Experiment No. : ....Date: ....Page No.:..................

Name of the Experiment : ....

5) i) Problem Statement :
For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithmn to output a description of the set of all hypothesis consistent.

ii) Objective :
The purpose of this program is to implement the find-S algorithm for finding the most specific hypothesis from a given set of training data stored in a .CSV file

iii) Algorithm :
* Import Necessary Libraries : Import the pandas library for handling the .CSV file
* Define Find-S Algorithm Function : Implement the find-S algorithm as a fuction that reads data from a .CSV file
c) Initialize Hypothesis : start with the most general hypothesis
d) update Hypothesis : For each positive example, refine the hypothesis by comparing attribute values
e) Return Final Hypothesis : Output the most specific hypothesis consistent with all positive training examples.

Experiment No. : .... Date: .... Page No.: ....

Name of the Experiment : ....

iv) Source | Program Code

```
import numpy as np
import matplotlib.pyplot as plt
from collections import counter

data = np.random.rand (100)

labels = ["class1" if x <= 0.5 else "class2" for
                              x in data[:50]]

def euclidean_distance(x1, x2):
    return abs(x1 - x2)

def knn_classifier(train_data, train_labels, test_point, k)
                                                          :
        distances = [(euclidean_distance(test_point, train_
                  data[i]), train_labels[i]) for i in
                         range(len(train_data))]

    distance.sort(key = lambda x: x[0])
    k_nearest_neighbours = distance[:k]
    k_nearest_labels = [label for _, label in k_
                                   nearest_neighbours]
    return Counter(k_nearest_labels).most_common(1)
                                              [0][0]

train_data = data[:50]
train_labels = labels
```

Experiment No. : .... Date: .... Page No.: ....

Name of the Experiment : ....

```
test_data = data[50:]
k_values = [1, 2, 3, 4, 5, 20, 30]
print("--- k-Nearest Neighbours Classification ---")
print("Training dataset: First 50 points labeled
  - based on the rule (x <= 0.5 -> class1, x > 0.5
  - -> Class2)")
print("Testing dataset: Remaining 50 points to be
  - classified \n")
results = {}

for k in k_values:
    print(f"Results for k = {k}:")
    classified_labels = [knn_classifier(train_data, train
      - _labels, test_point, k) for test_point
      - in test_data]
    results[k] = classified_labels

    for i, label in enumerate(classified_labels, start=
        print(f"Point x{i} (value: {test_data      51):
          - [i-51]:.4f}) is classified as {label}")
    print("\n")
print("Classification complete.\n")

for k in k_values:
    classified_labels = results[k]
    class1_points = [test_data[i] for i in range(len
      - (test_data)) if classified_labels[i] == 'class1"]
```

Experiment No. : ....Date: ....Page No.:..............................

Name of the Experiment : ....

```
class2_points = [test_data[i] for i in range(len
    (test_data)) if classified_labels[i] == "class2"]

plt.figure(figsize=(10,6))
plt.scatter(train_data, [0] * len(train_data), c=
    ["blue" if label == "class1" else "red" for
        label in train_labels],
        label="Training Data", marker="o")
plt.scatter(class1_points, [1] * len(class1_points),
        c="blue", label="class1 (Test)", marker
        ="x")

plt.scatter(class2_points, [1] * len(class2_points), c
    ="red", label="class 2 (Test)", marker="x")

plt.title(f"k-NN Classification Results for k={k}")
plt.xlabel("Data Points")
plt.ylabel("Classification Level")
plt.legend()
plt.grid(True)
plt.show()
```

v) Compilation and Execution steps

* Prepare a CSV file (e.g, training_data.csv) with training data examples.
* Save the program in a python file (e.g find_s.py)
* open a terminal or command prompt
* Navigate to the directory containing the python file and CSV file
* Run the program using the command:

Experiment No. : ....Date: ....Page No.:..................................

Name of the Experiment : ....

python find_s.py

vi) Sample Input and Output :

Sample Input :
A .CSV file containing training data examples with various attributes and a class label (Yes/NO)

Sample Output :
The final hypothesis generated by the Find-S algorithm, displayed as a list of attribute values or '?' indicating generalization

vii) Explaination of Output :
The output shows the specific hypothesis that covers all positive training examples. If a value differs between positive examples, the hypothesis will have a '?' at that position, indicating generalization.

viii) Observation and Analysis :
* The Find-S Algorithm produces a hypothesis that is as specific as possible, covering only positive examples
* It cannot handle noisy data or negative examples properly.

Experiment No. : ....Date: ....Page No.:.....................................

Name of the Experiment : ....

ix) Conclusion:

The find - S algorithm was successfully implemented and tested with a .csv file, the algorithm provides the most specific hypothesis that fits all positive training examples. However, it is sensitive to noise and incomplete data.

6) i) Problem Statement :

Develop a program to implement Locally weighted Regression (LWR) to fit a curve to randomly generated data points. Use a Gaussian kernal to compute the weights and visualize the resulting curve

ii) Objective :

The purpose of this program is to implement Locally Weighted Regression to predict values using a weighted Linear regression technique where weights decrease with distance from the query point

iii) Algorithm :

a) Import necessary libraries : import numpy and matplotlib for numerical computation and visualization
b) Generate Data : Create random data points following a sinsoidal pattern with noise
c) Define Gaussian Kernal Function : Calculate the weight for each ~~ea~~ point based on its distance from the query point
d) Implement LWR Algorithm : Compute weighted linear regression for each test point using calculated weights
e) Visualize Results : Plot the predicted curve along with training data points

Experiment No. : ....Date: ....Page No.:..................

Name of the Experiment : ....

Source / Program Code

```
import numpy as np
import matplotlib.pyplot as plt

def gaussian-kernal (x, xi, tau):
    return np.exp (-np.sum ((x - xi)**2) /
                   - (2 * tau ** 2))

def locally-weighted-regression (x, X, y, tau):
    m = X.shape [0]
    weights = np.array ([gaussian-kernal (x, X[i],
                         tau) for i in range(m)])
    w = np.diag (weights)
    X_transpose_W = X.T @ W
    theta = np.linalg.inv (X_transpose_W @ X) @
              - X_transpose_W @ y
    return x @ theta

np.random.seed (42)
X = np.linspace (0, 2 * np.pi, 100)
y = np.sin (X) + 0.1 * np.random.randn (100)
X-bias = np.c_[np.ones (X.shape), X]

x-test = np.linspace (0, 2 * np.pi, 200)
x_test_bias = np.c_[np.ones(x-test.shape), x_test]

tau = 0.5
y-pred = np.array ([locally-weighted-regression (xi,
           # X_bias, y, tau) for xi in x_test_
                                          bias])
```

Experiment No. : .... Date: .... Page No.: ....

Name of the Experiment : ....

```
plt.figure(figsize = (10, 6))
plt.scatter(x, y, color = 'red', label = 'Training Data',
                        alpha = 0.7)
plt.plot(x_test, y_pred, color = 'blue', label = f'LWR
          Fit (tau = {tau})', linewidth = 2)
plt.xlabel('x', fontsize = 12)
plt.ylabel('y', fontsize = 12)
plt.title('Locally Weighted Regression', fontsize = 14)
plt.legend(fontsize = 10)
plt.grid(alpha = 0.3)
plt.show()
```

v) Compilation and Execution steps:

* Save the program in a python file (eg lwr.py)
* open a terminal or command prompt.
* Navigate to the directory containing the python file
* Run the program using the command

```
python lwr.py
```

vi) Sample Input and output:

Sample Input:

* 100 randomly generated data points along a sinusoidal curve
* tau = 0.5

Sample output:

* A smooth curve fitted to the noisy data points using Locally Weighted Regression.

Experiment No. : .... Date: .... Page No.: ....

Name of the Experiment : ....

vii) Explaination of output :

The output displays a fitted curve that closely follows the underlying sinusoidal pattern, demonstra-ting the effectiveness of LWR for non-linear data fitting

viii) Observation and Analysis :

* Smaller values of tau result in a closer fit to the training data but may cause overfitting
* Larger values of tau produce a smoother curve but may underfit the data

xi) Conclusion :

The Locally Weighted Regression algorithm was successfully implemented and visualized. The model effectively fits a smooth curve to the data, demonstrating the capability of LWR for non-parametric regression.