ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2019-2020

ΜΥΕ002 - ΑΝΑΓΝΩΡΙΣΗ ΠΡΟΤΥΠΩΝ

ΔΙΔΑΣΚΩΝ: ΜΠΛΕΚΑΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΑΝΑΦΟΡΑ ΔΕΥΤΕΡΗΣ ΕΡΓΑΣΙΑΣ

ΓΚΙΤΣΑΚΗΣ ΔΗΜΟΣ Α.Μ.:2425

ΚΡΟΜΜΥΔΑΣ ΓΕΩΡΓΙΟΣ Α.Μ.:3260

ΙΩΑΝΝΙΝΑ, 2020

ΕΙΣΑΓΩΓΗ

Στην παρούσα αναφορά αναλύεται η υλοποίηση μεθόδων ομαδοποίησης δεδομένων στα πλαίσια του μαθήματος της Αναγνώρισης Προτύπων. Για τις μεθόδους αυτές χρησιμοποιήθηκαν 2 διαφορετικά datasets, το spam dataset και το occupancy dataset. Για την υλοποίηση των μεθόδων χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python, καθώς και κάποιες βιβλιοθήκες της. Ο κώδικας για τις μεθόδους αυτές βρίσκονται στο αρχείο **ask2.py**. Είναι σημαντικό να αναφερθεί πως λόγω των συνθηκών έχουμε δουλέψει στους προσωπικούς μας υπολογιστές και κάποιες από τις βιβλιοθήκες που έχουμε χρησιμοποιήσει, ενδέχεται να μην είναι εγκατεστημένες στους υπολογιστές των εργαστηρίων του τμήματος.

ΜΕΡΟΣ Α: ΠΡΟΕΤΟΙΜΑΣΙΑ ΔΕΔΟΜΕΝΩΝ

Στο πρώτο μέρος αναλύεται η διαδικασία που ακολουθήθηκε για το διάβασμα των δεδομένων που χρησιμοποιήθηκαν. Για το spam dataset δημιουργήθηκε η συνάρτηση readDataSpam(), η οποία παίρνει σαν παράμετρο το όνομα του αρχείου και αφού διάβασει όλες τις γραμμές του, χωρίζει κάθε γραμμή στο κόμμα και τοποθετεί τα δεδομένα στον πίνακα data αφού τα μετατρέψει σε float. Επιστρέφει τον πίνακα data. Για το occupancy dataset, δημιουργήθηκε η συνάρτηση readDataOcc(), η οποία παίρνει σαν όρισμα το όνομα του αρχείου και όπως και η προηγούμενη διαβάζει όλες τις γραμμές του αρχείου και χωρίζει την κάθε γραμμή στο κόμμα και τοποθετεί τα στοιχεία κάθε γραμμής στον πίνακα data αφού τα μετατρέψει σε float. Η διαφορά με την προηγούμενη είναι ότι δεν χρησιμοποιεί όλες τις στήλες του αρχείου (πχ date) και αγνοεί την πρώτη γραμμή που περιέχει τους τίτλους της κάθε στήλης. Επιστρέφει τον πίνακα data.

Η τελευταία στήλη περιέχει τις σωστές κατηγορίες κάθε παραδείγματος. Παρόλα αυτά, επειδή έχουμε ομαδοποίηση και επομένως εκπαίδευση χωρίς επίβλεψη δεν πρέπει να συμπεριληφθούν οι σωστές κατηγορίες στην εκπαίδευση. Επομένως, τις τοποθετούμε σε δύο άλλους πίνακες και τις αφαιρούμε από τους αρχικούς.

ΜΕΡΟΣ Β: ΥΛΟΠΟΙΗΣΗ ΜΕΘΟΔΩΝ

K-Means

Για την πρώτη μέθοδο δημιουργήθηκε η συνάρτηση KMeansFunc(), η οποία παίρνει σαν παραμέτρους τον αριθμό K των clusters που επιθυμούμε, το σύνολο δεδομένων και τις σωστές κατηγορίες των δεδομένων. Για την υλοποίηση της χρησιμοποιήθηκε η συνάρτηση KMeans της βιβλιοθήκης sklearn, η οποία παίρνει σαν παραμέτρους τον αριθμό K, init = "random" ώστε να πάρει τυχαία κάποια από τα δεδομένα για αρχικά κέντρα καθώς και n_init=20 για να επαναλάβει 20 runs στα δεδομένα και να κρατήσει ως τελικά αυτά με τα καλύτερα αποτελέσματα. Έπειτα, υπολογίζουμε τον contingency matrix

οποίος είναι ένας 2xK πίνακας που αναδεικνύει την κατανομή των δεδομένων των clusters. Έτσι, το position του μεγαλύτερο στοιχείου κάθε στήλης στον y άξονα είναι και η πλειοψηφούσα κατηγορία. Με αυτό τον τρόπο αθροίζουμε τον αριθμό των παραδειγμάτων στις πλειοψηφούσες κατηγορίες και διαιρούμε με τον αριθμό των παραδειγμάτων και το αποτέλεσμα είναι η μετρική purity. Στην συνέχεια, έχοντας τις πλειοψηφούσες κατηγορίες, υπολογίζουμε τα True Negative, True Positive, False Positive και False Negative για κάθε cluster και υπολογίζουμε το F1 για κάθε cluster. Το άθροισμα όλων των F1, όλων των clusters είναι η μετρική Total F-Measure. Η κλήση της συνάρτησης βρίσκεται κάτω από αυτή για τα 2 διαφορετικά dataset και κάθε φορά απλά αλλάζει ο αριθμός των K ομάδων.

Agglomerative Hierarchical Clustering

Για την μέθοδο αυτή δημιουργήθηκε η συνάρτηση Agglomerative HierarchicalClustering(), η οποία παίρνει σαν παραμέτρους τον αριθμό K των clusters, το σύνολο δεδομένων και τις σωστές κατηγορίες του συνόλου δεδομένων. Χρησιμοποιεί την συνάρτηση AgglomerativeClustering() της βιβλιοθήκης sklearn με παραμέτρους n_clusters=K για τον αριθμό των clusters, affinity='euclidean' ώστε να χρησιμοποιεί ευκλείδειες αποστάσεις και linkage='ward' το οποίο χρησιμοποιείται για να ελαχιστοποιήσει την διακύμανση των clusters που συγχωνεύονται. Με βάση τα αποτελέσματα που υπολογίζει αυτή η συνάρτηση, ακριβώς με τον ίδιο τρόπο με την προηγούμενη υπολογίζει τις μετρικές Purity και Total F-Measure.

Spectral Clustering

Για την μέθοδο αυτή δημιουργήθηκε η συνάρτηση SpectralClusteringFunc(),η οποία παίρνει σαν παραμέτρους τον αριθμό K των clusters, το σύνολο δεδομένων και τις σωστές κατηγορίες του συνόλου δεδομένων. Χρησιμοποιεί την συνάρτηση SpectralClustering() της βιβλιοθήκης sklearn με παραμέτρους n_clusters=K για τον αριθμό των clusters και affinity = 'rbf' για το είδος kernel που θέλουμε να χρησιμοποιήσουμε. Έτσι, για cosine kernel αρκεί να αλλαχτεί το rbf σε cosine. Με βάση τα αποτελέσματα που υπολογίζει αυτή η συνάρτηση, ακριβώς με τον ίδιο τρόπο με τις προηγούμενες υπολογίζει τις μετρικές Purity και Total F-Measure. Κατά την εκτέλεση με τα αρχικά δεδομένα παίρναμε ένα μήνυμα λάθους σχετικά με το ότι το γράφημα δεν είναι fully connected. Για αυτό το λόγο κανονικοποιήσαμε τα δεδομένα με την συνάρτηση normalize() και στην συνέχεια χρησιμοποιήσαμε τα κανονικοποιημένα δεδομένα για το οποία η μέθοδος δούλεψε. Όσο αφορά την εκτίμηση του βέλτιστου αριθμού ομάδων K και του διαγράμματος με τις τιμές των ιδιοτιμών του πίνακα Laplacian, δημιουργήθηκε η συνάρτηση eigenDecomposition(), η οποία παίρνει σαν παραμέτρους τον affinity_matrix που υπολογίζεται από την συνάρτηση SpectralClustering().fit() και μια Boolean μεταβλητή που ανάλογα με την τιμή της επιτρέπει ή όχι την εμφάνιση του γραφήματος. Αυτή η συνάρτηση υπολογίζει τον πίνακα Laplacian καθώς και τις ιδιοτιμές και τα ιδιοδιανύσματά του και με βάση αυτά κρατά την θέση της μεγαλύτερης διαφοράς μεταξύ 2 διαδοχικών ιδιοτιμών. Η θέση αυτή

είναι η εκτίμηση για τον βέλτιστο αριθμό ομάδων για συγκεκριμένο dataset και συγκεκριμένο kernel. Τα αποτελέσματα της εκτίμησης καθώς και τα purity και F-Measure για την εκτίμηση αυτή παρουσιάζονται στον παρακάτω πίνακα μαζί με τα αποτελέσματα των προηγούμενων μεθόδων για κάθε dataset.

ΑΠΟΤΕΛΕΣΜΑΤΑ ΜΕΘΟΔΩΝ

Spambase DataSet

Methods	**Number of Clusters (K)**		**Purity**		**F-Measure**	
K-Means	2		0.635949		0.878161	
	4		0.670506		2.517044	
	8		0.694197		4.952980	
Agglomerative Hierarchical Clustering	2		0.661595		0.841889	
	4		0.661595		1.669016	
	8		0.696153		3.368326	
Spectral Clustering	Best K Found (rbf kernel) 32	Best K Found (cosine kernel) 4600	(rbf kernel) 0.770267	(cosine kernel) 1.000000	(rbf kernel) 8.695408	(cosine kernel) 0.000000

Σύμφωνα με τον παραπάνω πίνακα, για το Spambase Dataset, η καλύτερη μέθοδος με βάση το F-Measure είναι η K-Means με K=4, ενώ με βάση το Purity η καλύτερη μέθοδος είναι η Spectral Clustering.

Occupancy DataSet

Methods	**Number of Clusters (K)**		**Purity**		**F-Measure**	
K-Means	2		0.933583		0.945761	
	4		0.978236		1.929257	
	8		0.978612		3.855080	
Agglomerative Hierarchical Clustering	2		0.841889		0.972000	
	4		0.978987		1.922020	
	8		0.978987		4.828698	
Spectral Clustering	Best K Found (rbf kernel) 2	Best K Found (cosine kernel) 2	(rbf kernel) 0.891932	(cosine kernel) 0.940338	(rbf kernel) 0.927835	(cosine kernel) 0.939577

Σύμφωνα με τον παραπάνω πίνακα, για το Occupancy Dataset, η καλύτερη μέθοδος με βάση το F-Measure και το Purity είναι η Agglomerative Hierarchical Clustering με K=8.

Σημείωση

Για το spambase dataset, στην μέθοδο Spectral Clustering και με cosine kernel, ο αριθμός του best k που υπολογίζει η μέθοδος είναι 4600, αριθμός πολύ μεγάλος και φυσικά χρονοβόρος. Δοκιμάσαμε και τρέξαμε την μέθοδο για k=4600 και πήραμε αποτελέσματα μετά από πολύ ώρα. Τα αποτελέσματα παρουσιάζονται στον παραπάνω πίνακα, ωστόσο δεν είναι εύκολη η επανάληψη της διαδικασίας για αυτό και αφήνουμε το k=2. Επίσης, αντιμετωπίσαμε ένα ζήτημα με την εμφάνιση του plot με το διάγραμμα με τις τιμές των ιδιοτιμών του πίνακα Laplacian και για αυτό το λόγο στην συνάρτηση eigenDecomposition() δίνουμε στην παράμετρο Plot την τιμή False ώστε να μην εμφανίζει μηνύματα λάθους κατά την εκτέλεση του προγράμματος και να μπορεί να τρέξει.

ΠΑΡΑΡΤΗΜΑ

Παρακάτω παρουσιάζεται ο κώδικας που γράψαμε για τις μεθόδους που αναφέρθηκαν.

```
#Gkitsakis Dimos 2425
#Krommydas Georgios 3260
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.spatial import distance
import scipy.cluster.hierarchy as shc
from sklearn.cluster import KMeans, AgglomerativeClustering, SpectralClustering
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd
from numpy import linalg as LA
from scipy.sparse import csgraph



def readDataSpam(filename):
    infile = open(filename)
    data = infile.readlines()
    for i in range(len(data)):
        data[i] = data[i].split(",")
        for j in range(len(data[i])):
            data[i][j] = float(data[i][j])

    return data
```

```
def readDataOcc(filename):
    infile = open(filename)
    data = infile.readlines()
    data = data[1:]
    for i in range(len(data)):
        data[i] = data[i].split(",")
        del data[i][0]
        del data[i][0]
        for j in range(len(data[i])):
            data[i][j] = float(data[i][j])

    return data



spamData = readDataSpam("spambase.data")
occData = readDataOcc("datatest.txt")
rightSpamData = []
rightOccData = []
for i in range(len(spamData)):
    rightSpamData.append(spamData[i][len(spamData[i])-1])
    spamData[i] = spamData[i][:len(spamData[i])-1]

for i in range(len(occData)):
    rightOccData.append(occData[i][len(occData[i])-1])
    occData[i] = occData[i][:len(occData[i])-1]
```

```
def KMeansFunc(K, dataset, rightdataset):
    kmeans = KMeans(n_clusters=K, init = "random",  n_init=20)
    kmeans.fit(dataset)
    contingency_matrix = metrics.cluster.contingency_matrix(rightdataset, kmeans.labels_)
    purity = np.sum(np.amax(contingency_matrix, axis=0)) / len(dataset)
    print ("Purity for %d Clusters is: %f" %(K, purity))

    #Gia thn pleiopsifia se kathe cluster
    clustersCategories = []
    for i in range(K):

        if contingency_matrix[0][i] > contingency_matrix[1][i]:
            clustersCategories.append(0)
        else:
            clustersCategories.append(1)

    #Gia to F-Measure
    TotalFMeasure = 0
    for i in range(K): #Gia kathe K
        TruePositive = 0
        TrueNegative = 0
        FalsePositive = 0
        FalseNegative = 0
        for j in range(len(dataset)): #Gia kathe paradeigma
            label = kmeans.labels_[j] #Krata to label tou paradeigmatos sumfwna me ton
kmeans
```

```
            if (label != i): #an den einai idio me to cluster pou eksetazoume
                continue
            else: #an einai idio
                if rightdataset[j] == clustersCategories[label] and clustersCategories[label] == 1:
                    TruePositive = TruePositive + 1
                elif rightdataset[j] == clustersCategories[label] and clustersCategories[label] ==
0:
                    TrueNegative = TrueNegative + 1
                elif rightdataset[j] != clustersCategories[label] and clustersCategories[label] ==
1:
                    FalsePositive = FalsePositive + 1
                elif rightdataset[j] != clustersCategories[label] and clustersCategories[label] ==
0:
                    FalseNegative = FalseNegative + 1

        if TruePositive != 0 and FalsePositive !=0:
            precision = TruePositive / (TruePositive + FalsePositive)
            recall = TruePositive / (TruePositive + FalseNegative)
            F1 = 2 / ((1/precision) + (1/recall))
        else:
            precision =0
            recall = 0
            F1 = 0

        TotalFMeasure = TotalFMeasure + F1

    print ("Total F-Measure for %d Clusters is: %f" %(K, TotalFMeasure))
```

```
print("KMeans for Spambase Data:")
KMeansFunc(8, spamData, rightSpamData)
print("\n")
print("KMeans for Occupancy Data:")
KMeansFunc(8, occData, rightOccData)
print("\n")


def AgglomerativeHierarchicalClustering(K, dataset, rightdataset):
    cluster = AgglomerativeClustering(n_clusters=K, affinity='euclidean', linkage='ward')
    cluster.fit_predict(dataset)

    contingency_matrix = metrics.cluster.contingency_matrix(rightdataset, cluster.labels_)
    purity = np.sum(np.amax(contingency_matrix, axis=0)) / len(dataset)
    print ("Purity for %d Clusters is: %f" %(K, purity))
    #Gia thn pleiopsifia se kathe cluster
    clustersCategories = []
    for i in range(K):

        if contingency_matrix[0][i] > contingency_matrix[1][i]:
            clustersCategories.append(0)
        else:
            clustersCategories.append(1)

    #Gia to F-Measure
    TotalFMeasure = 0
    for i in range(K): #Gia kathe K
        TruePositive = 0
```

```
    TrueNegative = 0
    FalsePositive = 0
    FalseNegative = 0
    for j in range(len(dataset)): #Gia kathe paradeigma
        label = cluster.labels_[j] #Krata to label tou paradeigmatos sumfwna me ton kmeans
        if (label != i): #an den einai idio me to cluster pou eksetazoume
            continue
        else: #an einai idio
            if rightdataset[j] == clustersCategories[label] and clustersCategories[label] == 1:
                TruePositive = TruePositive + 1
            elif rightdataset[j] == clustersCategories[label] and clustersCategories[label] == 0:
                TrueNegative = TrueNegative + 1
            elif rightdataset[j] != clustersCategories[label] and clustersCategories[label] == 1:
                FalsePositive = FalsePositive + 1
            elif rightdataset[j] != clustersCategories[label] and clustersCategories[label] == 0:
                FalseNegative = FalseNegative + 1

    if TruePositive != 0 and FalsePositive !=0:
        precision = TruePositive / (TruePositive + FalsePositive)
        recall = TruePositive / (TruePositive + FalseNegative)
        F1 = 2 / ((1/precision) + (1/recall))
    else:
        precision =0
        recall = 0
        F1 = 0
```

```
        TotalFMeasure = TotalFMeasure + F1

    print ("Total F-Measure for %d Clusters is: %f" %(K, TotalFMeasure))

print("Agglomerative Hierarchical Clustering for Spambase Data:")
AgglomerativeHierarchicalClustering(8, spamData, rightSpamData)
print("\n")
print("Agglomerative Hierarchical Clustering for Occupancy Data:")
AgglomerativeHierarchicalClustering(8, occData, rightOccData)
print("\n")

def normalize(dataset):
    array = dataset
    temp = np.amax(dataset, axis =0)
    temp = temp.tolist()
    for i in range(0, len(array)):
        for j in range(0, len(array[i])):
            array[i][j] = np.around(array[i][j] / temp[j], decimals =5)
    return array

spamDataNorm = normalize(spamData)
occDataNorm = normalize(occData)

def eigenDecomposition(AffMatrix, plot = False):
    # Compute Laplacian matrix, L = D - A
    L = csgraph.laplacian(AffMatrix)
    # Compute eigenvalues from |L- uI| = 0 and eigenvectors from Lx = ux
```

```
    eigenvalues, eigenvectors = LA.eigh(L)

    # A plot for the eigenvalues of the affinity matrix
    if plot:
        plt.figure(figsize=(14, 6))
        plt.title("Largest eigenvalues of input matrix")
        plt.scatter(np.arange(np.round(eigenvalues, decimals = 2)), np.round((eigenvalues),
decimals = 2))
        plt.grid()
        plt.show()
    # Compute the largest eigengap
    index_largest_gap = np.argmax(np.diff(np.round((eigenvalues), decimals = 2)))
    # Find the optimal number of clusters from the maximum eigengap
    nb_clusters = index_largest_gap + 2
    return nb_clusters, eigenvalues, eigenvectors

def SpectralClusteringFunc(K, dataset, rightdataset):
    cluster = SpectralClustering(n_clusters=K, affinity = 'rbf')
    cluster.fit(dataset)
    #print(cluster.labels_)



    affinity_matrix = cluster.affinity_matrix_
    k, _, _ = eigenDecomposition(affinity_matrix)
    print(f'Optimal number of clusters are: {k}')



    contingency_matrix = metrics.cluster.contingency_matrix(rightdataset, cluster.labels_)
```

```
    purity = np.sum(np.amax(contingency_matrix, axis=0)) / len(dataset)
    print("Purity for %d Clusters is: %f" % (K, purity))

    # Gia thn pleiopsifia se kathe cluster
    clustersCategories = []
    for i in range(K):

        if contingency_matrix[0][i] > contingency_matrix[1][i]:
            clustersCategories.append(0)
        else:
            clustersCategories.append(1)

    # Gia to F-Measure
    TotalFMeasure = 0
    for i in range(K):  # Gia kathe K
        TruePositive = 0
        TrueNegative = 0
        FalsePositive = 0
        FalseNegative = 0
        for j in range(len(dataset)):  # Gia kathe paradeigma
            label = cluster.labels_[j]  # Krata to label tou paradeigmatos sumfwna me ton kmeans
            if (label != i):  # an den einai idio me to cluster pou eksetazoume
                continue
            else:  # an einai idio
                if rightdataset[j] == clustersCategories[label] and clustersCategories[label] == 1:
                    TruePositive = TruePositive + 1
                elif rightdataset[j] == clustersCategories[label] and clustersCategories[label] == 0:
```

```
                TrueNegative = TrueNegative + 1
            elif rightdataset[j] != clustersCategories[label] and clustersCategories[label] ==
1:
                FalsePositive = FalsePositive + 1
            elif rightdataset[j] != clustersCategories[label] and clustersCategories[label] ==
0:
                FalseNegative = FalseNegative + 1

        if TruePositive != 0 and FalsePositive != 0:
            precision = TruePositive / (TruePositive + FalsePositive)
            recall = TruePositive / (TruePositive + FalseNegative)
            F1 = 2 / ((1 / precision) + (1 / recall))
        else:
            precision = 0
            recall = 0
            F1 = 0

        TotalFMeasure = TotalFMeasure + F1
    print("Total F-Measure for %d Clusters is: %f" % (K, TotalFMeasure))

print("Spectral Clustering for Spambase Data:")
SpectralClusteringFunc(32, spamDataNorm, rightSpamData)
print("\n")
print("Spectral Clustering for Occupancy Data:")
SpectralClusteringFunc(2, occDataNorm, rightOccData)
```