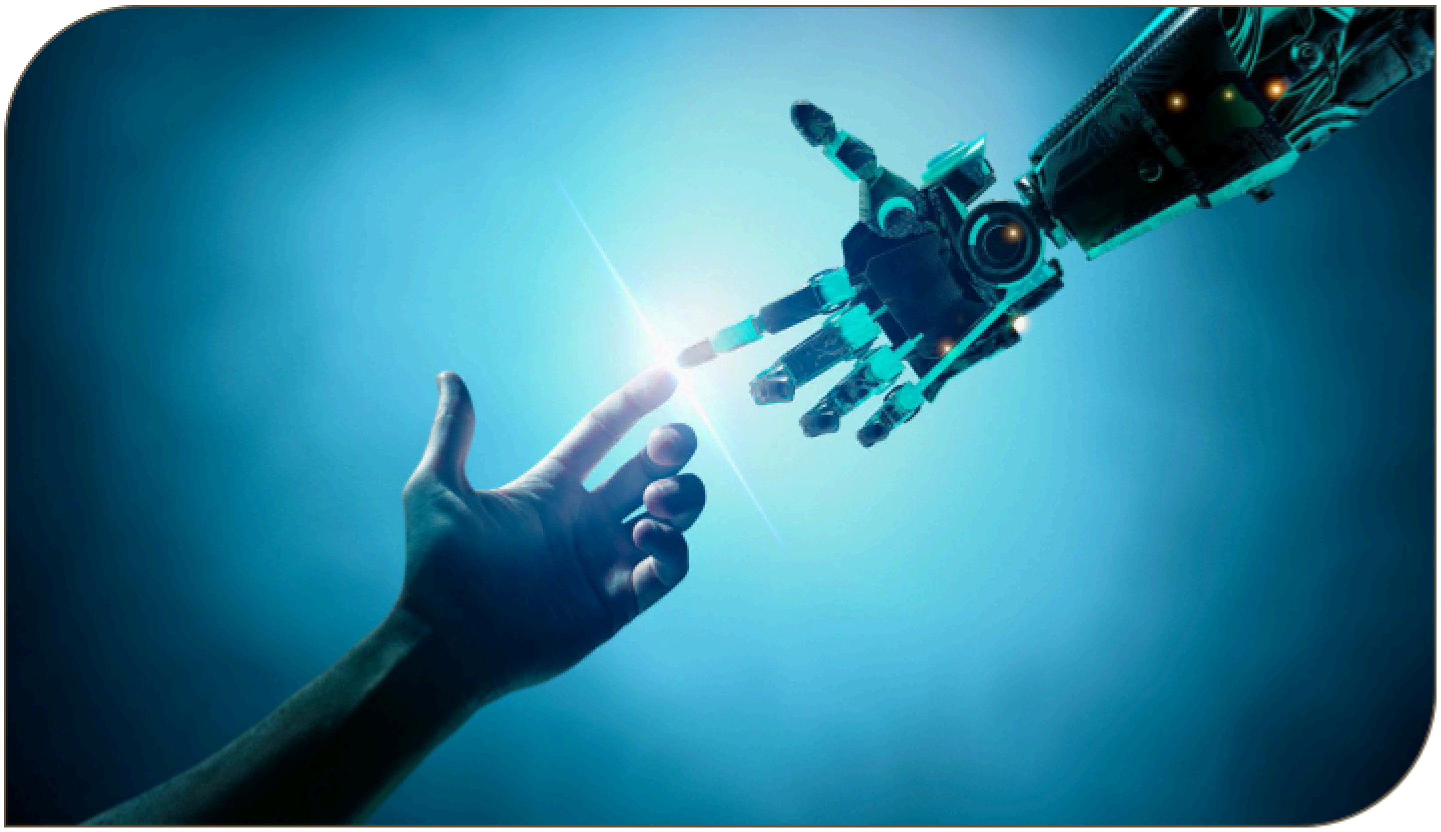


PATTERN RECOGNITION

Daily and Sports Activity Detection

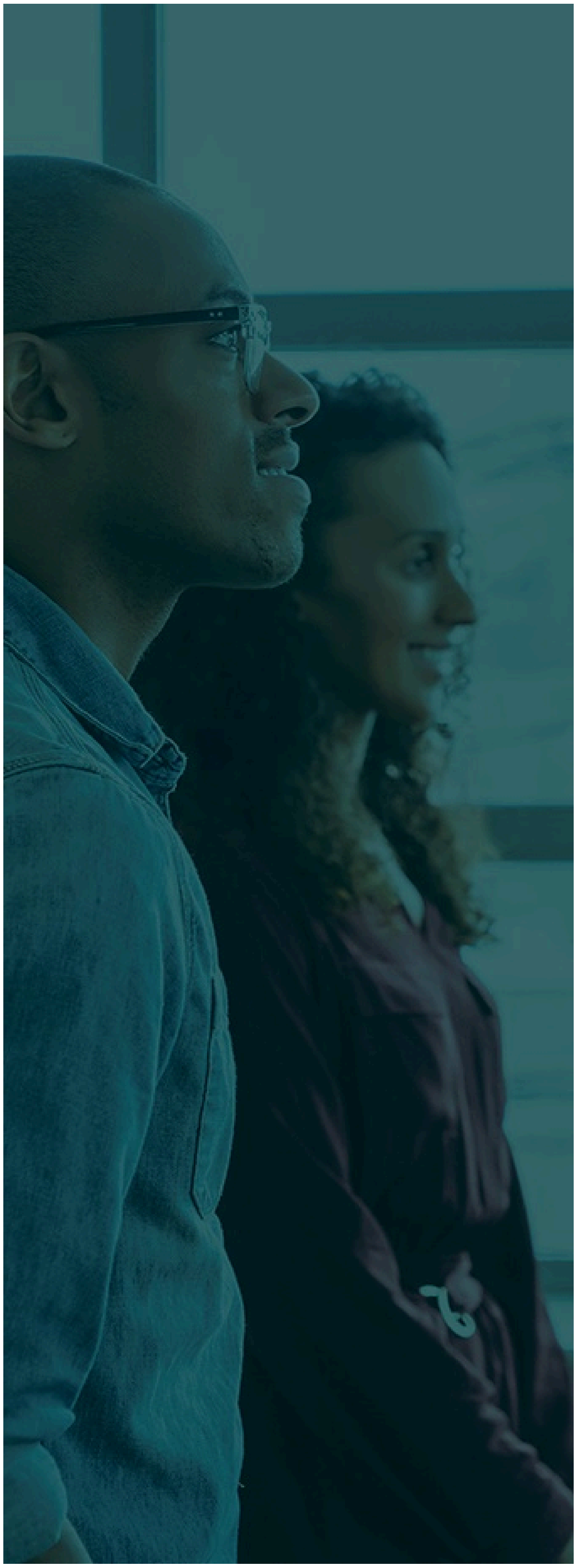


TABLE OF CONTENTS

Problem Statement

Clustering is a fundamental technique in unsupervised machine learning that plays a key role in organizing data into meaningful groups based on similarities. The objective of this assignment is to help students understand how K-Means and Normalized Cut algorithms can be used for the detection of daily and sports activities captured using motion sensors.

1. Download Dataset and Understand the Format
    - We will use the ”Daily and Sports Activity” dataset for this assignment.

2. Clustering Using K-Means and Normalized Cut (Your implementation)
    - We will use K-Means to cluster the daily and sports activity data and cluster them.
    - We will change the K of the K-means algorithm between {8, 13, 19, 28, 38} clusters. You will produce different clusters.

3. Normalized Cut (Your implementation)
    - Apply Normalized Cut algorithm to the two solutions to cluster the data into 19 clusters.
    - Compare the results of K-Means and Normalized Cut clustering regarding the number of detected activities and their characteristics.

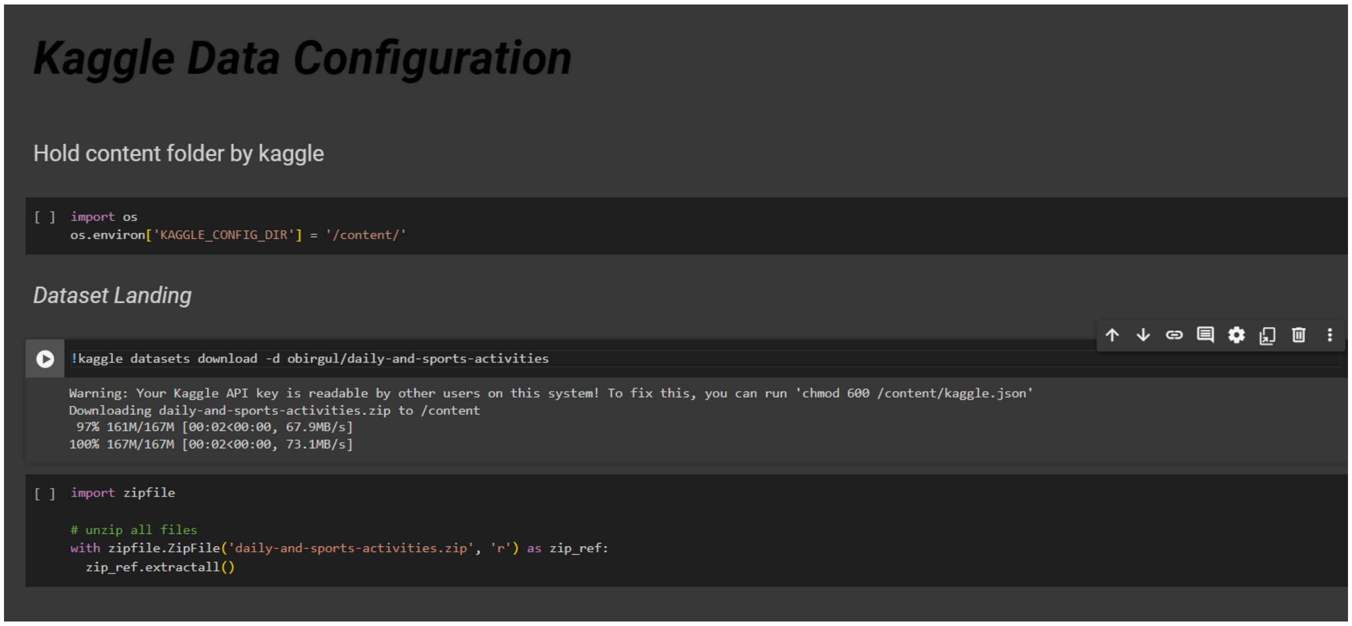
4. Evaluation
    - We will evaluate models based on their ability to detect daily and sports activities accurately. You will be required to use the following metrics to evaluate the quality of their models:
        - Precision
        - Recall
        - F1 score
        - Conditional Entropy

5. New Clustering Algorithm (Your implementation)
    - Choose any clustering technique of your own choice, implement it and repeat the above experiments.

1. Dataset Configuration

We have downloaded the "daily-and-sports-activities" dataset from Kaggle.

- **Downloading the dataset and understanding it's format:**



Kaggle Data Configuration

Hold content folder by kaggle

```
import os
os.environ['KAGGLE_CONFIG_DIR'] = '/content/'
```

Dataset Landing

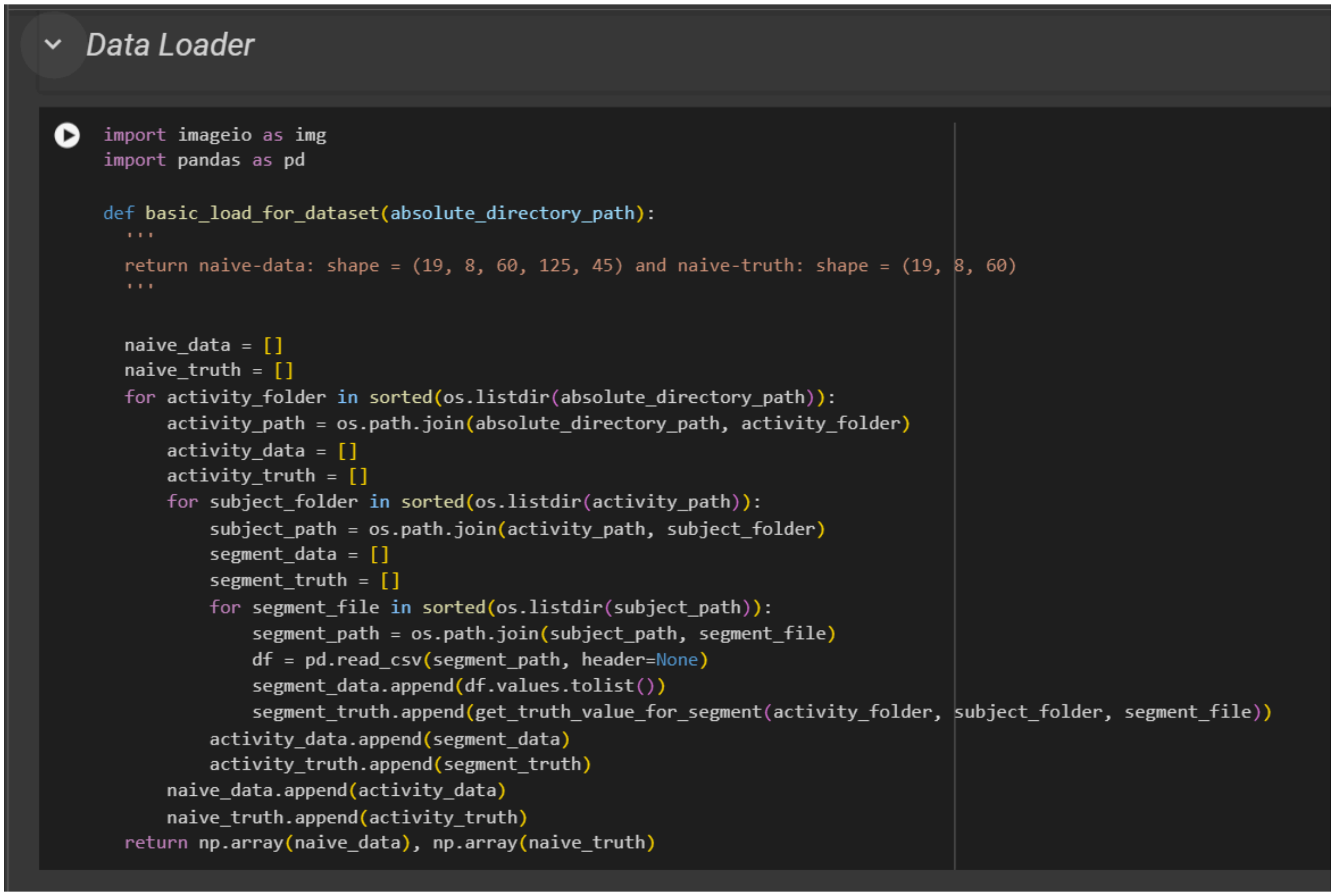
```
!kaggle datasets download -d obirgul/daily-and-sports-activities
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /content/kaggle.json'
Downloading daily-and-sports-activities.zip to /content
 97% 161M/167M [00:02<00:00, 67.9MB/s]
100% 167M/167M [00:02<00:00, 73.1MB/s]
```

```
import zipfile

# unzip all files
with zipfile.ZipFile('daily-and-sports-activities.zip', 'r') as zip_ref:
  zip_ref.extractall()
```

  - We've used the Kaggle CLI to download the dataset specifying it's slug (obirgul/daily-and-sports-activities).
  - Then we made a folder for activities and extracted the zip file of the dataset in it.
  - We have 19 labels and for each label we have 480 data points, so by total we have 9120 data points and each data point is 45 labels vector .

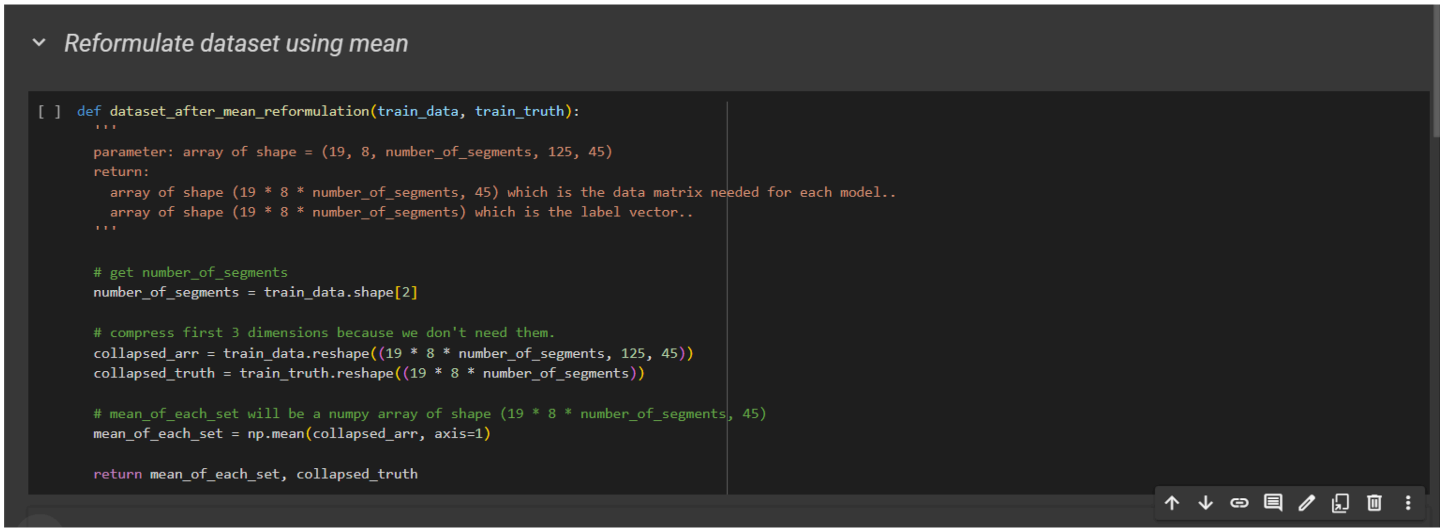
- **Generate the Data Matrix and the Label vector:**



```
import imageio as img
import pandas as pd

def basic_load_for_dataset(absolute_directory_path):
  '''
  return naive-data: shape = (19, 8, 60, 125, 45) and naive-truth: shape = (19, 8, 60)
  '''

  naive_data = []
  naive_truth = []
  for activity_folder in sorted(os.listdir(absolute_directory_path)):
      activity_path = os.path.join(absolute_directory_path, activity_folder)
      activity_data = []
      activity_truth = []
      for subject_folder in sorted(os.listdir(activity_path)):
          subject_path = os.path.join(activity_path, subject_folder)
          segment_data = []
          segment_truth = []
          for segment_file in sorted(os.listdir(subject_path)):
              segment_path = os.path.join(subject_path, segment_file)
              df = pd.read_csv(segment_path, header=None)
              segment_data.append(df.values.tolist())
              segment_truth.append(get_truth_value_for_segment(activity_folder, subject_folder, segment_file))
          activity_data.append(segment_data)
          activity_truth.append(segment_truth)
      naive_data.append(activity_data)
      naive_truth.append(activity_truth)
  return np.array(naive_data), np.array(naive_truth)
```

- We’ve created a function that takes the directory and iterates on each activity folder and for each subject.
- .For each subject read the 125x45 vector, and append it in the data matrix.
- for each point now we have two options:
  - take the mean for each column and now our labels vector is 1d of size 45.
  - Flatten the vector and perform PCA on it to reduce it to be 1d of size 45.

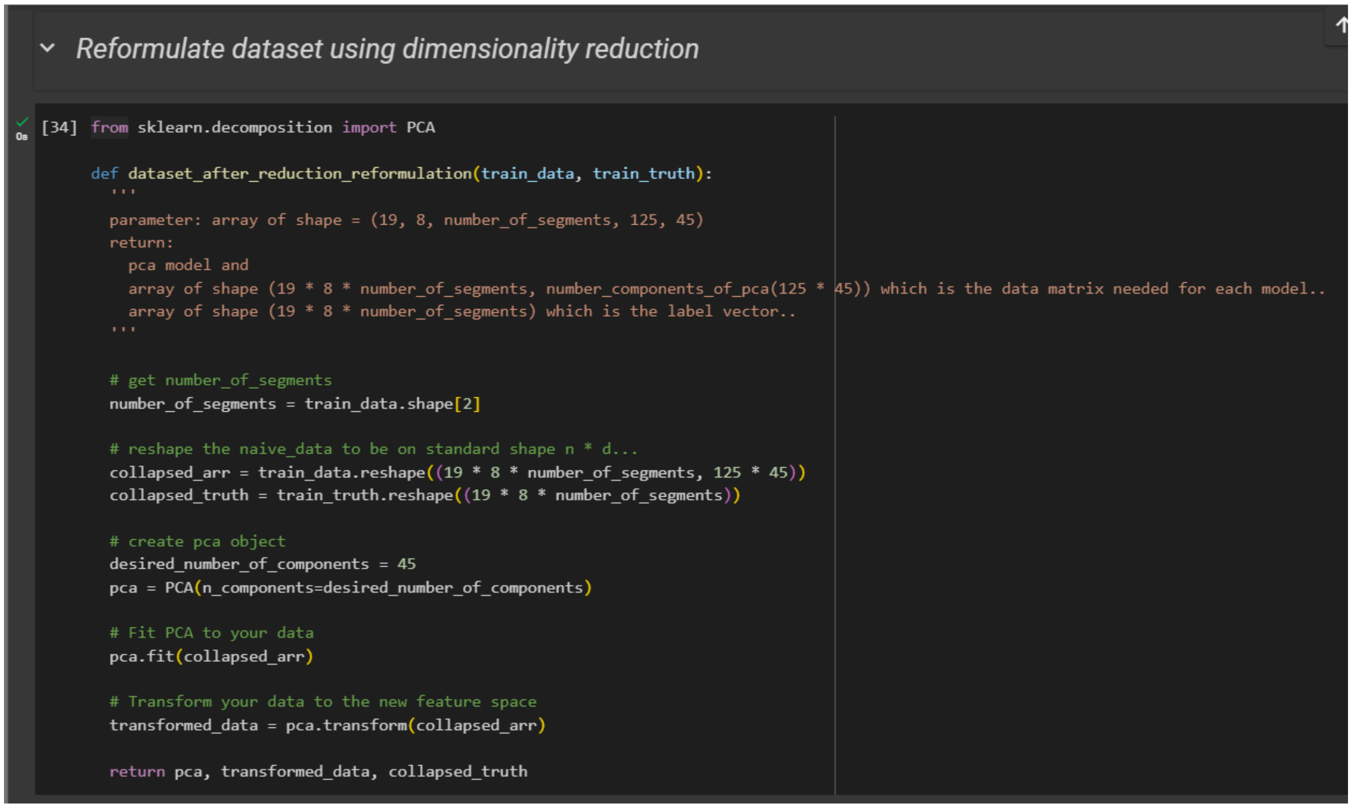
- **Reformulate the dataset using mean**



Reformulate dataset using mean

```
[ ] def dataset_after_mean_reformulation(train_data, train_truth):
      '''
      parameter: array of shape = (19, 8, number_of_segments, 125, 45)
      return:
        array of shape (19 * 8 * number_of_segments, 45) which is the data matrix needed for each model..
        array of shape (19 * 8 * number_of_segments) which is the label vector..
      '''

      # get number_of_segments
      number_of_segments = train_data.shape[2]

      # compress first 3 dimensions because we don't need them.
      collapsed_arr = train_data.reshape((19 * 8 * number_of_segments, 125, 45))
      collapsed_truth = train_truth.reshape((19 * 8 * number_of_segments))

      # mean_of_each_set will be a numpy array of shape (19 * 8 * number_of_segments, 45)
      mean_of_each_set = np.mean(collapsed_arr, axis=1)

      return mean_of_each_set, collapsed_truth
```

- **Reformulate the dataset using dimensionality reduction algorithm (PCA).**



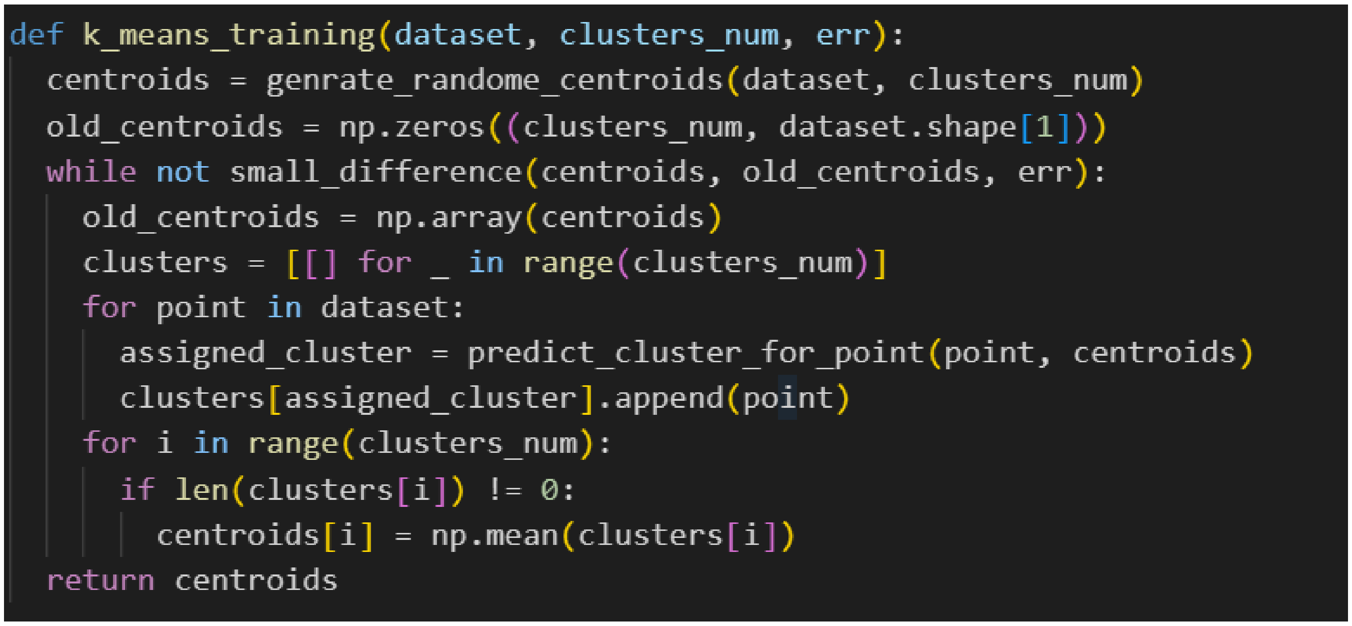
Reformulate dataset using dimensionality reduction

```
[34] from sklearn.decomposition import PCA

     def dataset_after_reduction_reformulation(train_data, train_truth):
       '''
       parameter: array of shape = (19, 8, number_of_segments, 125, 45)
       return:
         pca model and
         array of shape (19 * 8 * number_of_segments, number_components_of_pca(125 * 45)) which is the data matrix needed for each model..
         array of shape (19 * 8 * number_of_segments) which is the label vector..
       '''

       # get number_of_segments
       number_of_segments = train_data.shape[2]

       # reshape the naive_data to be on standard shape n * d...
       collapsed_arr = train_data.reshape((19 * 8 * number_of_segments, 125 * 45))
       collapsed_truth = train_truth.reshape((19 * 8 * number_of_segments))

       # create pca object
       desired_number_of_components = 45
       pca = PCA(n_components=desired_number_of_components)

       # Fit PCA to your data
       pca.fit(collapsed_arr)

       # Transform your data to the new feature space
       transformed_data = pca.transform(collapsed_arr)

       return pca, transformed_data, collapsed_truth
```

2. Clustering Using K-Means

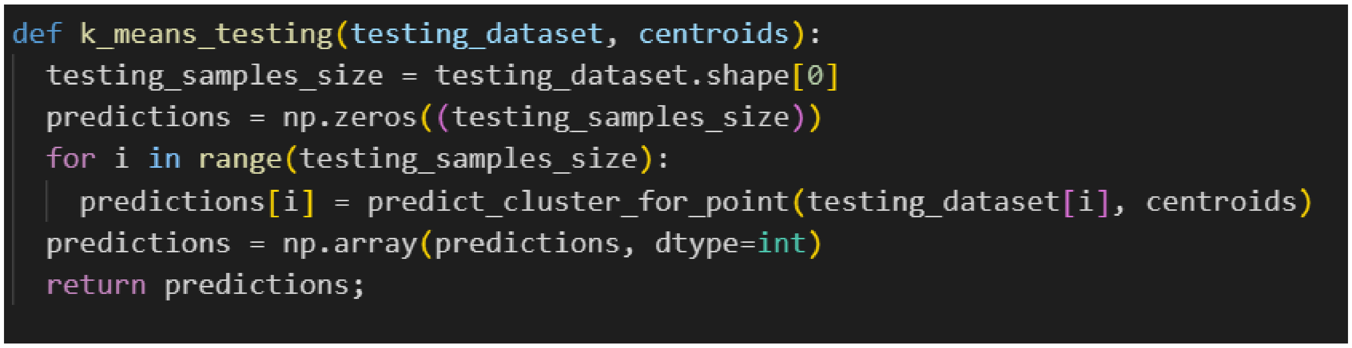
- **Code:**

- Training:
    - k-means clustering algorithm initializes random k centroids, in this implementation each centroid is a random point from the dataset. Then for each point in the training set, calculate the distance between it and each of the k centroids. Assign the point to the centroid of the smallest distance between it. After assigning each point to the nearest centroid, each centroid is updated to a new value that equals the mean of the points assigned to it. These steps are repeated until the norm of difference between old values and new values of each centroid is less than a given threshold.



```python
def k_means_training(dataset, clusters_num, err):
    centroids = genrate_randome_centroids(dataset, clusters_num)
    old_centroids = np.zeros((clusters_num, dataset.shape[1]))
    while not small_difference(centroids, old_centroids, err):
        old_centroids = np.array(centroids)
        clusters = [[] for _ in range(clusters_num)]
        for point in dataset:
            assigned_cluster = predict_cluster_for_point(point, centroids)
            clusters[assigned_cluster].append(point)
        for i in range(clusters_num):
            if len(clusters[i]) != 0:
                centroids[i] = np.mean(clusters[i])
    return centroids
```

- Testing:
    - For each point in the testing dataset, calculate the distance between it and each of the k centroids calculated in the training stage, and assign it to the centroid with the smallest distance between them.



```python
def k_means_testing(testing_dataset, centroids):
    testing_samples_size = testing_dataset.shape[0]
    predictions = np.zeros((testing_samples_size))
    for i in range(testing_samples_size):
        predictions[i] = predict_cluster_for_point(testing_dataset[i], centroids)
    predictions = np.array(predictions, dtype=int)
    return predictions;
```

- **Results of k-means:**

    - By splitting the data matrix into training and test sets by ratio 0.8 for training and 0.2 for testing.

By using mean reformulation.

K	Training set evaluation				Testing set evaluation			
	Precision	Recall	F1-Score	Entropy	Precision	Recall	F1-Score	Entropy
8	0.22957	0.4846	0.3539	3.07712	0.2655	0.45717	0.3403	3.1048
13	0.2509	0.3405	0.29804	2.944	0.25109	0.3407	0.2978	2.9196
19	0.27069	0.2571	0.26634	2.8509	0.26206	0.2489	0.2512	2.7965
28	0.2915	0.191	0.22959	2.758	0.29769	0.19504	0.2213	2.6282
38	0.3018	0.147	0.1953	2.7029	0.31907	0.15544	0.20614	2.5366

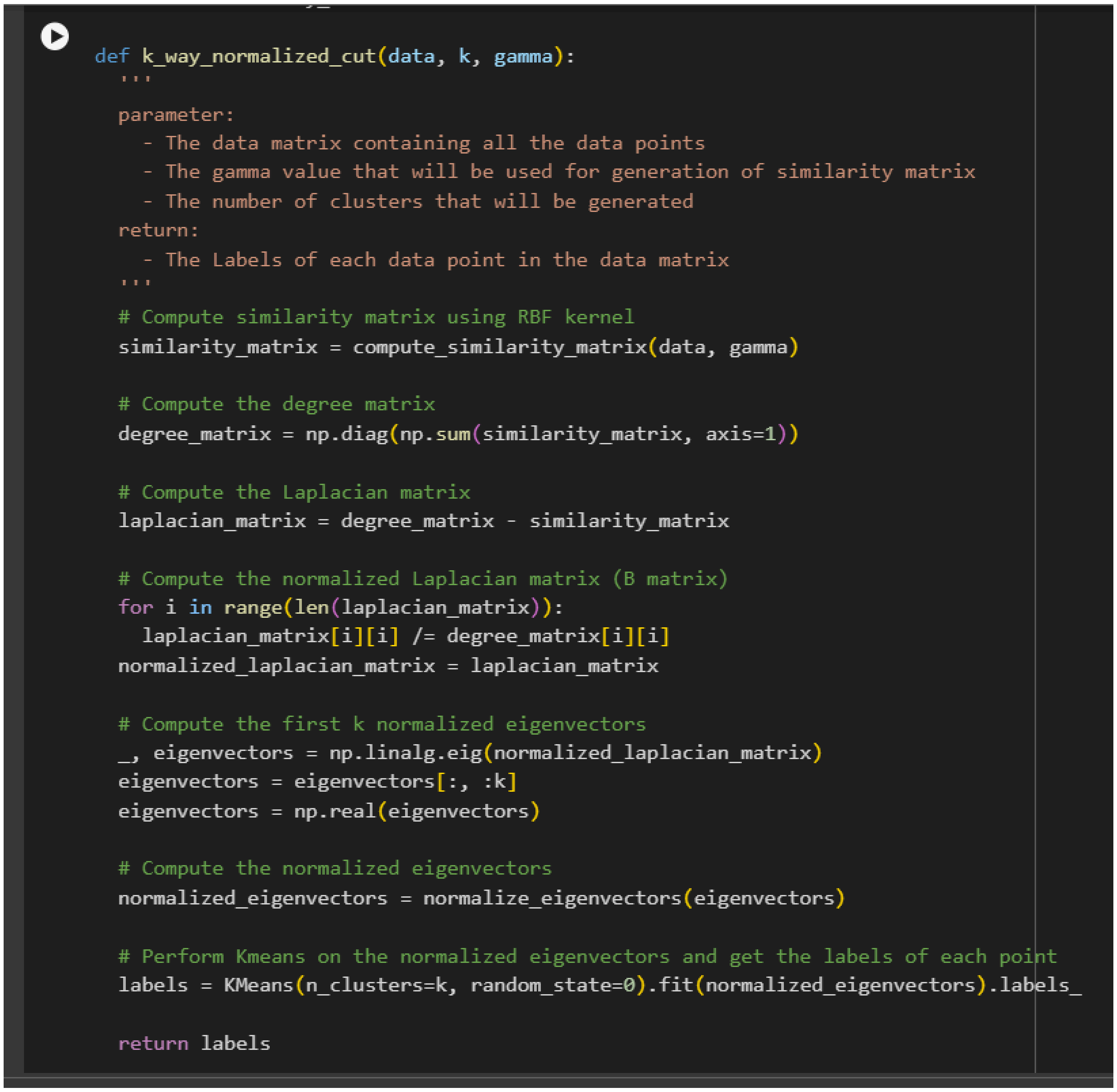
By using PCA reformulation

K	Training set evaluation				Testing set evaluation			
	Precision	Recall	F1-Score	Entropy	Precision	Recall	F1-Score	Entropy
8	0.18188	0.3839	0.28262	3.47597	0.18366	0.38773	0.29413	3.42913
13	0.19682	0.2671	0.23558	3.37264	0.20285	0.27529	0.24179	3.30504
19	0.20367	0.1934	0.20102	3.3442	0.21217	0.20156	0.21049	3.23586
28	0.22491	0.1473	0.17028	3.2857	0.22532	0.14762	0.17377	3.16323
38	0.23862	0.1162	0.15275	3.2419	0.25712	0.12526	0.16613	3.05277

- **Comments:**
    - The results are relatively worse than Spectral clustering which indicates that the data is not spherical so K-means is not the better approach to cluster it.
    - Precision score is better when near to 1 but average is 0.270146 and 0.217688 for mean reformulation and PCA reformulation respectively
    - F1 score is better when near to 1 but average is 0.270146 and 0.217688 for mean reformulation and PCA reformulation respectively
    - Entropy is better when near zero and worse is $\log_2(K)$ and average is 2.78738 and 3.349452 for mean reformulation and PCA reformulation respectively

3. Clustering Using Normalized-Cut

- **Code:**



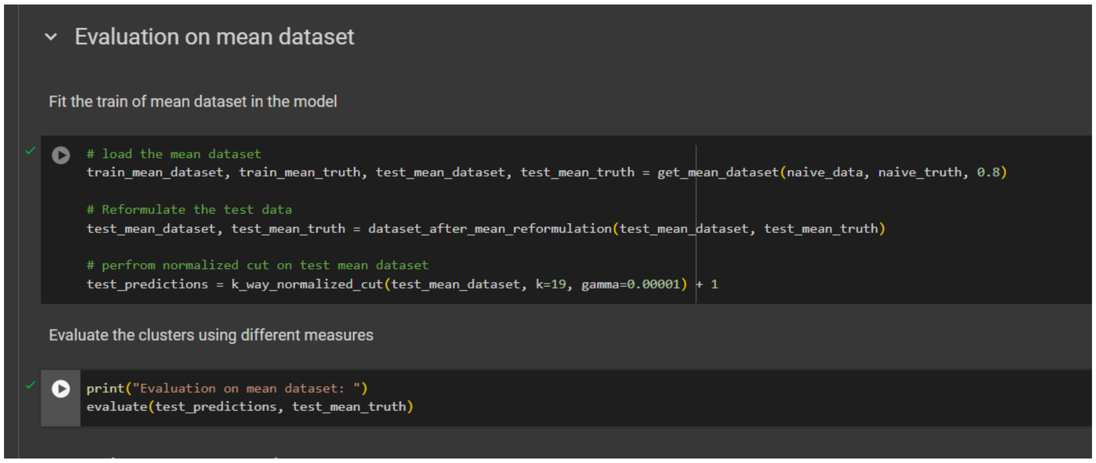
```
def k_way_normalized_cut(data, k, gamma):
  '''
  parameter:
    - The data matrix containing all the data points
    - The gamma value that will be used for generation of similarity matrix
    - The number of clusters that will be generated
  return:
    - The Labels of each data point in the data matrix
  '''
  # Compute similarity matrix using RBF kernel
  similarity_matrix = compute_similarity_matrix(data, gamma)

  # Compute the degree matrix
  degree_matrix = np.diag(np.sum(similarity_matrix, axis=1))

  # Compute the Laplacian matrix
  laplacian_matrix = degree_matrix - similarity_matrix

  # Compute the normalized Laplacian matrix (B matrix)
  for i in range(len(laplacian_matrix)):
    laplacian_matrix[i][i] /= degree_matrix[i][i]
  normalized_laplacian_matrix = laplacian_matrix

  # Compute the first k normalized eigenvectors
  _, eigenvectors = np.linalg.eig(normalized_laplacian_matrix)
  eigenvectors = eigenvectors[:, :k]
  eigenvectors = np.real(eigenvectors)

  # Compute the normalized eigenvectors
  normalized_eigenvectors = normalize_eigenvectors(eigenvectors)

  # Perform Kmeans on the normalized eigenvectors and get the labels of each point
  labels = KMeans(n_clusters=k, random_state=0).fit(normalized_eigenvectors).labels_

  return labels
```

- This function performs the algorithm of Normalized-Cut (Spectral Clustering).

- Compute the similarity matrix, which calculates the pairwise distance for all points.

- We've chosen the distance function as the RBF-Kernel function, with specified gamma = 0.00001.
- Calculate the diagonal matrix from the similarity matrix.
- Calculate the Laplacian matrix from the difference of degree matrix and similarity matrix.
- Calculate the normalized laplacian matrix from the dot product of inverse of degree matrix and laplacian matrix.
- Calculate the eigenvectors of the normalized laplacian matrix using eig function, then choose the first 19 eigenvectors then calculate the normalized eigenvectors.
- Perform K-means with 19 clusters on the normalized eigenvectors and return the labels for each datapoint.

- **Evaluation on two cases:**



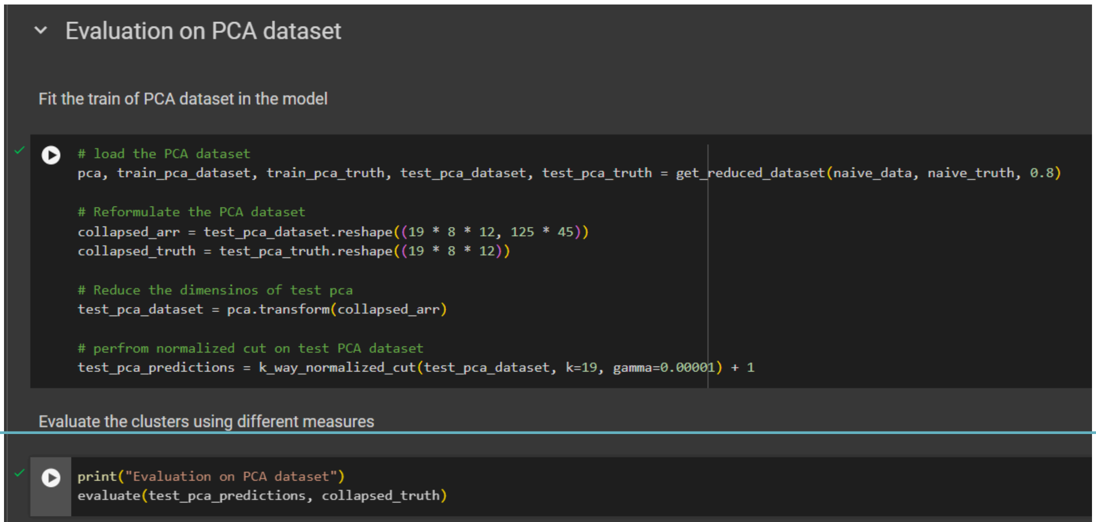
Evaluation on mean dataset

Fit the train of mean dataset in the model

```
# load the mean dataset
train_mean_dataset, train_mean_truth, test_mean_dataset, test_mean_truth = get_mean_dataset(naive_data, naive_truth, 0.8)

# Reformulate the test data
test_mean_dataset, test_mean_truth = dataset_after_mean_reformulation(test_mean_dataset, test_mean_truth)

# perfrom normalized cut on test mean dataset
test_predictions = k_way_normalized_cut(test_mean_dataset, k=19, gamma=0.00001) + 1
```

Evaluate the clusters using different measures

```
print("Evaluation on mean dataset: ")
evaluate(test_predictions, test_mean_truth)
```



Evaluation on PCA dataset

Fit the train of PCA dataset in the model

```
# load the PCA dataset
pca, train_pca_dataset, train_pca_truth, test_pca_dataset, test_pca_truth = get_reduced_dataset(naive_data, naive_truth, 0.8)

# Reformulate the PCA dataset
collapsed_arr = test_pca_dataset.reshape((19 * 8 * 12, 125 * 45))
collapsed_truth = test_pca_truth.reshape((19 * 8 * 12))

# Reduce the dimensinos of test pca
test_pca_dataset = pca.transform(collapsed_arr)

# perfrom normalized cut on test PCA dataset
test_pca_predictions = k_way_normalized_cut(test_pca_dataset, k=19, gamma=0.00001) + 1
```

Evaluate the clusters using different measures

```
print("Evaluation on PCA dataset")
evaluate(test_pca_predictions, collapsed_truth)
```

- **Compare with K-Means:**

- **Comparison on Testing set on mean reformulated data:**

K	K-Means				Spectral Clustering			
	Precision	Recall	F1-Score	Entropy	Precision	Recall	F1-Score	Entropy
19	0.26206	0.2489	0.2512	2.7965	0.42817	0.4067	0.42622	2.02945

- **Comparison on Testing set on dimensionality reduced data:**

K	K-Means				Spectral Clustering			
	Precision	Recall	F1-Score	Entropy	Precision	Recall	F1-Score	Entropy
19	0.26206	0.2489	0.2512	2.7965	0.43695	0.4151	0.44091	2.06317

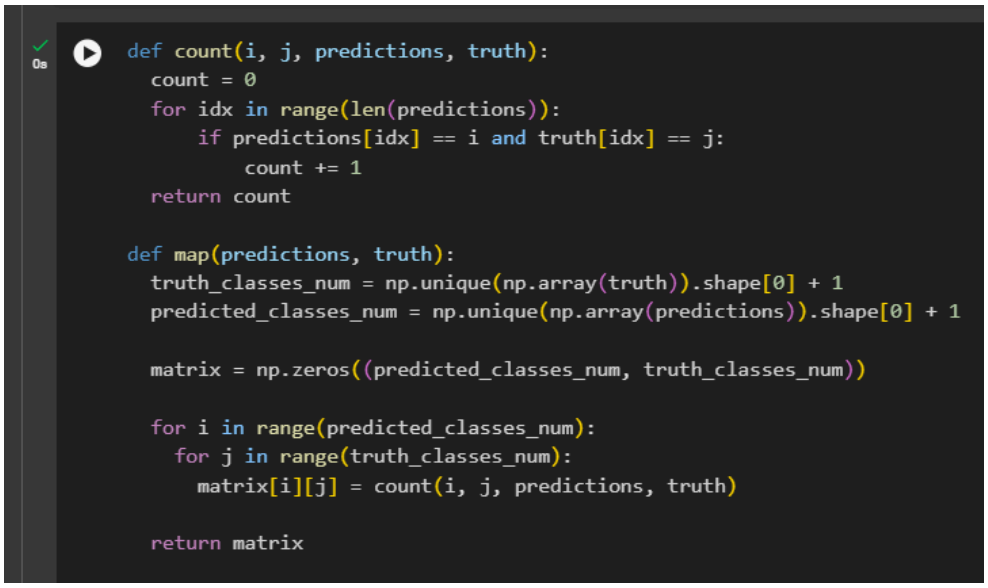
- **Comments:**

  - Spectral Clustering gives higher precision in both cases, as mentioned above the reason for this is that the data is not spherical.
  - It’s observed that the dimensionality reduced data gives higher precision than the mean reformulated data, and that because the mean summarizes the data without concerning the variance, while in PCA it captures the most important features of the data to increase the variance.

- If we perform the LDA algorithm instead of PCA, we'll get higher measures (precision) and that is because LDA exploits class information and enhances class separability.

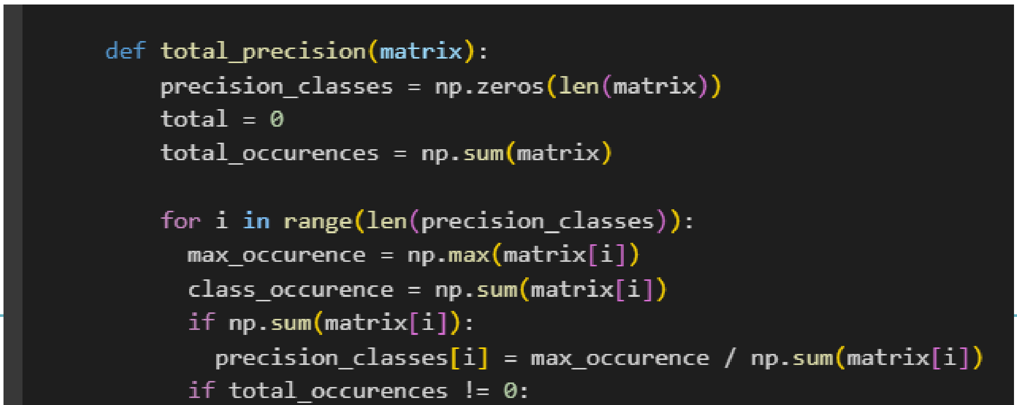
4. Evaluation Measures

- First, Build the contingency matrix.



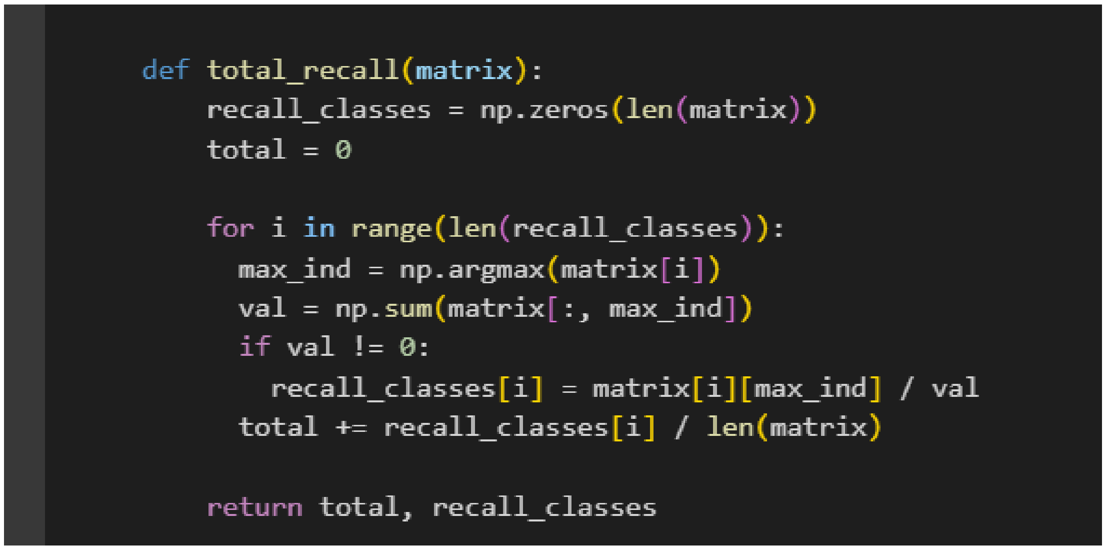
```
def count(i, j, predictions, truth):
  count = 0
  for idx in range(len(predictions)):
      if predictions[idx] == i and truth[idx] == j:
          count += 1
  return count

def map(predictions, truth):
  truth_classes_num = np.unique(np.array(truth)).shape[0] + 1
  predicted_classes_num = np.unique(np.array(predictions)).shape[0] + 1

  matrix = np.zeros((predicted_classes_num, truth_classes_num))

  for i in range(predicted_classes_num):
    for j in range(truth_classes_num):
      matrix[i][j] = count(i, j, predictions, truth)

  return matrix
```

- Given the contingency matrix, calculate the Precision.



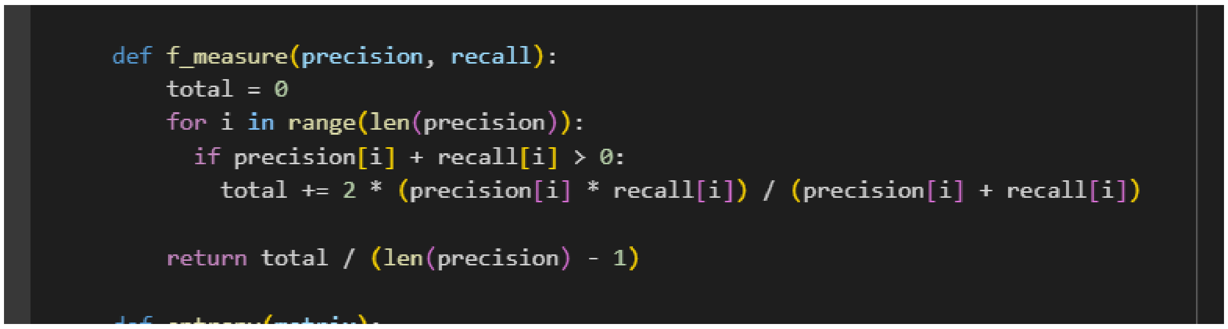
```
def total_precision(matrix):
    precision_classes = np.zeros(len(matrix))
    total = 0
    total_occurences = np.sum(matrix)

    for i in range(len(precision_classes)):
      max_occurence = np.max(matrix[i])
      class_occurence = np.sum(matrix[i])
      if np.sum(matrix[i]):
        precision_classes[i] = max_occurence / np.sum(matrix[i])
      if total_occurences != 0:
```

- Given the contingency matrix, calculate the Recall.



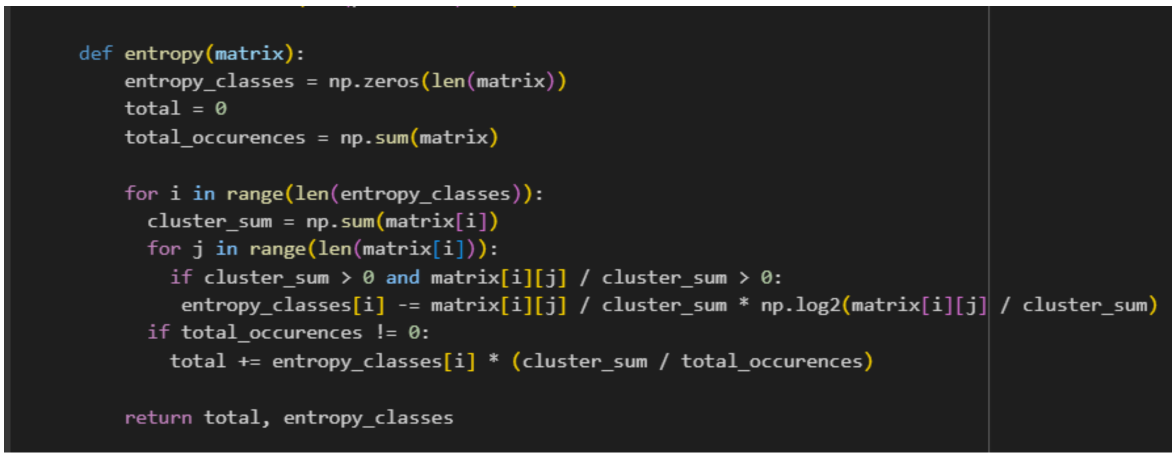
```python
def total_recall(matrix):
    recall_classes = np.zeros(len(matrix))
    total = 0

    for i in range(len(recall_classes)):
      max_ind = np.argmax(matrix[i])
      val = np.sum(matrix[:, max_ind])
      if val != 0:
        recall_classes[i] = matrix[i][max_ind] / val
      total += recall_classes[i] / len(matrix)

    return total, recall_classes
```

- Given the contingency matrix, calculate the F1-score.



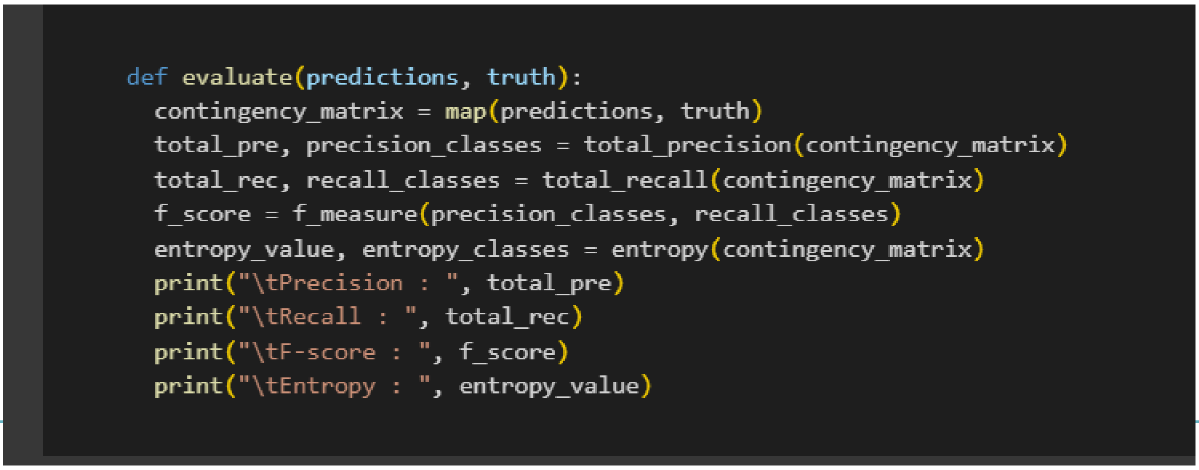
```python
def f_measure(precision, recall):
    total = 0
    for i in range(len(precision)):
      if precision[i] + recall[i] > 0:
        total += 2 * (precision[i] * recall[i]) / (precision[i] + recall[i])

    return total / (len(precision) - 1)

def entropy(matrix):
```

- Given the contingency matrix, calculate the Conditional Entropy.



```
def entropy(matrix):
    entropy_classes = np.zeros(len(matrix))
    total = 0
    total_occurences = np.sum(matrix)

    for i in range(len(entropy_classes)):
      cluster_sum = np.sum(matrix[i])
      for j in range(len(matrix[i])):
        if cluster_sum > 0 and matrix[i][j] / cluster_sum > 0:
         entropy_classes[i] -= matrix[i][j] / cluster_sum * np.log2(matrix[i][j] / cluster_sum)
      if total_occurences != 0:
        total += entropy_classes[i] * (cluster_sum / total_occurences)

    return total, entropy_classes
```

- Given the predictions and the ground truth, calculate all the measures.

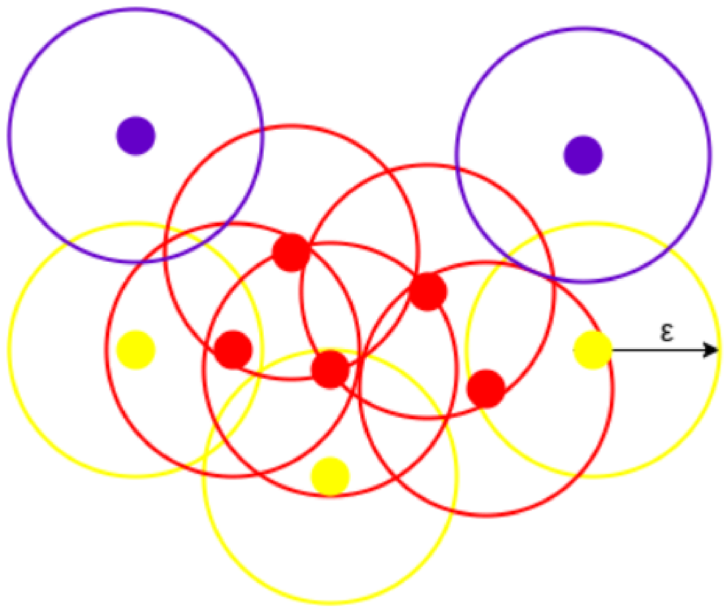


```
def evaluate(predictions, truth):
  contingency_matrix = map(predictions, truth)
  total_pre, precision_classes = total_precision(contingency_matrix)
  total_rec, recall_classes = total_recall(contingency_matrix)
  f_score = f_measure(precision_classes, recall_classes)
  entropy_value, entropy_classes = entropy(contingency_matrix)
  print("\tPrecision : ", total_pre)
  print("\tRecall : ", total_rec)
  print("\tF-score : ", f_score)
  print("\tEntropy : ", entropy_value)
```

5. Clustering Using DB-SCAN

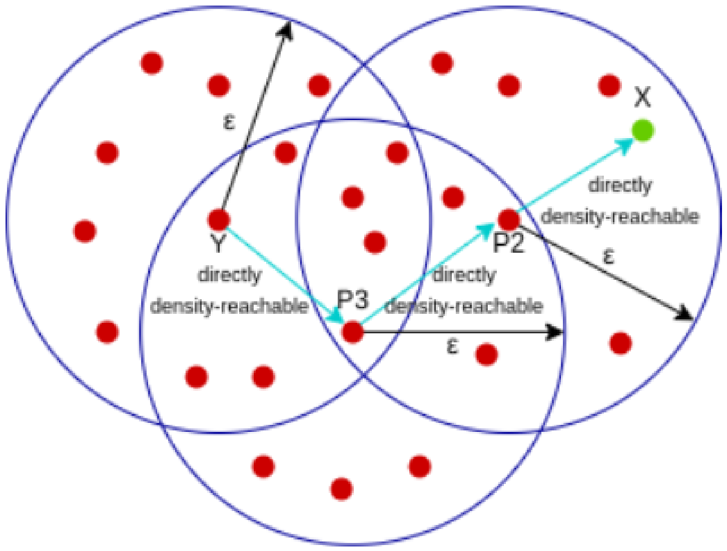
Intuition:

- Similar to the human naive clustering to each image of points, It groups 'densely grouped' data points into a single cluster so density of Circle is introduced to know how many nodes in the circle of a single point of radius ε.
- Moveover, different types of points are introduced: **Core**, Border and **Noise** points. Set of core points is a category of points that have at least k points in its ε-neighborhood. Furthermore, set of border points has less than k points in its ε-neighborhood but exists on a circle of a core point.

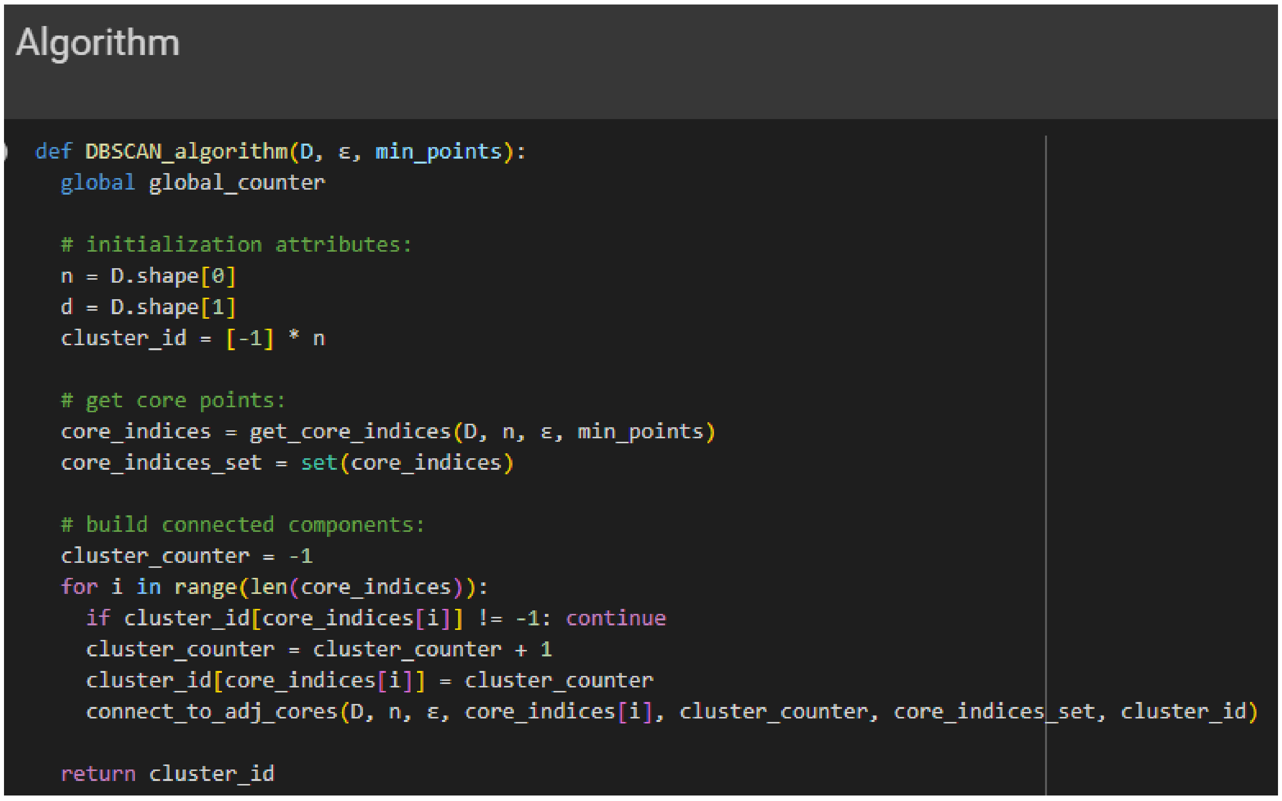


Algorithm:

- We get all core points in given data set according to the above intuition.
- Create a separate cluster:
    - To create Cluster, we start each core point with cluster i and for each density connected point, we will assign the same cluster i to this point.
    - After that, Recursively call each core point in the density connected points of previous point and apply same logic until no core point exists in the density connected points of the current point core point.
    - This totally means: assigning all reachable core points to same cluster.



Abstract Code:


Algorithm

```
def DBSCAN_algorithm(D, ε, min_points):
  global global_counter

  # initialization attributes:
  n = D.shape[0]
  d = D.shape[1]
  cluster_id = [-1] * n

  # get core points:
  core_indices = get_core_indices(D, n, ε, min_points)
  core_indices_set = set(core_indices)

  # build connected components:
  cluster_counter = -1
  for i in range(len(core_indices)):
    if cluster_id[core_indices[i]] != -1: continue
    cluster_counter = cluster_counter + 1
    cluster_id[core_indices[i]] = cluster_counter
    connect_to_adj_cores(D, n, ε, core_indices[i], cluster_counter, core_indices_set, cluster_id)

  return cluster_id
```

Advantages:

- is able to identify noise or outliers in the dataset

- It can identify clusters of arbitrary shape, making it suitable for datasets with complex cluster structures with flexibility of detecting the number of clusters.

Disadvantages:

- Very sensitive to two parameters ε and k.

- **Compare with K-Means:**

- **Comparison on Testing set on mean reformulated data:**

K	K-Means				DBscan			
	Precision	Recall	F1-Score	Entropy	Precision	Recall	F1-Score	Entropy
8	0.2655	0.4571	0.3403	3.1048	0.7338	0.25505	0.39253	0.8098

- **Comparison on Testing set on dimensionality reduced data:**

K	K-Means				DBscan			
	Precision	Recall	F1-Score	Entropy	Precision	Recall	F1-Score	Entropy
8	0.18366	0.3877	0.29413	3.42913	0.9999	0.1999	0.34386	0.0001

- **Comments:**

    - After Fitting DBSCAN over hyper-parameters that get best accuracy and evaluation values, We simply can observe that DBSCAN evaluation values are better than K-Means ones.
    - Observations almost make sense because DBSCAN can detect spherical data because of the perspective that each algorithm assign each cluster with more similarity.