Σπυρίδων Σιλίρας

3330

Μηχανική Μάθηση: 2η Εργασία

Μέρος 1:

Αρχικά, έχουν φορτωθεί όλα τα δεδομένα του συνόλου και έχουν μετατραπεί στην επιθυμητή μορφή. Στην αρχή του προγράμματος υπάρχουν global μεταβλητές που αφορούν στο ποιο είδος αναπαράστασης δεδομένων θέλουμε να επιλέξουμε, είδος απόστασης, αριθμός παραδειγμάτων που θα χρησιμοποιηθούν για την εκπαίδευση. Αν έχουμε επιλέξει τη μορφή του ιστογράμματος φωτεινότητας, έχουμε και μία επιπλέον global μεταβλητή, την numberOfBins που είναι ο αριθμός των κάδων που θα έχει η αναπαράσταση του ιστογράμματος. Στη συνέχεια, έχουν υλοποιηθεί οι τύποι απόστασης που χρειαζόμαστε για την άσκηση, ευκλείδεια, Manhattan και συνημιτονοειδής απόσταση. Έπειτα, χρειαζόμαστε κάποιες βοηθητικές υπορουτίνες για την κατασκευή της μεθόδου k-means. Πρώτον, πρέπει να αρχικοποιήσουμε τα κέντρα των k ομάδων (10 στην περίπτωσή μας), έχει γίνει επιλέγοντας τυχαία κάποιο παράδειγμα από το σύνολο εκπαίδευσης και θέτοντας τις τιμές κάθε διάστασης του κέντρου ίσες με αυτές του τυχαία επιλεγμένου παραδείγματος. Δεύτερον, χρειαζόμαστε και μία μέθοδο για την ενημέρωση των κέντρων κάθε φορά που έχει κληθεί ο k means. Στη συνέχεια αφού έχουμε αρχικοποιήσει τα κέντρα, καλούμε επαναληπτικά των k means αλγόριθμο, έως ότου το σφάλμα μεταξύ δύο διαδοχικών κλήσεων του αλγορίθμου γίνει μικρότερο από μία τιμή κατωφλίου (errorThreshold). Αυτή η τιμή στην δική μας περίπτωση είναι 100, αρκετά μεγάλη, αλλά υποθέτοντας μικρότερη τιμή σαφώς θα έχουμε ποιοτικότερο αποτέλεσμα, αλλά η πολυπλοκότητα χρόνου αυξάνεται σε μεγάλο βαθμό. Στο τέλος, αφού έχουμε βρεί τις κατάλληλες τιμές των κέντρων των k ομάδων, υπολογίζουμε την ακρίβεια του αλγορίθμου k-means μέσω των τιμών purity και f measure.

- Αποτελέσματα μετρήσεων:
  Μορφή αναπαράστασης R1, Ευκλείδεια απόσταση, 10000 samples:

  Purity = 60.77%
  F_measure = 6.174

  Μορφή αναπαράστασης R1, Manhattan distance, 10000 samples:

  Purity = 33.89%

  F_measure = 3.4

  Μορφή αναπαράστασης R2, Ευκλείδεια απόσταση, 10000 samples:

  Number of bins = 16:

  Purity = 28.93%

  F_measure = 2.73

Number of bins = 32:

Purity = 31.4%

F_measure = 3.07

Number of bins = 64:

Purity = 31.8%

F_measure = 3.008

Number of bins = 128:

Purity = 31.35%

F_measure = 3.06

Μορφή αναπαράστασης R2, Manhattan distance, 10000 samples:

Number of bins = 16:

Purity = 28.75%

F_measure = 2.67

Number of bins = 32:

Purity = 27.35%

F_measure = 2.52

Number of bins = 64:

Purity = 28.22%

F_measure = 2.56

Number of bins = 128:

Purity = 24.96%

F_measure = 2.14

Να σημειωθεί ότι για cosine distance δε μου έβγαινε αποτέλεσμα καθώς υπήρχαν clusters που ήτα κενά.

Επίσης, δεν υπάρχει υλοποίηση του k-medoids, καθώς δεν κατάφερα να το υλοποιήσω.

Μέρος 2:

Όπως και στο μέρος 1, έχουν χρησιμοποιηθεί οι δύο μορφές αναπαράστασης και οι ίδιοι τύποι αποστάσεων. Τώρα όσον αφορά την κατασκευή του δέντρου, αρχικά στη ρίζα τοποθετούμε όλα τα παραδείγματα του συνόλου εκπαίδευσης (είτε όλα είτε υποσύνολο αυτών), στη συνέχεια διασπάμε έναν κόμβο-φύλλο σε δύο αφού έχουμε επιλέξει προς διάσπαση αυτόν που έχει την μεγαλύτερη τιμή διακύμανσης. Τώρα, για να είναι αποδοτικό το split πρέπει να βρούμε δύο σημεία αυτού του κόμβου που θα είναι αυτά με βάση τα οποία θα γίνει ο χωρισμός στους δύο νέους κόμβους. Για να γίνει αυτό, εκτελούμε τον αλγόριθμο k-means με 2 clusters και με βάση τα δύο σημεία που προκύπτουν τοποθετούμε στους 2 νέους κόμβους τα υπόλοιπα δεδομένα*. Αυτή η διαδικασία συνεχίζεται έως ότου ο αριθμός των φύλλων του δέντρου γίνει ίσος με τον αριθμό τον ομάδων M που έχουμε επιλέξει να ομαδοποιήσουμε τα δεδομένα. Όταν έχει πραγματοποιηθεί αυτό, το αποτέλεσμα της ομαδοποίησης είναι οι ομάδες που περιέχουν αυτοί οι M κόμβοι-φύλλα. Τέλος, με όμοιο τρόπο όπως στο παράδειγμα 1 υπολογίζονται οι ποιοτικές τιμές που ζητούνται στην εκφώνηση.

*Να σημειωθεί ότι για την υλοποίηση του αλγορίθμου k-means, για την εύρεση αυτών των δύο σημείων έχει χρησιμοποιηθεί ο αλγόριθμος του πακέτου sk-learn.

- Αποτελέσματα μετρήσεων:

  Μορφή αναπαράστασης R1, Ευκλείδεια απόσταση, 10000 samples:
  - Purity = 28.81%
  - F_measure = 1.91

  Μορφή αναπαράστασης R1, Manhattan distance, 10000 samples:
  - Purity = 28.3%
  - F_measure = 1.83

  Μορφή αναπαράστασης R1, Cosine Distance, 10000 samples:
  - Purity = 31.96%
  - F_measure = 2.27

  Μορφή αναπαράστασης R2, Ευκλείδεια απόσταση, 10000 samples:
  - Number of bins = 16:
    - Purity = 25.92%
    - F_measure = 1.8
  - Number of bins = 32:
    - Purity = 27.2%
    - F_measure = 1.81
  - Number of bins = 64:
    - Purity = 26%
    - F_measure = 1.76

Number of bins = 128:

Purity = 26.03%

F_measure = 1.75

Μορφή αναπαράστασης R2, Manhattan distance, 10000 samples:

Number of bins = 16:

Purity = 29.63%

F_measure = 2.22

Number of bins = 32:

Purity = 29.45%

F_measure = 2.15

Number of bins = 64:

Purity = 29.26%

F_measure = 2.11

Number of bins = 128:

Purity = 28.79%

F_measure = 2.08

Μορφή αναπαράστασης R2, Cosine distance, 10000 samples:

Number of bins = 16:

Purity = 27.71%

F_measure = 2.36

Number of bins = 32:

Purity = 27.47%

F_measure = 2.31

Number of bins = 64:

Purity = 27.31%

F_measure = 2.25

Number of bins = 128:

Purity = 26.5%

F_measure = 2.17

Συμπέρασμα: Η μέθοδος με το καλύτερο αποτέλεσμα είναι αυτή του αλγορίθμου k-means με τιμή Purity = 61%.

Κώδικας:

Μέρος 1:

```
import tensorflow as tf
import math
import random
import numpy as np
import matplotlib.pyplot as plt

fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) =
fashion_mnist.load_data()

class_names = ['T-
shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Shirt','Sneaker','Bag','Ankl
e boot']

dimensions = 28*28

#method (=0 for R1 form of train images, =1 for histogram)
method = 1

#which distance type to use, 0=euclidean, 1=manhattan and 2=cosine
distance = 2

#number of clusters
k = 10

#the centers of the clusters
centroids = []

#keeps the id of train images of each one cluster
clusters = {}

#finishing error threshold
errorThreshold = 100

#the number of samples to use
train_number = 10000

#number of bins if we use R2 format (it can be 16, 32,64 or 128)
numberOfBins = 16
```

```
#normalize the data
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')
train_images = train_images/255.0
test_images = test_images/255.0

train_images = train_images.reshape(train_images.shape[0],
train_images.shape[1]*train_images.shape[2])
test_images = test_images.reshape(test_images.shape[0],
test_images.shape[1]*test_images.shape[2])

train_histogram = []

if method == 1:

        dimensions = numberOfBins

        bins = []
        bins.append(0)

        for i in range(1, numberOfBins+1):
                bins.append(i/numberOfBins)

        for i in range(0, len(train_images[0:train_number])):
                hist = np.histogram(train_images[i], bins)
                train_histogram.append(hist[0])

def euclidean_distance(s1, s2):
        total = 0

        for i in range(0, dimensions):
                total += (s1[i] - s2[i])**2

        return math.sqrt(total)

def manhattan_distance(s1, s2):
        total = 0

        for i in range(0, dimensions):
                total += abs(s1[i]-s2[i])

        return total
```

```
def cosine_distance(s1, s2):
	sum1 = 0
	sum2 = 0
	sum3 = 0

	for i in range(0, dimensions):
		sum1 += s1[i]*s2[i]
		sum2 += s1[i]**2
		sum3 += s2[i]**2

	total = sum1 / (math.sqrt(sum2)*math.sqrt(sum3))

	return total

#initialize centroids and clusters
def init_centers():
	for i in range(0, k):
		center = []

		#initialize with choosing a random sample from all train
samples
		ind = random.randint(0, train_number)

		if method == 0:
			for j in range (0, dimensions):
				#a random float between 0 and 1
				center.append(train_images[ind][j])

		elif method == 1:
			for j in range(0,dimensions):
				center.append(train_histogram[ind][j])

		#store the centers to global variable centroids
		centroids.append(center)

def update_centers():

	for i in range(0, k):
		for j in range(0, dimensions):
			val = 0
			for z in clusters[i]:
				if method == 0:
					val += train_images[z][j]
				elif method == 1:
```

```
                                        val += train_histogram[z][j]

                        if len(clusters[i]) == 0:
                                centroids[i][j] = 1
                        else:
                                centroids[i][j] = val / len(clusters[i])

def k_means():

        error = 0

        for i in range(0, k):
                clusters[i] = []


        #compute error and update clusters
        if method == 0:
                for i in range(0, train_number):
                        #print(i)
                        minimum = float('inf')
                        minIndex = -1

                        for j in range(0, k):
                                if distance == 0:
                                        d = euclidean_distance(train_images[i],
centroids[j])
                                        if d < minimum:
                                                minimum = d
                                                minIndex = j

                                elif distance == 1:
                                        d = manhattan_distance(train_images[i],
centroids[j])
                                        if d < minimum:
                                                minimum = d
                                                minIndex = j

                                elif distance == 2:
                                        d = cosine_distance(train_images[i],
centroids[j])
                                        if d < minimum:
                                                minimum = d
                                                minIndex = j
```

```
                        error += minimum
                        clusters[minIndex].append(i)

                update_centers()

        if method == 1:
                for i in range(0, train_number):

                        minimum = float('inf')
                        minIndex = -1

                        for j in range(0, k):
                                if distance == 0:
                                        d =
euclidean_distance(train_histogram[i], centroids[j])
                                        if d < minimum:
                                                minimum = d
                                                minIndex = j

                                elif distance == 1:
                                        d =
manhattan_distance(train_histogram[i], centroids[j])
                                        if d < minimum:
                                                minimum = d
                                                minIndex = j

                                elif distance == 2:
                                        d = cosine_distance(train_histogram[i],
centroids[j])
                                        if d < minimum:
                                                minimum = d
                                                minIndex = j

                        error += minimum
                        clusters[minIndex].append(i)

                update_centers()

        return error

def main():
        init_centers()
        #first call of k-means and keep the error value
        err1 = k_means()
```

```
        print('err1=',err1)

        #second call of k-means
        err2 = k_means()
        print('err2=',err2)

        while(err1-err2 > errorThreshold):

                err1 = err2

                err2 = k_means()

                print('err2=',err2)

main()

TP_total = 0
F_measure = 0

for i in range(0, k):

        find_cluster = {}

        for j in clusters[i]:
                label = train_labels[j]
                if label not in find_cluster:
                        find_cluster[label] = 1
                else:
                        find_cluster[label] += 1

        count = -1
        cluster_index = -1

        for j in find_cluster:
                if find_cluster[j] > count:
                        count = find_cluster[j]
                        cluster_index = j

        TP_total += find_cluster[cluster_index]

        #TP is eqwual to the number of the category with the most labels in
cluster
        true_positives = find_cluster[cluster_index]
```

```
        #FP is equal to all the labels contained in cluster minus that which
predicted as TP
        false_positives = len(clusters[i]) - find_cluster[cluster_index]

        #FN is equal to all labels in train set where its category is the same as
        #the majority label in the current cluster
        #each category has the same number of items
        false_negatives = (train_number/k) - find_cluster[cluster_index]

        precision = true_positives / (true_positives + false_positives)

        recall = true_positives / (true_positives + false_negatives)

        F1_score = 2*((precision*recall) / (precision+recall))

        F_measure += F1_score

print("Purity is: " +str(TP_total/train_number))
print("F_measure is: " + str(F_measure))
```

Μέρος 2:

```
import tensorflow as tf
import math
import random
import numpy as np
from sklearn.cluster import KMeans

fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) =
fashion_mnist.load_data()

class_names = ['T-
shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Shirt','Sneaker','Bag','Ankl
e boot']

#normalize the data
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')
train_images = train_images/255.0
test_images = test_images/255.0
```

```
train_images = train_images.reshape(train_images.shape[0],
train_images.shape[1]*train_images.shape[2])
test_images = test_images.reshape(test_images.shape[0],
test_images.shape[1]*test_images.shape[2])

dimensions = 28*28

#method (=0 for R1 form of train images, =1 for histogram)
method = 1

#number of bins if we use R2 format (it can be 16, 32,64 or 128)
numberOfBins = 128

#0= eucklidean distance, 1= manhattan distance and 2=cosine distance
distance = 2

#the number of train set photos used
train_number = 20000

train_histogram = []

if method == 1:

        dimensions = numberOfBins

        bins = []
        bins.append(0)

        for i in range(1, numberOfBins+1):
                bins.append(i/numberOfBins)

        for i in range(0, len(train_images[0:train_number])):
                hist = np.histogram(train_images[i], bins)
                train_histogram.append(hist[0])

def euclidean_distance(s1, s2):
        total = 0

        for i in range(0, dimensions):
                total += (s1[i] - s2[i])**2

        return math.sqrt(total)

def manhattan_distance(s1, s2):
```

```
        total = 0

        for i in range(0, dimensions):
                total += abs(s1[i]-s2[i])

        return total

def cosine_distance(s1, s2):
        sum1 = 0
        sum2 = 0
        sum3 = 0

        for i in range(0, dimensions):
                sum1 += s1[i]*s2[i]
                sum2 += s1[i]**2
                sum3 += s2[i]**2

        total = sum1 / (math.sqrt(sum2)*math.sqrt(sum3))

        return total

#number of clusters
M = 10

tree = []

leaf = []

leaf_index = []

train_images_index = []

for i in range(0, train_number):
        leaf.append(i)

train_images_index.append(leaf)

if method == 0:
        tree.append(train_images[0:train_number])

elif method == 1:
        tree.append(train_histogram[0:train_number])

def hierarchical_clustering():
```

```
k = 1

while k != M:
        print(k)
        if k == 1:

                new_indexes1 = []
                new_indexes2 = []

                new_node1 = []
                new_node2 = []

                if method == 0:

                        kmeans =
KMeans(n_clusters=2).fit(train_images[0:train_number])

                        for i in range(0, train_number):
                                if distance == 0:
                                        if
euclidean_distance(train_images[i], kmeans.cluster_centers_[0]) >
euclidean_distance(train_images[i], kmeans.cluster_centers_[1]):
                                                new_indexes1.append(i)


        new_node1.append(train_images[i])

                                        else:
                                                new_indexes2.append(i)


        new_node2.append(train_images[i])

                                elif distance == 1:
                                        if
manhattan_distance(train_images[i], kmeans.cluster_centers_[0]) >
manhattan_distance(train_images[i], kmeans.cluster_centers_[1]):
                                                new_indexes1.append(i)


        new_node1.append(train_images[i])

                                        else:
```

```
                                                    new_indexes2.append(i)


            new_node2.append(train_images[i])

                                            elif distance == 2:
                                                    if
cosine_distance(train_images[i], kmeans.cluster_centers_[0]) >
cosine_distance(train_images[i], kmeans.cluster_centers_[1]):
                                                    new_indexes1.append(i)


            new_node1.append(train_images[i])

                                                else:
                                                    new_indexes2.append(i)


            new_node2.append(train_images[i])

                                    tree.append(new_node1)
                                    tree.append(new_node2)

                                    leaf_index.append(len(tree) - 2)
                                    leaf_index.append(len(tree) - 1)

                                    train_images_index.append(new_indexes1)
                                    train_images_index.append(new_indexes2)

                            elif method == 1:

                                    kmeans =
KMeans(n_clusters=2).fit(train_histogram[0:train_number])

                                    for i in range(0, train_number):
                                        if distance == 0:
                                            if
euclidean_distance(train_histogram[i], kmeans.cluster_centers_[0]) >
euclidean_distance(train_histogram[i], kmeans.cluster_centers_[1]):
                                                    new_indexes1.append(i)


            new_node1.append(train_histogram[i])
```

```
                                        else:
                                            new_indexes2.append(i)

            new_node2.append(train_histogram[i])

                                    elif distance == 1:
                                        if
manhattan_distance(train_histogram[i], kmeans.cluster_centers_[0]) >
manhattan_distance(train_histogram[i], kmeans.cluster_centers_[1]):
                                            new_indexes1.append(i)

            new_node1.append(train_histogram[i])

                                        else:
                                            new_indexes2.append(i)

            new_node2.append(train_histogram[i])

                                    elif distance == 2:
                                        if
cosine_distance(train_histogram[i], kmeans.cluster_centers_[0]) >
cosine_distance(train_histogram[i], kmeans.cluster_centers_[1]):
                                            new_indexes1.append(i)

            new_node1.append(train_histogram[i])

                                        else:
                                            new_indexes2.append(i)

            new_node2.append(train_histogram[i])

                        tree.append(new_node1)
                        tree.append(new_node2)

                        leaf_index.append(len(tree) - 2)
                        leaf_index.append(len(tree) - 1)

                        train_images_index.append(new_indexes1)
                        train_images_index.append(new_indexes2)
```

```
                else:
                        max_var_index = -1
                        maxim = float("-inf")

                        for i in leaf_index:
                                var = np.var(tree[i])

                                if var > maxim:
                                        maxim = var
                                        max_var_index = i

                        leaf_index.remove(max_var_index)

                        print(len(tree[max_var_index]))

                        kmeans =
KMeans(n_clusters=2).fit(tree[max_var_index])

                        new_indexes1 = []
                        new_indexes2 = []

                        new_node1 = []
                        new_node2 = []

                        for i in range(0, len(tree[max_var_index])):

                                if distance == 0:
                                        if
euclidean_distance(tree[max_var_index][i], kmeans.cluster_centers_[0]) >
euclidean_distance(tree[max_var_index][i], kmeans.cluster_centers_[1]):

        new_indexes1.append(train_images_index[max_var_index][i])


        new_node1.append(tree[max_var_index][i])

                                        else:

        new_indexes2.append(train_images_index[max_var_index][i])


        new_node2.append(tree[max_var_index][i])
```

```
                elif distance == 1:
                    if
manhattan_distance(tree[max_var_index][i], kmeans.cluster_centers_[0]) >
manhattan_distance(tree[max_var_index][i], kmeans.cluster_centers_[1]):

                    new_indexes1.append(train_images_index[max_var_index][i])


                    new_node1.append(tree[max_var_index][i])

                    else:

                    new_indexes2.append(train_images_index[max_var_index][i])


                    new_node2.append(tree[max_var_index][i])

                elif distance == 2:
                    if
cosine_distance(tree[max_var_index][i], kmeans.cluster_centers_[0]) >
cosine_distance(tree[max_var_index][i], kmeans.cluster_centers_[1]):

                    new_indexes1.append(train_images_index[max_var_index][i])


                    new_node1.append(tree[max_var_index][i])

                    else:

                    new_indexes2.append(train_images_index[max_var_index][i])


                    new_node2.append(tree[max_var_index][i])

                        tree.append(new_node1)
                        tree.append(new_node2)

                        leaf_index.append(len(tree) - 2)
                        leaf_index.append(len(tree) - 1)

                        train_images_index.append(new_indexes1)
                        train_images_index.append(new_indexes2)
```

```
                k += 1

hierarchical_clustering()

TP_total = 0
F_measure = 0

for i in leaf_index:

        find_cluster = {}

        for j in train_images_index[i]:
                label = train_labels[j]
                if label not in find_cluster:
                        find_cluster[label] = 1
                else:
                        find_cluster[label] += 1

        count = -1
        cluster_index = -1

        for j in find_cluster:
                if find_cluster[j] > count:
                        count = find_cluster[j]
                        cluster_index = j

        TP_total += find_cluster[cluster_index]

        #TP is eqwual to the number of the category with the most labels in
cluster
        true_positives = find_cluster[cluster_index]

        #FP is equal to all the labels contained in cluster minus that which
predicted as TP
        false_positives = len(train_images_index[i]) -
find_cluster[cluster_index]

        #FN is equal to all labels in train set where its category is the same as
        #the majority label in the current cluster
        #each category has the same number of items
        false_negatives = (train_number/M) - find_cluster[cluster_index]

        precision = true_positives / (true_positives + false_positives)
```

```
            recall = true_positives / (true_positives + false_negatives)

            F1_score = 2*((precision*recall) / (precision+recall))

            F_measure += F1_score

print("Purity is: " +str(TP_total/train_number))
print("F_measure is: " + str(F_measure))
```