EXP.NO.01

Find s algorithm program

Program:

```
def find_s_algorithm(data):
    h = ['0', '0', '0', '0', '0']
    for instance in data:
        if instance[-1] == "yes":
            j = 0
            for x in instance[:-1]:
                if x != h[j] and h[j] == '0':
                    h[j] = x
                elif x != h[j] and h[j] != '0':
                    h[j] = '?'
                j += 1
    return h
def main():
    tennis_data = [
        ['sunny', 'hot', 'high', 'FALSE', 'no'],
        ['sunny', 'hot', 'high', 'TRUE', 'no'],
        ['overcast', 'mild', 'high', 'FALSE', 'yes'],
        ['rain', 'mild', 'high', 'FALSE', 'yes'],
        ['rain', 'cool', 'normal', 'FALSE', 'yes']
    ]
    hypothesis = find_s_algorithm(tennis_data)
    print("Most specific hypothesis is:", hypothesis)
if __name__ == "__main__":
    main()
```

Output:

```
Most specific hypothesis is: ['?', '?', '?', 'FALSE', '0']
```

EXP.NO.02

Candidate-Elimination algorithm

Program:

```
def candidate_elimination(data):
    specific_h = data[0][:-1]
    general_h = [["?" for _ in range(len(specific_h))] for _ in range(len(specific_h))]
    for instance in data:
        if instance[-1] == 'T':
            for i in range(len(specific_h)):
                if instance[i] != specific_h[i]:
                    specific_h[i] = '?'
                    general_h[i][i] = '?'
        else:
            for i in range(len(specific_h)):
                if instance[i] != specific_h[i]:
                    general_h[i][i] = specific_h[i]
                else:
                    general_h[i][i] = '?'
    general_h = [h for h in general_h if h != ['?' for _ in range(len(specific_h))]]
    return specific_h, general_h
data = [
    ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'T'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'T'],
    ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'F'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'T']]
s_final, g_final = candidate_elimination(data)
print("Final Specific Hypothesis:", s_final)
print("Final General Hypothesis:", g_final)
```

Output:

Final Specific Hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final General Hypothesis: [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

Expt no : 3

Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample

Program:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
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)
clf = DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
tree_rules = export_text(clf, feature_names=iris.feature_names)
print("\nDecision Tree Structure:\n", tree_rules)
```

Output:

```
Accuracy: 1.0

Decision Tree Structure:
 |--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) >  0.80
|   |--- petal length (cm) <= 4.75
|   |   |--- petal width (cm) <= 1.60
|   |   |   |--- class: 1
|   |   |--- petal width (cm) >  1.60
|   |   |   |--- class: 2
|   |--- petal length (cm) >  4.75
|   |   |--- petal width (cm) <= 1.75
|   |   |   |--- petal length (cm) <= 4.95
|   |   |   |   |--- class: 1
|   |   |   |--- petal length (cm) >  4.95
|   |   |   |   |--- petal width (cm) <= 1.55
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- petal width (cm) >  1.55
|   |   |   |   |   |--- sepal length (cm) <= 6.95
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- sepal length (cm) >  6.95
|   |   |   |   |   |   |--- class: 2
|   |   |--- petal width (cm) >  1.75
|   |   |   |--- petal length (cm) <= 4.85
|   |   |   |   |--- sepal width (cm) <= 3.10
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- sepal width (cm) >  3.10
|   |   |   |   |   |--- class: 1
|   |   |   |--- petal length (cm) >  4.85
|   |   |   |   |--- class: 2
```

EXP.NO:05 **DATE:**

Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file and compute the accuracy with a few test data sets

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.impute import SimpleImputer

data = pd.read_csv('/content/tested.csv')

data = data.drop(['Name', 'Ticket', 'Cabin', 'Embarked'], axis=1)

imputer = SimpleImputer(strategy='median')
data['Age'] = imputer.fit_transform(data[['Age']])

data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})

data.dropna(inplace=True)

X = data.drop('Survived', axis=1)
y = data['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

model = GaussianNB()
```

```
model.fit(X_train, y_train)


y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)


print("Accuracy on Test Dataset:", accuracy)
```

OUTPUT:

Accuracy on Test Dataset: 1.0

EXP.NO:06 **DATE:**

Implement naïve Bayesian Classifier model to classify a set of documents and measure the accuracy, precision, and recall.

PROGRAM:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split


titanic_df = pd.read_csv('/content/tested.csv')
text_data = titanic_df['Name']
target = titanic_df['Survived']
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(text_data)
X_train, X_test, y_train, y_test = train_test_split(X, target, test_size=0.2,
random_state=42)


clf = MultinomialNB()
```

```
clf.fit(X_train, y_train)

predicted = clf.predict(X_test)
accuracy = accuracy_score(y_test, predicted)
precision = precision_score(y_test, predicted)
recall = recall_score(y_test, predicted)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

OUTPUT:

Accuracy: 0.8690476190476191

Precision: 1.0

Recall: 0.6764705882352942

EXP.NO:07 **DATE:**

Write a program to construct a Bayesian network to diagnose CORONA infection using standard WHO Data Set.

PROGRAM:

```
from pgmpy.models import BayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
```

```
from pgmpy.inference import VariableElimination
import pandas as pd
data = pd.read_csv('/content/Covid Live.csv')
data.columns = data.columns.str.strip().str.replace('[^\w\s]', '')

if 'CountryOther' not in data.columns:
    print("Error: 'CountryOther' variable not found in the dataset.")
elif data['CountryOther'].isnull().any():
    print("Error: 'CountryOther' variable contains missing values.")
else:
    model = BayesianNetwork([
        ('TotalCases', 'CORONA'),
        ('TotalDeaths', 'CORONA'),
        ('NewDeaths', 'CORONA'),
        ('TotalRecovered', 'CORONA'),
        ('ActiveCases', 'CORONA'),
        ('SeriousCritical', 'CORONA'),
        ('TotCases1Mpop', 'CORONA'),
        ('Deaths1Mpop', 'CORONA'),
        ('TotalTests', 'CORONA'),
        ('Tests1Mpop', 'CORONA'),
        ('Population', 'CORONA')
    ])
    try:
        model.fit(data, estimator=MaximumLikelihoodEstimator)
        inference = VariableElimination(model)
        evidence = {
            'TotalCases': 'high',
            'TotalDeaths': 'low',
            'NewDeaths': 'moderate',
```

```
        'TotalRecovered': 'high',
        'ActiveCases': 'moderate',
        'SeriousCritical': 'low',
        'TotCases1Mpop': 'moderate',
        'Deaths1Mpop': 'low',
        'TotalTests': 'high',
        'Tests1Mpop': 'moderate',
        'Population': 'high'
    }

    probability = inference.query(['CORONA'], evidence=evidence)
    print(probability)
except KeyError as e:
    print(f"KeyError: {e}. Check if the target variable 'CORONA' exists in your dataset.")
except Exception as e:
    print(f"An error occurred: {e}")
```

OUTPUT:

KeyError: 'CORONA'. Check if the target variable 'CORONA' exists in your dataset.

EXP.NO:08 **DATE:**

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using the k-Means algorithm. Compare the results of these two algorithms.

PROGRAM:

```
import pandas as pd

from sklearn.mixture import GaussianMixture

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.impute import SimpleImputer

from sklearn.pipeline import Pipeline

from sklearn.metrics import silhouette_score


titanic_data = pd.read_csv('/content/tested.csv')

numeric_columns = ['Age', 'Fare']

titanic_numeric = titanic_data[numeric_columns]


imputer = SimpleImputer(strategy='mean')

titanic_numeric_imputed = pd.DataFrame(imputer.fit_transform(titanic_numeric),
columns=numeric_columns)


scaler = StandardScaler()

titanic_scaled = scaler.fit_transform(titanic_numeric_imputed)

em_model = GaussianMixture(n_components=2)

em_clusters = em_model.fit_predict(titanic_scaled)


kmeans_model = KMeans(n_clusters=2)

kmeans_clusters = kmeans_model.fit_predict(titanic_scaled)

em_silhouette_score = silhouette_score(titanic_scaled, em_clusters)

kmeans_silhouette_score = silhouette_score(titanic_scaled, kmeans_clusters)
```

```
print("Silhouette Score for EM Algorithm:", em_silhouette_score)
print("Silhouette Score for k-Means Algorithm:", kmeans_silhouette_score)
```

OUTPUT:

Silhouette Score for EM Algorithm: 0.4607062096491872

Silhouette Score for k-Means Algorithm: 0.5933834772375091

EXP.NO:09 **DATE:**

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score

data = pd.read_csv("/content/IRIS.csv")

X = data.drop('species', axis=1)
y = data['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_scaled, y_train)

y_pred = knn.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("\nCorrect Predictions:")
correct_predictions = X_test[y_test == y_pred]
print(correct_predictions)

print("\nWrong Predictions:")
wrong_predictions = X_test[y_test != y_pred]
print(wrong_predictions)
```

OUTPUT:

```
Accuracy Score: 1.0
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        19
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00        13

       accuracy                           1.00        45
      macro avg       1.00      1.00      1.00        45
   weighted avg       1.00      1.00      1.00        45


Correct Predictions:
     sepal_length  sepal_width  petal_length  petal_width
73            6.1          2.8           4.7          1.2
18            5.7          3.8           1.7          0.3
118           7.7          2.6           6.9          2.3
```

78	6.0	2.9	4.5	1.5
76	6.8	2.8	4.8	1.4
31	5.4	3.4	1.5	0.4
64	5.6	2.9	3.6	1.3
141	6.9	3.1	5.1	2.3
68	6.2	2.2	4.5	1.5
82	5.8	2.7	3.9	1.2
110	6.5	3.2	5.1	2.0
12	4.8	3.0	1.4	0.1
36	5.5	3.5	1.3	0.2
9	4.9	3.1	1.5	0.1
19	5.1	3.8	1.5	0.3
56	6.3	3.3	4.7	1.6
104	6.5	3.0	5.8	2.2
69	5.6	2.5	3.9	1.1
55	5.7	2.8	4.5	1.3
132	6.4	2.8	5.6	2.2
29	4.7	3.2	1.6	0.2
127	6.1	3.0	4.9	1.8
26	5.0	3.4	1.6	0.4
128	6.4	2.8	5.6	2.1
131	7.9	3.8	6.4	2.0
145	6.7	3.0	5.2	2.3
108	6.7	2.5	5.8	1.8
143	6.8	3.2	5.9	2.3
45	4.8	3.0	1.4	0.3
30	4.8	3.1	1.6	0.2
22	4.6	3.6	1.0	0.2
15	5.7	4.4	1.5	0.4
65	6.7	3.1	4.4	1.4
11	4.8	3.4	1.6	0.2
42	4.4	3.2	1.3	0.2
146	6.3	2.5	5.0	1.9
51	6.4	3.2	4.5	1.5
27	5.2	3.5	1.5	0.2
4	5.0	3.6	1.4	0.2
32	5.2	4.1	1.5	0.1
142	5.8	2.7	5.1	1.9
85	6.0	3.4	4.5	1.6
86	6.7	3.1	4.7	1.5
16	5.4	3.9	1.3	0.4
10	5.4	3.7	1.5	0.2

```
Wrong Predictions:
Empty DataFrame
Columns: [sepal_length, sepal_width, petal_length, petal_width]
Index: []
```

EXP.NO:10 **DATE:**

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

PROGRAM:

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd


def lwr_predict(X_train, y_train, x, tau):

    m = X_train.shape[0]

    X_train_bias = np.c_[np.ones((m, 1)), X_train]

    x_bias = np.array([1, x])

    weights = np.exp(-np.sum((X_train - x) ** 2, axis=1) / (2 * tau ** 2))

    theta =
np.linalg.inv(X_train_bias.T.dot(np.diag(weights)).dot(X_train_bias)).dot(X_train_bias.T).do
t(np.diag(weights)).dot(y_train)

    y_pred = x_bias.dot(theta)

    return y_pred

np.random.seed(0)

X_train = np.linspace(0, 10, 100).reshape(-1, 1)

y_train = np.sin(X_train) + np.random.normal(scale=0.1, size=(100, 1))

tau_values = [0.1, 1, 10]

plt.figure(figsize=(12, 8))

plt.scatter(X_train, y_train, color='blue', label='Training data')


for tau in tau_values:

    y_pred = [lwr_predict(X_train, y_train, x, tau) for x in X_train.ravel()]

    plt.plot(X_train, y_pred, label=f'Tau = {tau}')

    print(f"\nTau = {tau}:")

    print(pd.DataFrame({'X': X_train.ravel(), 'y_pred': y_pred}))
```

plt.title('Locally Weighted Regression')

plt.xlabel('X')

plt.ylabel('y')

plt.legend()

plt.show()

OUTPUT:

```
Tau = 0.1:
            X                   y_pred
0    0.00000   [0.16026994556830715]
1    0.10101   [0.20198442238472486]
2    0.20202   [0.32157146592079755]
3    0.30303   [0.45419426437196797]
4    0.40404      [0.50722781018871]
..       ...                     ...
95   9.59596  [-0.11752635137617062]
96   9.69697  [-0.20366390755344455]
97   9.79798  [-0.28126274399987317]
98   9.89899   [-0.3978801967056853]
99  10.00000   [-0.5155943358167292]

[100 rows x 2 columns]

Tau = 1:
            X                    y_pred
0    0.00000     [0.3455046389155173]
1    0.10101     [0.3999638343266886]
2    0.20202     [0.4510030678996318]
3    0.30303     [0.4984104325881321]
4    0.40404     [0.5419783742042416]
..       ...                      ...
95   9.59596  [-0.043131817912446024]
96   9.69697    [-0.1211173078671619]
97   9.79798   [-0.20195647212777335]
98   9.89899   [-0.28541575309705713]
99  10.00000    [-0.3712682632948008]

[100 rows x 2 columns]

Tau = 10:
            X                  y_pred
0    0.00000  [0.33762811134236187]
1    0.10101   [0.3328296100928286]
2    0.20202  [0.32809322516151196]
3    0.30303   [0.3234188939093193]
4    0.40404   [0.3188065525531287]
..       ...                    ...
95   9.59596   [0.1468571809197648]
96   9.69697  [0.14752411433367182]
97   9.79798  [0.14824219074400766]
98   9.89899  [0.14901124055300566]
99  10.00000   [0.1498310930136214]
```

```
[100 rows x 2 columns]
```