Expt. No. 1 Date 29/9/20

Page No. 1

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

```
from pandas import DataFrame
data = DataFrame.from_csv('C:/Users/hp/Desktop/4MT17CS043_Harshitha/labl.csv')
columnLength = data.shape[1]
print(data)
h = ['0'] * (columnLength - 1)
hp = []
hn = []
for trainingExample in data.values:
        if trainingExample[-1] != 'no':
            hp.append(list(trainingExample))
        else:
            hn.append(list(trainingExample))
for i in range(len(hp)):
    for j in range(columnLength - 1):
        if (h[j] == '0'):
            h[j] = hp[i][j]
        if (h[j] != hp[i][j]):
            h[j] = '?'
        else:
            h[j] = hp[i][j]
```

Teacher's Signature _____

```
print ("\n The positive hypothesis are : ", hp)
print ("\n The negative hypothesis are : ", hn)
print ("\n The maximally specific hypothesis is : ", h)
```

Sl. No	Sky	Airtemp	Humidity	Wind	Water	Forecast	Enjoy
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

The positive hypothesis are :

[['Sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'],
['Sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'],
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']]

The negative hypothesis are :

[['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']]

The maximally specific hypothesis is :

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

Expt. No. 2. Date 06/10/20

Page No. 3

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

```
import csv
with open('C:/Users/hp/Desktop/4MT17CS043/Harshitha/
          lab2.csv') as f:
    csv_file = csv.reader(f)
    data = list(csv_file)
print(data)
s = data[1][:-1]
print(s)
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] = "?"
```

Teacher's Signature _____

```
print ("steps of candidate elimination algorithm", data.index
                                                    (i)+1)
print (s)
print (g)
gh = []
for i in g:
    for j in i:
        if j != '?':
            gh.append (i)
            break
print ("\n Final specific hypothesis : \n", s)
print ("\n Final general hypothesis : \n", gh)
```

Teacher's Signature

[['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']]

['sunny', 'warm', 'high', 'strong', 'warm', 'same']

Steps of candidate elimination algorithm 1:

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

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

Steps of candidate elimination algorithm 2:

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

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

Steps of candidate elimination algorithm 3:

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

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

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

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

Steps of candidate elimination algorithm 4:

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

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

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

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

Final general hypothesis:

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

'?', '?']]

Expt. No. 3. Date 27/10/20

Page No. 5

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 this knowledge to classify a new sample.

```
import pandas as pd
from pandas import DataFrame
df_tennis = pd.read_csv('C:/Users/hp/Desktop/4MT17CS043_Harshitha/playtennis.csv')
attribute_names = list(df_tennis.columns)
attribute_names.remove('Play Tennis')
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])
```

Teacher's Signature

Expt. No. _____ Date _____

Page No. 6

```
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) * 1.0
    df_agg_ent = df_split.agg({target_attribute_name: [entropy_of_list, lambda x: len(x)/nobs]})
    df_agg_ent.columns = ['Entropy', 'propobservation']
    new_entropy = sum(df_agg_ent['Entropy'] * df_agg_ent['propobservations'])
    old_entropy = entropy_of_list(df[target_attribute_name])
    print(split_attribute_name, 'IG:', 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_attribute_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())
        gain = [
```

Teacher's Signature _____

```
information_gain(df, attr, target_attribute_name) for
attr in attribute_names]
index_of_max = gain.index(max(gain))
best_attr = attribute_names[index_of_max]

tree = {best_attr: {}}
remaining_attribute_names = [i for i in attribute_names
                             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 tree.

from pprint import pprint
tree = id3(df_tennis, 'PlayTennis', attribute_names)
print("\n The resultant Decision tree is: \n")
pprint(tree)
```

Teacher's Signature

['outlook', 'Temperature', 'Humidity', 'wind']

Outlook IG : 0.24674981977443911
Temperature IG : 0.029922565658954647
Humidity IG : 0.15183550136234136
Wind IG : 0.048127030408269 27

Temperature IG : 0.019973094021974 89
Humidity IG : 0.019973094021974 89
Wind IG : 0.9709505944546686

Temperature IG : 0.5709505944546686
Humidity IG : 0.9709505944546686
Wind IG : 0.019973094021974 89

The result decision tree is :

{'outlook': {'overcast': 'yes',
'Rain': {'wind': {'strong': 'NO', 'weak': 'Yes'}},
'Sunny': {'Humidity': {'high': 'No',
'Normal': 'Yes'}}}}

Date 02/12/20

Expt. No. 4 Page No. 8

Build an artificial neural network by implementing the backpropagation algorithm and test the same using appropriate dataset.

```
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))

def forward_propagate (network, row):
```

Teacher's Signature

```
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

def transfer_derivative(output):
    return output * (1.0 - output)

def backward_propogate_error(network, expected):
    for i in reverted(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]
        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 netwo
                      -rk [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_propogate (network, row)
            expected [row [-1]] = 1
            sum_error += sum ([(expected [i] - output [i])
                               ** 2 for i in range (len (expected))])
            backward_propogate_error (network, expected)
            update_weights (network, row, l_rate)
```

```
print ('>epoch=%d, l-rate=%.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_networks(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
    print(layer)
```

Teacher's Signature _____

```
>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 = 3.501
>epoch = 8 , lrate = 0.500, error = 3.192
> epoch = 9 , lrate = 0.500, error = 2.898
>epoch = 10 , lrate = 0.500, error = 2.628
>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
```

[{'weights': [-1.4688375095432327, 1.8508857325435315,
1.0858178629550297], 'output': 0.029980305604616,
'delta': -0.0059566041623236255},

{'weights': [0.3771109814246215 7, -0.06259098945529
987, 0.2765123702642716], 'output': 0.9456220
90002113 'delta': 0.0026279652850563837}]

[{'weights': [2.5153946493978499, -0.3391929502445985,
-0.9671565426390275], 'output': 0.2344679420935
7587, 'delta': -0.042700592783645874},

{'weights': [-2.5584149848484263, 1.0036422106209202,
0.42383086467582715], 'output':
0.7905352024386836, 'delta': 0.038031325
96437354}]

Date 22/12/2020

Expt. No. 5

Page No. 12

Write a program to implement the naive Bayesian classifier for a sample training dataset stored as a .csv file. Compute the accuracy of the classifier, considering few test data sets.

```
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 separateByClass(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 = separateByClass(dataset):
    summary ={}
    for classValue, instances in separated.items():
        summary[classValue] = compute-mean-std
                              (instances)
    return summary.

def estimateProbability(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, testVector):
    p={}
    for classValue, classSummaries in summaries.
        items():
```

```
p[classValue] = 1
for i in range (len(classSummaries)):
    mean, stdev = classSummaries[i]
    x = testVector[i]
    p[classValue] *= 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():
        if bestlabel is None or p > bestProb:
            bestProb = p
            bestlabel = lbl
return bestlabel

def performClassification(summaries, testset):
    predictions = []
    for i in range(len(testset)):
        result = 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

dataset = loadcsv('C:/users/hp/Desktop/4MT17CS043-
                  Harshitha/diabetes.csv');
print ('Pima Indian 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 testing
                set")
print ("Training examples = {0} \nTesting examples = {1}'
            format (len(trainingset), len(testset)))
Summaries = summariseByClass (trainingset);
predictions = perform - classification (Summaries, testset)
accuracy = getAccuracy (testset, predictions)
print ("\nAccuracy of the Naive Bayesian classifi
            -er is :", accuracy)
```

Teacher's Signature _____

Pima Indian Diabetes Dataset loaded..
Total instances available : 768
Total attributes present : 8

First five instances of dataset:

1 : [1.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]

2 : [1.0, 85.0, 66.0, 290, 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, 32.01, 1.0]

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 examples = 615

Testing examples = 153

Accuracy of the Naive Bayesian classifier is :
75.16339869281046.

Expt. No. 6. Date 22/12/20

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 recall for your dataset.

```
import pandas as pd
msg = pd.read_csv('C:/users/hp/Desktop/4MT17CS043_Harshitha/lab6.csv', names=['message', 'label'])
print('Total instances in the dataset :', msg.shape[0])
msg["labelnum"] = msg.label.map({'pos':1, 'neg':0})
x = msg.message
y = msg.labelnum
print("\n The message and its label of first 5 instances are listed below")
x5, y5 = x[0:5], msg.label[0:5]
for x, y in zip(x5, y5):
    print(x, ',', y)

from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y)
print("Dataset is split into training and testing samples")
print("Total training instances :", xtrain.shape[0])
print("Total testing instances :", xtest.shape[0])

from sklearn.feature_extraction.text import CountVectori
-zer.
```

Teacher's Signature

```
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtrain_dtm = count_vect.transform(xtest)
print("\n Total features extracted using CountVectorizer :: xtrain_dtm.shape[1])
print("\n Features for first 5 traing instances are listed below")
df = pd.DataFrame(xtrain_dtm.toarray(), columns=count_vect.get_feature_names())
print(df[0:5])

from sklearn.naivebayes import MultiNominalNB
df = MultiNominalNB().fit(xtrain_dtm, ytrain)
predicted = df.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 metrices
print("\n Accuracy metrices")
print("\n Accuracy of the classifier is", metrices.accuracy_score(ytest, predicted))
print("Recall:", metrices.recall_score(ytest, predicted))
print("confusion matrix")
print(metrices.confusion_matrix(ytest, predicted))
```

Teacher's Signature _____

Total instances in the dataset: 18

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

I love this 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 Samples

Total training instances: 13
Total testing instances: 5

Total features extracted using countvectorizer: 46
Features for first 5 training instances are listed below.

	about	am	an	awesome	beers	best	boss	can	dead	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 18/12/20

Expt. No. 9 Page No. 18

Write a program to Implement k-nearest neighbour algorithm to classify the iris dataset. Print both correct and wrong predictions. Java / Python ML library classes can be used for this problem.

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrices import classification_report
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrices import confusion_matrix.
from sklearn.metrices import accuracy_score
iris_dataset = load_iris()

print ("\n IRIS FEATURES\ TARGET NAMES : \n", iris_
        dataset, target_names)
for i in range (len (iris_dataset.target_names)):
    print ("\n [{0}]: [{1}]", format (i, iris_dataset.
                target_names[i]))
# print ("\n Iris data: \n", iris_dataset ["data"])

x_train, x_test, y_train, y_test = train_test_split (iris_
        dataset ["data"], iris_dataset ("target")
        random_state = 0)

classifier = kNeighborsClassifier (n_neighbors = 8, p=3,
        metric = "euclidean")
```

Teacher's Signature

```
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test
cm = confusion_matrix(y_test, y_pred)
print("confusion matrix is as follows \n", cm)
print("Accuracy metrices")
print(classification_report(y_test, y_pred))
print("correct prediction", accuracy_score(y_test,
                                    y_pred))
print("wrong prediction", (1-accuracy_score(y_test,
                                    y_pred)))
```

Teacher's Signature _____

IRIS FEATURES \ TARGET NAMES:

['setosa' 'versicolor' 'virginica']

[0]: [setosa]
[1]: [versicolor]
[2]: [virginica]

KNeighbors Classifier (algorithm = 'auto', leaf_size = 30, metric = 'euclidean' metric_params = none, n_jobs = none, n_neighbors = 8, P = 3; weights = 'uniform')

Confusion matrix is as follows:

```
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
```

Accuracy matrices

	prediction	recall	f1-score	support
0	1.00	1.00	1.00	13
1	1.00	0.94	0.97	6
2	0.90	1.00	0.95	9
accuracy			0.97	38
macro avg	0.97	0.98	0.97	38
weighted avg	0.98	0.97	0.97	38

Correct prediction : 0.9736842105263158

Wrong prediction : 0.02631578947368418