ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ
2η Σειρά Ασκήσεων

ΟΜΑΔΑ:
Tucaliuc Agnes Monalisa, 3346
Joanis Prifti, 3321

Θέμα: Ομαδοποίηση δεδομένων

Τις μεθόδους τις υλοποιήσαμε στο PyCharm Professional με έκδοση Python 3.7 (την κάθε μέθοδο σε ξεχωριστό αρχείο .py. Όπως και στο προηγούμενο σετ ασκήσεων έτσι και σε αυτό χρησιμοποιήσαμε το ίδιο dataset το σύνολο fashion MNIST το οποίο περιέχει ασπρόμαυρες εικόνες (διάστασης 28 x 28) από 10 κατηγορίες ρούχων.

Χρησιμοποιήσαμε έναν εναλλακτικό τύπο δεδομένων για την αναπαράσταση κάθε εικόνας, το R1 κάνοντας επιπλέον κανονικοποίηση.

Υλοποιήσαμε τον K-means clustering με τις εξής αποστάσεις:

1. Ευκλείδεια απόσταση(L2), ο κώδικας φαίνεται ακριβώς από κάτω:

```
import numpy as np
from tensorflow import keras
from sklearn import metrics
from sklearn.metrics import f1_score

#Downloading mnist dataset
fashion_mnist = keras.datasets.fashion_mnist
(trImages, trLabels), (tImages, tLabels) =
fashion_mnist.load_data()

# Conversion to float and normalize R1
trImages = trImages.astype('float32')
tImages = tImages.astype('float32')
trImages=trImages/255
tImages=tImages/255
# Reshaping input data
trImages = trImages.reshape(trImages.shape[0], trImages.shape[1]
* trImages.shape[2]) #one dimensional
```

```
tImages = tImages.reshape(tImages.shape[0], tImages.shape[1] *
tImages.shape[2])
#https://www.programmersought.com/article/21581162464/
#https://numpy.org/doc/stable/reference/generated/numpy.linalg.n
orm.html
#https://medium.com/ai-for-real/relationship-between-cosine-simi
larity-and-euclidean-distance-7e283a277dff

class KMeans:
   def __init__(self, k):
       self.k = k

   def train(self, X, MAXITER=100, TOL=1e-3):
       centroids = np.random.rand(self.k, X.shape[1])
       centroidsold = centroids.copy()
       for iter_ in range(MAXITER):
           dist = np.linalg.norm(X - centroids[0, :],
axis=1).reshape(-1, 1) #linalg is Euclidean distance
           for class_ in range(1, self.k):
               dist = np.append(dist, np.linalg.norm(X -
centroids[class_, :], axis=1).reshape(-1, 1), axis=1)
           classes = np.argmin(dist, axis=1)
           # update position
           for class_ in set(classes):
               centroids[class_, :] = np.mean(X[classes ==
class_, :], axis=0)
           if np.linalg.norm(centroids - centroidsold) < TOL:
               break
               print('Centroid converged')
       self.centroids = centroids

   def predict(self, X):
       dist = np.linalg.norm(X - self.centroids[0, :],
axis=1).reshape(-1, 1)
       for class_ in range(1, self.k):
           dist = np.append(dist, np.linalg.norm(X -
self.centroids[class_, :], axis=1).reshape(-1, 1), axis=1)
       classes = np.argmin(dist, axis=1)
       return classes

def purity_score(y_true, y_pred):
```

```
   # compute contingency matrix (also called confusion matrix)
   contingency_matrix =
metrics.cluster.contingency_matrix(y_true, y_pred)
   # return purity
   return np.sum(np.amax(contingency_matrix, axis=0)) /
np.sum(contingency_matrix)

kmeans = KMeans(10)
kmeans.train(trImages)
classes = kmeans.predict(trImages)
classes
print("PURITY: ")
print(purity_score(classes,trLabels)) #trLabels
print("F-MEASURE: ")
print(f1_score(classes, trLabels, average='weighted' ))
#trLabels
```

2. Manhattan distance (L1) , ο κώδικας φαίνεται από κάτω

```
#https://www.programmersought.com/article/21581162464/
#https://numpy.org/doc/stable/reference/generated/numpy.linalg.n
orm.html

import numpy as np
from tensorflow import keras
from sklearn import metrics
from sklearn.metrics import f1_score

#Downloading mnist dataset
fashion_mnist = keras.datasets.fashion_mnist
(trImages, trLabels), (tImages, tLabels) =
fashion_mnist.load_data()

# Conversion to float and normalize R1
trImages = trImages.astype('float32')
tImages = tImages.astype('float32')
trImages=trImages/255
tImages=tImages/255
# Reshaping input data
```

```
trImages = trImages.reshape(trImages.shape[0], trImages.shape[1]
* trImages.shape[2]) #one dimensional
tImages = tImages.reshape(tImages.shape[0], tImages.shape[1] *
tImages.shape[2])


class KMeans:
   def __init__(self, k):
       self.k = k

   def train(self, X, MAXITER=100, TOL=1e-3):
       centroids = np.random.rand(self.k, X.shape[1])
       centroidsold = centroids.copy()
       for iter_ in range(MAXITER):
           dist = np.linalg.norm(X - centroids[0, :],ord = 1,
axis=1).reshape(-1, 1) #linalg with ord=1 (L1)is Manhattan
distance
           for class_ in range(1, self.k):
               dist = np.append(dist, np.linalg.norm(X -
centroids[class_, :], ord = 1, axis=1).reshape(-1, 1), axis=1)
           classes = np.argmin(dist, axis=1)
           # update position
           for class_ in set(classes):
               centroids[class_, :] = np.mean(X[classes ==
class_, :], axis=0)
           if np.linalg.norm(centroids - centroidsold) < TOL:
               break
               print('Centroid converged')
       self.centroids = centroids

   def predict(self, X):
       dist = np.linalg.norm(X - self.centroids[0, :],ord = 1,
axis=1).reshape(-1, 1)
       for class_ in range(1, self.k):
           dist = np.append(dist, np.linalg.norm(X -
self.centroids[class_, :], ord =1, axis=1).reshape(-1, 1),
axis=1)
       classes = np.argmin(dist, axis=1)
       return classes

def purity_score(y_true, y_pred):
```

```
    # compute contingency matrix (also called confusion matrix)
    contingency_matrix =
metrics.cluster.contingency_matrix(y_true, y_pred)
    # return purity
    return np.sum(np.amax(contingency_matrix, axis=0)) /
np.sum(contingency_matrix)

kmeans = KMeans(10)
kmeans.train(trImages)
classes = kmeans.predict(trImages)
classes
print("PURITY: ")
print(purity_score(classes,trLabels)) #trLabels
print("F-MEASURE: ")
print(f1_score(classes, trLabels, average='weighted' ))
#trLabels
```

3. Συνημιτονοειδή απόσταση (cosine distance)

```
#https://www.programmersought.com/article/21581162464/
#https://numpy.org/doc/stable/reference/generated/numpy.lin
alg.norm.html
#https://medium.com/ai-for-real/relationship-between-cosine
-similarity-and-euclidean-distance-7e283a277dff

import numpy as np
from tensorflow import keras
from sklearn import metrics
from sklearn.metrics import f1_score

#Downloading mnist dataset
fashion_mnist = keras.datasets.fashion_mnist
(trImages, trLabels), (tImages, tLabels) =
fashion_mnist.load_data()

# Conversion to float and normalize R1
trImages = trImages.astype('float32')
tImages = tImages.astype('float32')
trImages=trImages/255
tImages=tImages/255
# Reshaping input data
```

```
trImages = trImages.reshape(trImages.shape[0],
trImages.shape[1] * trImages.shape[2]) #one dimensional
tImages = tImages.reshape(tImages.shape[0],
tImages.shape[1] * tImages.shape[2])


class KMeans:
   def __init__(self, k):
       self.k = k

   def train(self, X, MAXITER=100, TOL=1e-3):
       centroids = np.random.rand(self.k, X.shape[1])
       centroidsold = centroids.copy()
       for iter_ in range(MAXITER): #Euclidean Distance
(u,v) = 2 * Cosine Distance(u,v) for normalized vectors
           dist = (np.linalg.norm(X - centroids[0, :],
axis=1).reshape(-1, 1))/2 #linalg is Euclidean distance
           for class_ in range(1, self.k):
               dist = np.append(dist, (np.linalg.norm(X -
centroids[class_, :], axis=1).reshape(-1, 1))/2, axis=1)
           classes = np.argmin(dist, axis=1)
           # update position
           for class_ in set(classes):
               centroids[class_, :] = np.mean(X[classes ==
class_, :], axis=0)
           if np.linalg.norm(centroids - centroidsold) <
TOL:
               break
               print('Centroid converged')
       self.centroids = centroids

   def predict(self, X):
       dist = (np.linalg.norm(X - self.centroids[0, :],
axis=1).reshape(-1, 1))/2
       for class_ in range(1, self.k):
           dist = np.append(dist, (np.linalg.norm(X -
self.centroids[class_, :], axis=1).reshape(-1, 1))/2,
axis=1)
       classes = np.argmin(dist, axis=1)
       return classes
```

```
def purity_score(y_true, y_pred):
   # compute contingency matrix (also called confusion
matrix)
   contingency_matrix =
metrics.cluster.contingency_matrix(y_true, y_pred)
   # return purity
   return np.sum(np.amax(contingency_matrix, axis=0)) /
np.sum(contingency_matrix)

kmeans = KMeans(10)
kmeans.train(trImages)
classes = kmeans.predict(trImages)
classes
print("PURITY: ")
print(purity_score(classes,trLabels)) #trLabels
print("F-MEASURE: ")
print(f1_score(classes, trLabels, average='weighted' ))
#trLabels
```

ΑΠΟΤΕΛΕΣΜΑΤΑ

Χρησιμοποιήσαμε δύο μέτρα αξιολόγησης για να συγκρίνουμε τις μεθόδους ομαδοποίησης, το purity που υπολογίζει μετρώντας το ποσοστό των σωστών ταξινομημένων δεδομένων και το f-measure.

1. K-Means with Euclidean distance
   PURITY:
   0.6094166666666667
   F-MEASURE:
   0.01680615045103985

2. K-Means with Manhattan distance
   PURITY:
   0.6426333333333333
   F-MEASURE:
   0.012098786132408005

3. K-Means with cosine distance

PURITY:

0.6094333333333334

F-MEASURE:

0.05135549346760565

ΣΥΓΚΡΙΣΗ

Βάση των παραπάνω αποτελεσμάτων παρατηρούμε ότι με βάση το purity καλύτερο αποτέλεσμα βγάζει το k-means with Manhattan distance, ενώ με βάση το f-measure καλύτερο αποτέλεσμα βγάζει το k-means with cosine distance.