****

**CSE\_3284**

**PATTERN RECOGNITION LAB MANUAL**

**THIRD YEAR**

**(2022-CURRICULUM)**

**VI SEMESTER**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING-ARTIFICIL ENGINEERING**

MANIPAL INSTITUTE OF TECHNOLOGY MAHE, BENGALURU

KARNATAKA - 560064

Contents

|  |  |
| --- | --- |
| **Lab no** | **Title** |
|  | COURSE OBJECTIVES AND OUTCOMES  EVALUATION PLAN , RECORD AND OBSERVATION WRITING . |
|  | PYTHON SOFTWARE DOWNLOAD AND INSTALLATION. |
| 1 | Train a neuron to learn the AND pattern classification problem using Perceptron learning |
| 2 | Write a program to Match the given input patterns |
| 3 | Write a program to compute Mahalanobis Distance Between Polygons. . |
| 4 | Write a program to compute Shannon entropy of the given data |
| 5 | Implement Supervised and unsupervised learning of database images |
| 6 | Write a program to implement Bayesian classifier. |
| 7 | Write a Program for Hopfield Neural Network. |
| 8 | Implement Fuzzy c-means clustering on given data. |
| 9 | Write a code using Artificial Neural Networks (ANN) for classifying male/female from facial images |
| 10 | Using ANN (Convolution NN) implement Handwritten Digit Recognition |
| 11 | Implement Fuzzy classification for real-time data |
| 12 | Implement k-means clustering algorithm for a real time image data |

**Course objectives:**

This laboratory course enables students to

• Apply fundamental algorithms and techniques in the area of NLP • Implement, design and test Language modeling • Apply the concepts of NLP techniques to solve real-world problems • Implement basic algorithms in classification applied to text

**Course outcomes:**

At the end of this course, students will gain the

* To evaluate and implement neural network systems for standard pattern recognition problems
* To apply the concepts of pattern Recognition to solve real-world problems
* To implement basic algorithms for pattern classification applied to text, numeric and image data.
* Compare the performance of different Pattern Recognition algorithms.

**Evaluation plan**

• **Internal Assessment Marks: 60%**

✓ Students must work out the all the programs using python

✓ Students shall submit the lab code along with the results obtained under every program as per the schedule. Eg: At the end of lab 2, both lab 1 & lab 2 programs should be submitted.

• **End semester assessment of 2-hour duration: 40%**

✓ A complete Python program using Pattern Recognition will be asked. The given program must be worked out in code and compiled. The **Python code** should be submitted. The name of file should be like **regno.c Eg: 20100345.**

***Sample Lab Observation Note Preparation***

**Title: SIMPLE C PROGRAMS**

**Sample input and output: Screen shot of result – must be included.**

**Experiment 1**

**1.Train a neuron to learn the AND pattern classification problem using Perceptron learning.**

import numpy as np

# Define the activation function

def step\_function(x):

    return 1 if x >= 0 else 0

# Perceptron learning algorithm

def perceptron\_learning(X, y, learning\_rate=0.1, epochs=10):

    # Initialize weights and bias

    weights = np.zeros(X.shape[1])

    bias = 0

    for epoch in range(epochs):

        for i in range(len(X)):

            # Calculate the linear combination

            linear\_output = np.dot(X[i], weights) + bias

            # Apply the activation function

            prediction = step\_function(linear\_output)

            # Update weights and bias

            error = y[i] - prediction

            weights += learning\_rate \* error \* X[i]

            bias += learning\_rate \* error

    return weights, bias

# Input data for AND gate

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

y = np.array([0, 0, 0, 1])

# Train the perceptron

weights, bias = perceptron\_learning(X, y)

# Test the perceptron

def predict(X, weights, bias):

    linear\_output = np.dot(X, weights) + bias

    return step\_function(linear\_output)

# Print the results

for x in X:

    print(f"Input: {x}, Predicted Output: {predict(x, weights, bias)}")

**Experiment-2**

**Write a program to Match the given input patterns.**

import pandas as pd

import re

import seaborn as sns

# Load the Titanic dataset from seaborn

titanic = sns.load\_dataset('titanic')

# Function to match patterns

def match\_pattern(data, column, pattern):

matched\_results = []

# Loop through each entry in the specified column and apply the pattern

for value in data[column]:

# Use re.match() to apply the regular expression

if re.match(pattern, str(value)):

matched\_results.append(value)

return matched\_results

# Function to get user input for column and pattern

def get\_user\_input():

# Display available columns

print("Available columns to search: 'name', 'age', 'sex', 'survived', 'pclass'")

column = input("Enter the column name to search: ").lower()

# Check if the input column is valid

if column not in ['name', 'age', 'sex', 'survived', 'pclass']:

print("Invalid column name. Please enter a valid column name.")

return

# Get the pattern to search for

pattern = input(f"Enter the pattern to match in the '{column}' column: ")

# Match the pattern and display the results

matched\_results = match\_pattern(titanic, column, pattern)

if matched\_results:

print(f"\nFound {len(matched\_results)} matches for the pattern '{pattern}' in column '{column}':")

for result in matched\_results:

print(result)

else:

print(f"\nNo matches found for the pattern '{pattern}' in column '{column}'.")

# Main function to execute the program

def main():

print("Welcome to the Titanic Dataset Pattern Matching Program using Regular Expressions!")

while True:

# Get user input for pattern matching

get\_user\_input()

# Ask the user if they want to search again

again = input("\nDo you want to search again? (yes/no): ").strip().lower()

if again != 'yes':

print("Thank you for using the program! Goodbye.")

break

# Run the program

main()

**Experiment-3**

**Write a program to compute Mahalanobis Distance Between Polygons**

import numpy as np

from scipy.spatial.distance import mahalanobis

def compute\_mahalanobis\_distance\_between\_polygons(polygon1, polygon2):

"""

Compute the Mahalanobis Distance between two polygons.

Parameters:

polygon1 (list of tuples): List of (x, y) coordinates representing the first polygon.

polygon2 (list of tuples): List of (x, y) coordinates representing the second polygon.

Returns:

float: The Mahalanobis distance between the centroids of the two polygons.

"""

# Convert the polygons into numpy arrays

poly1\_array = np.array(polygon1)

poly2\_array = np.array(polygon2)

# Compute the centroids of each polygon

centroid1 = np.mean(poly1\_array, axis=0)

centroid2 = np.mean(poly2\_array, axis=0)

# Combine both polygons' points to compute covariance matrix

combined\_points = np.vstack((poly1\_array, poly2\_array))

covariance\_matrix = np.cov(combined\_points.T)

# Handle the case of singular covariance matrix

try:

inv\_cov\_matrix = np.linalg.inv(covariance\_matrix)

except np.linalg.LinAlgError:

raise ValueError("Covariance matrix is singular and cannot be inverted. Ensure the polygons are not collinear.")

# Compute the Mahalanobis distance between the centroids

distance = mahalanobis(centroid1, centroid2, inv\_cov\_matrix)

return distance

# Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

def get\_polygon\_input(polygon\_num):

print(f"Enter the coordinates of Polygon {polygon\_num} (e.g., x1 y1, x2 y2, ...):")

points = input("Coordinates: ").strip().split(",")

polygon = [tuple(map(float, point.strip().split())) for point in points]

return polygon

try:

# Get user input for the polygons

polygon1 = get\_polygon\_input(1)

polygon2 = get\_polygon\_input(2)

# Compute the Mahalanobis Distance

distance = compute\_mahalanobis\_distance\_between\_polygons(polygon1, polygon2)

print(f"Mahalanobis Distance between the polygons: {distance}")

except ValueError as e:

print(e)

except Exception as e:

print(f"An error occurred: {e}")

**Experiment-4**

**Write a program to compute Shannon entropy of the given data.**

import math

from collections import Counter

import pandas as pd

def calculate\_shannon\_entropy(data):

# Count the frequency of each unique element in the data

counts = Counter(data)

# Total number of elements

total\_count = sum(counts.values())

# Calculate the entropy

entropy = 0.0

for count in counts.values():

probability = count / total\_count

entropy -= probability \* math.log2(probability)

return entropy

# Load the Titanic dataset

url = "<https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv>"

df = pd.read\_csv(url)

# Prompt the user to select a column

print("Available columns in the Titanic dataset:")

print(df.columns.tolist())

column = input("Enter the column name to calculate Shannon entropy: ")

if column in df.columns:

data = df[column]

entropy = calculate\_shannon\_entropy(data)

print(f"Shannon Entropy of the column '{column}': {entropy}")

else:

print(f"Column '{column}' not found in the dataset. Please try again.")

**Experiment -8**

**Implement Fuzzy c-means clustering on given data. import numpy as np**

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.preprocessing import StandardScaler

from fcmeans import FCM

from sklearn.datasets import load\_wine

# Load publicly available dataset (Wine dataset)

wine = load\_wine()

X = wine.data[:, :2] # Taking only the first two features for visualization

# Standardizing the dataset

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Apply Fuzzy C-Means

fcm = FCM(n\_clusters=3, m=2.0, max\_iter=150, error=1e-5, random\_state=42)

fcm.fit(X\_scaled)

# Get cluster centers and labels

centers = fcm.centers

labels = fcm.predict(X\_scaled)

# Plot results

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

for i in range(3):

plt.scatter(X\_scaled[labels == i, 0], X\_scaled[labels == i, 1], label=f'Cluster {i+1}')

plt.scatter(centers[:, 0], centers[:, 1], marker='x', color='black', s=200, label='Centers')

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.legend()

plt.title("Fuzzy C-Means Clustering on Wine Dataset")

plt.show()

**Experiment -5**

**Implement Supervised and unsupervised learning of database images.**

import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.utils import to\_categorical

# Load images from directory

def load\_images\_from\_folder(folder, img\_size=(64, 64)):

images = []

labels = []

class\_names = os.listdir(folder)

for label, class\_name in enumerate(class\_names):

class\_path = os.path.join(folder, class\_name)

if not os.path.isdir(class\_path):

continue

for file in os.listdir(class\_path):

img\_path = os.path.join(class\_path, file)

img = cv2.imread(img\_path)

if img is not None:

img = cv2.resize(img, img\_size)

img = img / 255.0 # Normalize

images.append(img)

labels.append(label)

return np.array(images), np.array(labels), class\_names

# Load dataset

folder\_path = "path\_to\_your\_images"

X, y, class\_names = load\_images\_from\_folder(folder\_path)

# Split into training and testing

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# One-hot encoding labels for supervised learning

y\_train\_cat = to\_categorical(y\_train, num\_classes=len(class\_names))

y\_test\_cat = to\_categorical(y\_test, num\_classes=len(class\_names))

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Build CNN Model

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(64, 64, 3)),

MaxPooling2D(pool\_size=(2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D(pool\_size=(2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dropout(0.5),

Dense(len(class\_names), activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train\_cat, epochs=10, validation\_data=(X\_test, y\_test\_cat), batch\_size=32)

# Evaluate the model

test\_loss, test\_acc = model.evaluate(X\_test, y\_test\_cat)

print(f"Test Accuracy: {test\_acc:.2f}")

from sklearn.cluster import KMeans

# Flatten images for clustering

X\_flattened = X.reshape(X.shape[0], -1)

# Apply K-Means

k = len(class\_names) # Number of clusters (same as number of classes)

kmeans = KMeans(n\_clusters=k, random\_state=42)

labels = kmeans.fit\_predict(X\_flattened)

# Plot some clustered images

fig, axes = plt.subplots(1, 5, figsize=(10, 5))

for i, ax in enumerate(axes):

ax.imshow(X[i])

ax.set\_title(f"Cluster {labels[i]}")

ax.axis('off')

plt.show()

**Experiment -6**

**Write a program to implement Bayesian classifier**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.datasets import load\_iris

# Load dataset

data = load\_iris()

X = data.data # Features

y = data.target # Target labels

# Split dataset 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)

# Create and train the Bayesian classifier

model = GaussianNB()

model.fit(X\_train, y\_train)

# Predict on test data

y\_pred = model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred, target\_names=data.target\_names)

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

print("Classification Report:\n", report)

**Experiment -10**

**Implement k-means clustering algorithm for a real time image data.**

import cv2 import numpy as ncap = cv2.VideoCapture(0)

while True: ret, frame = cap.read() if not ret: break

# Resize frame to reduce processing time  
resized\_frame = cv2.resize(frame, None, fx=0.5, fy=0.5)  
  
# Reshape image into a 2D array of pixels (rows: pixels, columns: RGB)  
pixel\_data = resized\_frame.reshape((-1, 3))  
pixel\_data = np.float32(pixel\_data)  
  
# Apply k-means clustering  
\_, labels, centers = cv2.kmeans(  
 pixel\_data, k, None, criteria, attempts, cv2.KMEANS\_PP\_CENTERS  
)  
  
# Convert centers to 8-bit values  
centers = np.uint8(centers)  
  
# Replace pixel values with their corresponding center values  
segmented\_data = centers[labels.flatten()]  
segmented\_image = segmented\_data.reshape(resized\_frame.shape)  
  
# Display the segmented image  
cv2.imshow('Real-Time K-Means Clustering', segmented\_image)  
  
# Exit loop on 'q' key press  
if cv2.waitKey(1) & 0xFF == ord('q'):  
 break