POORNIMA

EXPERIMENT -1

Aim: Implement and demonstrate the FIND-S algorithm for finding hypothesis based on a given set of training data samples. Read the data from a .csv file.

FIND-S Algorithm :-

- Load dataset
- Initialize h to the most specific hypothesis H.
- For each positive training instance x :- (For each attribute constraint a_i in h)
  - If the constraint a_i in h is satisfied by x, do nothing.
  - else replace a_i in h by a more general constraint.
- Output hypothesis h.

Source Code :-

```
import pandas as pd
import numpy as np

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

Page No.....

Output ⇒

	Time	Weather	Temperature	Humidity	Wind	Plays
0	Morning	Sunny	Warm	Mild	Normal	Yes
1	Morning	Rainy	Cold	Mild	Strong	No
2	Morning	Sunny	Moderate	High	Normal	Yes
3	Evening	Sunny	Cold	Mild	Normal	Yes

The final hypothesis : ['?' 'Sunny' '?' '?' 'Normal']

```
for i, val in enumerate(c):
    if t[i] == "Yes":
        for x in range(len(specific_hypothesis)):
            if val[x] != specific_hypothesis[x]:
                specific_hypothesis[x] = '?'
            else:
                pass
    return specific_hypothesis

print("The final hypothesis is :", train(d, target))
```

POORNIMA

EXPERIMENT-2

Aim: For a given set of training data examples stored in a .csv file, implement and demonstrate the candidate elimination algorithm to output a description of the set of all hypothesis consistent with examples.

candidate-Elimination Algorithm:-

- Load dataset
- Initialize general and specific hypothesis-
- For each training example, if example is positive:
  - if attribute value == hypothesis value:
    - do nothing
  - else:
    - replace attribute_value with '?'
- if example is negative:
  - make general hypothesis more specific

Source code:

```
import pandas as pd
import numpy as np
data = pd.read_csv("data.csv")
print(data)
d = np.array(data)[:, :-1]
target = np.array(data)[:, -1]

def train(c, t):
    for i, val in enumerate(t):
```

Output ⇒

	Time	weather	Temperature	Humidity	Wind	Plays
0	Morning	Sunny	Warm	Mild	Normal	Yes
1	Morning	Rainy	Cold	Mild	Strong	No
2	morning	Sunny	Moderate	High	Normal	Yes
3	Evening	Sunny	Cold	High	Normal	Yes

The final specific hypothesis is : ['?' 'Sunny' '?' '?' 'Normal']

The final general hypothesis is : [['?' 'Sunny' '?' '?' '?'],['?' '?' '?'

POORNIMA

```
    if val == "Yes":
        specific_hypothesis = c[i].copy()
        break
    z = len(specific_hypothesis)
    general_hypothesis = [['?' for i in range(z)] for i in range(z)]

for i, val in enumerate(c):
    if t[i] == "Yes":
        for x in range(z):
            if val[x] != specific_hypothesis[x]:
                specific_hypothesis[x] = '?'
                general_hypothesis[x][x] = '?'
    if t[i] == "No":
        for x in range(z):
            if val[x] != specific_hypothesis[x]:
                general_hypothesis[x][x] = specific_hypothesis[x]
            else:
                general_hypothesis[x][x] = '?'
indices = [i for i, val in enumerate(general_hypothesis) if val == ['?', '?', '?', '?', '?']]
for i in indices:
    general_hypothesis.remove(['?', '?', '?', '?', '?'])
print("The final specific hypothesis is:", specific_hypothesis)
print("The final general hypothesis is:", general_hypothesis)

train(d, target)
```

EXPERIMENT-3

Aim : Write a program to demonstrate the working of the decision tree based ID3 Algorithm. Use an appropriate dataset for building the Decision tree and apply the knowledge to classify a new sample.

Source Code:

```
import pandas as pd, math, numpy as np.
data = pd.read_csv("data.csv")
features = [feat for feat in data]
features.remove("answer")
print(data)

class Node:
    def __init__(self):
        self.children=[]
        self.value = ""
        self.isleaf = false
        self.pred = ""
def entropy(examples):
    pos = 0.0
    neg=0.0
    for _,row in examples.iterrows():
        if row["answer"] = "yes":
            pos+=1
        else:
            neg +=1
```

Output:

	outlook	temperature	humidity	wind	answer
0	sunny	hot	high	weak	no
1	sunny	hot	high	strong	no
2	overcast	hot	high	weak	yes
3	rain	mild	high	weak	yes
4	rain	cool	normal	weak	yes
5	rain	cool	normal	strong	no
6	overcast	cool	normal	strong	yes
7	sunny	mild	high	weak	no
8	sunny	cool	normal	weak	yes
9	rain	mild	normal	weak	yes
10	sunny	mild	normal	strong	yes
11	overcast	mild	high	strong	yes
12	overcast	hot	normal	weak	yes
13	rain	mild	high	strong	no

POORNIMA

```
    if pos==0.0 or neg==0.0:
        return 0.0
    else:
        p = pos/(pos+neg)
        n = neg/(pos+neg)
        return -(p*math.log(p,2) + n*math.log(n,2))
def info_gain(examples, attrs):
    uniq = np.unique(examples[attrs])
    gain = entropy(examples)
    for u in uniq:
        subdata = examples[examples[attrs]==u]
        sub_e = entropy(subdata)
        gain -= (float(len(subdata))/float(len(examples)))*sub_e
    return gain

def ID3(examples, attrs):
    root = Node()
    max_gain = 0
    max_feat = ""
    for feature in attrs:
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    uniq = np.unique(example[max_feat])
```

Decision tree is:

outlook

overcast → ['yes']

rain

wind

strong → ['no']

weak → ['yes']

sunny

humidity

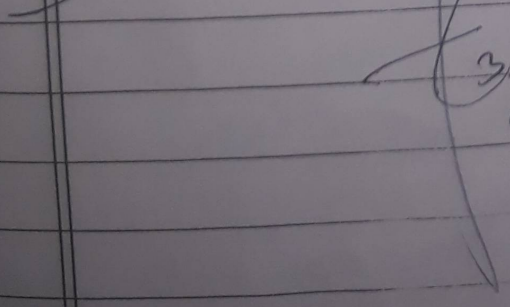
high → ['no']

normal → ['yes']

edicted Label for new example {'outlook': 'sunny', 'temperature': 'hot', 'humidity': 'normal', 'wind': 'strong'} is ['yes']

```
for u in uniq:
    subdata = examples[examples[max_feat] == u]
    if entropy(subdata) == 0.0:
        newNode = Node()
        newNode().isLeaf = True
        newNode.value = u
        newNode.pred = np.unique(subdata["answer"])
        root.children.append(newNode)
    else:
        dummyNode = Node()
        dummyNode.value = u
        new_attrs = attrs.copy()
        new_attrs.remove(max_feat)
        child = ID3(subdata, new_attrs)
        dummyNode.children.append(child)
        root.children.append(dummyNode)
return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print("->", root.pred)
    print()
```

```
for child in root.children
    printTree(child, depth+1)
def classify(root : Node, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isLeaf:
                print("Predicted Label for new example", new, "is:", child.pred
                exit
            else:
                classify(child.children[0], new)

root = ID3(data, features)
print("Decision tree is:")
printTree(root)

new = {"outlook": "sunny", "temperature": "hot", "humidity": "normal", "wind":
                                                              "strong"}

classify(root, new)
```



	Age	Income	Student	Credit	Buy
0	<30	High	No	Fair	No
1	<30	High	No	Excellent	No
2	31-40	High	No	Fair	Yes
3	>40	Medium	No	Fair	Yes
4	>40	Low	Yes	Fair	Yes
5	>40	Low	Yes	Excellent	No
6	31-40	Low	Yes	Excellent	Yes
7	<30	Medium	No	Fair	No
8	<30	Low	Yes	Fair	Yes
9	>40	Medium	Yes	Fair	Yes
10	<30	Medium	Yes	Excellent	Yes
11	31-40	Medium	No	Excellent	Yes
12	31-40	High	Yes	Fair	Yes
13	>40	Medium	No	Excellent	No

Aim :-

Progra

EXPERIMENT - 4

Aim :- Write a program to implement the ~~naive~~ naive Bayesian classifier for a sample training data set stored as a .csv file. Compute the accuracy of the classifier, considering few data sets.

Program :-

```
import numpy as np, pandas as pd
data = pd.read_csv("data.csv")
print(data)
X = data.drop('Buy', axis = 1).values
y = data['Buy'].values
Xn = X[y == 'No']
Xy = X[y == 'Yes']
pn = len(Xn) / len(X)
py = len(Xy) / len(X)

def cal_prob(arr):
    count = {}
    for row in arr:
        for feat in row:
            if feat not in count:
                count[feat] = 1
            else:
                count[feat] = +1
    for key, values in count.items():
        count[key] = value / len(arr)
    return count
```

```
def naive_bayes(instance):
    prob_n = pn
    prob_y = py
    for i in instances
        prob_y* = 0 if i not in list(psy.keys()) else psy[i]
        prob_n* = 0 if i not in list(psy.keys()) else psy[i]
    if prob_y > prob_n:
        return "Yes"
    else:
        return "No"

psn = cal_prob(Xn)
psy = cal_prob(Xy)
i = ['<30', 'Low', 'No', 'Excellent']
print(i, naive_bayes(i))
test_x = X[1:12]
test_y = y[1:12]
c = 0
for i in range(len(test_x)):
    if naive_bayes(test_x[i]) == test_y[i]:
        c += 1
print("Accuracy is :", (c/len(test_x)) * 100, "%")
```

['<30', 'Low', 'No', 'Excellent'] No

Accuracy is : 90.90909090909091 %