**LAB PROGRAM 4: For a given set of training data examples stored in a .csv file, implement and demonstrate the Find – S algorithm to output a description of the set of all hypotheses consistent with the training examples.**

**#Step 1: Define variables**

**num\_attributes = 6**

**a = []**

**#Step 2: Reading the Dataset**

**print("\nThe Given Training Data Set\n")**

**with open('enjoysport.csv', 'r') as csvfile:**

**reader = csv.reader(csvfile)**

**for row in reader:**

**a.append(row)**

**print(row)**

**#Step 3:** **Initializing the Hypothesis**

**print("\nThe initial value of hypothesis:")**

**hypothesis = ['0'] \* num\_attributes**

**print(hypothesis)**

**#Step 4: Applying the Find-S Algorithm**

**print("\nFind-S Algorithm: Finding a Maximally Specific Hypothesis\n")**

**for i in range(len(a)):**

**if a[i][num\_attributes] == 'yes': # Ensure correct column index**

**#Step 5: Initializing or Generalizing Hypothesis**

**if hypothesis == ['0'] \* num\_attributes:**

**# Initialize hypothesis with first positive example**

**hypothesis = a[i][:num\_attributes]**

**else:**

**# Generalize hypothesis**

**for j in range(num\_attributes):**

**if a[i][j] != hypothesis[j]:**

**hypothesis[j] = '?'**

**#Step 6: Printing Hypothesis After Each Training Instance**

**print(f"For Training instance No:{i}, the hypothesis is", hypothesis)**

**#Step 7: Final Hypothesis Output**

**print("\nThe Maximally Specific Hypothesis for the given Training Examples:\n")**

**print(hypothesis)**

**.csv file:**

**Create CSV File Manually**

1. Open **Notepad** or any text editor.
2. Enter the following data and save it as **enjoysport.csv** (Make sure the extension is .csv
3. Save the file as **enjoysport.csv** and place it in the same directory as your Python script.

Sunny,Warm,Normal,Strong,Warm,Same,yes

Sunny,Warm,High,Strong,Warm,Same,yes

Rainy,Cold,High,Strong,Warm,Change,no

Sunny,Warm,High,Strong,Cool,change,yes

**OUTPUT 1: First Instances = YES**

The Given Training Data Set

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'yes']

['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'yes']

['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'no']

['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'change', 'yes']

The initial value of hypothesis:

['0', '0', '0', '0', '0', '0']

Find-S Algorithm: Finding a Maximally Specific Hypothesis

For Training instance No:0, the hypothesis is ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

For Training instance No:1, the hypothesis is ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

For Training instance No:2, the hypothesis is ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

For Training instance No:3, the hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']

The Maximally Specific Hypothesis for the given Training Examples:

['Sunny', 'Warm', '?', 'Strong', '?', '?']

**OUTPUT 2: First Instances = NO**

The Given Training Data Set

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'no']

['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'yes']

['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'no']

['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'change', 'yes']

The initial value of hypothesis:

['0', '0', '0', '0', '0', '0']

Find-S Algorithm: Finding a Maximally Specific Hypothesis

For Training instance No:0, the hypothesis is ['0', '0', '0', '0', '0', '0']

For Training instance No:1, the hypothesis is ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same']

For Training instance No:2, the hypothesis is ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same']

For Training instance No:3, the hypothesis is ['Sunny', 'Warm', 'High', 'Strong', '?', '?']

The Maximally Specific Hypothesis for the given Training Examples:

['Sunny', 'Warm', 'High', 'Strong', '?', '?']

**LAB PROGRAM 5: Develop a program to implement k-Nearest Neighbour algorithm to classify the randomly generated 100 values of x in the range of [0,1]. Perform the following based on dataset generated.**

1. **Label the first 50 points {x1,……,x50} as follows: if (xi ≤ 0.5), then xi ε Class1, else xi ε Class2**
2. **Classify the remaining points, x51,……,x100 using KNN. Perform this for k=1,2,3,4,5,20,30**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from collections import Counter**

**# Step 1: Generate 100 random values in the range [0,1]**

**n\_samples = 100**

**np.random.seed(42)**

**x\_values = np.random.rand(n\_samples)**

**# Step 2: Label the first 50 points**

**labels = np.where(x\_values[:50] <= 0.5, "Class1", "Class2")**

**# Print x\_values and labels**

**print("Generated x\_values:")**

**print(x\_values)**

**print("\nAssigned labels for first 50 points:")**

**print(labels)**

**# Step 3: Define the KNN function**

**def knn\_classify(x\_train, y\_train, x\_test, k):**

**predictions = []**

**for x in x\_test:**

**distances = np.abs(x\_train - x)**

**k\_nearest\_labels = y\_train[np.argsort(distances)[:k]]**

**predictions.append(Counter(k\_nearest\_labels).most\_common(1)[0][0])**

**return np.array(predictions)**

**# Step 4: Classify the remaining 50 points using KNN for different k values**

**k\_values = [1, 2, 3, 4, 5, 20, 30]**

**results = {k: knn\_classify(x\_values[:50], labels, x\_values[50:], k) for k in k\_values}**

**# Step 5: Print classification results**

**for k, preds in results.items():**

**print(f"\nResults for k={k}:")**

**print(preds)**

**# Step 6: Visualization**

**plt.figure(figsize=(10, 6))**

**for k in k\_values:**

**plt.scatter(x\_values[50:], [k] \* 50, c=["blue" if lbl == "Class1" else "red" for lbl in results[k]], label=f"k={k}", marker='o')**

**plt.xlabel("x values")**

**plt.ylabel("k values (for visualization)")**

**plt.title("KNN Classification Results for Different k Values")**

**plt.colorbar(label="Predicted Class")**

**plt.legend()**

**plt.show()**

**OUTPUT:**

Generated x\_values:

[0.37454012 0.95071431 0.73199394 0.59865848 0.15601864 0.15599452

0.05808361 0.86617615 0.60111501 0.70807258 0.02058449 0.96990985

0.83244264 0.21233911 0.18182497 0.18340451 0.30424224 0.52475643

0.43194502 0.29122914 0.61185289 0.13949386 0.29214465 0.36636184

0.45606998 0.78517596 0.19967378 0.51423444 0.59241457 0.04645041

0.60754485 0.17052412 0.06505159 0.94888554 0.96563203 0.80839735

0.30461377 0.09767211 0.68423303 0.44015249 0.12203823 0.49517691

0.03438852 0.9093204 0.25877998 0.66252228 0.31171108 0.52006802

0.54671028 0.18485446 0.96958463 0.77513282 0.93949894 0.89482735

0.59789998 0.92187424 0.0884925 0.19598286 0.04522729 0.32533033

0.38867729 0.27134903 0.82873751 0.35675333 0.28093451 0.54269608

0.14092422 0.80219698 0.07455064 0.98688694 0.77224477 0.19871568

0.00552212 0.81546143 0.70685734 0.72900717 0.77127035 0.07404465

0.35846573 0.11586906 0.86310343 0.62329813 0.33089802 0.06355835

0.31098232 0.32518332 0.72960618 0.63755747 0.88721274 0.47221493

0.11959425 0.71324479 0.76078505 0.5612772 0.77096718 0.4937956

0.52273283 0.42754102 0.02541913 0.10789143]

Assigned labels for first 50 points:

['Class1' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2'

'Class2' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1'

'Class1' 'Class2' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class1'

'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1'

'Class1' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class1'

'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2'

'Class2' 'Class1']

Results for k=1:

['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'

'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'

'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'

'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'

'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'

'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1'

'Class1' 'Class1']

Results for k=2:

['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'

'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'

'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'

'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'

'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'

'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1'

'Class1' 'Class1']

Results for k=3:

['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'

'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'

'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'

'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'

'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'

'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1'

'Class1' 'Class1']

Results for k=4:

['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'

'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'

'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'

'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'

'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'

'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1'

'Class1' 'Class1']

Results for k=5:

['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'

'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'

'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'

'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'

'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'

'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1'

'Class1' 'Class1']

Results for k=20:

['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'

'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'

'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'

'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'

'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'

'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1'

'Class1' 'Class1']

Results for k=30:

['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'

'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'

'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'

'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'

'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'

'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1'

'Class1' 'Class1']

