Date. 29/9/20.....

Expt. No.....1..... Page No.....1.....

1. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data sample. Read the training data from csv file.

```
import random
import csv
attributes = [['sunny','rainy'],['warm','cold'],['normal',
    'high'],['strong','weak'],['warm','cool'],['same',
        'change']]
print(attributes)
num_attributes = len(attributes)
print(num_attributes)
print("\n the most general hypothesis : ['?','?','?',
        '?','?','?']\n")
print("\n the most specific hypothesis : ['0','0','0',
        '0','0','0']\n")
a=[]
print("\n the given training dataset\n")
with open('C:\Users\Admin\Desktop\ML\lab1\data.csv',
        'r') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        a.append(row)
        print(row)
print("\n the initial value of hypothesis:")
h = ['0']*num_attributes
print(h)
for j in range(0,num_attributes):
```

Teacher's Signature : .....

Date.....

Expt. No.....1..... Page No.....2.....

```
    b[j] = a[i][j]
for i in range(1, len(a)):
    if (a[i][num_attributes] == 'Yes'):
        for j in range(num_attributes):
            if(h[j] == '0' or h[j] == a[i][j]):
                h[j] = a[i][j]
            else:
                h[j] = '?'
print("\n for training examples : {0} the
    hypothesis") format(i+1), h)
```

Teacher's Signature : .....

Date.....

Expt. No..... Page No.....

Dataset:

Sl. No	sky	AirTemp	Humidity	wind	water	forecast	enjoy sport
1	Sunny	warm	normal	strong	warm	same	Yes
2	Sunny	warm	high	strong	warm	same	Yes
3	Rainy	cold	high	strong	warm	change	No
4	Sunny	warm	high	strong	cool	change	Yes

output:-

[['Sunny','Rainy'],['warm','cold'],['normal','high'],['strong','weak'],['warm','cool'],['same','change']]

b.

the most general hypothesis: ['?','?','?','?','?','?']

the most specific hypothesis: ['0','0','0','0','0','0']

The given training dataset

['Sky','AirTemp','Humidity','wind','water','forecast','EnjoySport']

['Sunny','warm','normal','strong','warm','same','Yes']

['Sunny','warm','high','strong','warm','same','Yes']

['Rainy','cold','High','strong','warm','change','Yes']

The initial hypothesis is:

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

Date.....

Expt. No.....

Page No.....

for training example : {0} the hypothesis =

['sunny','warm','?','strong','?','?']

Date.06.10.2020....

Expt. No.....2.....

Page No.....3.....

2. for a given set of set of training data example stored in a .csv file, implement and demonstrate the candidate-elimination algorithm to output a description of the set of all hypothesis consistent with the training example.

```
import csv
with open('c:\\Users\\Admin\\Desktop\\MLBDA\\all.csv')
as f:
csv_file = csv.reader(f)
data = list(csv_file)
print(data)
s= data[1][:-1]
g= [['?' for i in range(len(s))] for j in range(
    len(s))]
for i in data:
    if i[-1]=="Yes":
        for j in range(len(s)):
            if i[j]!= s[j]:
                s[j] = '?'
                g[j][j]='?'
    elif i[-1]=="No":
        for j in range(len(s)):
            if i[j]!= s[j]:
                g[j][j] = s[j]
            else:
                g[j][j] = "?"
print("\n steps of candidate elimination algorithm", data.index(i)+1)
```

Teacher's Signature : .....

Date.....

Expt. No.....2.....

Page No.....4.....

```
Print(s)
print(g)
gh=[]
for i in g:
    for j in i:
        if 'j'!='?':
            gb.append(i)
            break
printf("\n Final specific hypothesis:\n",s)
print("\n Final general hypothesis:\n",gh)
```

Teacher's Signature : .....

Date.....

Expt. No..... Page No.....

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	warm	change	Yes

Output:

steps of candidate elimination algorithm 1

['Sunny', 'warm', '?', 'strong', '?', '?']

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

steps of candidate elimination algorithm 2.

['sunny', 'warm', '?', 'strong', '?', '?']

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

steps of candidate elimination algorithm 3.

['Sunny', 'warm', '?', 'strong', '?', '?'].

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Date.....

Expt. No.....

Page No.....

steps of candidate elimination algorithm :

['sunny', 'warm', '?', 'strong', '?', '?']

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']],

final specific hypothesis

['sunny', 'warm', '?', 'strong', '?', '?']

final general hypothesis:

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

Date. 27/10/2020....

Expt. No.....3..... Page No.....5.....

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

```
import pandas as pd.
from pandas import Dataframe.
df_tennis = pd.read_csv('C:\Users\Admin\Desktop\
            Albin\playtennis.csv')
attribute_names = list(df_tennis.columns)
attribute_names.remove('playTennis').
print(attribute_names).
def entropy_of_list(lst):
    from collections import counter.
    count = counter(x for x in lst)
    num_instances = len(lst)*1.
    probs = [x/num_instances for x in count.values()]
    return entropy(probs).
def entropy(probs):
    import math
    return sum([-prob*math.log(prob,2) for
        prob in probs]).
Total_entropy = entropy_of_list(df_tennis['
playTennis'])
def information_gain(df, split_attribute_name,
target_attribute_name, trace=0):
    df_split = df.groupby(split_attribute_name).
    nobs = len(df.index)*1.
```

Teacher's Signature : .....

Date.....

Expt. No.....3..... Page No.....6.....

```
df_agg_ent = df_split_agg({target_attribute_name:
        [entropy_of_list, lambda x: len(x)/nobs(4)
df_agg_ent.columns = ['Entropy','prop observations']
new_entropy = sum(df_agg_ent['entropy'] * df_
                agg_ent['propobservations'])
old_entropy = entropy_of_list(df[target_
                attribute_name])
print('split_attribute_name,' Eq :', old_entropy_new
        entropy)
return old_entropy_new_entropy
def id3(df, target_attribute_name, attribute_
names, default_class = None):
from collections import Counter
count = Counter(x for x in df[target_attribut_
name])
if len(count)==1:
return next(iter(count))
elif df_empty or (not attribute_names):
    return default_class
else:
    default_class = max(count.keys()).
    gains = [information_gain(df, attr, target_
    attribute_name) for attr in attribute_
    names]
    print(i)
    index_of_max = gains.index(max(gain))
    best_attr = attributes.names[index_of_max]
    tree = {best_attr:{}}
remaining_attribute_names=[i for i in attribute_
```

Teacher's Signature : .....

Date.....

Expt. No.....3..... Page No.....7.....

```
name if i != best_attr]
for attr_val, data_subset in df.groupby(best_attr):
    subtree = ID3(data_subset, target_attribute_name, remaining_attribute_names, default_class)
    tree[best_attr][attr_val] = subtree

return

from pprint import pprint
tree = ID3(df_tennis, 'playTennis', attribute_names)
print("\n\n The resultant decision tree is:\n\n")
pprint(tree)
```

Teacher's Signature : .....

Date.....

Page No.....

Expt. No.....

output

['outlook', 'temperature', 'humidity', 'wind'].

outlook IG = 0.246749819774439

Temperature IG = 0.029225656598546 49.

Humidity IG : 0.151835501362341 36.

Wind IG : 0.048127030408269 27.

Temperature IG : 0.019273094021974 89.

Humidity IG : 0.019973094021974 89.

Wind IG : 0.92095059445466 86.

Temperature IG : 0.57095059445466 86.

Humidity IG : 0.97095059445466 86

Wind IG : 0.019973094021974 89.

The result of decision tree is

{'outlook': {'overcast': 'yes',

'rain': {'wind': {'strong': 'No', 'weak': 'yes'}},

'sunny': {'Humidity': {'High': 'No', 'Normal': 'yes'}}}

Date..02/12/2020...

Expt. No.....4.....

Page No.....8.....

4 Build an Artificial neural network by implementing the Backpropagation algorithm & test the same using appropriate data sets.

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

def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range(n_hidden)]
    network.append(hidden_layer)
    output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(n_outputs)]
    network.append(output_layer)
    return network

def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation

def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))
```

Teacher's Signature : .....

Date.....

Expt. No.....4..... Page No.....9.....

```
def forward_propegate(network, row):
    input = row
    for layers 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

def transfer_derivative(output):
    return output * (1.0 - output)
def backward_propagate_error(network,
expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i+1]:
                error += (neuron['weights'][j]*
                neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
            neuron = layer[j]
```

Teacher's Signature : .....

```
        errors.append(expected[j] - neuron['output'])
    for j in range(len(layer)):
        neuron = layer[j]
        neuron['delta'] = errors[j] * transfer_
                derivative(neuron['output'])

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']

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 i in range(n_
                outputs)]
            expected[row[-1]] = 1
```

Date.....

Expt. No.....4..... Page No.....11.....

```
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('> epoch=%d, lrate=%.3f, error=%.3f'
 % (epoch, l_rate, sum_error))
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, 0.5, 20,
n_outputs)
for layer in network:
    print(layer)
```

Teacher's Signature : .....

Date.....

Expt. No.....

Page No.....

epoch = 0 lrate = 0.500, error = 6.350.
epoch = 1, lrate = 0.500, error = 5.551
epoch = 2 lrate = 0.500 error = 5.041,
epoch = 3. lrate = 0.500 error = 4.595)
epoch = 4 lrate = 0.500 error 4.199,
epoch = 5. lrate = 0.500 error . 4.173
epoch = 6. lrate = 0500 error = 3.835.
epoch = 7 lrate = 0.500 error = 3.506
epoch = 8 lrate = 0.500 error = 3.198,
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 21.472.
epoch = 17. lrate = 0.500 error = 1.360.
epoch = 18. lrate = 0.500 error. = 1.283
epoch = 19. lrate = 0.500 error = 1.152

[{'weights': [-1.4688375095432307, 1.8508873254
39314, 1.0858178829550294], 'output':
0.029980305604426185, 'delta': -0.0995
66041623236e-}.
{'weights': [0.3791109814846157, -0.0625909894
552989, 0.2765123704264716], 'output':
0.9456022900211823, 'delta': 0.0026279052
850833837}].

Date.....

Page No.....

Expt. No.....

[{'weights': [2.515839464939849, -0.339190750044
5988, -0.96715654263902 75], 'output'
0.23648794202357588, 'delta': -
-0.04270059278364588 73,
{'weights': [-2.6589149848 4263, 1.0036422106 2
0920, 0.42383086467582 715]'
output': 0.77905352024 38867,
'delta': 0.83803132564643 73548}]

Date.22/12/2020...

Expt. No....5.....

Page No.....12.....

5. Write a program to implement the naive bayesian classifier for a sample training data stored as .csv file. Compute the accuracy of the classifier. Considering few test datasets.

```
import csv, random, math
import statistics as st

def loadCSV(filename):
    lines = csv.reader(open(filename, "r"));
dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset

def SplitDataset(dataset, splitRatio):
    testsize = int(len(dataset) * splitRatio):
    trainset = list(dataset):
    testset = []
    while len(testset) < testsize:
        index = random.randrange(len(trainset):
        testset.append(trainset.pop(index))
    return [trainset, testset]

def separatedByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        x = dataset[i]
        if (x[-1] not in separated):
```

Teacher's Signature : .....

```
        separated[x[-1]] = []
        separated[x[-1]].append(x)
    return separated.

def compute_mean_std(dataset):
    mean_std = [(st_mean(attribute), st_stdev(attribute))
                for attribute in zip(*dataset)]
    del mean_std[-1]
    return mean_std

def summarizeByclass(dataset):
    separated = separateBydass(dataset):
    summary = {}
    for classvalue, instances in separated.items():
        summary[classValue] = compute_mean_std(instances)
    return summary.

def estimatedprobability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1/math.sqrt(2*math.pi)*stdev))*exponent.

def calculateclassprobabilities(summaries, inputvector):
    p = {}
    for classvalue, classsummaries in summaries.items():
```

```
P[classvalues] = 1
    for i in range(len((class summaries)):
        mean, stdev = class summaries[i].
        x = testvector[i].
        P[classvalue, x] = estimateProbability(x,
                mean, stdev):
    return p.
def predict( summaries, testvector):
    all_p = calculateClassProbabilities( summaries,
        testvector).
    bestlabel, bestprob = None, -1.
    for lbl, p in all_p.items():
        bestprob = p
        bestlabel = lbl.
    return bestlabel.
def performs_classification (summaries, testset):
    predictions = []
    for i in range(len(testset)):
        results = predict( summaries, testset[i])
    predictions.append(result).
    return predictions.

def getAccuracy (testset, predictions):
    correct = 0.
    for i in range (len (testset)):
If testset[i][-1] == predictions[i]:
    correct += 1.
return ((correct/float(len(testset))) * 100.0.
```

Teacher's Signature : .....

Date.....

Expt. No.....5..... Page No.....15.....

```
dataset = loadcsv('c:\\users\\Admin\\Desktop\\AIML\\
          diabetes.csv')
print('Pima Indians Diabetes Dataset loaded...!')
print('Total instances available :', len(dataset))
print('Total attributes present :', len(dataset
    [0])-1)
print('First five instances of dataset:')
for i in range(5):
        print(i+1, ':', dataset[i])
splitRatio = 0.2
trainingset, Testset = splitDataset(dataset,
splitRatio)
print('\nDataset is split into training and
test set.')
print('Training example = {0}\nTesting examples
    = {1}'.format(len(trainingset), len(testset)))
summaries = summarizeByclass(trainingset)
prediction = performclassification(summaries,
testset)
accuracy = getAccuracy(testset, predictions)

print('\nAccuracy of the Naive Bayesian
    classifier is :', accuracy)
```

Teacher's Signature : .....

Date.....

Expt. No.....

Page No.....

Output -

Pima Indians diabetes dataset loaded...

Total instances available 768.

Total attributes present = 8.

first five instances of dataset

1: [6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 7.0]

2: [1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0, 0.0]

3: [8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0, 7.0]

4. [1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0, 0.0]

5: [0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0, 1.0]

dataset is split into training and testing set

Training example - 615.

testing example = 153.

Accuracy of the Naive Bayslan classifier is 76.47058823529412.

Date. 22/12/20.....

Expt. No.....6.....

Page No.....16.....

Assuming a set of documents that need to be classified use the naive bayesian classifier model to perform this task. Built in java classes / API can be used to write the program. calculate the accuracy, precision and recalls for your dataset.

```
import pandas as pd.
msg = pd.read_csv('data.csv', names=['message', 'label'])
print('Total instance in the dataset:', msg.shape[0])
msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})
x = msg.message
y = msg.labelnum.
print('In the message and its label of first 5 instance are listed below')
xs, ys = x[0:5], msg.label[0:5].
    for x, y in zip(xs, ys):
        print(x, '-', y)
from sklearn.model_selection import train_test_split(x,y).
print('In Dataset is split into Training and testing samples')
print('Total training instance', xtrain.shape[0]).
print('Total testing instance:', xtest.shape[0]).
from sklearn.feature_extraction.text import countvectorizer.
count_vect = countvectorizer()
```

Teacher's Signature : .....

Date.....

Expt. No.....6..... Page No.....17.....

```
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm = count_vect.transform(xtest)
print('\n Total features extracted using
countvectorizer:', xtrain_dtm.shape[1])
print('\n features for first 5 training instances
listed below')
df = pd.DataFrame(xtrain_dtm.toarray(), columns=
count_vect.get_feature_names())
print(df.loc[0:5])
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm, ytrain)
predicted = clf.predict(xtest_dtm)
print('\n classification results of testing samples
are given below')
for doc, p in zip(xtest, predicted):
    pred = 'pos' if p==1 else 'neg'
    print('%s -> %s' %(doc, pred))
from sklearn import metrics
print('\n Accuracy metrics')
print('Accuracy of the classifier is', metrics.
accuracy_score(ytest, predicted))
print('Recall:', metrics.recall_score(ytest, predicted,
'\n Precision:', metrics.precision_score(ytest,
predicted))
print('confusion matrix')
print(metrics.confusion_matrix(ytest, predi-
cted)).
```

Teacher's Signature : .....

Date.....

Expt. No.....

Page No.....

output

Total instance in the dataset : 18.

The message and its label of first 5 instance are listed below:

I love sandwich, pos.
This is an amazing place, pos
I feel very good about these beers. pos.
This is my best work, pos.
What an awesome view, pos

Dataset is split into training & testing sets.
Total training instance : 13.
Total testing instance : 5.

Total feature extracted using count vectorizer:
Feature for first 5 instance training are listed below.

	about	am	an	awesome	beers	best	boss	can	deal	do	...	today
0	0	0	0	0	0	0	0	0	0	1		0
1	0	0	0	0	0	0	0	0	0	0		1
2	0	0	0	0	0	0	0	1	1	0		0
3	0	0	1	1	0	0	0	0	0	0		0
4	0	0	0	0	0	0	0	0	0	0		0

Date.....

Page No.....

Expt. No.....

	tommorow	very	view	we	want	what	will	with	work
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0

[5 rows x 46 columns]

classification result of testing sample are gi
below:

I love to dance → pos.
I am sick and tired of this place → neg
This is an amazing place → pos
what a great holiday → pos.
This is a bad locality to stay → neg.

Accuracy metrices:

Accuracy of the classifier is 1.0.

Recall : 1.0

precision : 1.0.

confusion matrix.

[[2,0],[0,3]]