Μέθοδοι Data Clustering και Dimension Reduction

Αναφορά πρώτης φάσης: Μέθοδοι data clustering για ταξινόμηση τιμών κινητής τηλεφωνίας

Εισαγωγή

Αυτή η αναφορά παρουσιάζει τα αποτελέσματα της εφαρμογής δύο γνωστών αλγορίθμων ομαδοποίησης, του K-means και του Agglomerative Hierarchical Clustering, στο dataset ''Mobile Price Classification''. Αυτό το dataset, χωρισμένο σε υποσύνολα training και test, αποτελείται από 2.000 σημεία δεδομένων με 20 χαρακτηριστικά και κατηγοριοποιείται σε 4 classes (0, 1, 2, 3).

Μέθοδος

Η επιλεγμένη γλώσσα για αυτήν την εργασία είναι η Python, αξιοποιώντας βασικές βιβλιοθήκες για χειρισμό δεδομένων, μηχανική εκμάθηση και οπτικοποίηση. Το πρώτο βήμα περιλαμβάνει την εισαγωγή αυτών των βιβλιοθηκών, ακολουθούμενη από φόρτωση δεδομένων από τα αρχεία «training.csv» και «test.csv» χρησιμοποιώντας τη βιβλιοθήκη pandas. Το στάδιο προεπεξεργασίας των δεδομένων περιλαμβάνει την αφαίρεση του target variable «price_range» από το training set και του identifier «id» από το test set. Ακολούθησε τυποποίηση του συνόλου χαρακτηριστικών για να εξασφαλιστεί μέσος όρος 0 και τυπική απόκλιση 1, χρησιμοποιώντας το StandardScaler της Scikit-Learn.

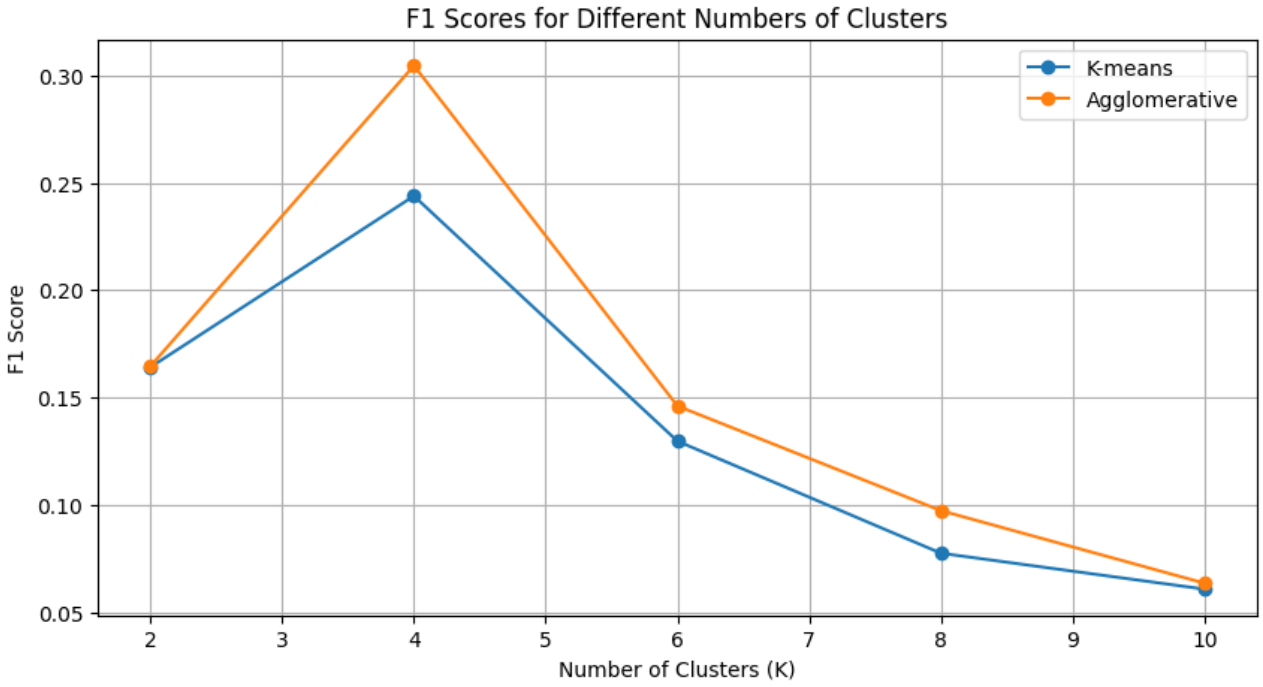
Για την υλοποίηση των δύο clustering algorithms, επαναλάβαμε διαφορετικούς αριθμούς clusters K={2, 4, 6, 8, 10}. Ειδικά για τον αλγόριθμο K-means, κάθε τιμή του K εκτελέστηκε 10 φορές για να ληφθούν υπόψη διαφορετικές αρχικοποιήσεις και τα αποτελέσματα υπολογίστηκαν κατά μέσο όρο. Το Agglomerative Clustering δεν απαιτούσε πολλαπλές εκτελέσεις λόγω της ντετερμινιστικής φύσης του, ανεξάρτητα από την αρχικοποίηση. Η απόδοση αυτών των μεθόδων αξιολογήθηκε από δύο μετρήσεις: purity και F1 score.

Πειραματικά αποτελέσματα

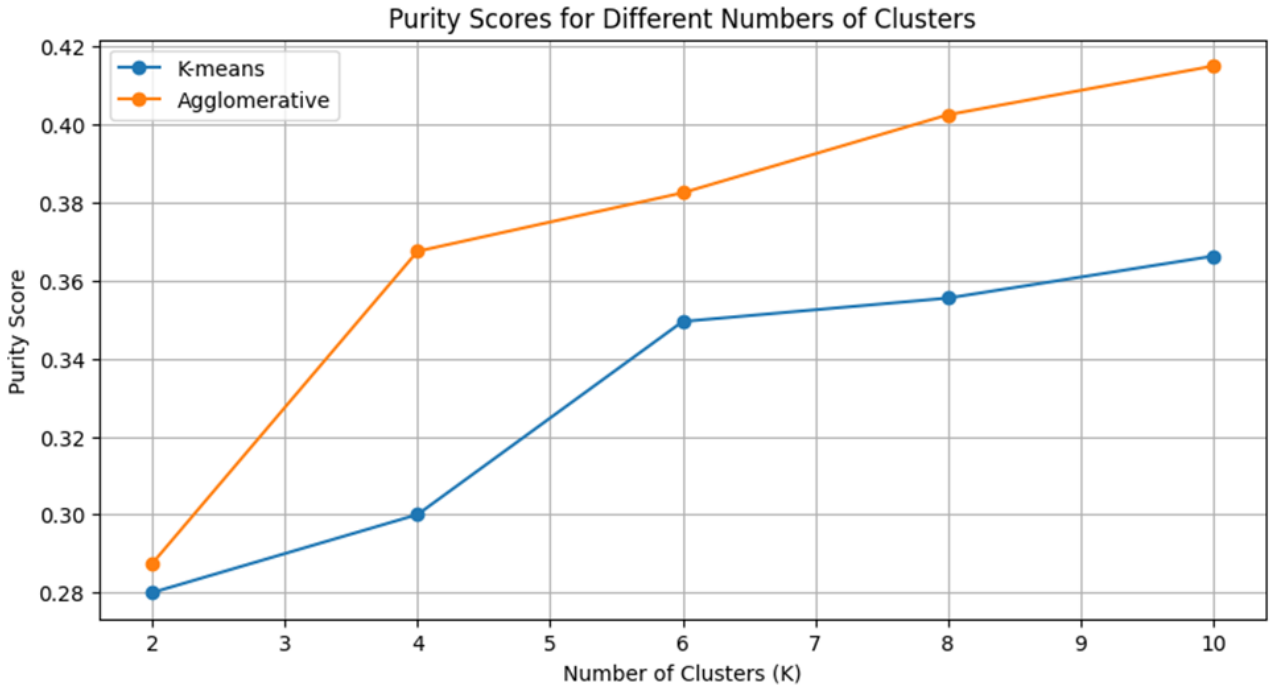
Τα αποτελέσματα από τον αλγόριθμο K-means και τη μέθοδο Agglomerative Clustering που εφαρμόζονται στο dataset παρουσιάζονται στον πίνακα 1 παρακάτω:

Method	Number of Clusters (K)	Purity	F-measure
K-means (Euclidean distance)	**2**	**0.280**	**0.164**
	4	**0.300**	**0.244**
	6	**0.349**	**0.130**
	8	**0.355**	**0.078**
	10	**0.366**	**0.061**
Agglomerative Hierarchical Clustering	**2**	**0.287**	**0.165**
	4	**0.368**	**0.305**
	6	**0.382**	**0.146**
	8	**0.403**	**0.097**
	10	**0.415**	**0.064**

Για την περαιτέρω κατανόηση της απόδοσης, το μέσο purity και τα F1 scores για διαφορετικές τιμές του K σχεδιάζονται για κάθε μέθοδο clustering.



Από το διάγραμμα F1, παρατηρούμε ότι η υψηλότερη βαθμολογία F1 και για τις δύο μεθόδους επιτυγχάνεται στο K = 4, υποδεικνύοντας τον βέλτιστο αριθμό clusters.



Επιπλέον, το διάγραμμα καθαρότητας δείχνει ότι οι δύο μέθοδοι αποκλίνουν περισσότερο στο K = 4, παρέχοντας πρόσθετη υποστήριξη για την επιλογή αυτού του αριθμού clusters.

Συμπέρασμα

Με βάση το purity και το F1 measure, το Agglomerative Clustering φαίνεται να αποδίδει καλύτερα από το K-means. Συγκεκριμένα, το Agglomerative Clustering με K = 4 παρέχει μια καλή ισορροπία με υψηλό **purity 0,368** και το υψηλότερο **F1 score 0,305**. Ωστόσο, αυτές οι μετρήσεις δεν καταγράφουν όλες τις πτυχές του clustering, όπως το σχήμα ή την πυκνότητα. Επιπλέον, ο βέλτιστος αριθμός clusters ενδέχεται να μην βρίσκεται στο εύρος που δοκιμάστηκε και η απόδοση θα μπορούσε να επηρεάζεται εάν οι υποκείμενες παραδοχές αυτών των μεθόδων δεν ευθυγραμμίζονται με το data structure. Επομένως, ενδέχεται να απαιτείται μια πιο ολοκληρωμένη ανάλυση για τα βέλτιστα αποτελέσματα.

Αναφορά δεύτερης φάσης: Μη εποπτευόμενη μάθηση με Autoencoders

Εισαγωγή

Στη δεύτερη φάση του πρότζεκτ, χρησιμοποιούμε autoencoders για να επιτύχουμε μείωση των διαστάσεων των δεδομένων μας πριν από την εκτέλεση του clustering. Οι autoencoders είναι μοντέλα νευρωνικών δικτύων που εκπαιδεύονται να ανακατασκευάζουν τα inputs τους. Περιορίζοντας τον αριθμό των νευρώνων στα κρυμμένα στρώματα, το μοντέλο αναγκάζεται να μάθει μια συμπιεσμένη, χαμηλών διαστάσεων αναπαράσταση των δεδομένων, η οποία θα μπορούσε ενδεχομένως να βελτιώσει την απόδοση του clustering.

Μέθοδος

Ο αυτόματος κωδικοποιητής κατασκευάζεται με ένα input layer που αντιστοιχεί στον αριθμό των χαρακτηριστικών στο dataset μας (20 σε αυτήν την περίπτωση), ένα encoding dimension που έχει οριστεί σε 100 και τρία διακριτά hidden layer dimensions ή τιμές «M», που ορίζονται ως [2, 10, 50]. Χρησιμοποιούμε το Dense layer 'Keras' για το σκοπό αυτό, όπως και το 'relu' activation για τα hidden layers και 'sigmoid' activation για το output layer.

Στη συνέχεια, εκτελούμε K-means και Agglomerative Clustering στα κωδικοποιημένα δεδομένα με παρόμοιο τρόπο με αυτόν που περιεγράφηκε στην φάση Ι, μόνο που αυτή τη φορά χρησιμοποιούμε dimensionally reduced representation. Το μέσο purity και τα F1 scores υπολογίζονται για κάθε συνδυασμό των «M» (διάσταση των κωδικοποιημένων δεδομένων) και «K» (αριθμός clusters).

Πειραματικά αποτελέσματα

Από τα αποτελέσματα, παρατηρείται ότι η χρήση autoencoders δεν βελτιώνει δραστικά την απόδοση των μοντέλων clustering για τις ίδιες τιμές του «K» σε συγκριτικά με ότι είδαμε στην πρώτη φάση. Είναι ενδιαφέρον ότι, ενώ το συνολικό purity των clusters βελτιώνεται ελαφρώς σε ορισμένες περιπτώσεις, τα F1 scores γενικά μειώνονται. Αυτό υποδηλώνει ότι ενώ οι ομάδες γίνονται όλο και πιο ομοιογενείς σε σχέση με την πλειοψηφική τάξη, κάνουν χειρότερη δουλειά στο να συλλαμβάνουν όλες τις περιπτώσεις αυτής της τάξης μέσα σε ένα δεδομένο cluster.

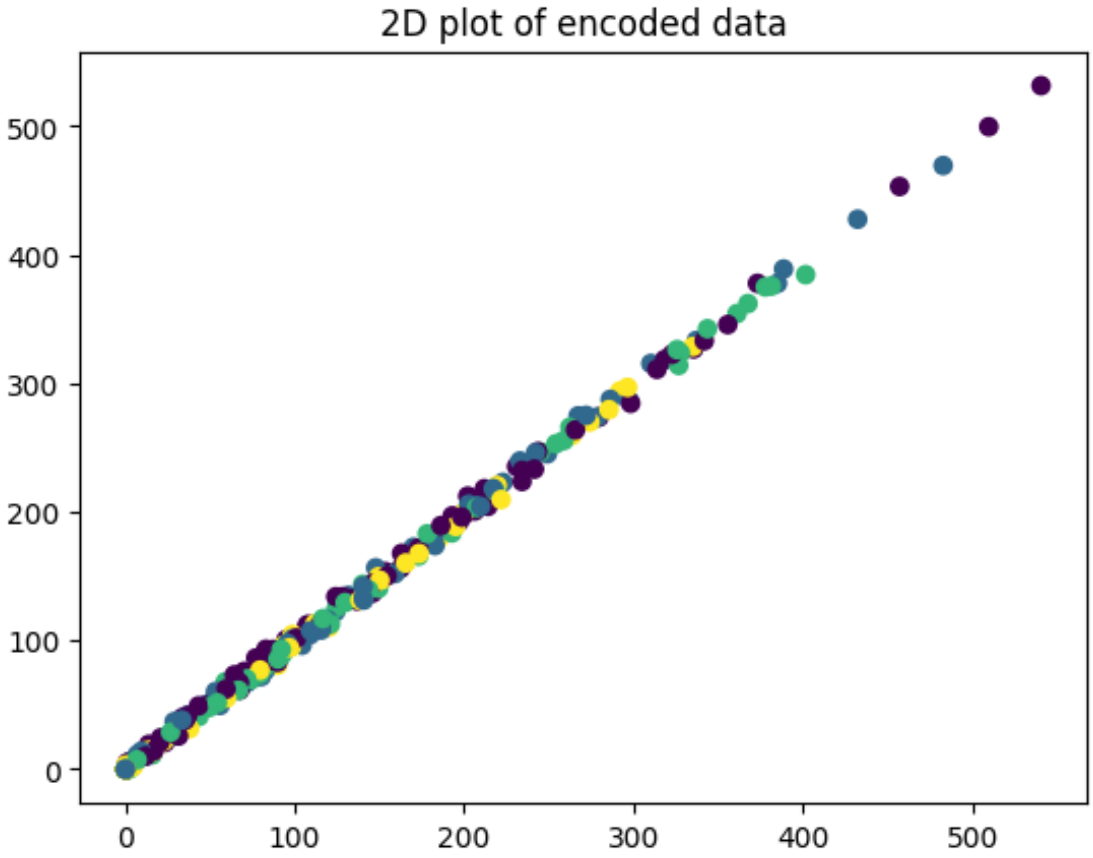
Συγκεκριμένα, η διαμόρφωση με τις καλύτερες επιδόσεις όσον αφορά τα F1 scores επιτεύχθηκε με τη μέθοδο Agglomerative Clustering, χρησιμοποιώντας K=4 clusters και M=10 dimensions για τα κωδικοποιημένα δεδομένα, η οποία πέτυχε μέσο F1 score **0,238**.

Τα αποτελέσματα θα τα δούμε στον παρακάτω πίνακα 2:

Πίνακας 2: Απόδοση clustering μειωμένων διαστάσεων με χρήση Autoencoder

Method	Dimension (M)	Number of Clusters (K)	Purity	F-measure
K-means (Euclidean distance)	2	2	0.338	0.153
		4	0.370	0.200
		6	0.382	0.111
		8	0.412	0.066
		10	0.417	0.049
	10	2	0.330	0.164
		4	0.375	0.187
		6	0.372	0.112
		8	0.389	0.075
		10	0.393	0.064
	50	2	0.343	0.161
		4	0.370	0.183
		6	0.382	0.113
		8	0.426	0.066
		10	0.410	0.051
Agglomerative Hierarchical Clustering	2	2	0.338	0.149
		4	0.360	0.176
		6	0.398	0.113
		8	0.403	0.069
		10	0.415	0.051
	10	2	0.325	0.155
		4	0.368	0.238
		6	0.372	0.132
		8	0.388	0.065
		10	0.393	0.047
	50	2	0.335	0.152
		4	0.335	0.176
		6	0.398	0.116
		8	0.400	0.057
		10	0.417	0.041

Για M=2, δημιουργείται ένα scatter plot, που παρουσιάζεται παρακάτω, των δισδιάστατων κωδικοποιημένων δεδομένων. Αυτό το διάγραμμα αντιπροσωπεύει τον μειωμένο χώρο δεδομένων, απεικονίζοντας οπτικά την κατανομή των πραγματικών ετικετών. Η γραμμική τάση που παρατηρείται στο διάγραμμα διασποράς σημαίνει ότι ο αυτόματος κωδικοποιητής μας έχει μάθει μια χαμηλών διαστάσεων αναπαράσταση των δεδομένων, όπου τα αρχικά σημεία δεδομένων υψηλής διάστασης που είναι παρόμοια μεταξύ τους είναι επίσης κοντά στον δισδιάστατο λανθάνοντα χώρο.



Συμπέρασμα δεύτερης φάσης

Η εφαρμογή των autoencoders για τη μείωση των διαστάσεων προσφέρει μια διαφορετική προοπτική στα δεδομένα και έχει αξιοσημείωτο αντίκτυπο στα clusters. Η μείωση του F1 score δείχνει ότι ενώ τα clusters είναι πιο ομοιογενή, δεν κάνουν απαραίτητα καλύτερη δουλειά στην καταγραφή όλων των περιπτώσεων της πλειοψηφικής κατηγορίας. Τα αποτελέσματα δείχνουν ότι υπάρχει ένας συμβιβασμός κατά τη μείωση της διάστασης των δεδομένων, όπου ορισμένες πληροφορίες χάνονται αναπόφευκτα.

Σύνοψη

Στις δύο φάσεις αυτού του πρότζεκτ, παρατηρήσαμε διαφορετικές επιδόσεις μοντέλων clustering με βάση την προσέγγιση. Στη πρώτη φάση, εφαρμόσαμε τεχνικές clustering απευθείας στα δεδομένα υψηλών διαστάσεων. Αυτό είχε ως αποτέλεσμα clusters που ήταν λιγότερο ομοιογενείς, όπως αντικατοπτρίζεται στις χαμηλότερες βαθμολογίες purity. Ωστόσο, τα clusters ήταν πιο περιεκτικά από όλες τις περιπτώσεις της πλειοψηφικής κατηγορίας, γεγονός που οδήγησε σε σχετικά υψηλότερα F1 scores.

Αντίθετα, η δεύτερη φάση μελέτησε την επίδραση της μείωσης των διαστάσεων στα μοντέλα clustering. Παρά το γεγονός ότι οδήγησαν σε πιο ομοιογενείς ομάδες, τα F1 scores γενικά μειώθηκαν, υποδηλώνοντας ότι ενώ οι ομάδες ήταν πιο καθαρές, δεν ήταν τόσο περιεκτικές σε όλες τις περιπτώσεις της πλειοψηφικής κατηγορίας.

Τελικά, η καλύτερη προσέγγιση θα εξαρτηθεί από πιο ειδικούς στόχους και περιορισμούς του εκάστοτε επιχειρήματος. Εάν το purity είναι κυρίαρχο, τότε η προσέγγιση dimensionality reduction της δεύτερης φασης θα μπορούσε να είναι καταλληλότερη. Από την άλλη, εάν ο στόχος είναι να συμπεριληφθούν όλες οι περιπτώσεις μιας συγκεκριμένης τάξης μέσα σε ένα cluster, η προσέγγιση της πρώτης φάσης θα είναι πιο επωφελής. Ωστόσο, όπως παρατηρήθηκε στα αποτελέσματά μας, η μέθοδος Agglomerative Clustering με

clusters K=4 στον διαστατικά μειωμένο χώρο M=10, παρείχε την καλύτερη συνολική απόδοση όσον αφορά την εξισορρόπηση τόσο του purity όσο και του F1 measure.

Ο κώδικας:

##Πρώτο μέρος

```
# Importing Required Libraries
import numpy as np
import pandas as pd
from sklearn import preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import accuracy_score, f1_score
import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")

# Downloading the Dataset

# Loading the Data
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

# Exploring the Data
print("Training Data:")
print(train_data.head())
print("\nTesting Data:")
print(test_data.head())



# Checking for Missing Values
```

```
print("\nMissing Values in Training Data:")
print(train_data.isnull().sum())
print("\nMissing Values in Testing Data:")
print(test_data.isnull().sum())

# Preprocessing the Data
# Assuming no missing values, we proceed to normalize the features.
# Note: The 'price_range' column in the training data is the target variable and should not be normalized.

features_train = train_data.drop('price_range', axis=1)
features_test = test_data.drop('id', axis=1)  # Assuming 'id' is not a feature

scaler = preprocessing.StandardScaler()

# Fit on training set only.
scaler.fit(features_train)

# Apply transform to both the training set and the test set.
train_scaled = scaler.transform(features_train)
test_scaled = scaler.transform(features_test)

# Convert back to DataFrame for convenience
train_scaled = pd.DataFrame(train_scaled, columns=features_train.columns)
test_scaled = pd.DataFrame(test_scaled, columns=features_test.columns)

print("\nScaled Training Data:")
print(train_scaled.head())
print("\nScaled Testing Data:")
print(test_scaled.head())
from sklearn.metrics import confusion_matrix
from scipy.stats import mode
```

```
# Function to calculate purity
def purity_score(y_true, y_pred):
    # compute confusion matrix
    confusion_matrix_ = confusion_matrix(y_true, y_pred)
    # return purity
    return np.sum(np.amax(confusion_matrix_, axis=0)) / np.sum(confusion_matrix_)

# Function to calculate F1 score
def f1_score_func(y_true, y_pred):
    return f1_score(y_true, y_pred, average='macro')
# Split the original training data into a new training set and a validation set
train_scaled, val_scaled, train_labels, val_labels = train_test_split(train_scaled, train_data['price_range'], test_size=0.2, random_state=42)

# Define the range of K
K_values = [2, 4, 6, 8, 10]

# Store the results
results = []

# Loop over K values
for K in K_values:
    purity_scores = []
    f1_scores = []
    for _ in range(10):  # 10 executions
        # Initialize KMeans
        kmeans = KMeans(n_clusters=K, random_state=None)

        # Fit on training data
        kmeans.fit(train_scaled)

        # Predict on validation data
        y_pred = kmeans.predict(val_scaled)
```

```
        # Calculate purity and F1 score
        purity = purity_score(val_labels, y_pred)
        f1 = f1_score_func(val_labels, y_pred)

        # Store the scores
        purity_scores.append(purity)
        f1_scores.append(f1)

    # Calculate average scores
    avg_purity = np.mean(purity_scores)
    avg_f1 = np.mean(f1_scores)

    # Store the results
    results.append((K, avg_purity, avg_f1))

# Print the results
for K, purity, f1 in results:
    print(f"K={K}, Average Purity={purity:.3f}, Average F1 Score={f1:.3f}")

# Store the results
results_agglo = []

# Loop over K values
for K in K_values:
    # Initialize scores list
    purity_scores_agglo = []
    f1_scores_agglo = []

    for _ in range(10):  # run the method 10 times
        # Initialize AgglomerativeClustering
        agglomerative = AgglomerativeClustering(n_clusters=K, linkage='ward')
```

```
        # Fit on training data
        agglomerative.fit(train_scaled)

        # Predict on validation data
       # Note: AgglomerativeClustering does not have a 'predict' method, so we fit_predict on the validation data
        y_pred_agglo = agglomerative.fit_predict(val_scaled)

        # Calculate purity and F1 score
        purity_agglo = purity_score(val_labels, y_pred_agglo)
        f1_agglo = f1_score_func(val_labels, y_pred_agglo)

        # Store the scores
        purity_scores_agglo.append(purity_agglo)
        f1_scores_agglo.append(f1_agglo)

    # Compute average purity and F1 score
    avg_purity_agglo = np.mean(purity_scores_agglo)
    avg_f1_agglo = np.mean(f1_scores_agglo)

    # Store the average results
    results_agglo.append((K, avg_purity_agglo, avg_f1_agglo))

# Print the results
for K, purity, f1 in results_agglo:
    print(f"K={K}, Average Purity={purity:.3f}, Average F1 Score={f1:.3f}")
import matplotlib.pyplot as plt

# Extract scores
k_values = [2, 4, 6, 8, 10]
purity_kmeans = [result[1] for result in results]
f1_kmeans = [result[2] for result in results]
```

```
purity_agglo = [result[1] for result in results_agglo]
f1_agglo = [result[2] for result in results_agglo]

# Plot purity scores
plt.figure(figsize=(10, 5))
plt.plot(k_values, purity_kmeans, label='K-means', marker='o')
plt.plot(k_values, purity_agglo, label='Agglomerative', marker='o')
plt.title('Purity Scores for Different Numbers of Clusters')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Purity Score')
plt.legend()
plt.grid(True)
plt.show()

# Plot F1 scores
plt.figure(figsize=(10, 5))
plt.plot(k_values, f1_kmeans, label='K-means', marker='o')
plt.plot(k_values, f1_agglo, label='Agglomerative', marker='o')
plt.title('F1 Scores for Different Numbers of Clusters')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('F1 Score')
plt.legend()
plt.grid(True)
plt.show()

##Δεύτερο Μέρος
from keras.layers import Input, Dense
from keras.models import Model
import matplotlib.pyplot as plt

# Define the dimensions of the autoencoder
input_dim = train_scaled.shape[1]  # 20
```

```
encoding_dim = 100
hidden_dim = [2, 10, 50]  # M values

# Loop over M values
for M in hidden_dim:
    # Define the encoder
    input_layer = Input(shape=(input_dim,))
    encoded = Dense(encoding_dim, activation='relu')(input_layer)
    encoded = Dense(M, activation='relu')(encoded)

    # Define the decoder
    decoded = Dense(encoding_dim, activation='relu')(encoded)
    decoded = Dense(input_dim, activation='sigmoid')(decoded)

    # Define the autoencoder model
    autoencoder = Model(input_layer, decoded)

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

    # Train the autoencoder
    autoencoder.fit(train_scaled, train_scaled,
                    epochs=50,
                    batch_size=256,
                    shuffle=True,
                    validation_data=(val_scaled, val_scaled))

    # Define the encoder model
    encoder = Model(input_layer, encoded)

    # Transform the data
    train_encoded = encoder.predict(train_scaled)
```

```
val_encoded = encoder.predict(val_scaled)

# Repeat the clustering process with the transformed data
for K in K_values:
    purity_scores_kmeans = []
    f1_scores_kmeans = []
    purity_scores_agglo = []
    f1_scores_agglo = []
    for _ in range(10):  # 10 executions
        # Initialize KMeans
        kmeans = KMeans(n_clusters=K, random_state=None)

        # Fit on training data
        kmeans.fit(train_encoded)

        # Predict on validation data
        y_pred_kmeans = kmeans.predict(val_encoded)

        # Calculate purity and F1 score
        purity_kmeans = purity_score(val_labels, y_pred_kmeans)
        f1_kmeans = f1_score_func(val_labels, y_pred_kmeans)

        # Store the scores
        purity_scores_kmeans.append(purity_kmeans)
        f1_scores_kmeans.append(f1_kmeans)

        # Initialize AgglomerativeClustering
        agglomerative = AgglomerativeClustering(n_clusters=K, linkage='ward')

        # Fit and predict on validation data
        y_pred_agglo = agglomerative.fit_predict(val_encoded)
```

```
            # Calculate purity and F1 score
            purity_agglo = purity_score(val_labels, y_pred_agglo)
            f1_agglo = f1_score_func(val_labels, y_pred_agglo)

            # Store the scores
            purity_scores_agglo.append(purity_agglo)
            f1_scores_agglo.append(f1_agglo)

        # Calculate average scores
        avg_purity_kmeans = np.mean(purity_scores_kmeans)
        avg_f1_kmeans = np.mean(f1_scores_kmeans)
        avg_purity_agglo = np.mean(purity_scores_agglo)
        avg_f1_agglo = np.mean(f1_scores_agglo)

        # Print the results
        print(f"\nM={M}, K={K}")
        print(f"K-means: Average Purity={avg_purity_kmeans:.3f}, Average F1 Score={avg_f1_kmeans:.3f}")
        print(f"Agglomerative Hierarchical Clustering: Purity={avg_purity_agglo:.3f}, F1 Score={avg_f1_agglo:.3f}")

    # When M=2, create a scatter plot of the data in the 2-dimensional space
    if M == 2:
        plt.scatter(val_encoded[:, 0], val_encoded[:, 1], c=val_labels)
        plt.title('2D plot of encoded data')
        plt.show()
```