Pattern Recognition
Lab 2

Ziad Reda Saad - 19015717 **Ali Mones Abd El-Mohsen - 20010946**
Marwan Mostafa Abd El-Kader - 20011867

March 28, 2024

[1]:
```
import torch
from torch import Tensor
from sklearn.decomposition import PCA
import os
from sklearn.cluster import KMeans
```

1 Generate the Data Matrix and the Label vector

Read data into torch.Tensor

[2]:
```
torch.set_default_dtype(torch.float64)

# Specify the top-level folder
top_folder = "data"

# Initialize an empty list to store flattened arrays
flattened_arrays = []
labels = torch.zeros(9120)
example_cnt, example_label = 0, 0

for root, dirs, files in os.walk(top_folder):
    for file in files:
        if file.endswith(".txt"):
            file_path = os.path.join(root, file)
            lines = []
            with open(file_path, "r") as file:
                for line in file:
                    values = line.strip().split(",")
                    lines.append([float(value) for value in values])

            flattened_array = torch.tensor(lines).view(-1)
            labels[example_cnt] = example_label

            flattened_arrays.append(flattened_array)
            example_cnt += 1
```

```
            if example_cnt % 480 == 0:
                example_label += 1

all_data = torch.stack(flattened_arrays)

# all_data_1 is a 2D tensor of shape (9120, 45) containing the mean of each␣
↪column in each segment resulting in 45 features for each data point
all_data_1 = torch.zeros((all_data.shape[0], 45))

for i in range(all_data_1.shape[1]):
    all_data_1[:, i] = all_data[:, i * 125 : i * 125 + 125].mean(1)

# all_data_2 is a 2D tensor of shape (9120, 5625) containing 45 x 125 features␣
↪for each data point
all_data_2 = all_data
```

2 Split the Dataset into Training and Test sets

Split dataset into training and test data

[3]:
```
training_indices = [i for i in range(len(all_data)) if (i % 60) < 48]
test_indices = [i for i in range(len(all_data)) if (i % 60) >= 48]

training_data_1 = all_data_1[training_indices]
test_data_1 = all_data_1[test_indices]

training_data_2 = all_data_2[training_indices]
test_data_2 = all_data_2[test_indices]

training_labels = labels[training_indices]
test_labels = labels[test_indices]
```

2.0.1 Reduced data using PCA

[4]:
```
pca = PCA(n_components=45)
pca.fit(training_data_2)
training_data_2_reduced = Tensor(pca.transform(training_data_2))

pca.fit(test_data_2)
test_data_2_reduced = Tensor(pca.transform(test_data_2))
```

3 K-Means Algorithm

[5]:
```
def k_means(points: Tensor, k: int, relative_error: float = 1e-6, max_iterations:
↪ int = 1000):
    n = len(points)
    means = points[torch.randperm(n)[:k]]

    error = 1
    iterations = 0
    while error > relative_error and iterations < max_iterations:
        iterations += 1
        distances = torch.cdist(points, means)
        closest_means = torch.argmin(distances, dim=1)

        new_means = torch.zeros_like(means)
        for i in range(k):
            new_means[i] = points[closest_means == i].mean(dim=0)

        error = torch.norm(means - new_means) / torch.norm(new_means)

        means = new_means

    return means, closest_means
```

3.1 K-Ways normalised Cut Algorithm

[6]:
```
def KNN_similarity_graph(data,k):
    n = data.shape[0]
    sim_graph = torch.zeros((n,n))
    distances = torch.cdist(data, data)
    distances.view(-1)[::distances.size(0) + 1] = float('inf')
    _, indices = torch.topk(distances, k, largest=False)
    row_indices = torch.arange(n).unsqueeze(1).expand(n,k)
    sim_graph[row_indices,indices] = 1

    return sim_graph
```

[7]:
```
def rbf_graph(data: Tensor, gamma: float):

    return torch.exp(-gamma*torch.cdist(data,data)**2)
```

[18]:
```
def k_ways_normalised_cut(a: Tensor, k: int):
    delta,inverse_delta = a.sum(dim=1).diag(),None
    if torch.allclose(a, a.T): inverse_delta = torch.diag(1 / delta.diag())
    else: inverse_delta = torch.inverse(delta)

    # Replace inf and nan values with 0
```

```
    inverse_delta.masked_fill_(torch.isnan(inverse_delta) | torch.
↪isinf(inverse_delta), 0)

    l_a = inverse_delta @ (delta - a)

    eigen_values, eigen_vectors = torch.linalg.eig(l_a)
    eigen_values = eigen_values.real
    eigen_vectors = eigen_vectors.real

    indices = torch.argsort(eigen_values)
    eigen_values = eigen_values[indices]
    eigen_vectors = eigen_vectors[:, indices]

    u = eigen_vectors[:, :k]
    y = u / torch.norm(u, dim=1, keepdim=True)
    y.masked_fill_(torch.isnan(y) | torch.isinf(y), 0)

    # Create KMeans object
    kmeans = KMeans(n_clusters=k)

    # Fit the model to the data
    kmeans.fit(y)

    # Predict the cluster labels
    centroids = Tensor(kmeans.cluster_centers_)
    predicted_labels = Tensor(kmeans.labels_)
    # centroids, predicted_labels = k_means(y, k)

    return centroids,predicted_labels
```

4 Evaluation fucntions

Precision

[31]:
```
def precision(clustering: Tensor, labels: Tensor):
    cluster_labels = torch.unique(clustering)

    total_precision = 0.0

    for cluster_label in cluster_labels:
        cluster_indices = (clustering == cluster_label).nonzero()

        actual_cluster_labels = labels[cluster_indices]
        mode = actual_cluster_labels.mode(dim=0)[0]
        total_precision += len(actual_cluster_labels[actual_cluster_labels ==␣
↪mode])
```

```python
    return total_precision / len(clustering)
```

recall

[30]:
```python
def contingency_matrix(actual_labels: Tensor, predicted_labels: Tensor):
    n_samples = len(actual_labels)

    n_actual = int(actual_labels.max().item()) + 1
    n_predicted = int(predicted_labels.max().item()) + 1
    matrix = torch.zeros(n_actual, n_predicted)

    for i in range(n_samples):
        matrix[int(actual_labels[i]), int(predicted_labels[i])] += 1

    return matrix


def confusion_matrix(contingency_matrix: Tensor, n: int):
    tp = 0.5 * (torch.sum(contingency_matrix ** 2, dim=(0, 1)) - n)

    column_sum = contingency_matrix.sum(0)
    fp = torch.sum(column_sum * (column_sum - 1) / 2) - tp

    row_sum = contingency_matrix.sum(1)
    fn = torch.sum(row_sum * (row_sum - 1) / 2) - tp

    tn = n * (n - 1) / 2 - tp - fp - fn

    return tp, fp, tn, fn


def recall(predicted_labels: Tensor, actual_labels: Tensor):
    contingency = contingency_matrix(actual_labels, predicted_labels)
    tp, _, _, fn = confusion_matrix(contingency, len(actual_labels))

    return tp / (tp + fn)
```

F1 score

[32]:
```python
def f1_score(clustering: Tensor, labels: Tensor):
    cluster_labels = torch.unique(clustering)

    total_f = 0.0

    for cluster_label in cluster_labels:
        cluster_indices = (clustering == cluster_label).nonzero()

        actual_cluster_labels = labels[cluster_indices]
        mode = actual_cluster_labels.mode(dim=0)[0]
```

```
        precision = len(actual_cluster_labels[actual_cluster_labels == mode]) /␣
↪len(cluster_indices)
        recall = len(actual_cluster_labels[actual_cluster_labels == mode]) /␣
↪len(labels[labels == mode])
        total_f += 2 * precision * recall / (precision + recall)

    return total_f / len(cluster_labels)
```

Conditional entropy

[11]:
```
def conditional_entropy(clustering: Tensor, labels: Tensor):
    cluster_labels = torch.unique(clustering)
    partition_labels = torch.unique(labels)

    total_entropy = 0.0
    for cluster_label in cluster_labels:
        cluster_entropy = 0.0
        cluster_indices = (clustering == cluster_label).nonzero()

        for partition_label in partition_labels:
            partition_indices = (labels == partition_label).nonzero()
            cluster_in_partition_count = (clustering[partition_indices] ==␣
↪cluster_label).sum()
            cluster_entropy -= cluster_in_partition_count / len(cluster_indices)␣
↪* torch.log2(torch.Tensor([cluster_in_partition_count /␣
↪len(cluster_indices)])) if cluster_in_partition_count > 0 else 0

        total_entropy += len(cluster_indices) / len(labels) * cluster_entropy

    return total_entropy
```

5 Clustering Using K-Means and Normalized Cut

5.0.1 Solution 1: Taking the mean of each column in each segment for each data point

Using kmeans

[113]:
```
ks = [8, 13, 19, 28,38]
for k in ks:
    centroids,training_predicted_labels = k_means(training_data_1,k)
    test_predicted_labels = torch.empty_like(test_labels, dtype=torch.long)
    for i,point in enumerate(test_data_1):
        distances = torch.norm(point - centroids, dim=1)
        test_predicted_labels[i] = torch.argmin(distances)

    prec_train     = precision(training_predicted_labels,training_labels)
    rec_train      = recall(training_predicted_labels,training_labels)
```

```
    f_score_train  = f1_score(training_predicted_labels,training_labels)
    entropy_train  =␣
↪conditional_entropy(training_predicted_labels,training_labels)

    prec_test     = precision(test_predicted_labels,test_labels)
    rec_test      = recall(test_predicted_labels,test_labels)
    f_score_test = f1_score(test_predicted_labels,test_labels)
    entropy_test  = conditional_entropy(test_predicted_labels,test_labels)

    print(f'------ For k = {k} ------')
    print("training:")
    print(f'Precision for training set = {prec_train}')
    print(f'Recall for training set = {rec_train}')
    print(f'Fscore for training set = {f_score_train}')
    print(f'Entropy for training set= {entropy_train.item()}')
    print("test:")
    print(f'Precision for test set = {prec_test}')
    print(f'Recall for test set = {rec_test}')
    print(f'Fscore for test set = {f_score_test}')
    print(f'Entropy for test set = {entropy_test.item()}')
```

```
------ For k = 8 ------
training:
Precision for training set = 0.22930372807017543
Recall for training set = 0.4881848060098026
Fscore for training set = 0.1934170586164415
Entropy for training set= 3.150761364192166
test:
Precision for test set = 0.22861842105263158
Recall for test set = 0.4821445060018467
Fscore for test set = 0.1892518812832141
Entropy for test set = 3.1473447107732375
------ For k = 13 ------
training:
Precision for training set = 0.2735745614035088
Recall for training set = 0.45972470340341715
Fscore for training set = 0.1929711768982569
Entropy for training set= 2.9612778801742037
test:
Precision for test set = 0.2631578947368421
Recall for test set = 0.45090027700831026
Fscore for test set = 0.18049885992695217
Entropy for test set = 2.994771398422983
------ For k = 19 ------
training:
Precision for training set = 0.2974232456140351
Recall for training set = 0.33198061243186294
Fscore for training set = 0.18951288596348762
```

```
Entropy for training set= 2.8459469469925933
test:
Precision for test set = 0.27960526315789475
Recall for test set = 0.3311634349030471
Fscore for test set = 0.18461182102459173
Entropy for test set = 2.8373268477357594
------ For k = 28 ------
training:
Precision for training set = 0.31661184210526316
Recall for training set = 0.2578250251935321
Fscore for training set = 0.1856180759519761
Entropy for training set= 2.6638988491583566
test:
Precision for test set = 0.31469298245614036
Recall for test set = 0.26188827331486614
Fscore for test set = 0.18738902423917084
Entropy for test set = 2.5900846965996314
------ For k = 38 ------
training:
Precision for training set = 0.39418859649122806
Recall for training set = 0.2284566671247309
Fscore for training set = 0.20093379991854274
Entropy for training set= 2.3857082459831958
test:
Precision for test set = 0.3843201754385965
Recall for test set = 0.2315904893813481
Fscore for test set = 0.20035347685425572
Entropy for test set = 2.3665942213263893
```

Using Normalized Cut

[23]:
```
alpha,k = 0.1,19

#sim_graph = rbf_graph(test_data_1, alpha)
sim_graph = KNN_similarity_graph(test_data_1,100)
centroids,test_predicted_labels = k_ways_normalised_cut(sim_graph,k)

prec    = precision(test_predicted_labels,test_labels)
rec     = recall(test_predicted_labels,test_labels)
f_score = f1_score(test_predicted_labels,test_labels)
entropy = conditional_entropy(test_predicted_labels,test_labels)

print(f'Precision for k:{k} = {prec}')
print(f'Recall for k:{k} = {rec}')
print(f'Fscore for k:{k} = {f_score}')
print(f'Entropy for k:{k} = {entropy.item()}')
```

```
Precision for k:19 = 0.3344298245614035
Recall for k:19 = 0.2443674976915974
```

```
Fscore for k:19 = 0.3328502748492976
Entropy for k:19 = 2.5005078562956684
```

5.0.2 Solution 2: Flattening all the features together for each data point

Using kmeans

[114]:
```python
ks = [8, 13, 19, 28,38]
centroids,training_predicted_labels = None,None
for k in ks:
    centroids,training_predicted_labels = k_means(training_data_2_reduced,k)

    test_predicted_labels = torch.empty_like(test_labels, dtype=torch.long)
    for i,point in enumerate(test_data_2_reduced):
        distances = torch.norm(point - centroids, dim=1)
        test_predicted_labels[i] = torch.argmin(distances)

    prec_train     = precision(training_predicted_labels,training_labels)
    rec_train      = recall(training_predicted_labels,training_labels)
    f_score_train  = f1_score(training_predicted_labels,training_labels)
    entropy_train  =␣
 ↪conditional_entropy(training_predicted_labels,training_labels)

    prec_test    = precision(test_predicted_labels,test_labels)
    rec_test     = recall(test_predicted_labels,test_labels)
    f_score_test = f1_score(test_predicted_labels,test_labels)
    entropy_test  = conditional_entropy(test_predicted_labels,test_labels)

    print(f'------ For k = {k} ------')
    print("training:")
    print(f'Precision for training set = {prec_train}')
    print(f'Recall for training set = {rec_train}')
    print(f'Fscore for training set = {f_score_train}')
    print(f'Entropy for training set= {entropy_train.item()}')
    print("test:")
    print(f'Precision for test set = {prec_test}')
    print(f'Recall for test set = {rec_test}')
    print(f'Fscore for test set = {f_score_test}')
    print(f'Entropy for test set = {entropy_test.item()}')
```

```
------ For k = 8 ------
training:
Precision for training set = 0.32140899122807015
Recall for training set = 0.7234308437542943
Fscore for training set = 0.24722184039224185
Entropy for training set= 2.5221248885709384
test:
Precision for test set = 0.27028508771929827
Recall for test set = 0.685387811634349
```

```
Fscore for test set = 0.22514587733034058
Entropy for test set = 2.6703336474780945
------ For k = 13 ------
training:
Precision for training set = 0.38061951754385964
Recall for training set = 0.5910767658375704
Fscore for training set = 0.2683288237831322
Entropy for training set= 2.290392241607976
test:
Precision for test set = 0.33497807017543857
Recall for test set = 0.5798822714681441
Fscore for test set = 0.23940930361108204
Entropy for test set = 2.4266484048341783
------ For k = 19 ------
training:
Precision for training set = 0.49917763157894735
Recall for training set = 0.4785633102010902
Fscore for training set = 0.3710778996591631
Entropy for training set= 1.8843416426164439
test:
Precision for test set = 0.4479166666666667
Recall for test set = 0.4583333333333333
Fscore for test set = 0.3272703297427825
Entropy for test set = 2.091418050904501
------ For k = 28 ------
training:
Precision for training set = 0.5209703947368421
Recall for training set = 0.4046625211854702
Fscore for training set = 0.3608960164634376
Entropy for training set= 1.7475111806102102
test:
Precision for test set = 0.45997807017543857
Recall for test set = 0.4007733148661127
Fscore for test set = 0.3116644152533625
Entropy for test set = 2.0018586138195995
------ For k = 38 ------
training:
Precision for training set = 0.5705866228070176
Recall for training set = 0.3553776739498878
Fscore for training set = 0.37633433885612944
Entropy for training set= 1.5566305500433548
test:
Precision for test set = 0.5054824561403509
Recall for test set = 0.34344413665743306
Fscore for test set = 0.3190341914248035
Entropy for test set = 1.7697984020572328
```

Using Normalized Cut

[115]:
```
alpha,k = 0.001,19

#sim_graph = rbf_graph(test_data_2_reduced, alpha)
sim_graph = KNN_similarity_graph(test_data_2_reduced,100)
centroids,test_predicted_labels = k_ways_normalised_cut(sim_graph,k)

prec    = precision(test_predicted_labels,test_labels)
rec     = recall(test_predicted_labels,test_labels)
f_score = f1_score(test_predicted_labels,test_labels)
entropy = conditional_entropy(test_predicted_labels,test_labels)

print(f'Precision for k:{k} = {prec}')
print(f'Recall for k:{k} = {rec}')
print(f'Fscore for k:{k} = {f_score}')
print(f'Entropy for k:{k} = {entropy.item()}')
```

```
Precision for k:19 = 0.5060307017543859
Recall for k:19 = 0.4137119113573407
Fscore for k:19 = 0.36683126519124987
Entropy for k:19 = 1.844753891319193
```

6 Hierarchical clustering

[25]:
```
def hierarchical_clustring(data,k):
    num_of_clusters = len(data)

    labels = torch.arange(num_of_clusters)
    distances = torch.cdist(data,data)
    distances.fill_diagonal_(float('inf'))

    while(num_of_clusters > k):

        #get min distance between clusters
        indices = torch.argmin(distances)
        row_index, col_index = torch.unravel_index(indices, distances.shape)

        # get the elements of the two clusters obtained from step above
        cluster1 = torch.nonzero(labels == labels[row_index]).flatten()
        cluster2 = torch.nonzero(labels == labels[col_index]).flatten()

        # Apply the mask to replace the labels to make them one cluster
        labels[cluster1] = labels[col_index].item()

        grouped_cluster = torch.flatten(torch.cat((cluster1, cluster2)))

        # Broadcast the row indices to match the shape of the column indices
```

```
        broadcasted_rows_indices = grouped_cluster.unsqueeze(1).expand(-1,␣
↪grouped_cluster.size(0))

        # Set values using broadcasted indices
        distances[broadcasted_rows_indices, grouped_cluster] = float('inf')
        num_of_clusters-=1

    return labels
```

[33]:
```
k = 19
test_predicted_labels = hierarchical_clustring(test_data_1,k)

prec    = precision(test_predicted_labels,test_labels)
rec     = recall(test_predicted_labels,test_labels)
f_score = f1_score(test_predicted_labels,test_labels)
entropy = conditional_entropy(test_predicted_labels,test_labels)

print(f'Precision for k:{k} = {prec}')
print(f'Recall for k:{k} = {rec}')
print(f'Fscore for k:{k} = {f_score}')
print(f'Entropy for k:{k} = {entropy.item()}')
```

```
Precision for k:19 = 0.06304824561403509
Recall for k:19 = 0.9802746999076639
Fscore for k:19 = 0.025912171811011103
Entropy for k:19 = 4.2028529319221
```

[116]:
```
k = 19
test_predicted_labels = hierarchical_clustring(test_data_2_reduced,k)

prec    = precision(test_predicted_labels,test_labels)
rec     = recall(test_predicted_labels,test_labels)
f_score = f1_score(test_predicted_labels,test_labels)
entropy = conditional_entropy(test_predicted_labels,test_labels)

print(f'Precision for k:{k} = {prec}')
print(f'Recall for k:{k} = {rec}')
print(f'Fscore for k:{k} = {f_score}')
print(f'Entropy for k:{k} = {entropy.item()}')
```

```
Precision for k:19 = 0.12828947368421054
Recall for k:19 = 0.959960757156048
Fscore for k:19 = 0.11003390808171278
Entropy for k:19 = 3.850657794069117
```