Μπαζάκας Τηλέμαχος Μάρκος 3281

Ντόντης Βασίλειος 3300

Μηχανική Μάθηση 2η Σειρά Ασκήσεων

Θέμα: Μέθοδοι μείωσης Ομαδοποίησης δεδομένων και μείωσης διάστασης (Clustering and Dimension Reduction methods)

Πανεπιστήμιο Ιωαννίνων

1η Φάση

Στη πρώτη φάση της εργασίας, στόχος είναι να μελετήσουμε τις επιδόσεις δύο γνωστών αλγορίθμων ομαδοποίησης (clustering):

• αλγόριθμος **k-means** χρησιμοποιώντας Ευκλείδια απόσταση,

• **agglomerative hierarchical clustering** (συνθετική ιεραρχική ομαδοποίηση) χρησιμοποιώντας την στρατηγική ward για την εύρεση των ομάδων με την μικρότερη απόσταση και την συνένωσή τους (merge).

Θέλουμε να συγκρίνουμε τους δυο αλγορίθμους με βάση τα γνωστά μέτρα αξιολόγησης Purity και F-Measure,

Στο πρόβλημα της ομαδοποίησης δεν είναι γνωστή εκ των προτέρων η κατηγορία. Θα υποθέσουμε διαφορετικό αριθμό ομάδων (K clusters) και συγκεκριμένα τις τιμές: K={2, 4, 6, 8, 10}. Επίσης εφαρμόζουμε 10-fold cross validation για την αντιμετώπιση του overfitting και της αύξησης της γενίκευσης του εκάστοτε ομαδοποιητή.

Αρχικά, διαβάζουμε το csv αρχείο, κρατάμε τα δεδομένα στο Dataframe 'trainingData' και αφού αφαιρέσουμε τη στήλη που έχει την κατηγορία, χωρίζουμε τα δεδομένα σε train και test.

```
print('-Kmeans Algorithm-')

trainingData = pd.read_csv('train.csv')
target_col = 'price_range'
y = trainingData[target_col]
x = trainingData.drop(target_col, axis=1)

X, X_, Y, Y_ = train_test_split(x, y, test_size=0.2, random_state=42)
X = X.to_numpy()
Y = Y.to_numpy()
```

Data

Για κάθε πλήθος clusters(K) κάνουμε την εξής διαδικασία:

- Χωρίζουμε τα δεδομένα μας σε train και validation.
- Εφαρμόζουμε τον k-means αλγόριθμο στα δεδομένα μας, μέσο της βιβλιοθήκης sklearn, για να πάρουμε τις αναθέσεις των clusters.
- Δημιουργούμε ένα λεξικό που το ονομάζουμε cluster_data, το οποίο χρησιμοποιούμε για να αποθηκεύσουμε εκεί τα data_points για κάθε cluster. Δίνουμε labels στα clusters ανάλογα τη πλειοψηφούσα κατηγορία των δεδομένων που βρίσκονται σε κάθε cluster. Επειδή έχουμε τα ground truth labels, συγκρίνουμε τις αναθέσεις των clusters με τα true labels για να καθορίσουμε τη πλειοψηφούσα κατηγορία του κάθε cluster.
- Συγκρίνοντας τις αναθέσεις με τα true labels, υπολογίζουμε τα TP, FP και FN(True Positives, False Positives και False Negatives)
- Για τα True Positives, μετράμε το πλήθος των δεδομένων που ανήκουν στο ίδιο label του cluster και του ground truth label.
- Για τα False Positives, μετράμε το πλήθος των δεδομένων που έχουν ανατεθεί σε ένα cluster, αλλά ανήκουν σε διαφορετική κατηγορία με βάση το ground truth label.
- Για τα False Negatives, μετράμε το πλήθος των δεδομένων που ανήκουν στη σωστή κατηγορία(label), αλλά έχουν ανατεθεί σε διαφορετικό cluster.

```python
k_values = [2, 4, 6, 8, 10]
totalFMeasures = []
print('----------------------')
for k in k_values:
    print(f"K = {k}")
    purity_scores = []
    fMeasures_scores = []
    cluster_data = defaultdict(list)
    kf = KFold(n_splits=10)
    for train_index, test_index in kf.split(X):
        X_train, X_val = X[train_index], X[test_index]
        y_train, y_val = Y[train_index], Y[test_index]
        kmeans = KMeans(n_clusters=k, n_init=10)
        kmeans.fit(X_train)
        labels = kmeans.predict(X_val)
        for i in range(len(X_val)):
            data_point = X_val[i]
            cluster_label = labels[i]
            cluster_data[cluster_label].append(data_point)
        cluster_majority = {}
        for cluster_label, points in cluster_data.items():
            true_labels = [y_val[i] for i in range(len(X_val)) if labels[i] == cluster_label]
            majority_class = max(set(true_labels), key=true_labels.count)
            cluster_majority[cluster_label] = majority_class
        true_positives = 0
        false_positives = 0
        false_negatives = 0

        for i in range(len(X_val)):
            cluster_label = labels[i]
            true_label = y_val[i]
            if true_label == cluster_majority[cluster_label]:
                true_positives += 1
            else:
                if true_label in cluster_majority.values():
                    false_positives += 1
                else:
                    false_negatives += 1
```

Clustering and TP, FP, FN

Για να υπολογίσουμε το Purity δημιουργούμε μέθοδο η οποία κάνει το εξής:

Αρχικά, εντοπίζουμε τις μοναδικές ομάδες (clusters) από τις προβλεπόμενες ετικέτες (predicted_labels) με τη χρήση της συνάρτησης np.unique(). Για κάθε ομάδα: δημιουργούμε μια μάσκα (mask) που επιλέγει τα δείγματα που ανήκουν στη συγκεκριμένη ομάδα. Κρατάμε στην μεταβλητή cluster_labels τις πραγματικές ετικέτες των δειγμάτων που ανήκουν στην ομάδα. Υπολογίζουμε τον αριθμό εμφάνισης κάθε ετικέτας μέσω της συνάρτησης np.bincount() και τους αποθηκεύουμε στη μεταβλητή counts. Βρίσκουμε τον αριθμό της πιο πλειοψηφούσας ετικέτας (majority_count) μέσω της συνάρτησης np.max(counts). Υπολογίζουμε τον αριθμό του purity (cluster_purity) της ομάδας, δηλαδή τον αριθμό των δειγμάτων που ανήκουν στην πλειοψηφούσα ετικέτα διαιρούμενος με τον συνολικό αριθμό των δειγμάτων στην ομάδα (len(cluster_labels)). Προσθέτουμε το purity του cluster (cluster_purity * len(cluster_labels)) στη μεταβλητή purityy. Τέλος, διαιρούμε το purityy με τον συνολικό αριθμό των δειγμάτων (len(predicted_labels)) για να υπολογίσουμε τον επιθυμητό μέσο όρο purity των clusters.

```python
def compute_purity(predicted_labels, true_labels):
    clusters = np.unique(predicted_labels)
    purityy = 0

    for cluster in clusters:
        mask = predicted_labels == cluster
        cluster_labels = true_labels[mask]
        counts = np.bincount(cluster_labels)
        majority_count = np.max(counts)
        cluster_purity = majority_count / len(cluster_labels)
        purityy += cluster_purity * len(cluster_labels)

    purityy /= len(predicted_labels)

    return purityy
```

Purity

Για να υπολογίσουμε το F-Measure χρησιμοποιούμε τα TP, FP και FN και κάνουμε το εξής:

- Υπολογίζουμε το Precision, το οποίο είναι το πλήθος των TP διαιρούμενος με το άθροισμα των TP και FP.
- Υπολογίζουμε το Recall, το οποίο είναι το πλήθος των TP διαιρούμενος με το άθροισμα των TP και FN.
- Τέλος, υπολογίζουμε το F-Measure, το οποίο είναι: 1+a/(1/precision) + (1+recall) , όπου το a το θεωρούμε 1 οπότε έχουμε το τελικό F-Measure που φαίνεται πατακάτω.

```python
precision = true_positives/(true_positives + false_positives)
recall = true_positives/(true_positives + false_negatives)
fMeasure = 2 / ((1 / precision) + (1 / recall))
```

F-Measure

Παρακάτω φαίνονται τα αποτελέσματα των υλοποιήσεων μας για τους δύο αλγορίθμους.

Method	***Number of Clusters (K)***	***Purity***	***F-measure***	***Total F-measure***	***Mean Purity***
K-means (Euclidean distance)	2	0.503749	0.669691	***3.958400334779919***	***0.720625***
	4	0.676875	0.807042		
	6	0.685625	0.812916		
	8	0.711875	0.831352		
	10	0.720625	0.837397		
Agglomerative Hierarchical Clustering	2	0.506875	0.672057	***3.9889193873870235***	***0.764375***
	4	0.6475	0.784997		
	6	0.695624	0.819911		
	8	0.733749	0.846150		
	10	0.764375	0.865801		

Results

$2^{η}$ Φάση

Στη πρώτη φάση της εργασίας, καλούμαστε να μελετήσουμε την αξία του μετασχηματισμού του αρχικού χώρου των δεδομένων μέσω της τεχνικής νευρωνικών δικτύων **Autoencoder.**

Θα εκπαιδεύσουμε αρχικά πάνω στα δεδομένα (του συνόλου εκπαίδευσης) έναν Autoencoder με την εξής αρχιτεκτονική νευρωνικού δικτύου:

20-100-M-100-20

υποθέτοντας τις εξής τιμές του M (διάσταση του μετασχηματισμένου χώρου): M={2, 10, 50}.

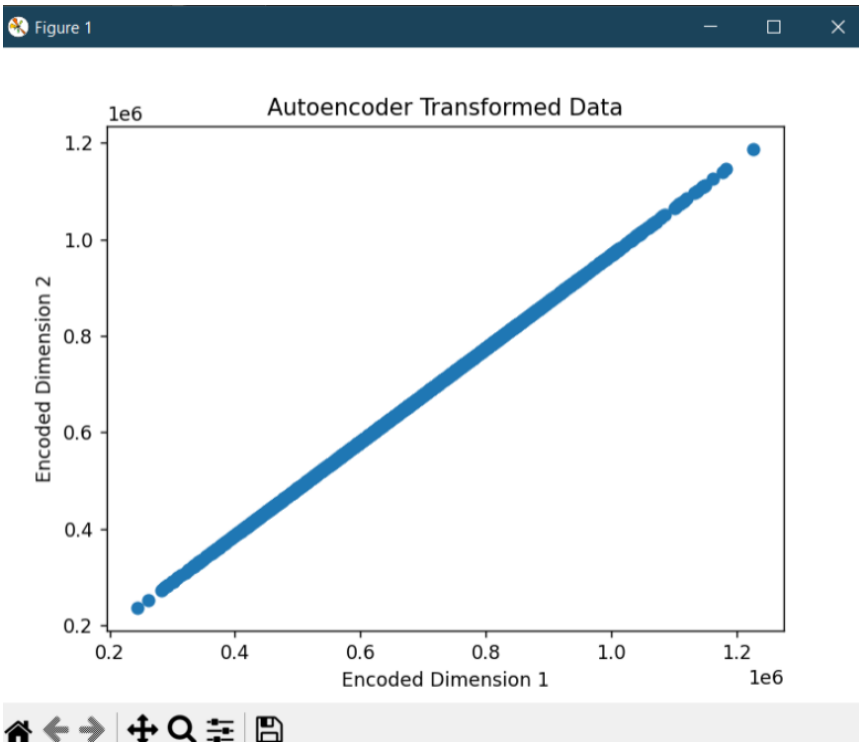
Στη συνέχεια και μετά την μάθηση του Autoencoder, θα χρησιμοποιήσουμε μόνο τον encoder (1ο μισό του δικτύου) για να προβάλλουμε όλα τα δεδομένα (και των δύο αρχείων) στον νέο χώρο M-διάστασης.

Παρακάτω υλοποιούμε έναν autoencoder, χρησιμοποιώντας την βιβλιοθήκη Keras. Με το m που συμβολίζει τις διαστάσεις του μετασχηματισμένου χώρου, να παίρνει τις τιμές 2,10 και 50.

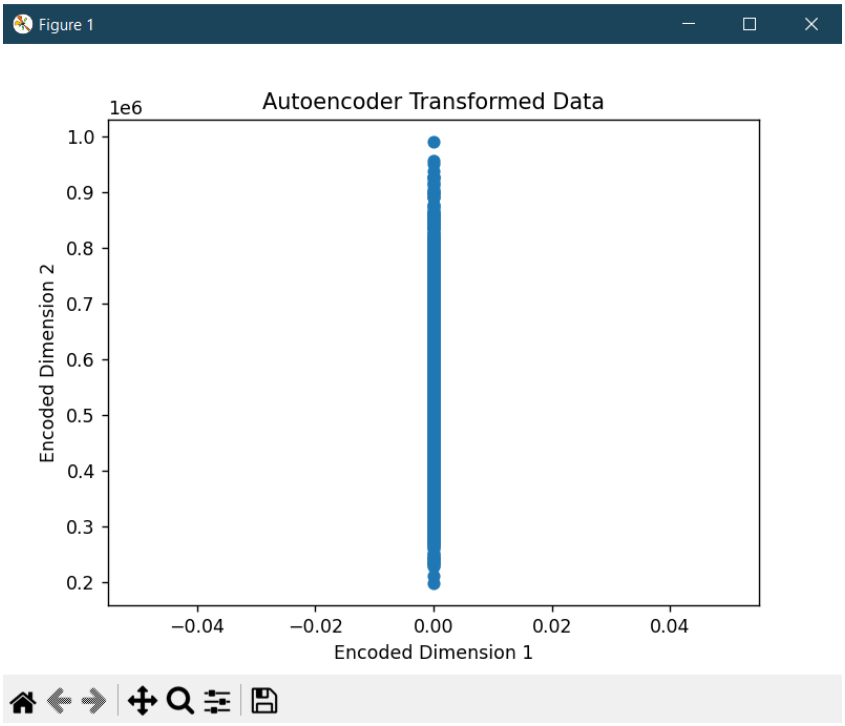
```
for m in encoding_dim:
    input_layer = Input(shape=(input_dim,))
    encoder = Dense(100, activation='relu')(input_layer)
    encoder = Dense(m, activation='relu')(encoder)

    decoder = Dense(100, activation='relu')(encoder)
    decoder = Dense(input_dim, activation='sigmoid')(decoder)

    autoencoder = Model(inputs=input_layer, outputs=decoder)

    encoder_model = Model(inputs=input_layer, outputs=encoder)

    autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

    autoencoder.fit(x, x, epochs=10, batch_size=32)

    encoded_data = encoder_model.predict(x)
```

Αρχικά, ορίζουμε το input layer με σχήμα (input_dim,), όπου το input_dim συμβολίζει την διάσταση των δεδομένων εισόδου. Στην συνέχεια, δημιουργούμε ένα πυκνό (dense) επίπεδο με 100 νευρώνες και συνάρτηση ενεργοποίησης relu, το οποίο λαμβάνει ως είσοδο την διάσταση των δεδομένων. Έχοντας δημιουργήσει το πυκνό επίπεδο που αναφέραμε μόλις, το χρησιμοποιούμε για να δημιουργήσουμε ένα πυκνό επίπεδο με m νευρώνες. Ορίζουμε επίσης, έναν πυκνό αποκωδικοποιητή (decoder) με 100 νευρώνες και συνάρτηση ενεργοποίησης Relu, το οποίο λαμβάνει ως είσοδο το προηγούμενο πυκνό επίπεδο. Έπειτα δημιουργούμε και έναν πυκνό αποκωδικοποιητή με σχήμα (input_dim,) και συνάρτηση ενεργοποίησης sigmoid, το οποίο λαμβάνει ως είσοδο το προηγούμενο πυκνό επίπεδο. Στην συνεχεια, δημιουργούμε το μοντέλο του autoencoder, το οποίο έχει ως είσοδο το input layer και ως έξοδο τον decoder. Επίσης ορίζουμε ένα διαφορετικό μοντέλο του κωδικοποιητή, το οποίο έχει ως είσοδο το input layer και ως έξοδο τον κωδικοποιητή (encoder). Αυτό το μοντέλο χρησιμοποιείται για να εξάγουμε τις συμπιεσμένες αναπαραστάσεις των δεδομένων. Χρησιμοποιούμε τον βελτιστοποιητή 'adam' και τη συνάρτηση απώλειας 'binary_crossentropy' για το μοντέλο του autoencoder. Εκπαιδεύουμε αυτό το μοντέλου για 10 εποχές, χρησιμοποιώντας τα δεδομένα x ως είσοδο και τις ετικέτες. Χρησιμοποιώντας το μοντέλο του encoder, προβλέπουμε τις συμπιεσμένες αναπαραστάσεις των δεδομένων x και τις αποθηκεύουμε στη μεταβλητή "encoded_data", η οποία θα χρησιμοποιηθεί στη θέση του trainingData για το χώρισμα των δεδομένων σε train και test. Στη συνέχεια ακολουθούμε την ίδια διαδικασία με το πρώτο μέρος, με τη μόνη διαφορά ότι τα input δεδομένα μας είναι του encoder(1ο μισό του δικτύου) όπου προβάλλονται όλα τα δεδομένα στον νέο χώρο M-διάστασης.

Στην περίπτωση που το m=2 και ο encoder μας επιστρέφει δεδομένα 2 διαστάσεων. Δημιουργούμε το αντίστοιχο plot λόγο του ότι τα λιγότερων διαστάσεων δεδομένα μπορούμε να τα κάνουμε visualise για την καλύτερη αντίληψή τους. Παρακάτω απεικονίζονται δύο περιπτώσεις των δυσδιάστατων δεδομένων.



KMEANS PLOT M=2



AGGLOMERATIVE PLOT M=2

Συμπεράσματα

Παρακάτω φαίνονται τα αποτελέσματα των υλοποιήσεων μας για τους δύο αλγορίθμους.

- Στο Clustering των αρχικών δεδομένων καταλήγουμε πως η βέλτιστη μέθοδος που θα προκύψει από την σύγκρισή τους είναι η Agglomerative Hierarchical Clustering αλλά με ελάχιστες διαφορές, τέτοιες ώστε να μη μπορούμε να καταλήξουμε ότι είναι 'καλύτερη'. Παρεμπιπτόντως, σε κάθε πείραμα ήταν πάντα ελάχιστα βέλτιστη.
- Στον Autoencoder: καταλήγουμε πως η βέλτιστη μέθοδος που θα προκύψει από την σύγκρισή τους είναι η K-means αλλά με ελάχιστες διαφορές, ομοίως και με το clustering των αρχικών δεδομένων.

Οπότε, τελικά δε μπορούμε να καταλήξουμε ότι κάποια μέθοδος είναι με διαφορά βέλτιστη.

Method	***Dimension (M)***	***Number of Clusters (K)***	***Purity***	***F-measure***	***Total F-measure***	***Mean Purity***
K-means (Euclidean distance)	**2**	2	0.497499	0.664069	***11.807211067955626***	***0.71375***
		4	0.664374	0.798105		
		6	0.696249	0.820129		
		8	0.703750	0.825961		
		10	0.704375	0.826353		
	10	2	0.498125	0.664661		
		4	0.6625	0.796763		
		6	0.691874	0.817299		
		8	0.695000	0.819827		
		10	0.7	0.823302		
	50	2	0.499375	0.665795		
		4	0.669374	0.801800		
		6	0.7025	0.824498		
		8	0.703750	0.825901		
		10	0.71375	0.832741		

Agglomerative Hierarchical Clustering	**2**	2	0.497499	0.663706	***11.759871608728329***	***0.71***
		4	0.64375	0.782079		
		6	0.675	0.805404		
		8	0.695624	0.820148		
		10	0.703750	0.825895		
	10	2	0.495	0.661520		
		4	0.645625	0.783151		
		6	0.691249	0.817271		
		8	0.7025	0.825086		
		10	0.7075	0.828499		
	50	2	0.503125	0.669032		
		4	0.665625	0.798535		
		6	0.696874	0.821237		
		8	0.706874	0.828078		
		10	0.71	0.830221		

Results