών (Action, Adventure, …, Western), και η 19η τα ratings.

Για να τοποθετηθούν τα δεδομένα στο πρόγραμμα, χρησιμοποιήθηκε το αρχείο python saveasnumpy.py.

Αρχικά, αρχικοποιήθηκε ο πίνακας ratings

ratings = np.arange(400000).reshape(100000, 4)

που περιέχει τα ids και τα ratings. Διαβάστηκαν από το αρχείο όλα τα ratings και τοποθετήθηκαν στον πίνακα ανά γραμμή.

fileStream = open('u.data', 'r')  
lines = fileStream.readlines()  
i= 0  
for line in lines:  
 ratings[i]= line.split("\t")  
 i += 1  
fileStream.close()

Ομοίως για τους χρήστες και τις ταινίες.

Το τελικό dataset τοποθετήθηκε στο array dataset, μεγέθους 100.000\*19.

dataset = np.zeros((100000, 19), dtype=float)

Στις πρώτες 18 στήλες του dataset τοποθετήθηκαν οι 18 στήλες των ειδών στο movies,

for j in range(18):  
 dataset[i][j] = movies[movieId][j + 6]

ενώ στην τελευταία στήλη τοποθετήθηκε το rating της ταινίας του κάθε χρήστη κανονικοποιημένο, για να βρίσκεται σε τιμές μεταξύ 0 και 1

dataset[i][18] = rating[2]\*0.25-0.25

Τέλος, γίνεται αποθήκευση του αρχείου σε μορφή .npy, για πιο εύκολη και γρήγορη εκτέλεση του υπόλοιπου κώδικα.

np.save('dataset.npy', dataset)

# Basic Sequential Algorithmic Scheme (BSAS)

Για τον αλγόριθμο BSAS χρειάστηκε να οριστούν 2 σταθερές. Τον μέγιστο αριθμό ομαδοποιήσεων και τον δείκτη διαφοροποίησης.

# maximum number of clusters to create  
MAX\_CLUSTER\_NUMBER = 20  
# threshold for how different 2 vectors can be based on their distance  
THRESHOLD = 2.45

Επιπλέον, ορίζεται η συνάρτηση FindDistance, που παίρνει 2 διανύσματα -μονοδιάστατα arrays- και επιστρέφει την ευκλείδεια απόστασή τους,

# finds the euclidean distance between 2 vectors  
def FindDistance(vec1, vec2):  
 distance = 0  
 for i in range(vec1.size):  
 distance += (vec1[i]-vec2[i])\*\*2  
 distance = math.sqrt(distance)  
 return distance

και η συνάρτηση FindClosestCluster, η οποία δέχεται σαν ορίσματα ένα διάνυσμα, ένα array με όλες τις θέσεις των υπαρχόντων clusters και τον αριθμό τους.

# finds the closest possible cluster from a vector, given the vector itself,  
# the current clusters locations and the number of existing clusters  
def FindClosestCluster(vec1, clusterPositions, currentClustersNumber):  
 numberOfCluster = 0  
 minumunClusterDistance = FindDistance(vec1, clusterPositions[0])  
 for i in range(currentClustersNumber):  
 currentClusterDistance = FindDistance(vec1, clusterPositions[i])  
 if (minumunClusterDistance> currentClusterDistance):  
 minumunClusterDistance = currentClusterDistance  
 numberOfCluster = i  
 return numberOfCluster

Η συνάρτηση επιστρέφει τον αριθμό του cluster που βρίσκεται πιο κοντά στο διάνυσμα.

Για να τρέξει ο αλγόριθμος, χρειάζεται να φορτωθεί ο πίνακας dataset από το αρχείο dataset.npy:

# loads the dataset from saveasnumpy  
dataset = np.load('dataset.npy')

Οι μεταβλητές που θα χρησιμοποιηθούν είναι αρχικά, ο πίνακας ClusterPositions, μεγέθους MAX\_CLUSTER\_NUMBER \* Αριθμός διαστάσεων dataset (20x18). Στον πίνακα αυτόν θα τοποθετηθεί η θέση του κάθε cluster στον διανυσματικό χώρο.

Ο επόμενος πίνακας assignedCluster, περιέχει τον αριθμό του cluster στον οποίο ανήκει (label).

Τέλος, o ακέραιος currentClustersNumber δείχνει τον αριθμό των υπαρχόντων cluster.

Μετά την αρχικοποίηση των μεταβλητών,

clusterPositions[0] = dataset[0]  
assignedCluster[0] = 0  
currentClustersNumber = 1

γίνεται προσπέλαση όλων των διανυσμάτων του dataset.

for i in range(dataset.shape[0]):

Για κάθε ένα dataset βρίσκεται το πιο κοντινό cluster σε αυτό,

currentClosestCluster = FindClosestCluster(dataset[i],clusterPositions,currentClustersNumber)

και γίνεται έλεγχος για το αν η απόσταση από αυτό είναι μεγαλύτερη από το threshold, και αν δεν έχει συμπληρωθεί ο μέγιστος αριθμός cluster.

if(FindDistance(clusterPositions[currentClosestCluster], dataset[i])>THRESHOLD and currentClustersNumber<MAX\_CLUSTER\_NUMBER):

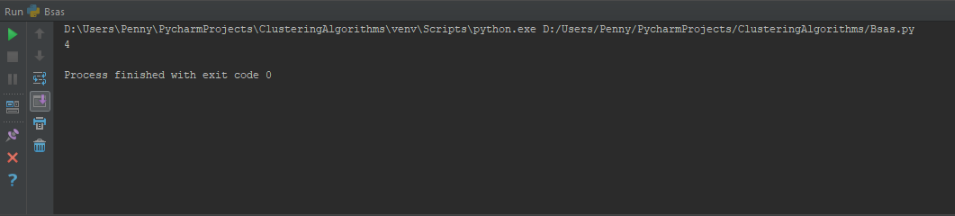
Αν όντως γίνει αυτό, δημιουργείται το καινούργιο cluster, και ανανεώνονται οι κατάλληλοι πίνακες και μεταβλητές, αλλιώς το σημείο τοποθετείται σε ένα ήδη υπάρχον cluster, αυτό που επέστρεψε η συνάρτηση.

clusterPositions[currentClustersNumber] = dataset[i]  
 assignedCluster[i] = currentClustersNumber  
 currentClustersNumber += 1  
else:  
 assignedCluster[i] = currentClosestCluster

Όταν τελειώσει η επανάληψη, γίνεται εκτύπωση του αριθμού των cluster.

print(currentClustersNumber)

## Παραδειγμα εκτελεσησ



## Επεξήγηση Threshold και MAx\_Number\_OF\_CLUSTERS

Στον παραπάνω κώδικα, τέθηκαν οι σταθερές για τον μέγιστο αριθμό cluster και τον δείκτη διαφοροποίησης αυθαίρετα. Στην πραγματικότητα, θα έπρεπε να γίνει εκτέλεση του αλγόριθμου με for από την μικρότερη απόσταση μεταξύ των cluster, μέχρι την μεγαλύτερη απόσταση, με ένα βήμα, και το πιο συνηθισμένο θ που θα προκύψει, να επιλεγεί.

Το θ = 2,45 επιλέχθηκε με βάση τους επόμενους αλγορίθμους ομαδοποίησης. Ο λόγος που δεν υλοποιήθηκε ο παραπάνω τρόπος, είναι επειδή για το μεγάλο dataset της εργασίας, παίρνει πάρα πολύ χρόνο, οπότε δεν ήταν δυνατό να ελεγχθεί το αν έτρεχε σωστά.

# Αλγόριθμος k-means

Για τον αλγόριθμο k-means, χρησιμοποιήθηκε η βιβλιοθήκη scikit-learn.

Αφού γίνει φόρτωση του dataset από το npy αρχείο,

# loads the dataset from saveasnumpy  
dataset = np.load('dataset.npy')

Ορίζεται ο αριθμός των cluster που θα δημιουργηθούν.

# total cluster number based on the result of BSAS  
k = 4

Ο αριθμός που επιλέχθηκε είναι 4, με βάση το αποτέλεσμα που εμφάνισε ο αλγόριθμος BSAS.

Στη συνέχεια εφαρμόζεται ο αλγόριθμος k-means της βιβλιοθήκης.

# apply k-means with sk learn library, based on the number of clusters we found via BSAS  
kmeans = KMeans(n\_clusters=k, random\_state=0).fit(dataset)

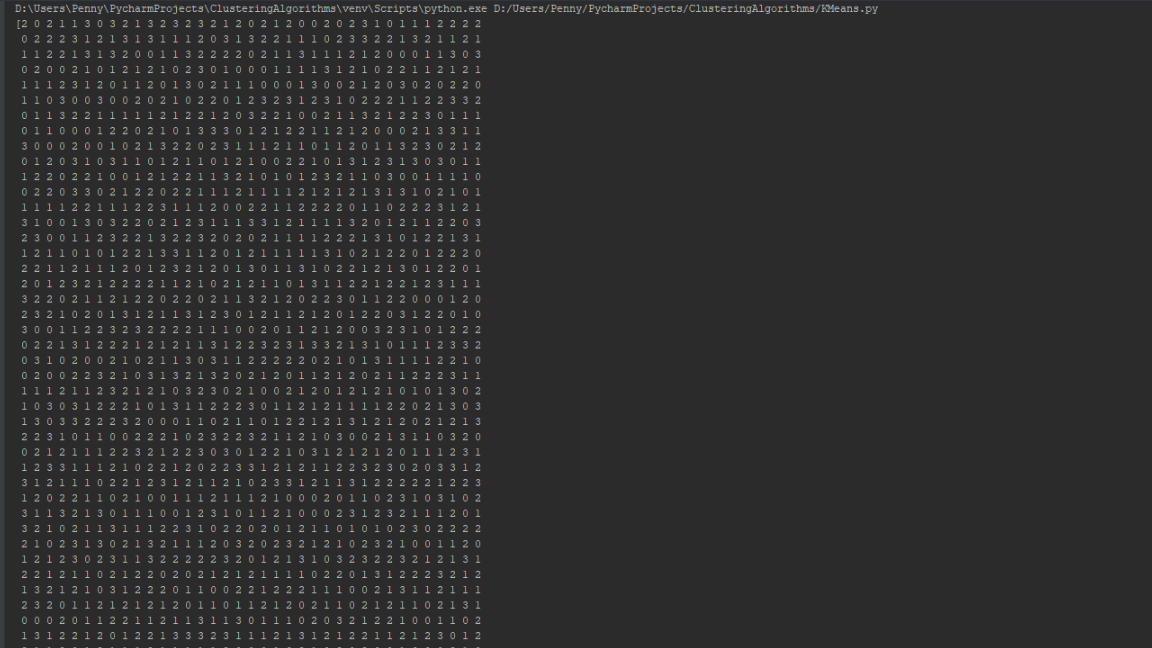
Από εκεί αρκεί να βρεθούν τα labels του κάθε διανύσματος, δηλαδή τον αριθμό του cluster του οποίου ανήκουν.

# get the labels for each rating, basically the number of cluster every rating belongs to  
kmeansLabels = kmeans.labels\_  
print(kmeans.labels\_)

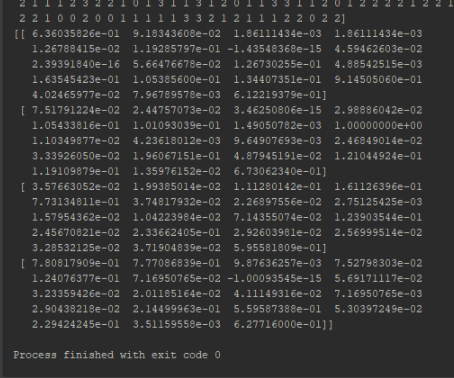
Τέλος, εμφανίζονται τα κέντρα του κάθε cluster:

# get the position of every cluster center  
clusterCenters = kmeans.cluster\_centers\_  
print(clusterCenters)

## Παράδειγμα εκτέλεσης κώδικα



Labels κάθε διανύσματος



Θέσεις κέντρων

# Ιεραρχικός αλγόριθμος ομαδοποίησης

Για τον αλγόριθμο αυτό, χρησιμοποιήθηκε η βιβλιοθήκη scipy, και για την αναπαράσταση του δενδρογράμματος η βιβλιοθήκη matplotlib.

Επειδή οι αλγόριθμοι ιεραρχικής ομαδοποίησης δεν έχουν καλή κλιμάκωση με μεγάλα δεδομένα, χρησιμοποιήθηκε ένα μικρότερο μέρος του dataset. Αν χρησιμοποιηθεί ολόκληρο το dataset, θα υπάρξει πρόβλημα μη επαρκής μνήμης.

Έτσι λοιπόν, φορτώνουμε τον πίνακα dataset, και παίρνουμε μόνο τα πρώτα 1000 δεδομένα:

# importing dataset from saveasnumpy  
dataset = np.load('dataset.npy')  
  
# reducing the size of dataset to the first 1000 ratings  
X = dataset[:-99000]

Χρησιμοποιώντας την βιβλιοθήκη, κάνουμε την ένωση των διανυσμάτων.

# creating links between ratings(vectors) based on ward linking  
Z = linkage(X, 'ward')

Ο τρόπος ένωσης που επιλέχθηκε ήταν ο ward, επειδή είναι πιο ακριβής στις αποστάσεις μεταξύ των διανυσμάτων και των clusters.

Χρησιμοποιώντας την βιβλιοθήκη matplotlib, δημιουργούμε και εμφανίζουμε το δενδρόγραμμα:

# calculating full dendrogram  
plt.figure(figsize=(25, 10))  
plt.title('Hierarchical Clustering Dendrogram')  
plt.xlabel('sample index')  
plt.ylabel('distance')  
dendrogram(  
 Z,  
 leaf\_rotation=90., # rotates the x axis labels  
 leaf\_font\_size=8., # font size for the x axis labels  
)  
plt.show()

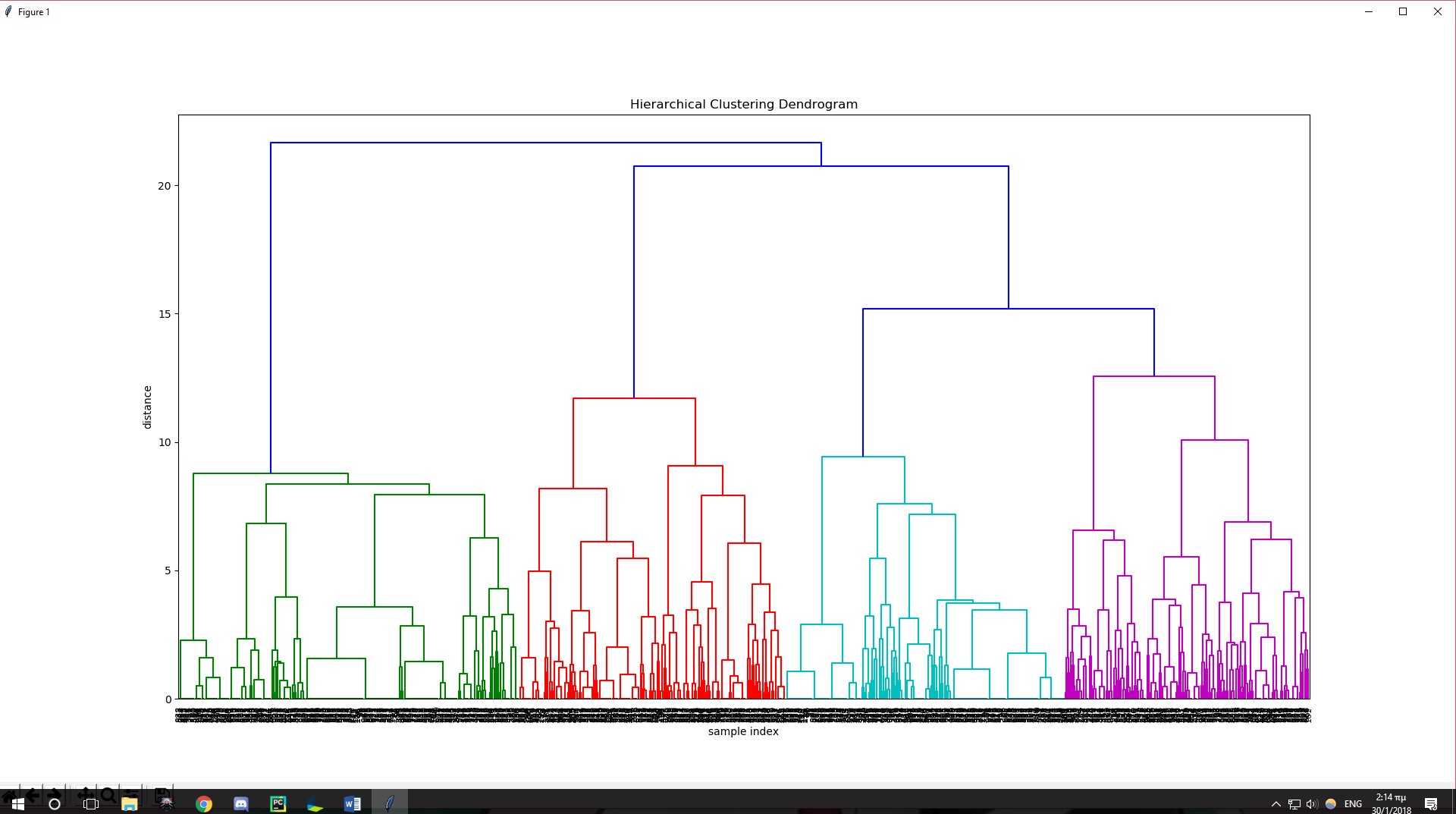
Με βάση τον αριθμό των cluster που βρήκαμε από τον BSAS, επιλέγουμε τα clusters του dataset.

# maximum number of clusters, based on BSAS algorithm  
k = 4  
  
# get clusters based on k  
clusters = fcluster(Z, k, criterion='maxclust')

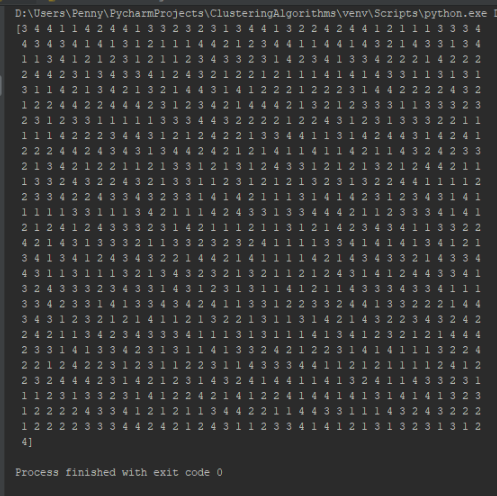
Τέλος, εμφανίζουμε σε ποιον αριθμό cluster ανήκει το κάθε διάνυσμα (labels):

# print on which number of clusters each rating belongs to  
print(clusters)

## Παραδειγματα εκτέλεσης κώδικα



Δενδρόγραμμα

]

Labels του dataset

## Συμπερασμα – Σημασια Ομαδοποιησησ

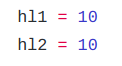
Το συμπέρασμα που μπορούμε να βγάλουμε από την ομαδοποίηση είναι ότι οι προτιμήσεις των χρηστών, χωρίζονται κυρίως σε 4 κατηγορίες. Δηλαδή, κάποιος μπορεί να πει, ότι οι διαφοροποίηση των χρηστών δεν είναι μεγαλύτερη από τις 4 και ότι οι χρήστες και οι προτιμήσεις τους είναι σχετικά όμοιες με το ¼ των υπόλοιπων χρηστών.

# Ερώτημα 4

Σύμφωνα με το 5-fold σχήμα δημιουργήσαμε 2 ταξινομητες. 'Εναν νευρωνικού δικτύου και εναν ελαχίστων τετραγώνων.

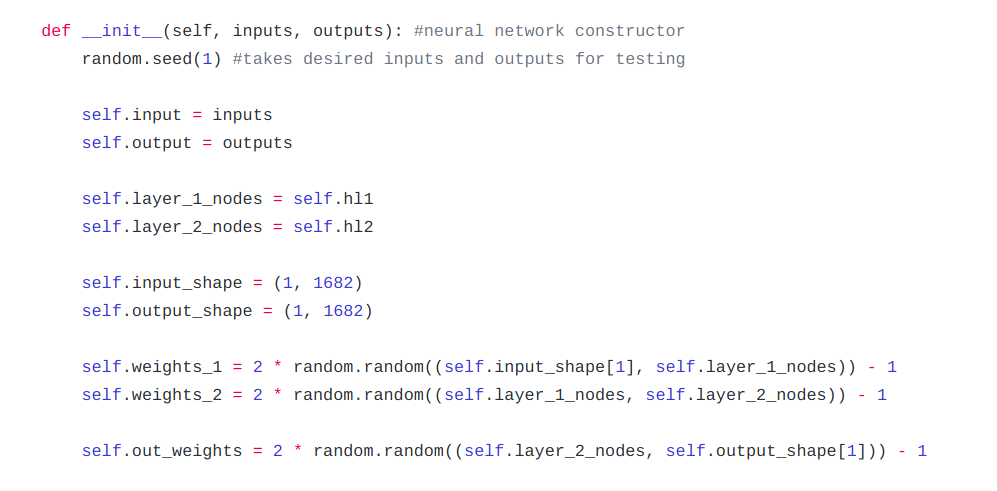
## Νευρωνικό Δίκτυο

Το νευρωνικό δίκτυο που σχεδιάσαμε περιεχει 2 κρυφά επίπεδα (hidden layers) των 10 επιπέδων το καθένα.



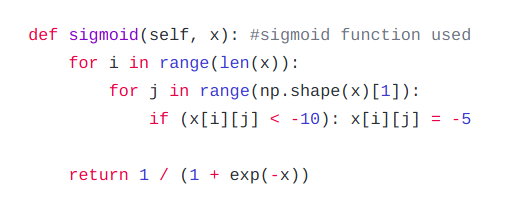
Γενικά, ο σκοπός του δικτύου μας είναι να δέχεται ως είσοδο έναν πίνακα με βαθμολογίες και να βγάζει ως έξοδο τον ίδιο πίνακα.

Η συνάρτηση της αρχικοποίησης του αντικειμένου είναι η εξής:

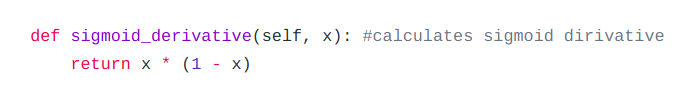


όπου αρχικοποιούνται τα βάρη με τυχαίες τιμές μεταξύ του -1 και του 1 αλλά και οι επιθυμητές διαστάσεις των εισόδων και εξόδων όπως είναι λογικό εφόσον κάθε χρήστης βαθμολογεί 1682 διαφορετικές ταινίες το πολύ.

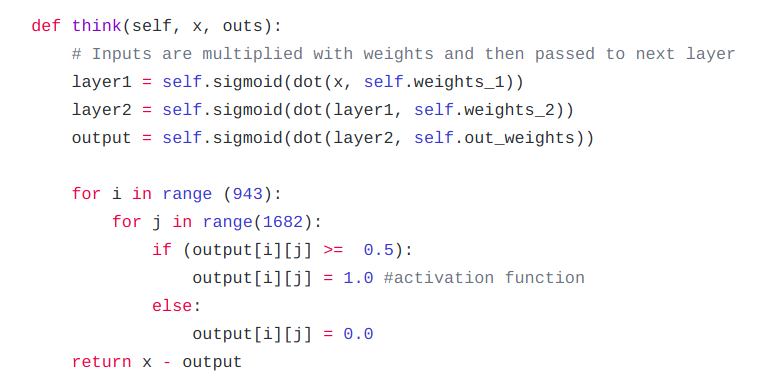
Μετά, ορίσαμε την συνάρτηση "sigmoid" η οποία κάνει κανονικοποίηση των δεδομένων ώστε να μην παίρνουν ποτέ τιμές μικρότερες του -10,



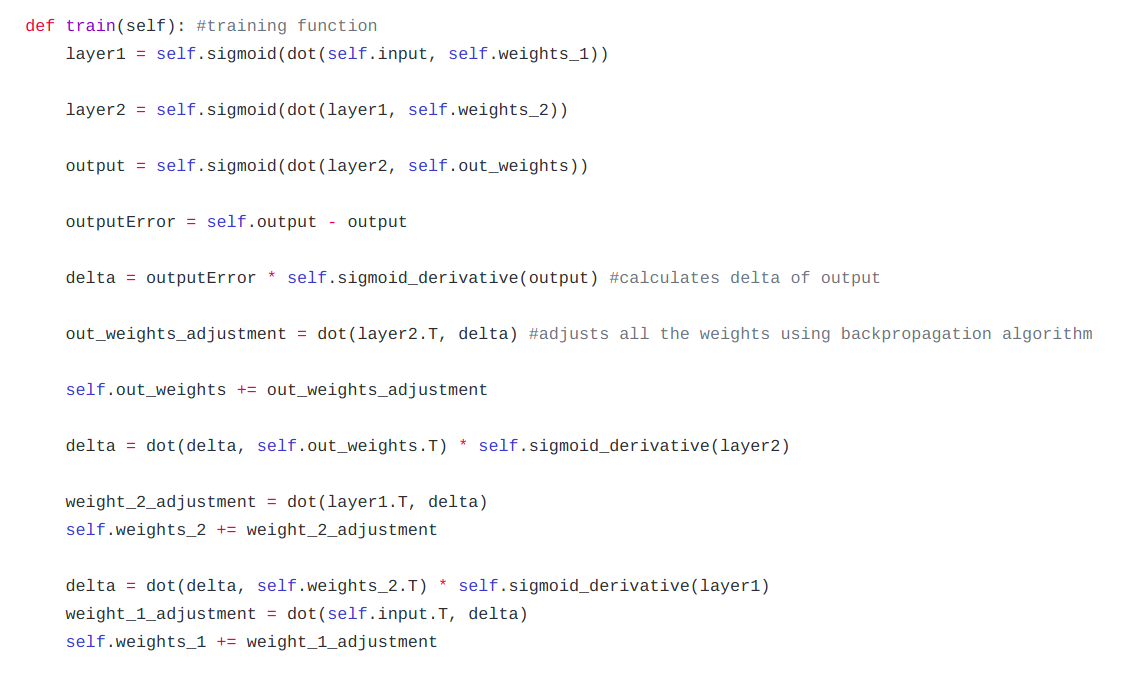
και την "sigmoid\_derivative" μέσω της οποίας υπολογίζεται η παράγωγος της απόστασης "X" απο τα δεδομένα μας.



Στην συνέχεια ορίσαμε την συνάρτηση "think". Η συνάρτηση αυτή κάνει μία 'πρόβλεψη' των αποτελεσμάτων του δικτύου. Με βάση αυτά τα αποτελέσματα βλέπουμε κατά πόσο είναι ακριβές το δίκτυό μας.

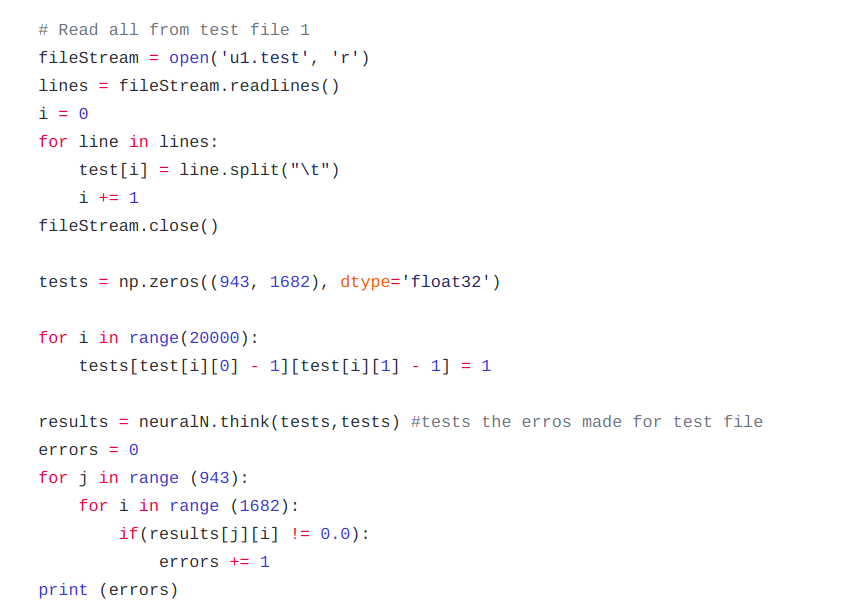


Η τελευταία συνάρτηση που ορίζεται, είναι η "train". Αυτή στην ουσία πραγματοποιεί τον σκοπό που πρέπει εφόσον είναι αυτή η οποία υπολογίζει τα αποτελέσματα τα οποία συγκρίνει με τα αποτελέσματα που εικάζονται να έρθουν, και με βάση τους υπολογισμούς αυτούς αλλάζει τις τιμές των βαρών με σκοπό να κινηθεί προς την σωστή κατεύθυνση.



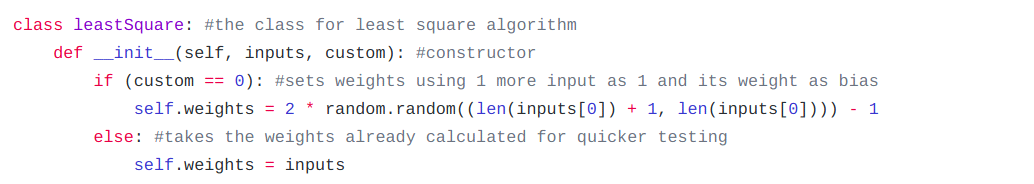
Τέλος, σας παρουσιάζουμε των κώδικα που χρησιμοποιεί τις παραπάνω συναρτήσεις στα δεδομένα του αρχείου "u1.base":



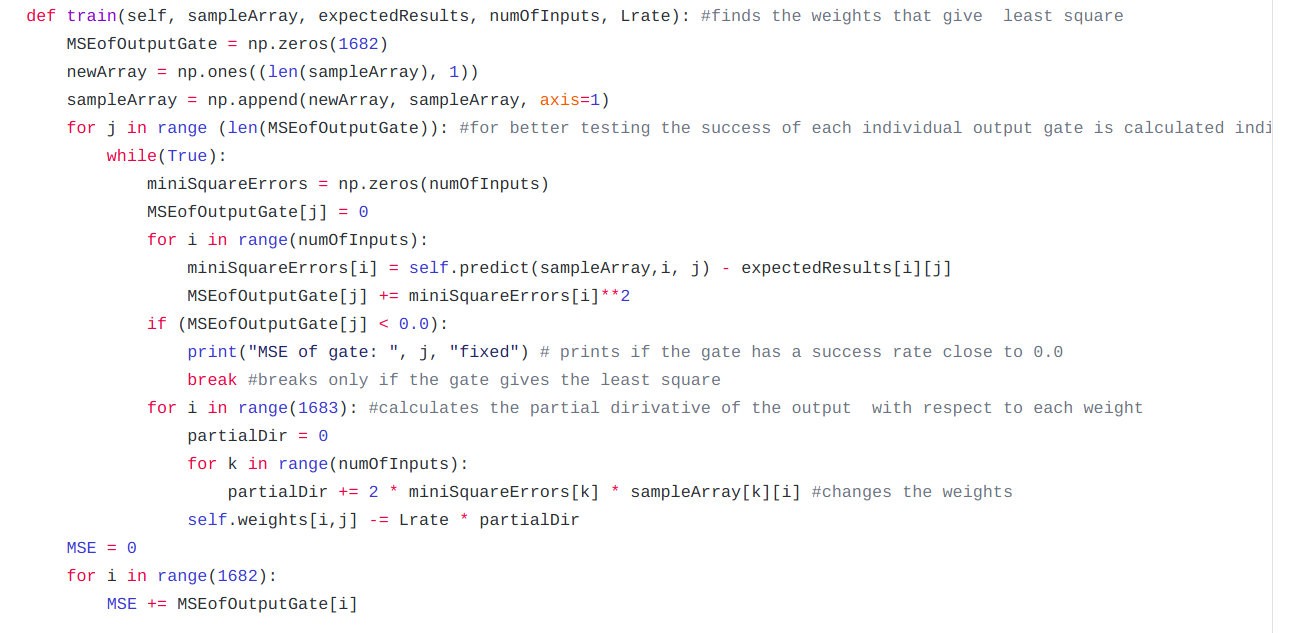


## ΤΑΞΙΝΟΜΗΤΗΣ ΕΛΑΧΙΣΤΩΝ ΤΕΤΡΑΓΩΝΩΝ

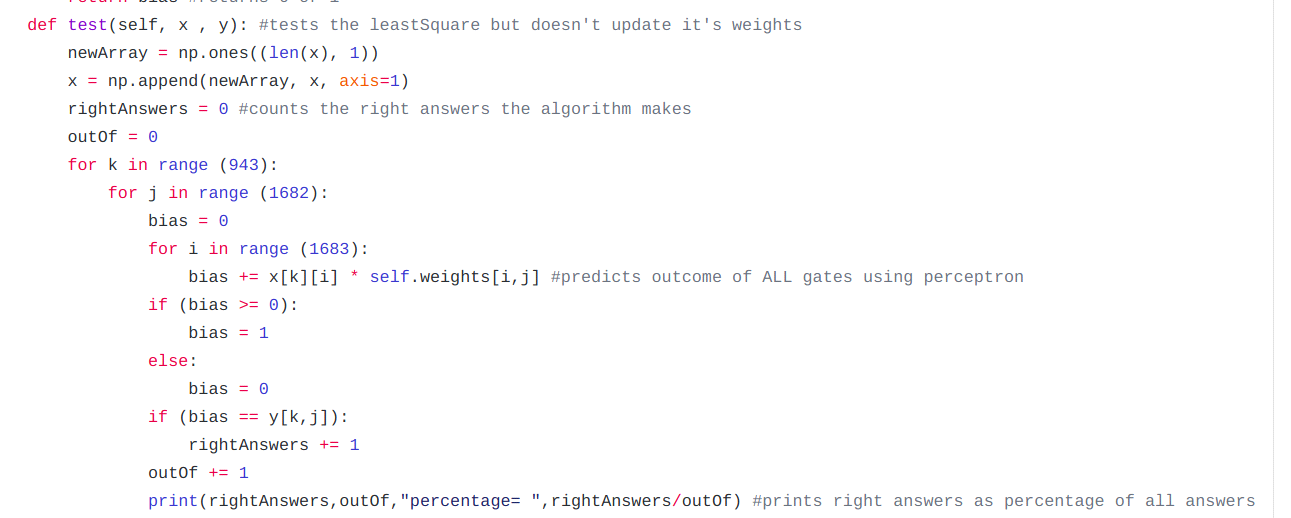
Για την υλοποίηση του εξής ταξινομητή κάναμε χρήση μιας αρχιτεκτονικής όμοιας με τον αλγόριθμο perceptron. Πιο λεπτομερώς, αφού έχουν ορισθεί και αρχικοποιηθεί τα βάρη, γίνεται μια πρόβλεψη των αποτελεσμάτων με βάση τον perceptron και τα λάθη που προκύπτουν διορθώνεται με βάση την απόσταση ελαχίστων τετραγώνων. Συγκεκριμένα, η αρχικοποίηση γινεται ως εξής:



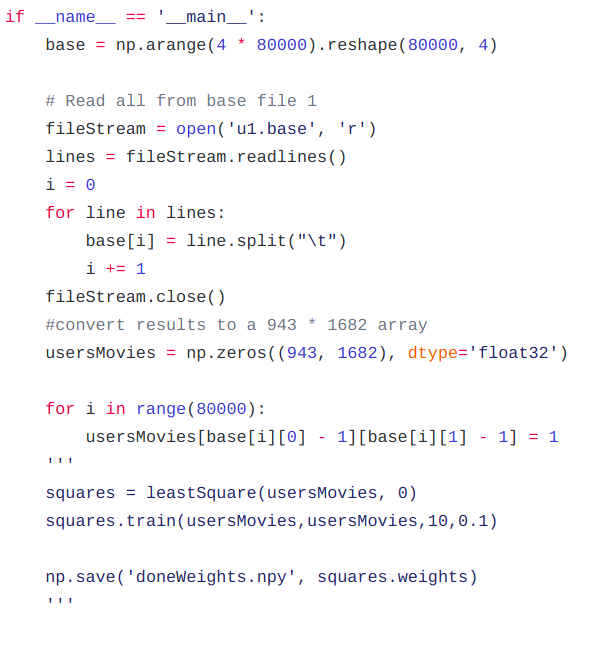
Ενώ η εκπαίδευση:

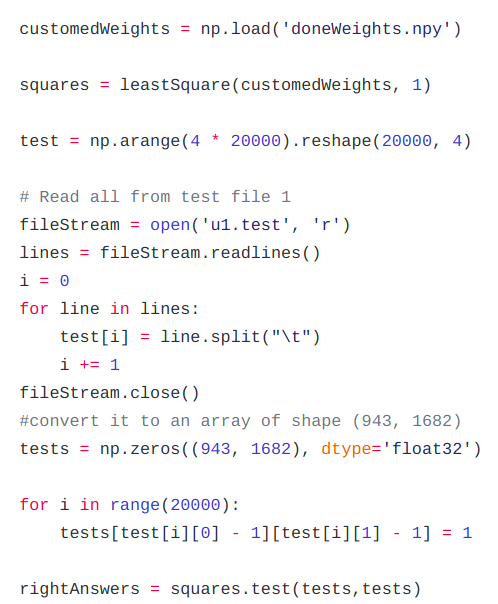


Για την αποτίμηση του ταξινομητή "γράψαμε" την συνάρτηση test:



Και φυσικά όλα τα παραπάνω τα κάναμε πράξη στην main συνάρτηση της τάξης.

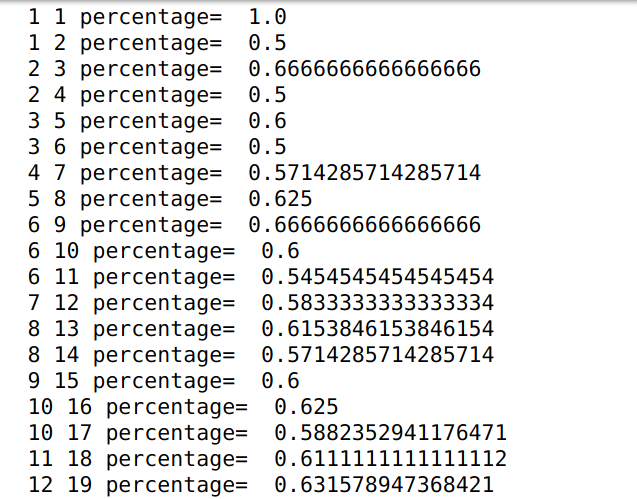


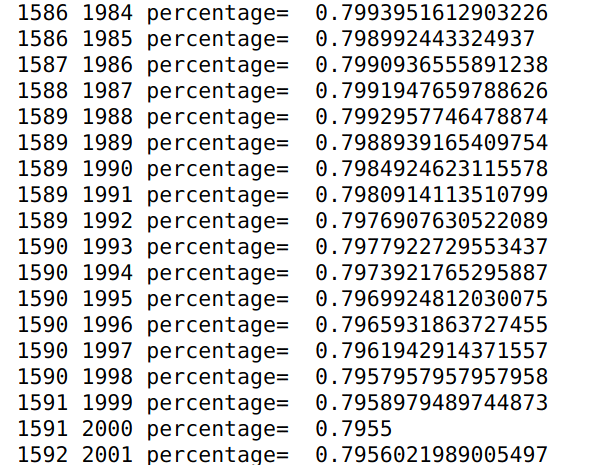


## Παράδειγμα Εκτέλεσης κώδικα



Νευρωνικό δίκτυο





Ταξινομητής Ελαχίστων Τετραγώνων