PROGRAM 1

```
import csv

# Step 1: Create Weather.csv with training data
data = [
    ['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']
]

# Write to Weather.csv
with open('Weather.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)

print("✅ Weather.csv created successfully!\n")



# Step 2: Load CSV
def loadCsv(filename):
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        dataset = list(reader)
    return dataset

# Step 3: Define attributes
attributes = ['Sky','Temp','Humidity','Wind','Water','Forecast']
print("Attributes:", attributes)

# Step 4: Load dataset and extract target labels
```

```
filename = "Weather.csv"
dataset = loadCsv(filename)

# Display the dataset
print("\nDataset:")
for row in dataset:
    print(row)

# Step 5: Extract target values (last column)
target = [row[-1] for row in dataset]
print("\nTarget Labels:", target)

# Step 6: Initialize most specific hypothesis
hypothesis = ['0'] * len(attributes)
print("\nInitial Hypothesis:", hypothesis)

# Step 7: Apply FIND-S algorithm
print("\nLearning Hypothesis:")
for i in range(len(dataset)):
    if target[i] == 'Yes':
        for j in range(len(attributes)):
            if hypothesis[j] == '0':
                hypothesis[j] = dataset[i][j]
            elif hypothesis[j] != dataset[i][j]:
                hypothesis[j] = '?'
        print(f"Instance {i+1} → {hypothesis}")

# Step 8: Final Hypothesis
print("\n✅ Final Hypothesis:")
print(hypothesis)
```

OUTPUT

```
Weather.csv created successfully!

Attributes: ['Sky', 'Temp', 'Humidity', 'Wind', 'Water', 'Forecast']

Dataset:
['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']

Target Labels: ['Yes', 'Yes', 'No', 'Yes']

Initial Hypothesis: ['0', '0', '0', '0', '0', '0']

Learning Hypothesis:
Instance 1 → ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
Instance 2 → ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
Instance 4 → ['Sunny', 'Warm', '?', 'Strong', '?', '?']

✅ Final Hypothesis:
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

2.CANDIDATE ELIMINATION

```
import pandas as pd

import numpy as np

import csv


# Step 1: Create Training_examples.csv directly

data = [

   ['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']

]


# Column names

columns = ['Sky', 'AirTemp', 'Humidity', 'Wind', 'Water', 'Forecast', 'EnjoySport']


# Write the data to a CSV file
```

```
with open('Training_examples.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(columns)
    writer.writerows(data)

print("✅ Training_examples.csv created successfully!\n")

# Step 2: Load the data
dataset = pd.read_csv('Training_examples.csv')
print("📄 Dataset:\n", dataset, "\n")

# Step 3: Separate features and target
concepts = np.array(dataset.iloc[:, :-1])
target = np.array(dataset.iloc[:, -1])

# Step 4: Candidate Elimination Algorithm
def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [["?" for _ in range(len(specific_h))] for _ in range(len(specific_h))]

    print("🔍 Initial Specific Hypothesis:", specific_h)
    print("🔍 Initial General Hypothesis:", general_h, "\n")

    for i, instance in enumerate(concepts):
        if target[i] == 'Yes':
            for j in range(len(specific_h)):
                if instance[j] != specific_h[j]:
                    specific_h[j] = '?'
                    general_h[j][j] = '?'
        elif target[i] == 'No':
            for j in range(len(specific_h)):
```

```
            if instance[j] != specific_h[j]:
                general_h[j][j] = specific_h[j]
            else:
                general_h[j][j] = '?'

        print(f"Step {i+1}:")
        print("Specific Hypothesis:", specific_h)
        print("General Hypothesis:", general_h, "\n")

    # Remove all rows in general_h that are all '?'
    general_h = [gh for gh in general_h if gh != ['?' for _ in range(len(specific_h))]]
    return specific_h, general_h

# Step 5: Run learning algorithm
s_final, g_final = learn(concepts, target)

# Step 6: Output final results
print("✅ Final Specific Hypothesis:\n", s_final)
print("✅ Final General Hypothesis:\n", g_final)
```

OUTPUT

```
✅ Training_examples.csv created successfully!

📄 Dataset:
     Sky AirTemp Humidity    Wind Water Forecast EnjoySport
0  Sunny    Warm   Normal  Strong  Warm     Same        Yes
1  Sunny    Warm     High  Strong  Warm     Same        Yes
2  Rainy    Cold     High  Strong  Warm   Change         No
3  Sunny    Warm     High  Strong  Cool   Change        Yes

🔍 Initial Specific Hypothesis: ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
🔍 Initial General Hypothesis: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 1:
Specific Hypothesis: ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
General Hypothesis: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

Step 2:
Specific Hypothesis: ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
General Hypothesis: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?
', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 3:
Specific Hypothesis: ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
General Hypothesis: [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Step 4:
Specific Hypothesis: ['Sunny' 'Warm' '?' 'Strong' '?' '?']
General Hypothesis: [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

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

3. ID3 DESION TREE

```
import pandas as pd

import numpy as np

from io import StringIO


# Include dataset directly in the code

data = """

outlook,temperature,humidity,wind,class

0,0,0,0,0

0,0,0,1,0

1,0,0,0,1

2,1,0,0,1

2,2,1,0,1

2,2,1,1,0

1,2,1,1,1
```

```
0,1,0,0,0
0,2,1,0,1
2,1,1,0,1
0,1,1,1,1
1,1,0,1,1
1,0,1,0,1
2,1,0,1,0
0,1,1,1,1
1,1,1,1,1
"""

# Read the CSV data from the string
dataset = pd.read_csv(StringIO(data))

# Entropy calculation
def entropy(target_col):
    elements, counts = np.unique(target_col, return_counts=True)
    return np.sum([(-counts[i]/np.sum(counts)) *
np.log2(counts[i]/np.sum(counts)) for i in range(len(elements))])

# Information Gain calculation
def InfoGain(data, split_attribute_name, target_name="class"):
    total_entropy = entropy(data[target_name])
```

```
    vals, counts = np.unique(data[split_attribute_name],
return_counts=True)
    weighted_entropy = np.sum([
        (counts[i]/np.sum(counts)) *
entropy(data[data[split_attribute_name] == vals[i]][target_name])
        for i in range(len(vals))
    ])
    return total_entropy - weighted_entropy


# ID3 Algorithm
def ID3(data, originaldata, features, target_attribute_name="class",
parent_node_class=None):
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]
    elif len(data) == 0:
        return
np.unique(originaldata[target_attribute_name])[np.argmax(
            np.unique(originaldata[target_attribute_name],
return_counts=True)[1])]
    elif len(features) == 0:
        return parent_node_class
    else:
        parent_node_class =
np.unique(data[target_attribute_name])[np.argmax(
            np.unique(data[target_attribute_name],
return_counts=True)[1])]
```

```
        item_values = [InfoGain(data, feature, target_attribute_name)
for feature in features]
        best_feature_index = np.argmax(item_values)
        best_feature = features[best_feature_index]
        tree = {best_feature: {}}
        features = [i for i in features if i != best_feature]
        for value in np.unique(data[best_feature]):
            sub_data = data[data[best_feature] == value]
            subtree = ID3(sub_data, data, features,
target_attribute_name, parent_node_class)
            tree[best_feature][value] = subtree
        return tree


# Predict function
def predict(query, tree, default=1):
    for key in query.keys():
        if key in tree:
            try:
                result = tree[key][query[key]]
            except:
                return default
            if isinstance(result, dict):
                return predict(query, result)
            else:
```

```
            return result
    return default

# Simple train-test split
def train_test_split(dataset):
    training_data = dataset.iloc[:14].reset_index(drop=True)
    return training_data

# Test function
def test(data, tree):
    queries = data.iloc[:, :-1].to_dict(orient="records")
    predicted = pd.DataFrame(columns=["predicted"])
    for i in range(len(data)):
        predicted.loc[i, "predicted"] = predict(queries[i], tree, 1.0)
    accuracy = np.sum(predicted["predicted"] == data["class"]) /
len(data)
    print("Prediction accuracy: {:.2f}%".format(accuracy * 100))

# Train the model
training_data = train_test_split(dataset)
features = training_data.columns[:-1]
tree = ID3(training_data, training_data, features)
print("Decision Tree:\n", tree)
```

```
# Test the model on training data
test(training_data, tree)


# Predict on a new sample
sample_query = {'outlook': 2, 'temperature': 1, 'humidity': 0, 'wind': 0}
prediction = predict(sample_query, tree)
print("Prediction for sample {} is: {}".format(sample_query, prediction))
```

OUTPUT

```
Decision Tree:
 {'outlook': {0: {'humidity': {0: 0, 1: 1}}, 1: 1, 2: {'wind': {0: 1, 1: 0}}}}
Prediction accuracy: 100.00%
Prediction for sample {'outlook': 2, 'temperature': 1, 'humidity': 0, 'wind': 0} is: 1
```

4. ANN BACK PROPAGATION ALGORITHM

```
from math import exp
from random import seed, random


# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{'weights': [random() for _ in range(n_inputs + 1)]} for _ in range(n_hidden)]
```

```
    network.append(hidden_layer)
    output_layer = [{'weights': [random() for _ in range(n_hidden + 1)]} for _ in range(n_outputs)]
    network.append(output_layer)
    return network

# Calculate neuron activation for an input
def activate(weights, inputs):
    activation = weights[-1]  # bias
    for i in range(len(weights) - 1):
        activation += weights[i] * inputs[i]
    return activation

# Transfer neuron activation
def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
```

```
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs

# Calculate the derivative of a neuron's output
def transfer_derivative(output):
    return output * (1.0 - output)

# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = []
        if i != len(network) - 1:
            for j in range(len(layer)):
                error = sum([neuron['weights'][j] * neuron['delta'] for
neuron in network[i + 1]])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)):
```

```
        neuron = layer[j]
        neuron['delta'] = errors[j] *
transfer_derivative(neuron['output'])

# Update network weights with error
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']

# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for _ in range(n_outputs)]
            expected[row[-1]] = 1
```

```
            sum_error += sum([(expected[i] - outputs[i]) ** 2 for i in
range(len(expected))])
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)
        print(f'>epoch={epoch}, lrate={l_rate:.3f}, error={sum_error:.3f}')

# Test training backprop algorithm
seed(1)
dataset = [
    [2.7810836, 2.550537003, 0],
    [1.465489372, 2.362125076, 0],
    [3.396561688, 4.400293529, 0],
    [1.38807019, 1.850220317, 0],
    [3.06407232, 3.005305973, 0],
    [7.627531214, 2.759262235, 1],
    [5.332441248, 2.088626775, 1],
    [6.922596716, 1.77106367, 1],
    [8.675418651, -0.242068655, 1],
    [7.673756466, 3.508563011, 1]
]

n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, 2, n_outputs)
```

```
train_network(network, dataset, l_rate=0.5, n_epoch=20,
n_outputs=n_outputs)


# Show the final network
print("\nFinal network structure with weights and outputs:")
for i, layer in enumerate(network):
    print(f"Layer {i+1}:")
    for neuron in layer:
        print(neuron)
```

OUTPUT

```
>epoch=0, lrate=0.500, error=6.350
>epoch=1, lrate=0.500, error=5.531
>epoch=2, lrate=0.500, error=5.221
>epoch=3, lrate=0.500, error=4.951
>epoch=4, lrate=0.500, error=4.519
>epoch=5, lrate=0.500, error=4.173
>epoch=6, lrate=0.500, error=3.835
>epoch=7, lrate=0.500, error=3.506
>epoch=8, lrate=0.500, error=3.192
>epoch=9, lrate=0.500, error=2.898
>epoch=10, lrate=0.500, error=2.626
>epoch=11, lrate=0.500, error=2.377
>epoch=12, lrate=0.500, error=2.153
>epoch=13, lrate=0.500, error=1.953
>epoch=14, lrate=0.500, error=1.774
>epoch=15, lrate=0.500, error=1.614
>epoch=16, lrate=0.500, error=1.472
>epoch=17, lrate=0.500, error=1.346
>epoch=18, lrate=0.500, error=1.233
>epoch=19, lrate=0.500, error=1.132

Final network structure with weights and outputs:
Layer 1:
{'weights': [-1.4688375095432327, 1.850887325439514, 1.0858178629550297], 'output': 0.029980305604426185, 'delta': -0.0059546604162323625}
{'weights': [0.37711098142462157, -0.0625909894552989, 0.2765123702642716], 'output': 0.9456229000211323, 'delta': 0.0026279652850863837}
Layer 2:
{'weights': [2.515394649397849, -0.3391927502445985, -0.9671565426390275], 'output': 0.23648794202357587, 'delta': -0.04270059278364587}
{'weights': [-2.5584149848484263, 1.0036422106209202, 0.42383086467582715], 'output': 0.7790535202438367, 'delta': 0.03803132596437354}
```

6.program

```
# 1. Importing necessary libraries
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer,
TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn import metrics
import numpy as np

# 2. Loading the training data
twenty_train = fetch_20newsgroups(subset='train', shuffle=True)
print("Number of training documents:", len(twenty_train.data))
print("Categories:", twenty_train.target_names)

# 3. Creating the Naive Bayes Pipeline
text_clf = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', MultinomialNB())
])

# 4. Training the model
text_clf.fit(twenty_train.data, twenty_train.target)
```

```
# 5. Loading test data
twenty_test = fetch_20newsgroups(subset='test', shuffle=True)

# 6. Predicting the categories
predicted = text_clf.predict(twenty_test.data)

# 7. Evaluation: Accuracy, Precision, Recall
accuracy = np.mean(predicted == twenty_test.target)
print(f"\nAccuracy: {accuracy:.4f}")
print("Classification Report:\n")
print(metrics.classification_report(twenty_test.target, predicted,
target_names=twenty_test.target_names))

# Optionally: Print Confusion Matrix
# import matplotlib.pyplot as plt
# import seaborn as sns
# cm = metrics.confusion_matrix(twenty_test.target, predicted)
# plt.figure(figsize=(12,10))
# sns.heatmap(cm, annot=True, fmt='d',
xticklabels=twenty_test.target_names,
yticklabels=twenty_test.target_names, cmap='Blues')
# plt.xlabel('Predicted')
# plt.ylabel('True')
# plt.title('Confusion Matrix')
# plt.show()
```

OUTPUT

Number of training documents: 11314
Categories: ['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']

Accuracy: 0.7739
Classification Report:

	precision	recall	f1-score	support
alt.atheism	0.80	0.52	0.63	319
comp.graphics	0.81	0.65	0.72	389
comp.os.ms-windows.misc	0.82	0.65	0.73	394
comp.sys.ibm.pc.hardware	0.67	0.78	0.72	392
comp.sys.mac.hardware	0.86	0.77	0.81	385
comp.windows.x	0.89	0.75	0.82	395
misc.forsale	0.93	0.69	0.80	390
rec.autos	0.85	0.92	0.88	396
rec.motorcycles	0.94	0.93	0.93	398
rec.sport.baseball	0.92	0.90	0.91	397
rec.sport.hockey	0.89	0.97	0.93	399
sci.crypt	0.59	0.97	0.74	396
sci.electronics	0.84	0.60	0.70	393
sci.med	0.92	0.74	0.82	396
sci.space	0.84	0.89	0.87	394
soc.religion.christian	0.44	0.98	0.61	398
talk.politics.guns	0.64	0.94	0.76	364
talk.politics.mideast	0.93	0.91	0.92	376
talk.politics.misc	0.96	0.42	0.58	310
talk.religion.misc	0.97	0.14	0.24	251
accuracy			0.77	7532
macro avg	0.83	0.76	0.76	7532
weighted avg	0.82	0.77	0.77	7532