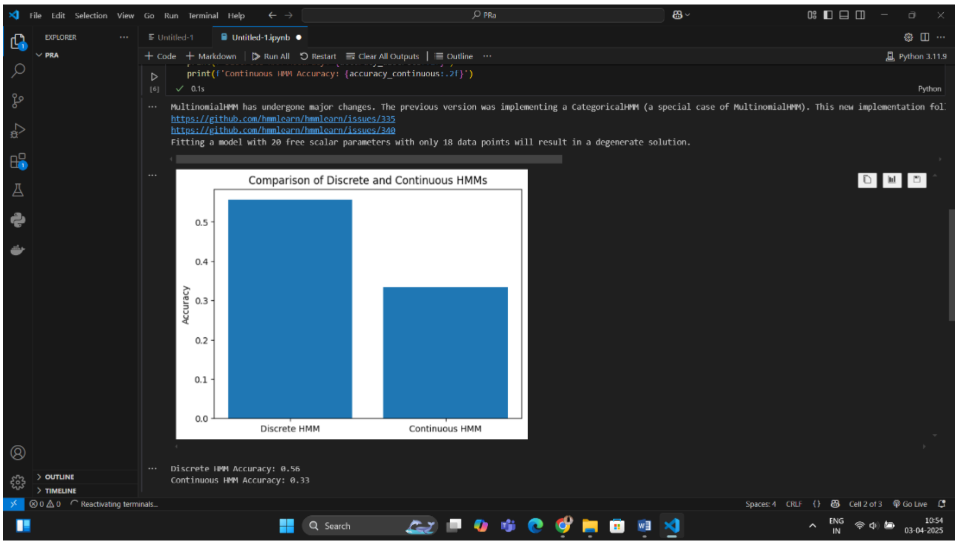
Expriment No.7

Code:-

```
import numpy as np

import matplotlib.pyplot as plt

from hmmlearn import hmm

from sklearn.preprocessing import KBinsDiscretizer, LabelEncoder

from sklearn.metrics import accuracy_score

# Simulated dataset: Temperature (Celsius), Humidity (%), and Actual Weather Labels

data = np.array([

    [30, 40, 'sunny'], [32, 42, 'sunny'], [35, 50, 'sunny'],

    [28, 60, 'cloudy'], [26, 65, 'cloudy'], [25, 70, 'cloudy'],

    [22, 80, 'rainy'], [20, 85, 'rainy'], [18, 90, 'rainy']

])

# Extract temperature, humidity, and labels

temp_humidity = data[:, :2].astype(float)

labels = data[:, 2]

# Encoding states

weather_encoder = LabelEncoder()

states = weather_encoder.fit_transform(labels)  # sunny=2, cloudy=1, rainy=0

n_states = len(set(states))

# Discrete HMM: Discretizing temperature and humidity into 3 bins

discretizer = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')

discrete_obs = discretizer.fit_transform(temp_humidity).astype(int)

# Train Discrete HMM
```

```
discrete_hmm = hmm.MultinomialHMM(n_components=n_states, n_iter=100)

discrete_hmm.fit(discrete_obs)

predicted_states_discrete = discrete_hmm.predict(discrete_obs)

accuracy_discrete = accuracy_score(states, predicted_states_discrete)

# Continuous HMM: Train a Gaussian HMM

continuous_hmm = hmm.GaussianHMM(n_components=n_states,
covariance_type='diag', n_iter=100)

continuous_hmm.fit(temp_humidity)

predicted_states_continuous = continuous_hmm.predict(temp_humidity)

accuracy_continuous = accuracy_score(states, predicted_states_continuous)

# Plot Accuracy Comparison

plt.bar(['Discrete HMM', 'Continuous HMM'], [accuracy_discrete,
accuracy_continuous])

plt.ylabel('Accuracy')

plt.title('Comparison of Discrete and Continuous HMMs')

plt.show()

# Print Results

print(f'Discrete HMM Accuracy: {accuracy_discrete:.2f}')

print(f'Continuous HMM Accuracy: {accuracy_continuous:.2f}')
```

Output:-


File
Edit
Selection
View
Go
Run
Terminal
Help
PRa
EXPLORER
PRA
Untitled-1
Untitled-1.ipynb
Code
Markdown
Run All
Restart
Clear All Outputs
Outline
Python 3.11.9
print(f'Continuous HMM Accuracy: {accuracy_continuous:.2f}')
[6]
0.1s
Python
MultinomialHMM has undergone major changes. The previous version was implementing a CategoricalHMM (a special case of MultinomialHMM). This new implementation fol
https://github.com/hmmlearn/hmmlearn/issues/335
https://github.com/hmmlearn/hmmlearn/issues/340
Fitting a model with 20 free scalar parameters with only 18 data points will result in a degenerate solution.
Comparison of Discrete and Continuous HMMs
Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
Discrete HMM
Continuous HMM
Discrete HMM Accuracy: 0.56
Continuous HMM Accuracy: 0.33
OUTLINE
TIMELINE
Reactivating terminals...
Spaces: 4
CRLF
Cell 2 of 3
Go Live
Search
ENG
IN
10:54
03-04-2025

Expriment No.8

Code:-

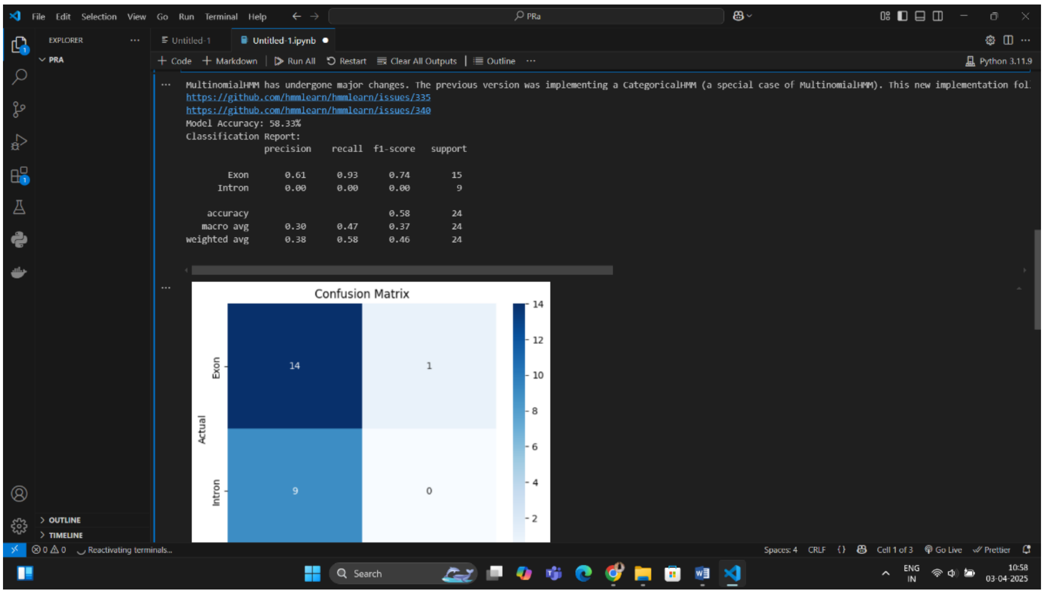
```
import numpy as np

import matplotlib.pyplot as plt

from hmmlearn import hmm

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

import seaborn as sns

# Define DNA sequence encoding: A=0, C=1, G=2, T=3

dna_mapping = {'A': 0, 'C': 1, 'G': 2, 'T': 3}

state_mapping = {'Exon': 0, 'Intron': 1}

reverse_state_mapping = {0: 'Exon', 1: 'Intron'}

# Sample training dataset (Observed DNA sequences and their corresponding
states)

sequences = ['ATGCGT', 'CGTTAG', 'GGATCC', 'TACGTA']

states = [['Exon', 'Exon', 'Intron', 'Intron', 'Exon', 'Exon'],

      ['Intron', 'Intron', 'Exon', 'Exon', 'Intron', 'Exon'],

      ['Exon', 'Exon', 'Exon', 'Intron', 'Intron', 'Exon'],

      ['Intron', 'Intron', 'Exon', 'Exon', 'Exon', 'Exon']]

# Convert sequences and states to numerical format

encoded_sequences = [np.array([dna_mapping[n] for n in seq]) for seq in
sequences]

encoded_states = [np.array([state_mapping[s] for s in state]) for state in
states]

# Flatten the sequences for training

X = np.concatenate(encoded_sequences).reshape(-1, 1)

lengths = [len(seq) for seq in encoded_sequences]
```

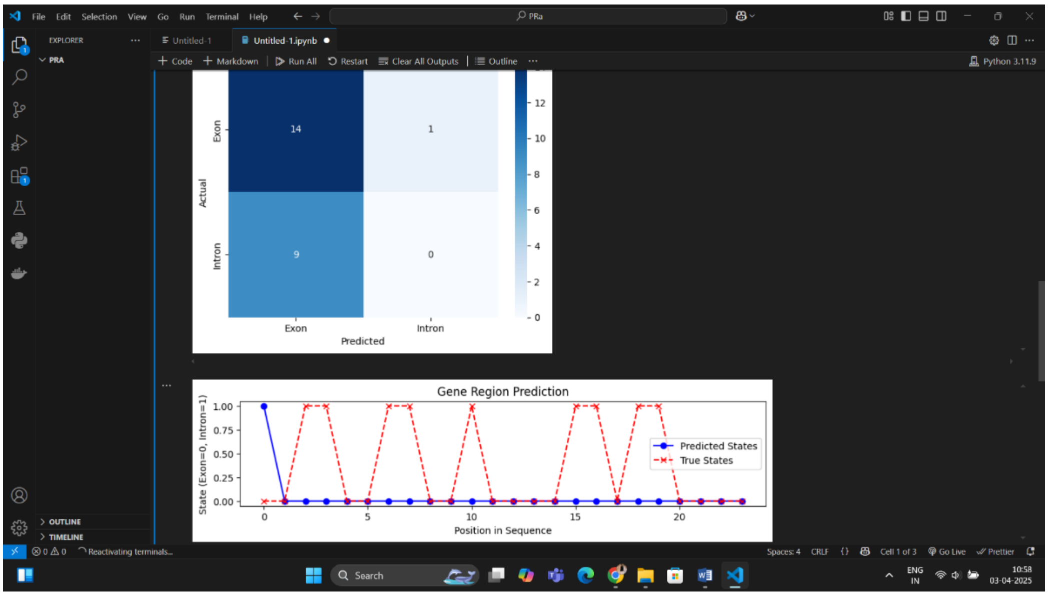
```
y_true = np.concatenate(encoded_states)

# Train Discrete HMM

model = hmm.MultinomialHMM(n_components=2, n_iter=100, tol=1e-4,
random_state=42)

model.fit(X, lengths)

# Predict hidden states using Viterbi Algorithm

logprob, y_pred = model.decode(X, algorithm="viterbi")

# Evaluate model accuracy

accuracy = accuracy_score(y_true, y_pred)

print(f"Model Accuracy: {accuracy:.2%}")

print("Classification Report:\n", classification_report(y_true, y_pred,
target_names=['Exon', 'Intron']))

# Confusion Matrix

cm = confusion_matrix(y_true, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Exon',
'Intron'], yticklabels=['Exon', 'Intron'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

# Visualization of predicted gene regions

plt.figure(figsize=(10, 2))

plt.plot(y_pred, label='Predicted States', marker='o', linestyle='-', color='b')

plt.plot(y_true, label='True States', marker='x', linestyle='--', color='r')

plt.xlabel('Position in Sequence')

plt.ylabel('State (Exon=0, Intron=1)')

plt.legend()
```

plt.title('Gene Region Prediction')

plt.show()

Output:-





Expriment No.10

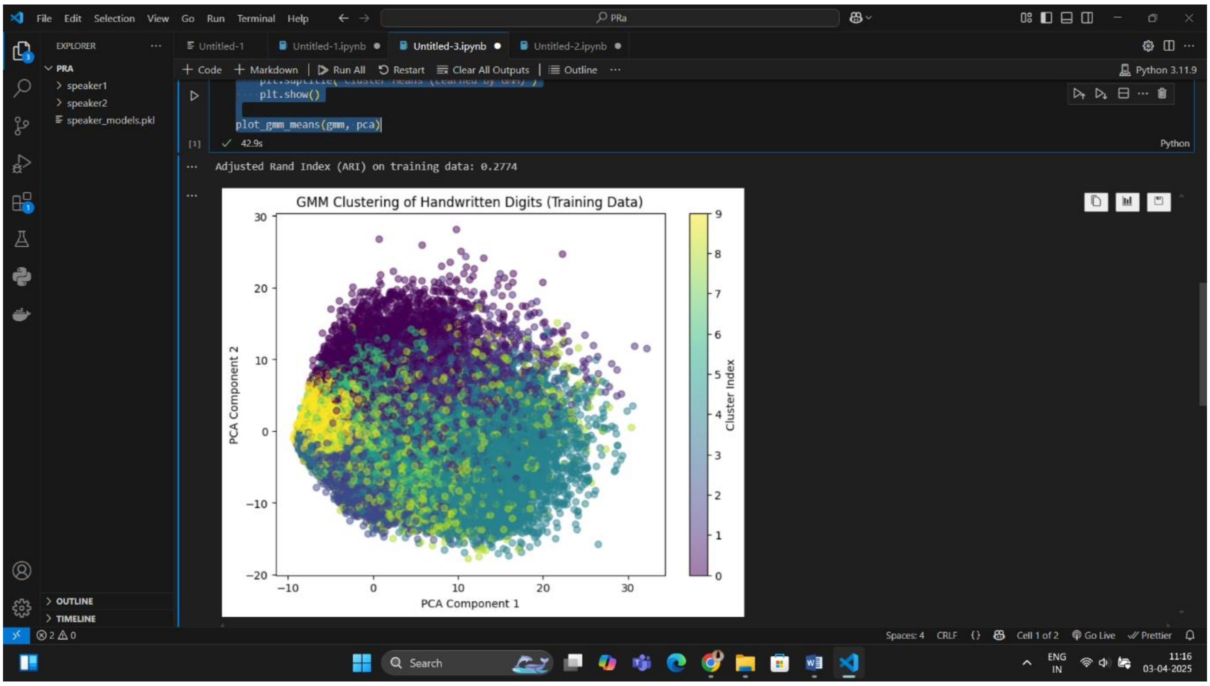
Code:-

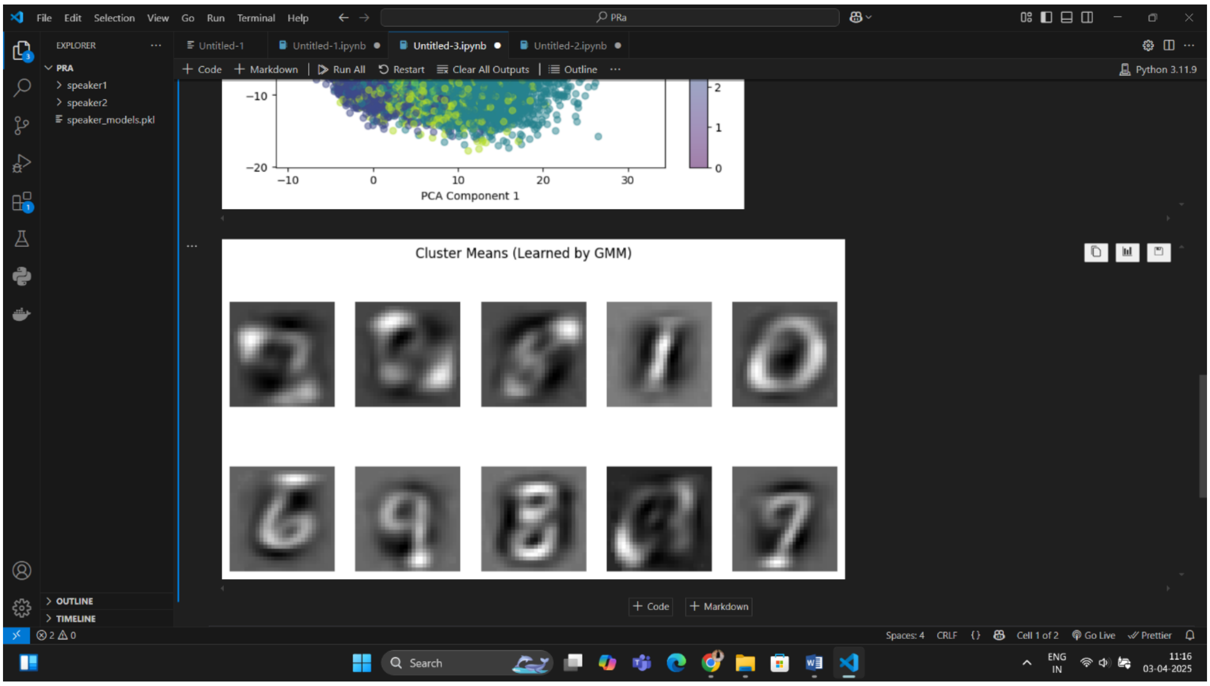
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.decomposition import PCA
from sklearn.metrics import adjusted_rand_score
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.datasets import mnist
# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# Flatten images to 784-dimensional vectors
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)
# Normalize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Reduce dimensionality for better clustering
pca = PCA(n_components=50)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Define and fit Gaussian Mixture Model (GMM)
n_components = 10  # Number of clusters (digits 0-9)
```

```
gmm = GaussianMixture(n_components=n_components,
covariance_type='full', random_state=42)

gmm.fit(X_train_pca)

# Predict cluster labels

train_clusters = gmm.predict(X_train_pca)

test_clusters = gmm.predict(X_test_pca)

# Evaluate clustering performance using Adjusted Rand Index (ARI)

ari_score = adjusted_rand_score(y_train, train_clusters)

print(f"Adjusted Rand Index (ARI) on training data: {ari_score:.4f}")

# Visualize the clusters using PCA (first two components)

def plot_clusters(data, labels, title):

    plt.figure(figsize=(8, 6))

    scatter = plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis', alpha=0.5)

    plt.colorbar(scatter, label='Cluster Index')

    plt.title(title)

    plt.xlabel('PCA Component 1')

    plt.ylabel('PCA Component 2')

    plt.show()

plot_clusters(X_train_pca, train_clusters, 'GMM Clustering of Handwritten
Digits (Training Data)')


# Visualize cluster means (reshaped to 28x28 images)

def plot_gmm_means(gmm, pca):

    mean_images =
pca.inverse_transform(gmm.means_).reshape(n_components, 28, 28)

    plt.figure(figsize=(10, 5))

    for i in range(n_components):
```

```
        plt.subplot(2, 5, i + 1)
        plt.imshow(mean_images[i], cmap='gray')
        plt.axis('off')
    plt.suptitle('Cluster Means (Learned by GMM)')
    plt.show()
plot_gmm_means(gmm, pca)
```

Output:-





Expriment No.11

Code:-

```
import numpy as np
import cv2
import os

# Define dataset directory
dataset_path = "shapes_dataset"
shapes = ["circle", "square", "triangle"]
num_images = 100  # Number of images per shape

# Ensure directories exist
for shape in shapes:
    os.makedirs(os.path.join(dataset_path, shape), exist_ok=True)

# Function to draw shapes
def generate_shape(shape, size=(64, 64)):
    img = np.zeros(size, dtype=np.uint8)  # Black background
    center = (size[0] // 2, size[1] // 2)
    radius = 20

    if shape == "circle":
        cv2.circle(img, center, radius, 255, -1)
    elif shape == "square":
        cv2.rectangle(img, (center[0] - radius, center[1] - radius),
                      (center[0] + radius, center[1] + radius), 255, -1)
```

```
    elif shape == "triangle":
        pts = np.array([[center[0], center[1] - radius],
                        [center[0] - radius, center[1] + radius],
                        [center[0] + radius, center[1] + radius]], np.int32)
        pts = pts.reshape((-1, 1, 2))
        cv2.fillPoly(img, [pts], 255)

    return img

# Generate and save images
for shape in shapes:
    for i in range(num_images):
        img = generate_shape(shape)
        img_path = os.path.join(dataset_path, shape, f"{shape}_{i}.png")
        cv2.imwrite(img_path, img)

print("✅ Synthetic dataset generated successfully!")
import os
import numpy as np
import cv2
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KernelDensity, KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

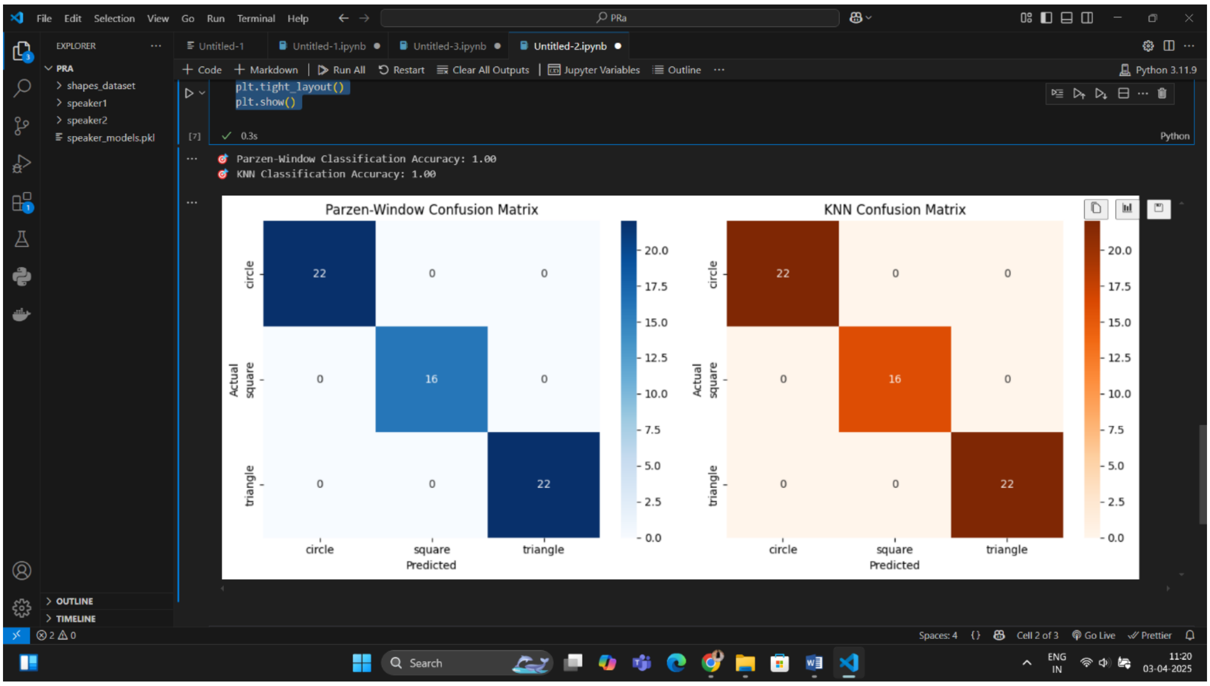
```
# Load dataset
dataset_path = "shapes_dataset"
shapes = ["circle", "square", "triangle"]
img_size = (64, 64)
X = []  # Feature vectors
y = []  # Labels
# Load images and extract features
for label, shape in enumerate(shapes):
    shape_path = os.path.join(dataset_path, shape)
    for img_name in os.listdir(shape_path):
        img_path = os.path.join(shape_path, img_name)
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)  # Load as grayscale
        img = cv2.resize(img, img_size)  # Resize (optional)
        X.append(img.flatten())  # Flatten image to 1D feature vector
        y.append(label)
# Convert to NumPy arrays
X = np.array(X)
y = np.array(y)
# Split dataset (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# --- Parzen-Window Classifier ---
kde_models = {}  # Dictionary to store KDE models per class
bandwidth = 0.2  # Smoothing parameter
for label in np.unique(y_train):
    kde = KernelDensity(kernel="gaussian", bandwidth=bandwidth)
    kde.fit(X_train[y_train == label])
```

```
    kde_models[label] = kde
# Classification using Parzen-Window method
y_pred_parzen = []
for x in X_test:
    probs = {label: kde_models[label].score_samples([x]) for label in
kde_models}
    y_pred_parzen.append(max(probs, key=probs.get))
# Compute accuracy
accuracy_parzen = accuracy_score(y_test, y_pred_parzen)
print(f"🎯 Parzen-Window Classification Accuracy: {accuracy_parzen:.2f}")


# --- K-Nearest Neighbors Classifier ---
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
# Compute accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f"🎯 KNN Classification Accuracy: {accuracy_knn:.2f}")
# --- Confusion Matrices ---
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
# Parzen-Window Confusion Matrix
sns.heatmap(confusion_matrix(y_test, y_pred_parzen), annot=True, fmt="d",
cmap="Blues",
            xticklabels=shapes, yticklabels=shapes, ax=axes[0])
axes[0].set_title("Parzen-Window Confusion Matrix")
axes[0].set_xlabel("Predicted")
axes[0].set_ylabel("Actual")
```

KNN Confusion Matrix

```
sns.heatmap(confusion_matrix(y_test, y_pred_knn), annot=True, fmt="d", cmap="Oranges",
            xticklabels=shapes, yticklabels=shapes, ax=axes[1])
axes[1].set_title("KNN Confusion Matrix")
axes[1].set_xlabel("Predicted")
axes[1].set_ylabel("Actual")
plt.tight_layout()
plt.show()
```

Output:-



Expriment No.12

Code:-

```
import numpy as np

import cv2

import os

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, make_scorer

import seaborn as sns

def create_shape_image(shape, size=(50, 50)):

    img = np.zeros(size, dtype=np.uint8)

    if shape == "circle":

        cv2.circle(img, (25, 25), 20, 255, -1)

    elif shape == "square":

        cv2.rectangle(img, (10, 10), (40, 40), 255, -1)

    elif shape == "triangle":

        points = np.array([[25, 5], [5, 45], [45, 45]], np.int32)

        cv2.fillPoly(img, [points], 255)

    return img

def generate_dataset(samples_per_class=100):

    shapes = ["circle", "square", "triangle"]

    images, labels = [], []

    for shape in shapes:
```

```
        for _ in range(samples_per_class):
            img = create_shape_image(shape)
            images.append(img.flatten())
            labels.append(shape)
    return np.array(images, dtype=np.float32), np.array(labels)
# Generate synthetic dataset
X, y = generate_dataset(samples_per_class=200)
# Encode labels
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
# Normalize features
scaler = StandardScaler()
X = scaler.fit_transform(X)
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)
# Custom scoring function to handle potential errors
scorer = make_scorer(accuracy_score)
# Hyperparameter tuning using GridSearchCV
param_grid = {'n_neighbors': [3, 5, 7, 9], 'metric': ['euclidean', 'manhattan',
'minkowski']}
knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring=scorer,
error_score='raise')
grid_search.fit(X_train, y_train)
# Best model selection
best_knn = grid_search.best_estimator_
```

y_pred = best_knn.predict(X_test)

Model evaluation

accuracy = accuracy_score(y_test, y_pred)

print(f'Best KNN Model: {grid_search.best_params_}')

print(f'Accuracy: {accuracy:.2f}')

print('Classification Report:\n', classification_report(y_test, y_pred))

Confusion matrix visualization

conf_matrix = confusion_matrix(y_test, y_pred)

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

Output:-