

D	D	M	M	Y	Y	Y	Y

1. Aim: Illustrate and Demonstrate the working model and principle of Find-S algorithm.

Program: 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.

```
import numpy as np
import pandas as pd
```

```
data = pd.read_csv("ENJOYSPORT.csv")
```

```
d = np.array(data)[ :, :-1]
```

```
target = np.array(data)[ :, -1]
```

```
def train(c, t):
```

```
    for i, val in enumerate(t):
```

```
        if val == "yes":
```

```
            specific_hypothesis = c[i].copy()
```

```
            break
```

```
    for i, val in enumerate(c):
```

```
        if t[i] == "yes":
```

```
            for x in range(len(specific_hypothesis)):
```

```
                if val[x] != specific_hypothesis[x]:
```

```
                    and specific_hypothesis[x] == "?":
```

```
                        specific_hypothesis[x] = val[x]
```

```
            else:
```

```
                pass
```

```
    return specific_hypothesis
```

~~18/6/24~~

$$[[c_{\text{army}}^*, c_2^*, c_3^*, c_4^*, c_5^*, c_6^*], [c_7^*, c_{\text{army}}^*, c_8^*, c_9^*, c_{10}^*], [c_{11}^*]]$$

```
print("Final General h:", g-final, sep = "\n")
```




3. Aim: Understand and analyse the concept of Regression algorithm techniques.

Program: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

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

```
def locally_weighted_regression(x_query, x_train, y_train, tau=0.1):
    m = x_train.shape[0]
    weights = np.exp(-(np.sum((x_train - x_query)**2, axis=1) / (2 * tau * tau)))
    w = np.diag(weights)
    theta = np.linalg.pinv(x_train.T @ w @ x_train) @ (x_train.T @ w @ y_train)
    prediction = x_query @ theta
    return prediction
```

```
np.random.seed(0)
x_train = np.linspace(0, 10, 50)
y_train = np.sin(x_train) + np.random.normal(0, 0.1, x_train.shape[0])
x_query = np.linspace(0, 10, 100)
```

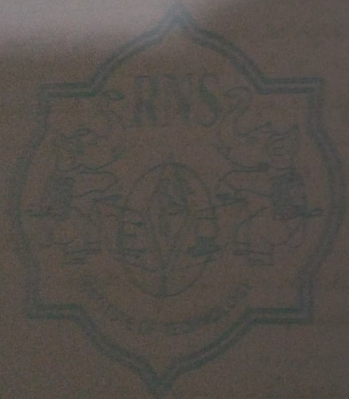
```
tau = 0.5
```

```
predictions = []
```

```
for xq in x_query:
```

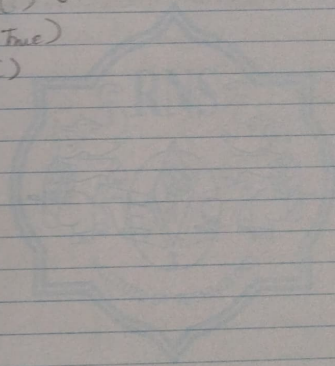
```
    x_query = np.array([1, xq])
```

```
    prediction = locally_weighted_regression(x_query,
```



```
np.c_[np.ones(X_train.shape[0]), X_train], y_train, stage  
predictions.append(prediction)
```

```
plt.figure(figsize=(10,6))  
plt.scatter(X_train, y_train, color='blue', label='training data')  
plt.plot(X_query, predictions, color='red', label='locally weighted')  
plt.xlabel('x')  
plt.ylabel('y')  
plt.title('Locally weighted Regression')  
plt.legend()  
plt.grid(True)  
plt.show()
```



Output:
Accuracy: 1.0

Correct Predictions:

Input: [6.1 2.8 4.7 1.2] Actual Class: versicolor Predicted Class: versicolor

Input: [5.7 3.8 1.7 0.3] Actual Class: setosa Predicted Class: setosa

Input: [7.7 2.6 6.9 2.3] Actual Class: virginica Predicted Class: virginica

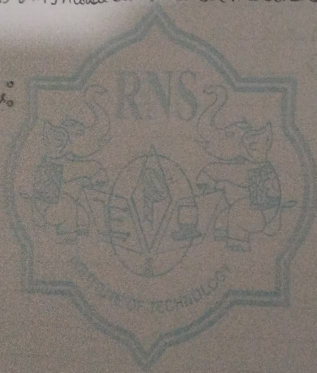
Input: [6.2 2.9 4.5 1.5] Actual Class: versicolor Predicted Class: versicolor

Input: [6.8 2.8 4.8 1.4] Actual Class: versicolor Predicted Class: versicolor

Input: [5.4 3.4 1.5 0.4] Actual Class: setosa Predicted Class: setosa

....

Wrong Predictions:



RNSIT

4. Aim: Demonstrate and analyse the results of classification based on KNN Algorithm.

Program: Write a program to implement k-Nearest Neighbors algorithm to classify the iris dataset. Print both correct and wrong predictions. Javal bython ML library classes can be used for this problem.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
iris = load_iris()
```

```
X = iris.data
```

```
y = iris.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
k = 3
```

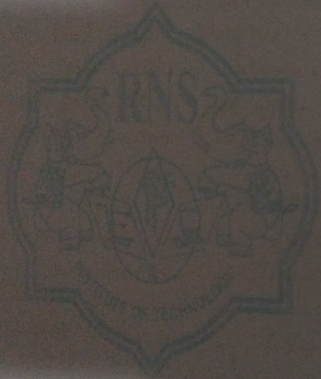
```
knn = KNeighborsClassifier(n_neighbors=k)
```

```
knn.fit(X_train, y_train)
```

```
predictions = knn.predict(X_test)
```

```
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: ", accuracy)
```

Confusion Matrix

$$\begin{bmatrix} 19 & 0 & 0 \\ 0 & 13 & 0 \\ 0 & 0 & 13 \end{bmatrix}$$


```
correct_predictions = []
wrong_predictions = []
for i in range(len(predictions)):
    if predictions[i] == y_test[i]:
        correct_predictions.append((x_test[i], y_test[i], predictions[i]))
    else:
        wrong_predictions.append((x_test[i], y_test[i], predictions[i]))
```

```
print("In Correct Predictions:")
for prediction in correct_predictions:
    print("Input:", prediction[0], "Actual Class:", rns.target_names[
        prediction[1]], "Predicted Class:", rns.target_name[prediction[2]])
```

```
print("In Wrong Predictions:")
for prediction in wrong_predictions:
    print("Input:", prediction[0], "Actual Class:", rns.target_names[
        prediction[1]], "Predicted Class:", rns.target_names[
        prediction[2]])
```

```
conf_matrix = confusion_matrix(y_test, predictions)
print("In Confusion Matrix:")
print(conf_matrix)
```