PDEU

PANDIT DEENDAYAL ENERGY UNIVERSITY

Formerly Pandit Deendayal Petroleum University (PDPU)

[redacted]

Laboratory Manual 20CP412P: Pattern Recognition

DEPARTMENT of COMPUTER SC. & ENGINEERING

SCHOOL OF TECHNOLOGY

Name of Student:Roll No:

Branch: Sem./Year:

Academic Year:

PANDIT DEENDAYAL ENERGY UNIVERSITY

Raisan, Gandhinagar – 380 007, Gujarat, India

Department of Computer Science & Engineering

Certificate

This is to certify that

Mr./Ms. _____ Roll no._____

Exam No. _____ of 7th Semester Degree course in Computer Science and Engineering has satisfactorily completed his/her term work in Pattern Recognition Lab (20CP412P) subject during the semester from_____ to _____ at School of Technology, PDEU.

Date of Submission:

Signature:

Faculty In-charge Head of Department

Index

Name:

Roll No:

Exam No:

Sr. No.	Experiment Title	Pages		Date of Completion	Marks (out of 10)	Sign.
		From	To			
1	Edge Detection using Sobel and Canny Operators					
2	Boundary Detection using Contour Tracing					
3	Feature Extraction using Hu Moments and HOG Descriptors					
4	K-Means and DBSCAN Clustering on Synthetic Dataset					
5	Decision Tree and Random Forest Classification on Iris Dataset					
6	Dimensionality Reduction using PCA on Face Dataset					
7	LDA for Class Separability and Visualization					
8	Pattern Matching using Euclidean Distance and Correlation Measures					
9	Implementation of Discrete HMM for Simple Weather Prediction					
10	Speech Signal Modeling using Continuous HMM (HTK or Python)					

Pattern Recognition Lab Manual

Language: Python
Libraries: NumPy, SciPy, OpenCV, scikit-learn, matplotlib

Experiment 1: Edge Detection using Sobel and Canny Operators

Objective:

To implement and compare edge detection techniques using Sobel and Canny operators on grayscale images.

Theory:

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness.

- **Sobel Operator** computes the gradient magnitude in both horizontal and vertical directions.
- **Canny Edge Detector** is a multi-stage algorithm that provides good noise suppression and strong edge detection using:
    1. Noise reduction
    2. Gradient calculation
    3. Non-maximum suppression
    4. Double thresholding
    5. Edge tracking by hysteresis

Libraries Required:

NumPy, SciPy, OpenCV, scikit-learn, matplotlib

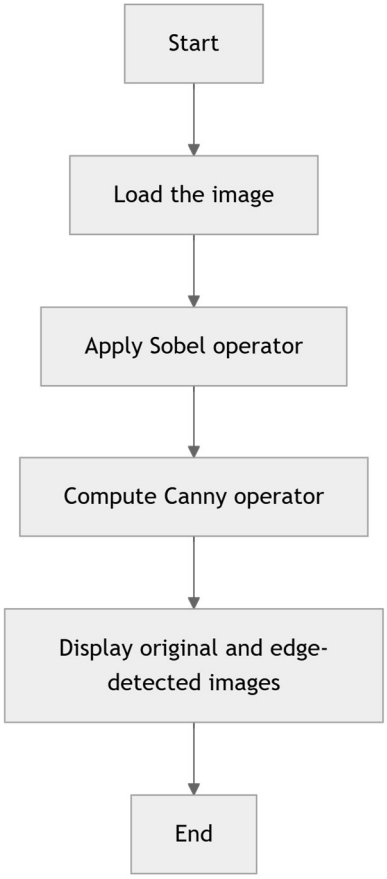
Dataset:

Use any standard grayscale image or download from online repositories.

Algorithm:

Step 1: Load the input image in grayscale
Step 2: Apply Gaussian blur to reduce noise (for Canny)
Step 3: Compute Sobel gradient (x and y directions)
Step 4: Combine gradients to obtain final Sobel edges
Step 5: Apply Canny edge detector
Step 6: Display the original, Sobel, and Canny results

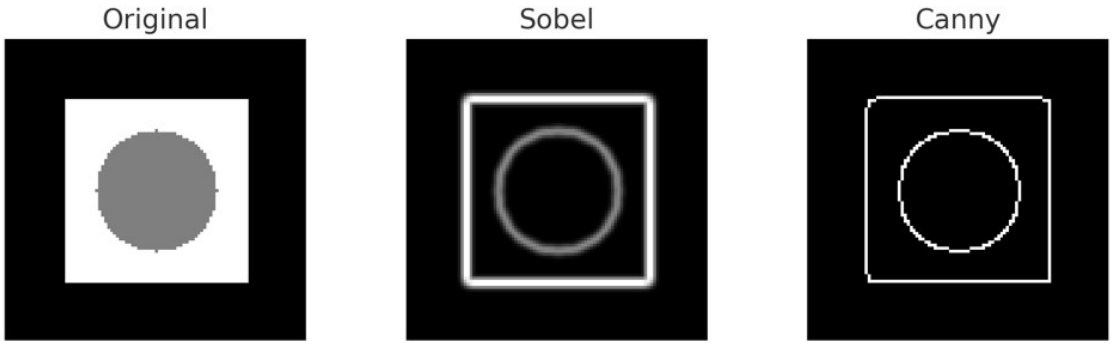
Flowchart:



Pseudo Code:

```
Input: Grayscale image
1. Read image in grayscale
2. Apply Gaussian blur for noise reduction
3. Compute Sobel edges:
   a. sobel_x = Sobel(image, dx=1, dy=0)
   b. sobel_y = Sobel(image, dx=0, dy=1)
   c. sobel_combined = sqrt(sobel_x^2 + sobel_y^2)
4. Compute Canny edges using cv2.Canny()
5. Display all results for comparison
```

Sample Output (using matplotlib):



Experiment 2: Boundary Detection using Contour Tracing

Objective:
To detect and trace object boundaries using contours in a binary image.

Theory:
Contours are continuous curves joining all the points along the boundary with the same color or intensity. In OpenCV, cv2.findContours() helps in tracing these contours..

Libraries Required:
NumPy, SciPy, OpenCV, scikit-learn, matplotlib

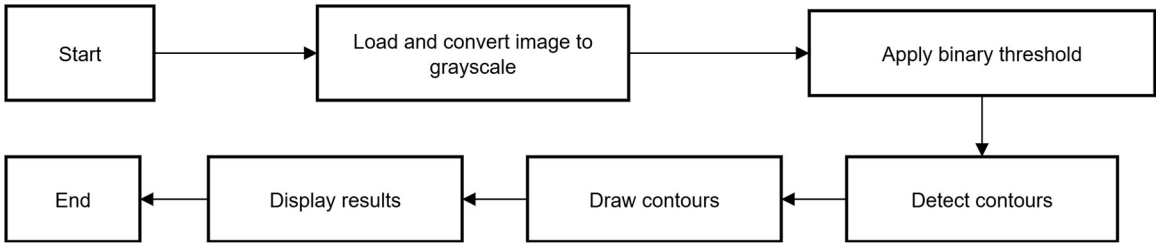
Dataset:
Binary shape images (e.g., shapes.png).

Algorithm:
Step 1. Read image and convert to binary

Step 2. Find contours using cv2.findContours()

Step 3. Draw contours on a copy of the image

Step 4. Display result

Flowchart:


Pseudo Code:
```
Input: Binary image
1. Read input image
2. Convert to grayscale and apply threshold
3. Detect contours
4. Draw contours
5. Display original and contour image
```

Sample Output (using matplotlib):
Similar example like experiment 2.

Experiment 3: Feature Extraction using Hu Moments and HOG Descriptors

Objective:

To extract shape-based features using Hu Moments and texture/edge-based features using Histogram of Oriented Gradients (HOG).

Theory:

- **Hu Moments**: Shape descriptors invariant to scale, rotation, and translation.
- **HOG**: Captures the gradient direction histogram for object detection.

Libraries Required:

```
import cv2
import numpy as np
from skimage.feature import hog
```

Dataset:

Use standard datasets or synthetic data depending on the experiment.

Algorithm:

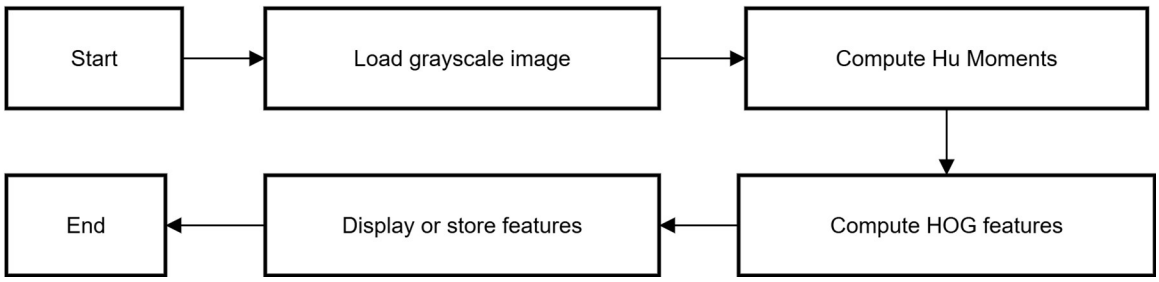
Step 1. Load image and convert to grayscale

Step 2. Compute Hu moments

Step 3. Compute HOG features

Step 4. Print or visualize the descriptors

Flowchart:



Pseudo Code:

```
Input: Grayscale image
1. Convert image to grayscale
2. Compute Hu Moments using cv2.HuMoments()
3. Compute HOG features using skimage.feature.hog
4. Display/print features
```

Sample Output (using matplotlib):

Output should present both global (Hu Moments) and local (HOG) shape descriptors, ready for use in a pattern recognition model.

Experiment 4: K-Means and DBSCAN Clustering on Synthetic Dataset

Objective:

To cluster data points using K-Means and DBSCAN algorithms.

Theory:

- **K-Means**: Partitions data into K clusters by minimizing intra-cluster variance.
- **DBSCAN**: Density-based clustering, effective for non-spherical clusters.

Libraries Required:

```
from sklearn.cluster import KMeans, DBSCAN
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
```

Dataset:

Use standard datasets or synthetic data depending on the experiment. Synthetic data can be generated using make_blobs.

Algorithm:

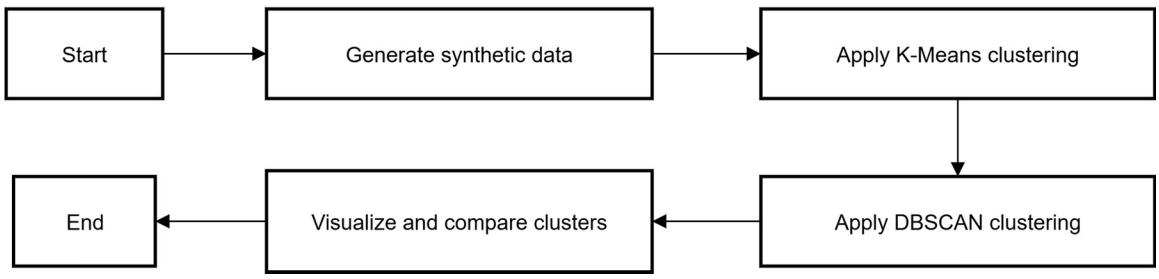
Step 1. Generate synthetic dataset

Step 2. Apply K-Means

Step 3. Apply DBSCAN

Step 4. Visualize results

Flowchart:



Pseudo Code:

```
Input: Data points
1. Generate or load data
2. Apply KMeans.fit_predict()
3. Apply DBSCAN.fit_predict()
4. Plot results with labels
```

Sample Output (using matplotlib):

K-Means:

Imagine a scatter plot here: 300 points plotted on an X-Y plane. The points should be colored distinctly into 3 groups (e.g., red, blue, green), representing the clusters found by K-Means. The centroids should be marked, e.g., with 'X' symbols of corresponding colors.

DBSCAN:

Imagine a scatter plot here: 300 points plotted on an X-Y plane. The majority of points should be colored distinctly into 3 groups (e.g., red, blue, green), representing the clusters found by DBSCAN. A small number of points, particularly those isolated or far from dense regions, should be colored differently (e.g., black or grey) to represent noise points.

Experiment 5: Decision Tree and Random Forest Classification on Iris Dataset

Objective:

To classify data using Decision Tree and Random Forest classifiers.

Theory:

- **Decision Tree** splits features based on entropy/gini.
- **Random Forest** combines multiple decision trees to improve generalization.

Libraries Required:

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

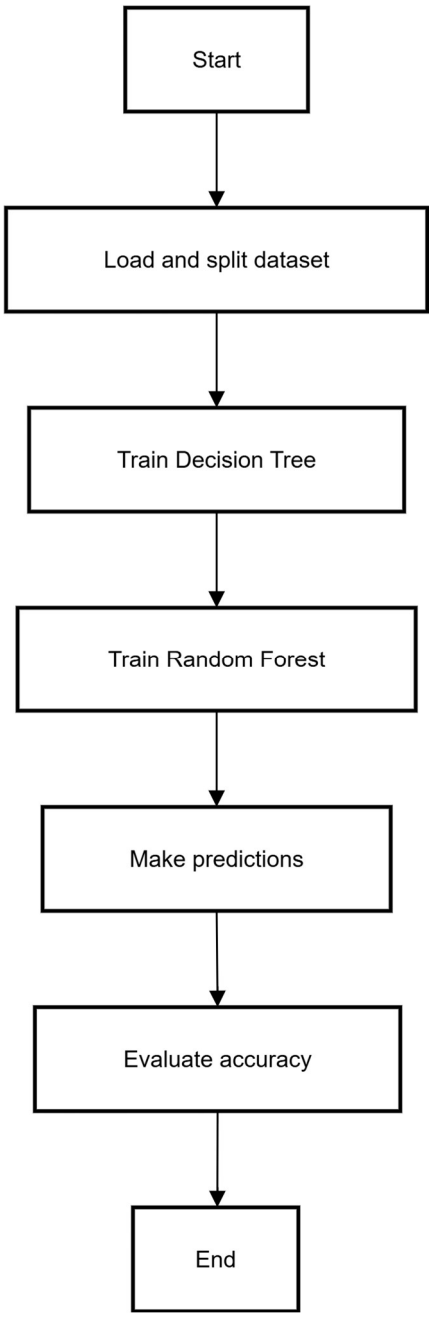
Dataset:

Use standard datasets like Iris Dataset on the experiment.

Pseudo Code:

```
Input: Labeled dataset
1. Load dataset and split into train/test
2. Train DecisionTreeClassifier
3. Train RandomForestClassifier
4. Predict and calculate accuracy
```

Flowchart:



Sample Output (using matplotlib):

Classification report and confusion matrix should be generated.

Experiment 6: Dimensionality Reduction using PCA on Face Dataset

Objective:

To apply Principal Component Analysis for feature reduction and visualization.

Theory:

PCA reduces dimensions by projecting data along principal components which capture maximum variance.

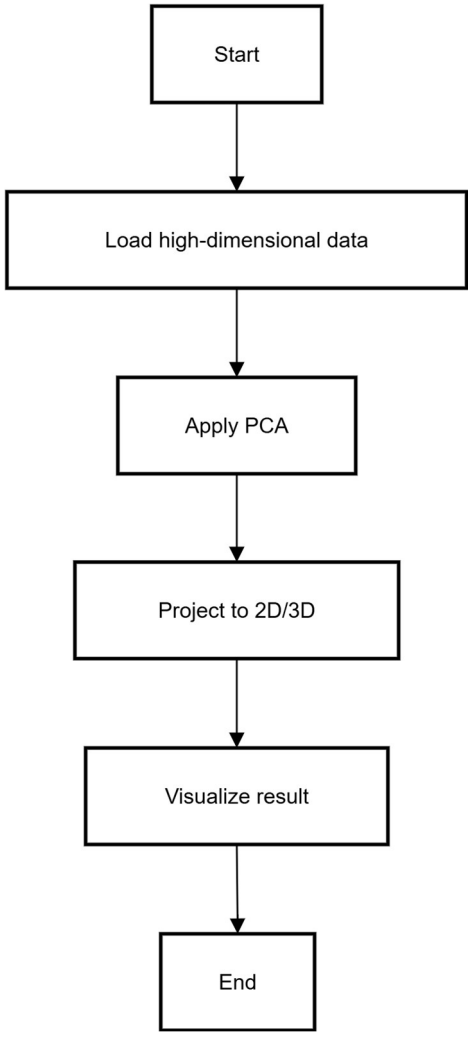
Libraries Required:

```
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
```

Dataset:

Use standard datasets like Digits dataset (from sklearn.datasets) on the experiment.

Mermaid Flowchart Code:



Pseudo Code:

```
Input: High-dimensional data
1. Load dataset
2. Apply PCA(n_components=2)
3. Plot projected data
```

Sample Output:

Results should be shown with following components:

- **Explained Variance Ratio:**

  - **Plot: Cumulative Explained Variance vs. Number of Components:**
  - **Summary:**
    - Original Dimensions
    - Reduced Dimensions
    - Variance Explained

- **Top Principal Components (Eigenfaces):**

Imagine a grid of small grayscale images here: Display 5-10 of the leading principal components (eigenvectors), reshaped back into 64x64 pixel images. These would typically look like ghostly, generalized face features, showing patterns of light and dark that correspond to variations in the dataset.

- **Image Reconstruction Example:**

This demonstrates how well the original image can be approximated using only the reduced set of principal components.

Original Image	Reconstructed with 10 Components	Reconstructed with 50 Components	Reconstructed with 100 Components

- **Reduced Dimensionality Data (Example for a Single Image):**

The original image, a 4096-dimensional vector, is now represented by a 50-dimensional vector.

Experiment 7: LDA for Class Separability and Visualization

Objective:

To apply Linear Discriminant Analysis (LDA) for dimensionality reduction and improved classification.

Theory:

LDA maximizes between-class variance and minimizes within-class variance.

Libraries Required:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris
```

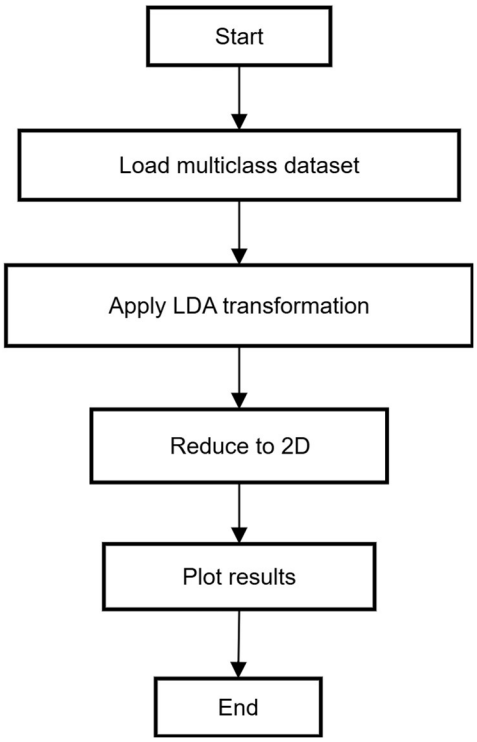
Dataset:

Use standard datasets like Iris Dataset on the experiment.

Pseudo Code:

```
Input: Multiclass dataset
1. Load dataset
2. Apply LDA(n_components=2)
3. Plot class-wise transformed features
```

Flowchart:



Sample Output (using matplotlib):

Results should be shown with following components:

- **Original Feature Space Visualization (Example using 2 of 4 features):**
  A scatter plot is required here: X-axis "Petal Length (cm)", Y-axis "Petal Width (cm)". Points for the 3 Iris species are plotted using different colors. Iris-setosa would be clearly separated, but Iris-versicolor and Iris-virginica would show significant overlap.

- **LDA Transformed Data Visualization:**
  A scatter plot is required here: X-axis "LDA Component 1", Y-axis "LDA Component 2". Points for the 3 Iris species are plotted using the *same colors* as the original plot. The points for all three classes should now appear much more distinctly separated, forming compact, well-defined clusters in the 2D LDA space.

- **Explained Variance Ratio by LDA Components:**
  The values like LDA Component 1, LDA Component 2, Total Explained Variance, etc. indicate the proportion of discriminatory information (class separability) captured by each component. Component 1 captures almost all of it for the Iris dataset.

- **Example of Transformed Data (First 5 samples):** Table required for Original (sepal_l, sepal_w, petal_l, petal_w), LDA Transformed (Component 1, Component 2), Class

Experiment 8: Pattern Matching using Euclidean Distance and Correlation Measures

Objective:

To match unknown input against known patterns using Euclidean distance.

Theory:

Pattern matching involves comparing feature vectors using similarity or distance metrics.

Libraries Required:

```
from scipy.spatial.distance import euclidean
```

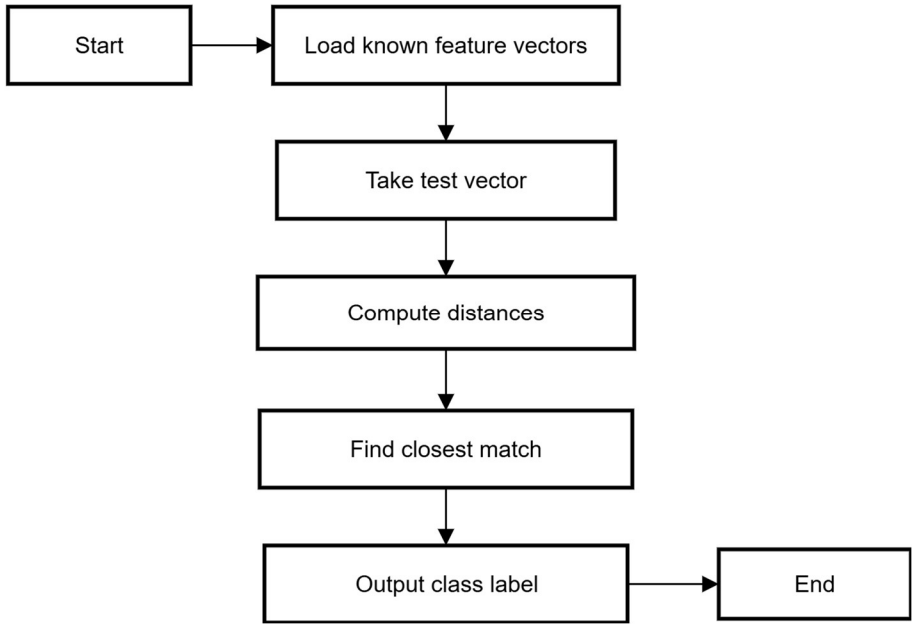
Dataset:

Use standard datasets or synthetic data depending on the experiment.

Algorithm:

Step 1. Define pattern feature vectors
Step 2. Take a test vector
Step 3. Compute distance from each known pattern
Step 4. Assign label of closest match

Flowchart:



Pseudo Code:

```
Input: Known pattern vectors, test vector
1. Compute Euclidean distance from each known vector
2. Find minimum distance
3. Assign matching label
```

Sample Output (using matplotlib):

Calculated Similarity/Dissimilarity Measures:

Comparison	Euclidean Distance	Pearson Correlation Coefficient
P_Q vs. P_A	0.2236	1.0000
P_Q vs. P_B	4.4721	1.0000
P_Q vs. P_C	4.4721	1.0000
P_Q vs. P_D	4.2426	-1.0000
P_Q vs. P_E	3.1623	-0.1980

.

Ranking of Reference Patterns:

Rank	By Euclidean Distance (Lower is Better)	By Pearson Correlation (Higher is Better)
1	P_A (0.2236)	P_A (1.0000)
2	P_E (3.1623)	P_B (1.0000)
3	P_D (4.2426)	P_C (1.0000)
4	P_B (4.4721)	P_E (-0.1980)
5	P_C (4.4721)	P_D (-1.0000)

Experiment 9: Implementation of Discrete HMM for Simple Weather Prediction

Objective:

To model weather sequences using a simple discrete Hidden Markov Model.

Theory:

HMM defines states and observable outputs with transition and emission probabilities.

Libraries Required:

```
from hmmlearn import hmm
import numpy as np
```

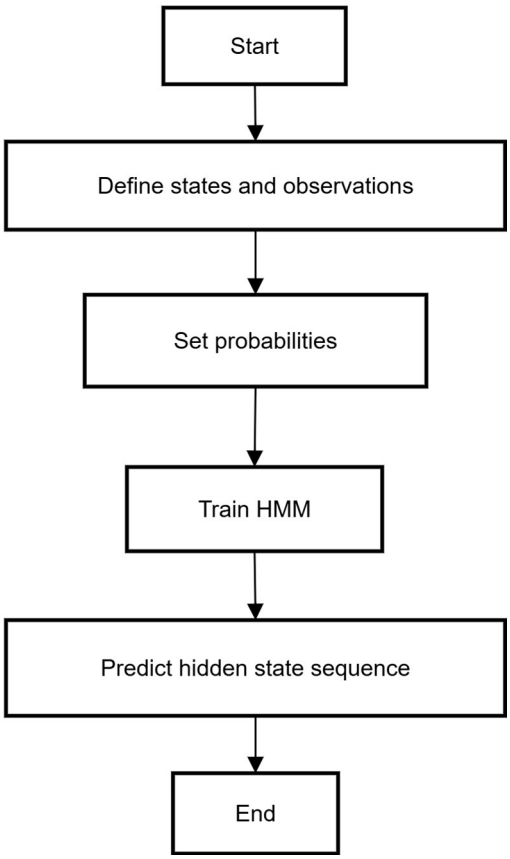
Dataset:

Toy dataset with discrete observations (e.g., Rainy, Sunny).

Algorithm:

Step 1. Define states and symbols
Step 2. Define transition/emission probabilities
Step 3. Train HMM on sequences
Step 4. Predict hidden states

Flowchart:



Pseudo Code:

```
Input: Sequence of observations
1. Define transition/emission matrices
2. Create DiscreteHMM model
3. Fit model
4. Decode hidden states
```

Sample Output:

Followings are required as output:

- Decoding (Viterbi Algorithm): Finding the Most Likely Hidden State Sequence
- Forecasting: Probability of Future Observation
- Probability of an Observation Sequence (Forward Algorithm)

Experiment 10: Speech Signal Modeling using Continuous HMM (HTK or Python)

Objective:

To model speech features using continuous HMMs.

Theory:

Continuous HMMs use Gaussian mixture models for representing observations like MFCC features in speech.

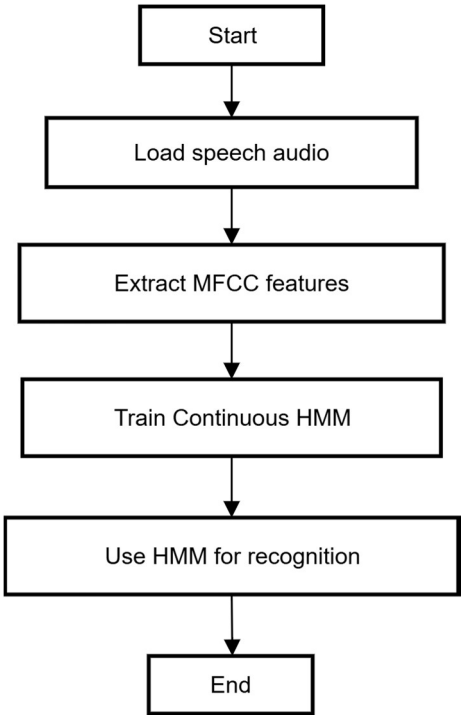
Libraries Required:

```
from hmmlearn import hmm
from python_speech_features import mfcc
```

Dataset:

Spoken digit samples or speech dataset

Mermaid Flowchart Code:



Pseudo Code:

```
Input: Audio files
1. Extract MFCC features
2. Create GaussianHMM model
3. Train on features
4. Predict or classify new samples
```

Sample Output (using matplotlib):

- Trained Model Parameters
- Speech Recognition (Decoding - Viterbi Algorithm)
- Performance Evaluation (for the entire test set)