ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΜΥΕ002 - ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ
ΑΣΚΗΣΗ 2

Κίτσιος Κωνσταντίνος 4388
Λιόντος Αναστάσιος 4409

June 20, 2021

Contents

1 K-means clustering **3**

2 Hierarchical clustering **7**

3 K-medoids algorithm **12**

4 Βοηθητική συνάρτηση histogram **16**

5 Βοηθητική συνάρτηση reduce **16**

6 Η βέλτιστη μέθοδος **16**

1 K-means clustering

```
import idx2numpy as idx2numpy
import numpy as np
import Histogram
from sklearn.metrics import confusion_matrix


trainImagePath = "../dataset/train-images-idx3-ubyte"
trainLabelPath = "../dataset/train-labels-idx1-ubyte"

testImagePath = "../dataset/t10k-images-idx3-ubyte"
testLabelPath="../dataset/t10k-labels-idx1-ubyte"

x_train = idx2numpy.convert_from_file(trainImagePath)
y_train = idx2numpy.convert_from_file(trainLabelPath)

x_test = idx2numpy.convert_from_file(testImagePath)
y_test = idx2numpy.convert_from_file(testLabelPath)

x_train=x_train.reshape((x_train.shape[0], 28*28))
x_test=x_test.reshape((x_test.shape[0], 28*28))

x_train=x_train/255.0
x_test=x_test/255.0

k=10; # number of clusters
centroids = {}
def fit(data, distance):

    for i in range(k):
        centroids[i] = data[i]

    while(True):
        classifications = {}

        for i in range(k):
            classifications[i] = []

        for featureset in data:
            if (distance=='Euclidean'):
                distances = [np.linalg.norm(featureset-
centroids[centroid]) for centroid in  centroids]
            elif (distance == 'Manhattan'):
                distances = [np.linalg.norm(featureset -
centroids[centroid], ord=1) for centroid in centroids
]
            elif (distance == 'Cosine'):
                distances = []
                for centroid in centroids:
                    dotP = np.dot(featureset, centroids[
centroid].T)
                    if (isinstance(centroids[centroid],
np.float64)):dotP = 0
                    norm1 = np.linalg.norm(featureset)
                    if (np.isnan(norm1).any()): norm1 =
10**10
                    norm2 = np.linalg.norm(centroids[
centroid])
                    if (np.isnan(norm2).any()):  norm2 =
```

```
    10**10
52                      distances.append(dotP/(norm2*norm1))
53
54
55
56              minn=np.amin(distances)
57              classification = distances.index(minn)
58              classifications[classification].append(
    featureset)
59
60          prev_centroids = dict(centroids)
61
62          for classification in  classifications:
63              centroids[classification] = np.average(
    classifications[classification],axis=0)
64
65          optimized = True
66
67          for c in  centroids:
68              original_centroid = prev_centroids[c]
69              current_centroid =  centroids[c]
70              if np.sum((current_centroid-original_centroid
    )/original_centroid*100.0) > 0.001:
71                  optimized = False
72
73          if optimized:
74              break
75
76  def predict(data):
77      classification = []
78      for d in data:
79          distances = [np.linalg.norm(d-centroids[centroid
    ]) for centroid in centroids]
80          classification.append(distances.index(min(
    distances)))
81      return classification
82
83  def purity(y_true,y_pred):
84          np.seterr(divide='ignore', invalid='ignore')
85          cm = confusion_matrix(y_true, y_pred, labels=[i
    for i in range(10)])
86          return np.sum(np.amax(cm,axis=0)) / np.sum(cm)
87
88  def f_measure(y_true,y_pred):
89      np.seterr(divide='ignore', invalid='ignore')
90      cm = confusion_matrix(y_true, y_pred, labels=[i for i
      in range(10)])
91      tp = np.diag(cm)
92      fp=cm.sum(axis=0) - np.diag(cm)
93      fn = cm.sum(axis=1) - np.diag(cm)
94      tn=cm.sum()-(tp+fn+fp)
95      recall = tp/(tp+fn)
96      precision = tp/(tp+fp)
97      f1 = 2*((precision*recall)/(precision+recall))
98      f1[np.isnan(f1)] = 0
99      return f1.sum()
100 print ("----------------------Kmeans clustering
    algorithm----------------------------------")
101 print("#########################R1-Data Represantation
    ############################")
102 fit(x_train, "Euclidean")
```

```
classes = predict(y_test)
print("-----------------Euclidean------------------")
print("Purity score {:.3f}%".format(purity(y_test,np.
    array(classes))*100))
print("F-measure score {:.3f}%".format(f_measure(y_true=
    y_test, y_pred=np.array(classes))*100))
fit(x_train, "Manhattan")
classes = predict(y_test)
print("-----------------Manhattan------------------")
print("Purity score {:.3f}%".format(purity(y_test,np.
    array(classes))*100))
print("F-measure score {:.3f}%".format(f_measure(y_true=
    y_test, y_pred=np.array(classes))*100))
fit(x_train, "Cosine")
classes = predict(y_test)
print("-----------------Cosine------------------")
print("Purity score {:.3f}%".format(purity(y_test,np.
    array(classes))*100))
print("F-measure score {:.3f}%".format(f_measure(y_true=
    y_test, y_pred=np.array(classes))*100))

print("########################R2-Data Represantation
    ############################")

histArray=Histogram.histogram(x_train,32)
testhistArray=Histogram.histogram(x_test,32)

fit(histArray, "Euclidean")
clusters=predict(testhistArray)
print("-----------------Euclidean------------------")
print("Purity score {:.3f}%".format(purity(y_test,np.
    array(clusters))*100))
print("F-measure score {:.3f}%".format(f_measure(y_true=
    y_test, y_pred=np.array(clusters))*100))

fit(histArray, "Manhattan")
clusters=predict(testhistArray)
print("-----------------Manhattan------------------")
print("Purity score {:.3f}%".format(purity(y_test,np.
    array(clusters))*100))
print("F-measure score {:.3f}%".format(f_measure(y_true=
    y_test, y_pred=np.array(clusters))*100))

fit(histArray, "Cosine")
clusters=predict(testhistArray)
print("-----------------Cosine------------------")
print("Purity score {:.3f}%".format(purity(y_test,np.
    array(clusters))*100))
print("F-measure score {:.3f}%".format(f_measure(y_true=
    y_test, y_pred=np.array(clusters))*100))
```

Αρχικά, αρχικοποιούμε τα κέντρα επιλέγοντας τις πρώτες k είκονες από το σύνολο δεδομένων. Στη συνέχεια, για κάθε πρότυπο, βρίσκουμε ποιο κέντρο βρίσκεται πιο κόντα σε αυτό το πρότυπο με βάση μία από τις τρεις συναρτήσεις απόστασης και το τοποθετούμε στη συστάδα του κέντρου αυτού. Έπειτα, ενημερώνουμε την τιμή κάθε κέντρου, θέτοντάς το ίσο με τη μέση τιμή των προτύπων που ανήκουν στη συστάδα του κέντρου αυτού, επαναλαμβάνοντας αυτή τη διαδικασία εώς ότου να μην υπάρχει αλλαγή στα κέντρα C_k. Τα αποτελέσματα της μεθόδου K-means

για τις ζητούμενες αποστάσεις και τις ζητούμενες αναπαραστάσεις των δεδομένων είναι τα εξής:

```
----------Kmeans clustering algorithm----------
##############################R1-Data Represantation##############################
----------Euclidean----------
Purity score 20.000%
F-measure score 20.000%
----------Manhattan----------
Purity score 20.000%
F-measure score 20.000%
----------Cosine----------
Purity score 10.000%
F-measure score 18.182%
##############################R2-Data Represantation##############################
----------Euclidean----------
Purity score 31.020%
F-measure score 75.881%
----------Manhattan----------
Purity score 27.990%
F-measure score 80.428%
----------Cosine----------
Purity score 10.000%
F-measure score 18.182%
```

2 Hierarchical clustering

```
import math
from sklearn.metrics import confusion_matrix
import numpy as np
import idx2numpy as idx2numpy
import Histogram, ReduceDataSet


class Hierarchical:
  def __init__(self,data):
    '''
    constructor of the class, it takes the main data
     frame as input
    '''
    self.data = data
    self.n_samples, self.n_features = data.shape

  def DistanceMatrix(self,data,distance):
    '''
    arguement
    ---------
    data - the dataset whose Similarity matrix we are
     going to calculate
    returns
    -------
    the distance matrix.
    '''
    N = len(data)
    if (distance == "Cosine"):
      similarity_mat = np.ones([N, N]) #for cosine np.
     ones
    else:
      similarity_mat = np.zeros([N, N]) #for cosine np.
     ones
    for i in range(N):
      for j in range(N):
        similarity_mat[i][j]=self.SimilarityMeasure(data[
     i],data[j],distance)
    return similarity_mat

  def SimilarityMeasure(self,data1, data2, distance):
    '''
    arguements
    ----------
    data1, data2 - vectors, between which we are going to
      calculate similarity
    returns
    -------
    distance between the two vectors
    '''
    N=self.n_features
    if distance=='Euclidean':
      # for L2 norm or pythagorean distance
      dist = 0
      for i in range(N):
        dist += (data1[i] - data2[i])**2

      dist = math.sqrt(dist)
      return dist
```

```
53
54      if distance=='Cosine':
55        # form cosine similarity = a.b/|a|.|b|
56        dot_prod = 0
57        data1_mod = np.linalg.norm(data1)
58        data2_mod = np.linalg.norm(data2)
59        for x in range(N):
60          dot_prod += data1[x]*data2[x]
61        return (1-dot_prod/(data1_mod*data2_mod))
62
63      if distance=='Manhattan':
64        # for L1 norm or pythagorean distance
65        dist = 0
66        for i in range(N):
67          dist += abs(data1[i] - data2[i] )
68        return dist
69
70    def fit(self,k,distance):
71      '''
72      this method uses the main Divisive Analysis algorithm
         to do the clustering
73      arguements
74      ----------
75      k - integer
76              number of clusters we want
77
78      returns
79      -------
80      cluster_labels - numpy array
81               an array where cluster number of a sample
        corrosponding to
82               the same index is stored
83      '''
84      similarity_matrix = self.DistanceMatrix(self.data,
        distance) # similarity matrix of the data
85      clusters = [list(range(self.n_samples))]      # list
        of clusters, initially the whole dataset is a single
        cluster
86      while True:
87        c_diameters = [np.max(similarity_matrix[cluster][:,
          cluster]) if (len(similarity_matrix[cluster][:,
          cluster]) != 0) else -1 for cluster in clusters]  #
          cluster diameters
88        max_cluster_dia = np.argmax(c_diameters)  #maximum
          cluster diameter
89        max_difference_index = np.argmax(np.mean(
          similarity_matrix[clusters[max_cluster_dia]][:,
          clusters[max_cluster_dia]], axis=1))
90        splinters = [clusters[max_cluster_dia][
          max_difference_index]] #spinter group
91        last_clusters = clusters[max_cluster_dia]
92        del last_clusters[max_difference_index]
93        while True:
94          split = False
95          for j in range(len(last_clusters))[::-1]:
96            splinter_distances = similarity_matrix[
          last_clusters[j], splinters]
97            last_distances = similarity_matrix[
          last_clusters[j], np.delete(last_clusters, j, axis=0)
          ]
98            if np.mean(splinter_distances) <= np.mean(
```

```
    last_distances):
99              splinters.append(last_clusters[j])
100             del last_clusters[j]
101             split = True
102             break
103         if split == False:
104           break
105       del clusters[max_cluster_dia]
106       clusters.append(splinters)
107       clusters.append(last_clusters)
108       if len(clusters) == k:
109         break
110
111     cluster_labels = np.zeros(self.n_samples)
112     for i in range(len(clusters)):
113       cluster_labels[clusters[i]] = i
114
115     return (clusters, cluster_labels)
116
117
118   def purity(self,y_true,y_pred):
119     np.seterr(divide='ignore', invalid='ignore')
120     cm = confusion_matrix(y_true, y_pred, labels=[i for i
        in range(10)])
121     return np.sum(np.amax(cm,axis=0)) / np.sum(cm)
122
123   def f_measure(self,y_true,y_pred):
124     np.seterr(divide='ignore', invalid='ignore')
125     cm = confusion_matrix(y_true, y_pred, labels=[i for i
        in range(10)])
126     tp = np.diag(cm)
127     fp=cm.sum(axis=0) - np.diag(cm)
128     fn = cm.sum(axis=1) - np.diag(cm)
129     tn=cm.sum()-(tp+fn+fp)
130     recall = tp/(tp+fn)
131     precision = tp/(tp+fp)
132     f1 = 2*((precision*recall)/(precision+recall))
133     f1[np.isnan(f1)] = 0
134     return f1.sum()
135
136
137 trainImagePath = "../dataset/train-images-idx3-ubyte"
138 trainLabelPath = "../dataset/train-labels-idx1-ubyte"
139 testImagePath = "../dataset/t10k-images-idx3-ubyte"
140 testLabelPath="../dataset/t10k-labels-idx1-ubyte"
141
142 x_train = idx2numpy.convert_from_file(trainImagePath)
143 x_test = idx2numpy.convert_from_file(testImagePath)
144
145 y_train = idx2numpy.convert_from_file(trainLabelPath)
146 y_test = idx2numpy.convert_from_file(testLabelPath)
147
148 x_train=x_train.reshape((x_train.shape[0], 28*28))
149 x_test=x_test.reshape((x_test.shape[0], 28*28))
150
151
152
153
154 Hx_train, Hy_train = ReduceDataSet.reduce(x_train,y_train
      ,10,5000)
155
```

```
156
157 histArray=Histogram.histogram(Hx_train,16)
158
159 print ("-----------------------Hierarchical clustering
      algorithm---------------------------------")
160
161 print("#########################R2-Data Represantation
      ############################")
162 Hmodel = Hierarchical(histArray)
163
164 print("-----------------Euclidean-----------------")
165 Hclusters, Hlabels = Hmodel.fit(10, "Euclidean")
166 print("Purity score {:.3f}%".format(Hmodel.purity(y_true=
      Hy_train, y_pred=Hlabels)*100))
167 print("F-measure score {:.3f}%".format(Hmodel.f_measure(
      y_true=Hy_train, y_pred=Hlabels)*100))
168
169 print("-----------------Manhattan-----------------")
170 Hclusters, Hlabels = Hmodel.fit(10, "Manhattan")
171 print("Purity score {:.3f}%".format(Hmodel.purity(y_true=
      Hy_train, y_pred=Hlabels)*100))
172 print("F-measure score {:.3f}%".format(Hmodel.f_measure(
      y_true=Hy_train, y_pred=Hlabels)*100))
173
174 print("-----------------Cosine-----------------")
175 Hclusters, Hlabels = Hmodel.fit(10, "Cosine")
176 print("Purity score {:.3f}%".format(Hmodel.purity(y_true=
      Hy_train, y_pred=Hlabels)*100))
177 print("F-measure score {:.3f}%".format(Hmodel.f_measure(
      y_true=Hy_train, y_pred=Hlabels)*100))
178
179 print("#########################R1-Data Represantation
      ############################")
180
181
182 x_train, y_train = ReduceDataSet.reduce(x_train,y_train
      ,10,2000)
183 x_train=x_train/255.0
184
185 model = Hierarchical(x_train)
186
187 print("-----------------Euclidean-----------------")
188 clusters, labels = model.fit(10, "Euclidean")
189 print("Purity score {:.3f}%".format(model.purity(y_true=
      y_train, y_pred=labels)*100))
190 print("F-measure score {:.3f}%".format(model.f_measure(
      y_true=y_train, y_pred=labels)*100))
191
192 print("-----------------Manhattan-----------------")
193 clusters, labels = model.fit(10, "Manhattan")
194 print("Purity score {:.3f}%".format(model.purity(y_true=
      y_train, y_pred=labels)*100))
195 print("F-measure score {:.3f}%".format(model.f_measure(
      y_true=y_train, y_pred=labels)*100))
196
197 print("-----------------Cosine-----------------")
198 clusters, labels = model.fit(10, "Cosine")
199 print("Purity score {:.3f}%".format(model.purity(y_true=
      y_train, y_pred=labels)*100))
200 print("F-measure score {:.3f}%".format(model.f_measure(
      y_true=y_train, y_pred=labels)*100))
```

Αρχικά, ξεκινάμε με όλα τα σημεία σε μία συστάδα, η οποία στη συνέχεια, χωρίζεται σε δύο λιγότερο όμοιες συστάδες. Υπολογίζουμε τον distance matrix $C(i, j)$, ο οποίος αναπαριστά τις αποστάσεις των σημείων i,j (σύμφωνα με κάποια από τις τρεις συναρτήσεις απόστασης.) Έπειτα, η συστάδα διασπάται με βάση τη μεγαλύτερη μέση απόσταση σε δύο νέες συστάδες και τα εναπομείναντα σημεία μετακινούνται, εάν η διαφορά μεταξύ της μέσης απόστασης και της απόστασης με την άλλη συστάδα είναι θετική. Στη συνέχεια, υπολογίζουμε τη διάμετρο της κάθε συστάδας, η οποία ορίζεται ως η απόσταση μεταξύ δύο οποιονδήποτε σημείων της συστάδας και επιλέγεται η συστάδα με τη μεγαλύτερη διάμετρο για περαιτέρω διάσπαση. Η παραπάνω διαδικασία, επαναλαμβάνεται έως ότου πάρουμε τον επιθυμητό αριθμό συστάδων. Τα αποτελέσματα του Hierarchical clustering για τις ζητούμενες αποστάσεις και τις ζητούμενες αναπαραστάσεις των δεδομένων είναι τα εξής:

```
----------Hierarchical clustering algorithm----------
##############################R2-Data Represantation##############################
--------------------Euclidean--------------------
Purity score 28.240%
F-measure score 12.355%
--------------------Manhattan--------------------
Purity score 26.260%
F-measure score 42.998%
--------------------Cosine--------------------
Purity score 29.480%
F-measure score 74.590%
##############################R1-Data Represantation##############################
--------------------Euclidean--------------------
Purity score 47.400%
F-measure score 79.292%
--------------------Manhattan--------------------
Purity score 43.300%
F-measure score 98.818%
--------------------Cosine--------------------
Purity score 40.150%
F-measure score 98.405%
```

3 K-medoids algorithm

```
import numpy as np
from math import log2
from sklearn.metrics import confusion_matrix
import idx2numpy as idx2numpy
import Histogram, ReduceDataSet


class k_medoids:
    def __init__(self, k=10):  # total number of clusters
    =10
        self.k = k

    def fit(self, data):

        # For 1st clustering
        self.medoids = []
        self.cost = 0
        self.clusters = []
        for i in range(self.k):
            self.medoids.append(data[i])  # selecting k
    random points out of the dataset as the medoids (1st
    step)
        for j in range(self.k):
            self.clusters.append([])

        for point in data:
            # list containing distances of each point in
    the dataset from the medoids and finding minimum
    distance

            distance = [np.sum(np.where(point != 0, point
     * np.log(point / m), 0)) for m in self.medoids]
            min_distance = min(distance)
            # calculating the cost (cost is the distance
    of each point from its medoid i.e., total of min
    distances)
            self.cost += min_distance
            o = distance.index(min_distance)
            # clustering (i.e., associating each point to
     the closest medoid)
            self.clusters[o].append(point)

        # now, at each iteration we are finding the
    replacement of our medoids
        # we will stop if there is no changes in the cost
        while (True):

            # lists that will store new clusters, new
    cost and the new medoids
            new_clusters = []
            self.new_cost = 0
            self.new_medoids = []
            for j in range(self.k):
                new_clusters.append([])

            for j in range(self.k):
                # list for storing the total distance of
    a particular point from all other points in the same
```

```
cluster
                dist_with_each_point_in_same_cluster = []
                l = []  # list for storing the distance
of each and every point from all other points of the
same cluster

                for point in self.clusters[j]:
                    dist_with_each_point_in_same_cluster
= sum([np.sum(np.where(point != 0, point * np.log(
point / d), 0)) for d in self.clusters[j]])
                    l.append(
dist_with_each_point_in_same_cluster)
                minima = min(l)  # finding the minimum
distance
                q = l.index(minima)
                # point with the min distance from all
the points in the same cluster is taken as the new
medoid
                self.new_medoids.append(self.clusters[j][
q])


                # now, finding the new clustering and the
 new cost
            for point in data:
                # list containing distances of each point
 in the dataset from the new medoids and finding
minimum distance
                distance = [np.sum(np.where(point != 0,
point * np.log(point / m), 0)) for m in self.
new_medoids]
                min_distance = min(distance)
                # calculating the new cost
                self.new_cost += min_distance
                o = distance.index(min_distance)
                # new clustering
                new_clusters[o].append(point)

            # if the cost decreases with the new medoids
we will replace the old medoids, old clusters and the
 old cost with the new one
            # if the cost increases with the new medoids
we will not replace the old medoids, old clusters and
 the oldcost with the new one

            if self.new_cost < self.cost:
                self.medoids = self.new_medoids
                self.clusters = new_clusters
                self.cost = self.new_cost

            # if the cost remains same we will stop and
come out of the iterations

            elif abs(self.new_cost - self.cost) <= 0.01:
                break

    # final clustering according to the new medoids
    def predict(self, test_data):
        pred = []
        for point in test_data:
            # finding distance of each point from the
```

```
        final medoids and get the min distance
88              distance = [np.sum(np.where(point != 0, point
         * np.log(point / m), 0)) for m in self.medoids]
89              minDistance = min(distance)
90              l = distance.index(minDistance)
91              pred.append(l)  # associating each point to
        the closest medoid
92          return pred
93
94      def purity(self, y_true, y_pred):
95          np.seterr(divide='ignore', invalid='ignore')
96          cm = confusion_matrix(y_true, y_pred, labels=[i
        for i in range(10)])
97          return np.sum(np.amax(cm, axis=0)) / np.sum(cm)
98
99      def f_measure(self, y_true, y_pred):
100         np.seterr(divide='ignore', invalid='ignore')
101         cm = confusion_matrix(y_true, y_pred, labels=[i
        for i in range(10)])
102         tp = np.diag(cm)
103         fp = cm.sum(axis=0) - np.diag(cm)
104         fn = cm.sum(axis=1) - np.diag(cm)
105         tn = cm.sum() - (tp + fn + fp)
106         recall = tp / (tp + fn)
107         precision = tp / (tp + fp)
108         f1 = 2 * ((precision * recall) / (precision +
        recall))
109         f1[np.isnan(f1)] = 0
110         return f1.sum()
111
112
113 trainImagePath = "../dataset/train-images-idx3-ubyte"
114 trainLabelPath = "../dataset/train-labels-idx1-ubyte"
115 testImagePath = "../dataset/t10k-images-idx3-ubyte"
116 testLabelPath = "../dataset/t10k-labels-idx1-ubyte"
117
118 x_train = idx2numpy.convert_from_file(trainImagePath)
119 x_test = idx2numpy.convert_from_file(testImagePath)
120
121 y_train = idx2numpy.convert_from_file(trainLabelPath)
122 y_test = idx2numpy.convert_from_file(testLabelPath)
123
124 x_train = x_train.reshape((x_train.shape[0], 28 * 28))
125 x_test = x_test.reshape((x_test.shape[0], 28 * 28))
126
127 x_train,y_train = ReduceDataSet.reduce(x_train,y_train
        ,10,6000)
128 print(x_train.shape)
129 bins = 16
130 histArray = Histogram.histogram(x_train,bins)
131 testhistArray=Histogram.histogram(x_test,bins)
132
133 model = k_medoids()
134 model.fit(histArray)
135 clusters = model.predict(testhistArray)
136
137 print ("----------------------Kmedoid clustering
        algorithm----------------------------------")
138 print("#########################R2-Data Represantation
        ############################")
139 print("Purity score {:.3f}%".format(model.purity(y_true=
```

```
            y_test, y_pred=clusters)*100))
print("F-measure score {:.3f}%".format(model.f_measure(
            y_true=y_test, y_pred=clusters)*100))
```

Αρχικά, αρχικοποιούμε τις διαμέσους (medoids) επιλέγοντας τις πρώτες k είκονες από το σύνολο δεδομένων. Στη συνέχεια, για κάθε πρότυπο, βρίσκουμε ποιο medoid είναι περισσότερο όμοιο σε αυτό το πρότυπο με βάση την symmetric Kullback-Libler distance και το τοποθετούμε στη συστάδα του medoid αυτού. Έπειτα, ενημερώνουμε την τιμή κάθε medoid, θέτοντάς το ίσο με το σημείο του cluster που έχει τη μεγαλύτερη ομοιότητα με όλα τα υπόλοιπα σημεία, επαναλαμβάνοντας αυτή τη διαδικασία εώς ότου να μην υπάρχει αλλαγή στα κέντρα C_k. Τα αποτελέσματα της μεθόδου K-medoids είναι τα εξής:

```
----------Kmedoid clustering algorithm----------
##############################R2-Data Represantation##############################
Purity score 17.880%
F-measure score 89.048%
```

4 Βοηθητική συνάρτηση histogram

```
import numpy as np
def histogram(X,bins):
  histArray=[]
  for i in X:
    histArray.append(np.array(np.histogram(i,bins)[0]))
  return np.array(histArray, dtype='float64')
```

Η συνάρτηση histogram που βρίσκεται στο αρχείο Histogram.py παίρνει ως όρισμα ένα dataset και τον αριθμό των bins που επιθυμούμε και δημιουργεί το τελικό dataset που περιέχει το ιστόγραμμα της κάθε εικόνας.

5 Βοηθητική συνάρτηση reduce

```
import numpy as np
def reduce(X,Y,k, n):
    dataset = []
    labels =[]
    for i in  range(k):
        c=n//k
        j=0
        while c>0:
            if Y[j] == i:
                dataset.append(X[j])
                labels.append(Y[j])
                c-=1
            j+=1
    return np.array(dataset), np.array(labels)
```

Η συνάρτηση reduce που βρίσκεται στο αρχείο ReduceDataSet.py παίρνει ως όρισμα ένα dataset μαζί με τα labels, το πλήθος των κλάσεων και τέλος τον αριθμό των στοιχείων που θα έχει το μειωμένο (τελικό) dataset. Για κάθε κατηγορία, υπολογίζουμε το πλήθος των στοιχείων που θα έχει αυτή η κατηγορία και έπειτα τα τοποθετούμε στο τελικό dataset.

6 Η βέλτιστη μέθοδος

Έπειτα από σύγκριση των μέτρων αξιολόγησης Purity, F-measure καταλήγουμε πως η βέλτιση μέθοδος για την R_1 αναπαράσταση δεδομένων είναι η μέθοδος της ιεραρχικής συσταδοποίησης με χρήση της ευκλείδιας απόστασης καθώς έχει το μεγαλύτερο Purity (47.4%) και για την R_2 αναπαράσταση δεδομένων είναι η μέθοδος της ιεραρχικής συσταδοποίησης με χρήση της συνημιτονοειδούς απόστασης καθώς έχει το μεγαλύτερο Purity (29.48%).