Data Mining Project

Aditya Suryawanshi, 210150004
Harsh Raj , 210150011

1 Introduction

In this data mining project, we aim to utilize clustering techniques to identify underlying patterns in a dataset and then employ a neural network model to predict cluster labels for new data points. Clustering is a fundamental unsupervised learning technique used to group similar data points together based on certain features or characteristics. Once clusters are identified, a neural network model can be trained to classify new data points into these predefined clusters.

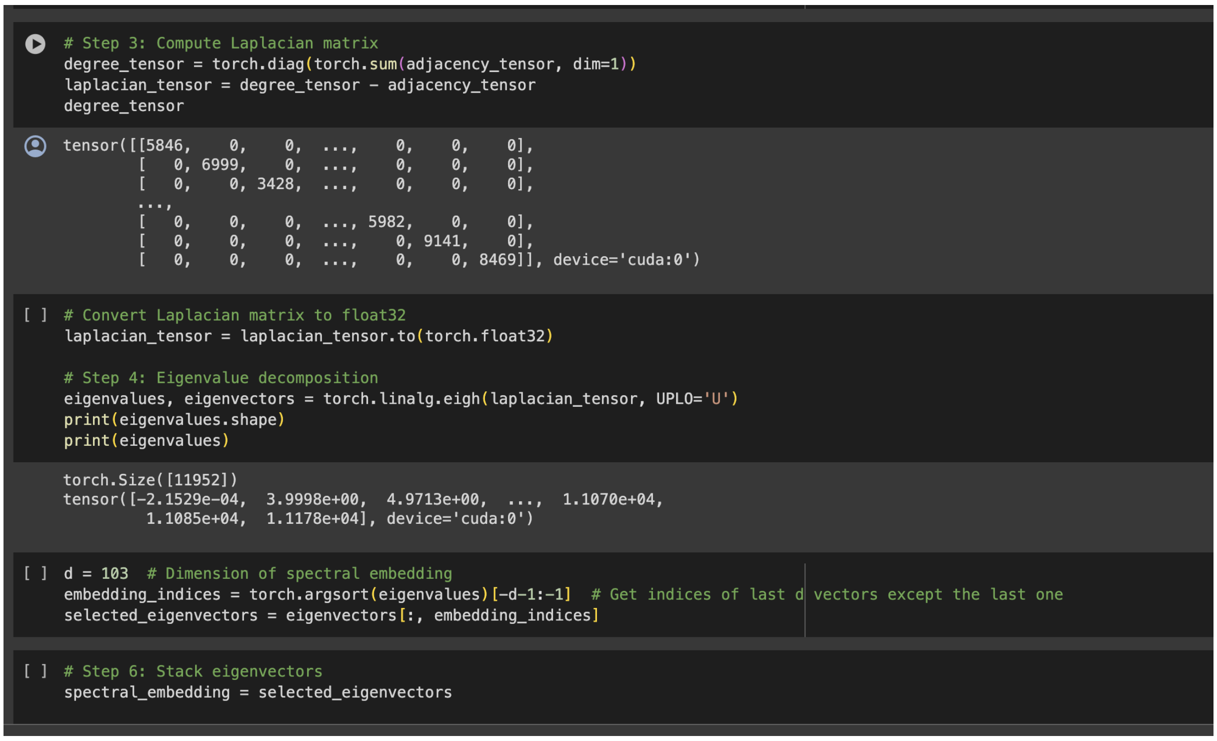
1.1 Tasks

- Cluster the first 10,952 data samples into 10 clusters using non-neural network methods taught in class.
- Train a neural network using the clustered data.
- Predict cluster labels for 1000 samples using the trained neural network.

2 Methodology

This project employs a multi-step approach for feature extraction and clustering of data points followed by classification using a neural network. The methodology consists of the following key steps:

- Spectral Embedding with Normalization: Initially, the adjacency matrix is utilized to compute spectral embeddings for the data points. These embeddings are then normalized to ensure consistency and stability in subsequent analysis.
- PCA Dimensionality Reduction: To handle the high dimensionality of the attribute matrix (100 features), Principal Component Analysis (PCA) is applied. This technique reduces the dimensionality of the attribute matrix to 50 while preserving essential features of the data points.
- Other techniques: Also tried with LLE, MDP, SNE and t-SNE to reduce dimension because there can be fold type structure of data. Also tried to get Node2Vec and DeepWalk embeddings but it was taking a lot of time to get embedding and that's why we just worked with Spectral embeddings.
- Feature Matrix Construction: The normalized embeddings and the PCA-reduced attribute data are concatenated to form a unified feature matrix, facilitating comprehensive representation of the data.
- Clustering with K-Means: K-Means clustering algorithm is employed to partition the dataset into 10 clusters based on the feature matrix. Each data point is assigned to a cluster, resulting in meaningful groupings.



```
# Step 3: Compute Laplacian matrix
degree_tensor = torch.diag(torch.sum(adjacency_tensor, dim=1))
laplacian_tensor = degree_tensor - adjacency_tensor
degree_tensor
```

```
tensor([[5846,    0,    0,  ...,    0,    0,    0],
        [   0, 6999,    0,  ...,    0,    0,    0],
        [   0,    0, 3428,  ...,    0,    0,    0],
        ...,
        [   0,    0,    0,  ..., 5982,    0,    0],
        [   0,    0,    0,  ...,    0, 9141,    0],
        [   0,    0,    0,  ...,    0,    0, 8469]], device='cuda:0')
```

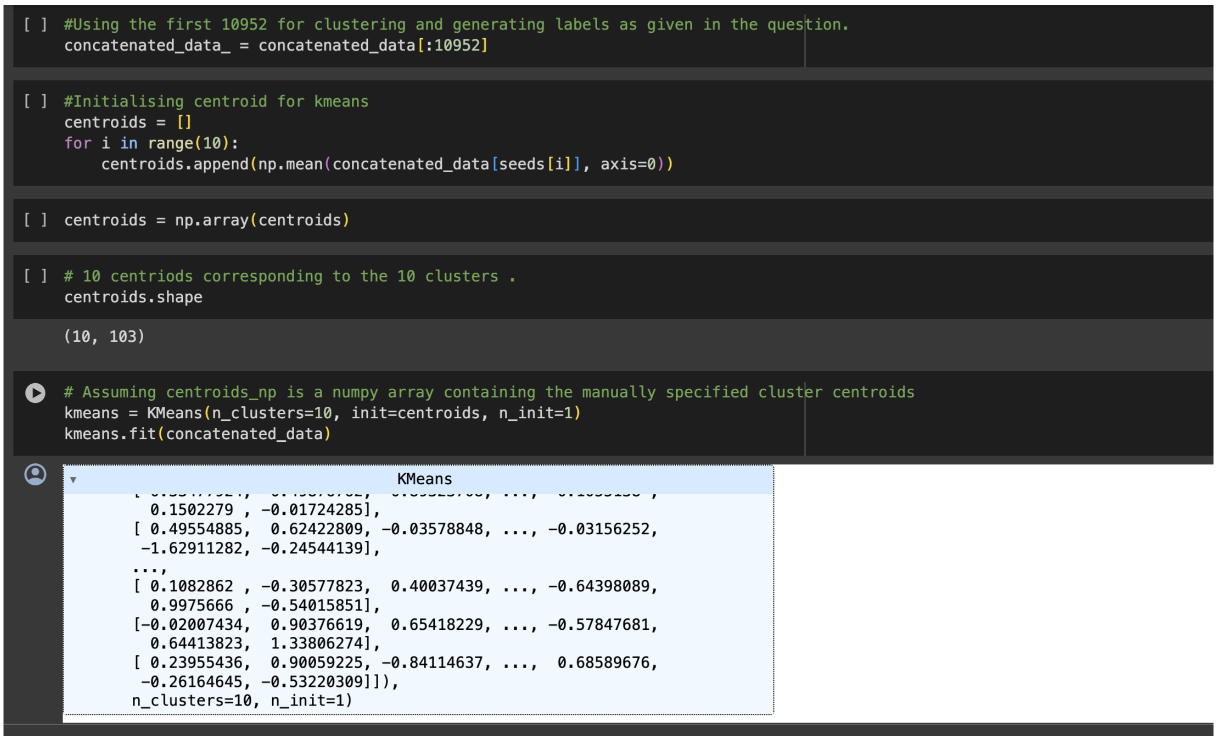
```
[ ] # Convert Laplacian matrix to float32
laplacian_tensor = laplacian_tensor.to(torch.float32)

# Step 4: Eigenvalue decomposition
eigenvalues, eigenvectors = torch.linalg.eigh(laplacian_tensor, UPLO='U')
print(eigenvalues.shape)
print(eigenvalues)
```

```
torch.Size([11952])
tensor([-2.1529e-04,  3.9998e+00,  4.9713e+00,  ...,  1.1070e+04,
         1.1085e+04,  1.1178e+04], device='cuda:0')
```

```
[ ] d = 103  # Dimension of spectral embedding
embedding_indices = torch.argsort(eigenvalues)[-d-1:-1]  # Get indices of last d vectors except the last one
selected_eigenvectors = eigenvectors[:, embedding_indices]
```

```
[ ] # Step 6: Stack eigenvectors
spectral_embedding = selected_eigenvectors
```

Figure 1: Code snippet for Spectral embedding and Feature extraction.



```
[ ] #Using the first 10952 for clustering and generating labels as given in the question.
concatenated_data_ = concatenated_data[:10952]
```

```
[ ] #Initialising centroid for kmeans
centroids = []
for i in range(10):
    centroids.append(np.mean(concatenated_data[seeds[i]], axis=0))
```

```
[ ] centroids = np.array(centroids)
```

```
[ ] # 10 centriods corresponding to the 10 clusters .
centroids.shape
```

```
(10, 103)
```

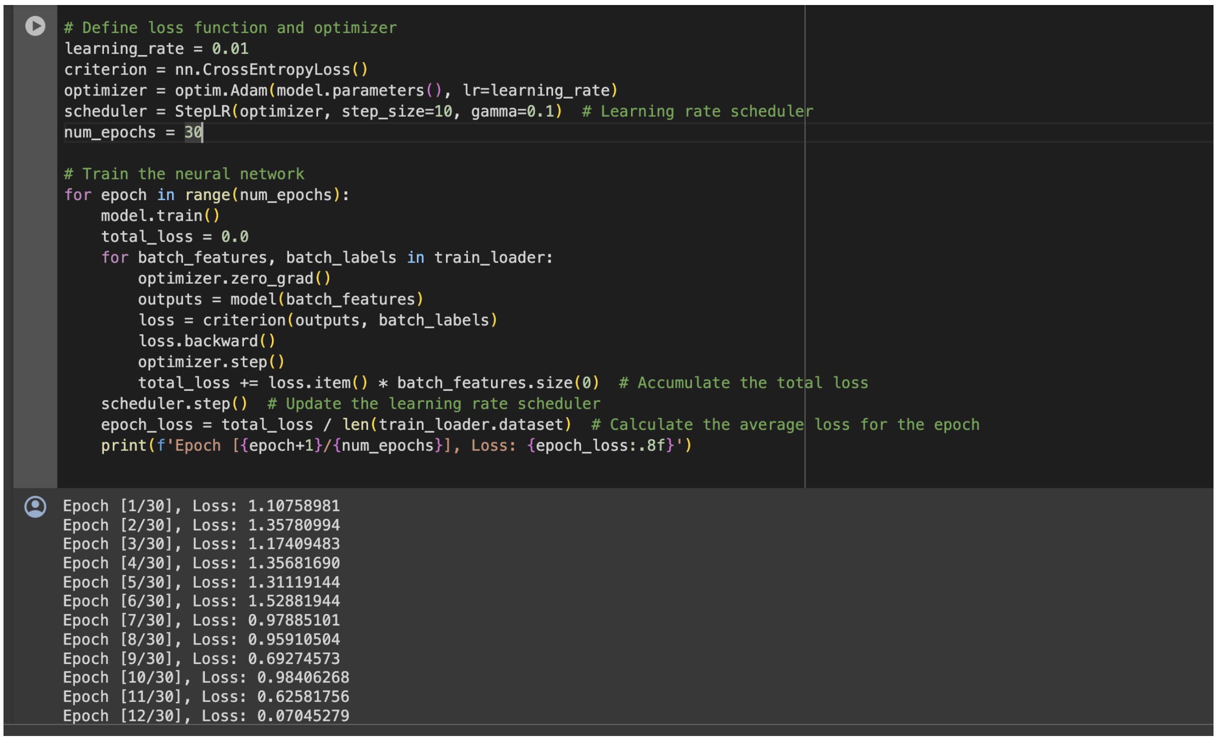
```
# Assuming centroids_np is a numpy array containing the manually specified cluster centroids
kmeans = KMeans(n_clusters=10, init=centroids, n_init=1)
kmeans.fit(concatenated_data)
```

```
KMeans
         0.1502279 , -0.01724285],
        [ 0.49554885,  0.62422809, -0.03578848, ..., -0.03156252,
         -1.62911282, -0.24544139],
        ...,
        [ 0.1082862 , -0.30577823,  0.40037439, ..., -0.64398089,
          0.9975666 , -0.54015851],
        [-0.02007434,  0.90376619,  0.65418229, ..., -0.57847681,
          0.64413823,  1.33806274],
        [ 0.23955436,  0.90059225, -0.84114637, ...,  0.68589676,
         -0.26164645, -0.53220309]]),
        n_clusters=10, n_init=1)
```

Figure 2: Code snippet for kmeans clustering .

- Classification Neural Network: A simple neural network architecture is constructed for classification. The concatenated feature matrix, along with the assigned cluster labels obtained from K-Means, serve as the training features for the neural network. By learning the weights through training, the neural network generates a model capable of predicting the cluster labels for new data points.

- Prediction of Cluster Labels: Utilizing the trained neural network model, the cluster labels for the remaining 1000 data points are predicted, enabling classification of unseen instances.



```python
# Define loss function and optimizer
learning_rate = 0.01
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
scheduler = StepLR(optimizer, step_size=10, gamma=0.1)  # Learning rate scheduler
num_epochs = 30

# Train the neural network
for epoch in range(num_epochs):
    model.train()
    total_loss = 0.0
    for batch_features, batch_labels in train_loader:
        optimizer.zero_grad()
        outputs = model(batch_features)
        loss = criterion(outputs, batch_labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item() * batch_features.size(0)  # Accumulate the total loss
    scheduler.step()  # Update the learning rate scheduler
    epoch_loss = total_loss / len(train_loader.dataset)  # Calculate the average loss for the epoch
    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.8f}')
```

```
Epoch [1/30], Loss: 1.10758981
Epoch [2/30], Loss: 1.35780994
Epoch [3/30], Loss: 1.17409483
Epoch [4/30], Loss: 1.35681690
Epoch [5/30], Loss: 1.31119144
Epoch [6/30], Loss: 1.52881944
Epoch [7/30], Loss: 0.97885101
Epoch [8/30], Loss: 0.95910504
Epoch [9/30], Loss: 0.69274573
Epoch [10/30], Loss: 0.98406268
Epoch [11/30], Loss: 0.62581756
Epoch [12/30], Loss: 0.07045279
```

Figure 3: Code snippet for model training.

3 Model Description

The model We've used is a neural network classifier for classification tasks. It consists of three main components: an input layer, a hidden layer, and an output layer. The input layer receives the input features, and its size is determined by the dimensionality of the input data. Moving to the hidden layer, it comprises multiple neurons, each performing a linear transformation followed by a non-linear activation function, typically ReLU (Rectified Linear Unit). I've set the number of neurons in this hidden layer to 128, for a balance between model complexity and computational efficiency. Finally, the output layer produces the classification predictions, with its size equal to 10 for the number of clusters in the classification problem.

During the training phase, our focus is on minimizing a predefined loss function, typically cross-entropy loss for classification tasks. This loss function quantifies the disparity between the predicted class probabilities and the ground truth labels. To optimize the model's parameters, including weights and biases in each layer, we employ an appropriate optimizer, such as Adam. This optimizer iteratively adjusts the parameters in the direction that minimizes the loss function, thereby enhancing the model's predictive performance.

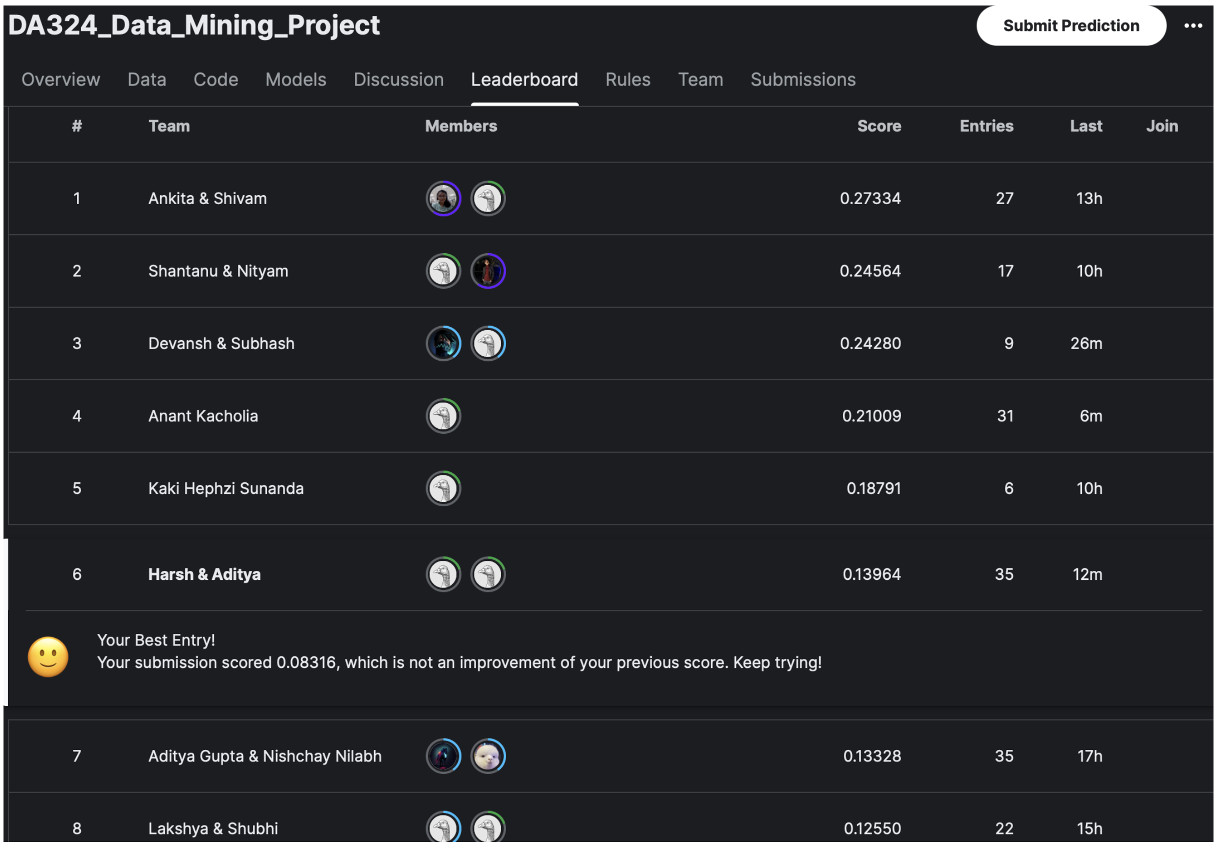
Furthermore, to fine-tune the learning process, we utilize a learning rate scheduler. This scheduler dynamically adjusts the learning rate throughout training, typically decreasing it over time to facilitate convergence. This adjustment helps prevent the model from overshooting the optimal parameter values or getting stuck in local minima.

The training process involves iterating through the entire dataset multiple times (30 epochs). Within each epoch, we divide the dataset into batches(32), and update the model's parameters based on the gradients computed from each batch. This iterative process continues until a predefined number of epochs is reached or until a convergence criterion is met, indicating that further training is unlikely to significantly improve the model's performance.

4 Failed Attempts

Throughout our project, we encountered several challenges that prompted us to explore various approaches, some of which did not yield the desired outcomes. One significant hurdle was the computational infeasibility of extracting a large number of features, which resulted in excessive processing times. Conversely, limiting the number of features led to decreased accuracy in clustering, highlighting the need for a delicate balance between computational efficiency and clustering performance. Additionally, we experimented with dimensionality reduction techniques, considering t-SNE as an alternative to PCA. However, due to the dense graph structure of the data, t-SNE proved ineffective in extracting meaningful local features. Consequently, we reverted to PCA to retain the global information of the data, crucial for clustering 10592 points into only 10 clusters.

We also found that while normalized spectral cut was efficient, attempts with ratio cut did not yield satisfactory clustering results. Furthermore, exploring alternative clustering algorithms like DBSCAN proved unsuccessful due to the high-dimensional nature of the data and its high density, resulting in an unclear cluster structure. Another significant challenge arose during neural network training, where instances of overfitting were encountered. This was attributed to the poor quality of clustering in the initial stage, leading to the model's poor generalization to unseen data.


DA324_Data_Mining_Project
Submit Prediction
Overview Data Code Models Discussion Leaderboard Rules Team Submissions

#	Team	Members	Score	Entries	Last	Join
1	Ankita & Shivam		0.27334	27	13h	
2	Shantanu & Nityam		0.24564	17	10h	
3	Devansh & Subhash		0.24280	9	26m	
4	Anant Kacholia		0.21009	31	6m	
5	Kaki Hephzi Sunanda		0.18791	6	10h	
6	**Harsh & Aditya**		0.13964	35	12m	
7	Aditya Gupta & Nishchay Nilabh		0.13328	35	17h	
8	Lakshya & Shubhi		0.12550	22	15h	

Your Best Entry!
Your submission scored 0.08316, which is not an improvement of your previous score. Keep trying!

Figure 4: Kaggle leaderboard status.

5 Source code

You can find the source code for tried attempts on: Google Colab.

6 Analysis

Follwing is the analysis of no. of nodes in a Class.

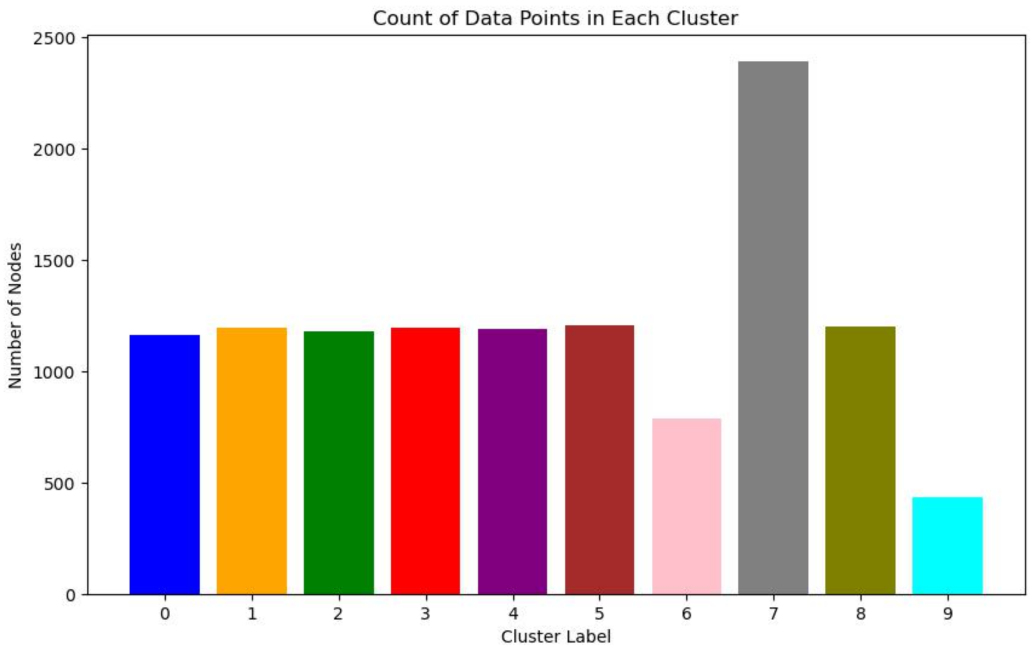


Figure 5: Analysis of number of nodes in a cluster.