1 FIND-S Algorithm

```
importcsv
a = []
with open('enjoysport.csv', 'r') as csvfile:
next(csvfile)
for row in csv.reader(csvfile):
a.append(row)
print(a)
print("\nThe total number of training instances are : ",len(a))
num_attribute = len(a[0])-1
print("\nThe initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)
for i in range(0, len(a)):
if a[i][num_attribute] == 'yes':
print ("\nInstance ", i+1, "is", a[i], " and is Positive Instance")
for j in range(0, num_attribute):
if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
hypothesis[j] = a[i][j]
else:
hypothesis[j] = '?'
print("The hypothesis for the training instance", i+1, " is: " , hypothesis, "\n")
if a[i][num_attribute] == 'no':
print ("\nInstance ", i+1, "is", a[i], " and is Negative Instance Hence Ignored")
print("The hypothesis for the training instance", i+1, " is: " , hypothesis, "\n")
print("\nThe Maximally specific hypothesis for the training instance is ", hypothesis)
```

DATA SET

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
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:
[['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast', 'enjoysport'], ['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']]
The total number of training instances are : 5
The initial hypothesis is : ['0', '0', '0', '0', '0', '0']
Instance 2 is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'] and is Positive Instance
The hypothesis for the training instance 2 is: ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
Instance 3 is ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'] and is Positive Instance
The hypothesis for the training instance 3 is: ['sunny', 'warm', '?', 'strong', 'warm', 'same']
Instance 4 is ['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no'] and is Negative Instance Hence Ignored
The hypothesis for the training instance 4 is: ['sunny', 'warm', '?', 'strong', 'warm', 'same']
Instance 5 is ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes'] and is Positive Instance
The hypothesis for the training instance 5 is: ['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally specific hypothesis for the training instance is [‘sunny’, ‘warm’, ‘?’, ‘strong’, ‘?’, ‘?’]

2 Candidate-Elimination algorithm

```
importnumpy as np
import pandas as pd
data = pd.read_csv(path+'/enjoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)
def learn(concepts, target):
specific_h = concepts[0].copy()
print("\nInitialization of specific_h and genearal_h")
print("\nSpecific Boundary: ", specific_h)
general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
print("\nGeneric Boundary: ",general_h)
for i, h in enumerate(concepts):
print("\nInstance", i+1 , "is ", h)
if target[i] == "yes":
print("Instance is Positive ")
for x in range(len(specific_h)):
if h[x]!= specific_h[x]:
specific_h[x] ='?'
general_h[x][x] ='?'
if target[i] == "no":
print("Instance is Negative ")
for x in range(len(specific_h)):
if h[x]!= specific_h[x]:
general_h[x][x] = specific_h[x]
else:
general_h[x][x] = '?'
print("Specific Boundary after ", i+1, "Instance is ", specific_h)
print("Generic Boundary after ", i+1, "Instance is ", general_h)
print("\n")
indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
for i in indices:
general_h.remove(['?', '?', '?', '?', '?', '?'])
return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

Sky	Temperature	Humid	Wind	Water	Forest	Output
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

OUTPUT:
Instances are:
[[‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]
[‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘warm’ ‘same’]

[‘rainy’ ‘cold’ ‘high’ ‘strong’ ‘warm’ ‘change’]
[‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘cool’ ‘change’]]
Target Values are: [‘yes’ ‘yes’ ‘no’ ‘yes’]
Initialization of specific_h and genearal_h
Specific Boundary: [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]
Generic Boundary: [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]
Instance 1 is [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’] Instance is Positive
Specific Bundary after 1 Instance is [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]
Generic Boundary after 1 Instance is [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]
Instance 2 is [‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘warm’ ‘same’] Instance is Positive
Specific Bundary after 2 Instance is [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘warm’ ‘same’]
Generic Boundary after 2 Instance is [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]
Instance 3 is [‘rainy’ ‘cold’ ‘high’ ‘strong’ ‘warm’ ‘change’] Instance is Negative
Specific Bundary after 3 Instance is [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘warm’ ‘same’]
Generic Boundary after 3 Instance is [[‘sunny’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘warm’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘same’]]
Instance 4 is [‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘cool’ ‘change’] Instance is Positive
Specific Bundary after 4 Instance is [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘?’ ‘?’]
Generic Boundary after 4 Instance is [[‘sunny’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘warm’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]
Final Specific_h: [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘?’ ‘?’]
Final General_h: [[‘sunny’, ‘?**’, ‘?’, ‘?’, ‘?**’, ‘?’], [‘?’, ‘warm’, ‘?’, ‘?’, ‘?’, ‘?’]

Decision tree based ID3 algorithm

```
import pandas as pd
import math
importnumpy as np
data = pd.read_csv("Dataset/4-dataset.csv")
features = [feat for feat in data]
features.remove("answer")
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
ifpos == 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))
definfo_gain(examples, attr):
uniq = np.unique(examples[attr])
#print ("\n",uniq)
gain = entropy(examples)
#print ("\n",gain)
for u in uniq:
subdata = examples[examples[attr] == u]
#print ("\n",subdata)
sub_e = entropy(subdata)
gain -= (float(len(subdata)) / float(len(examples))) * sub_e
#print ("\n",gain)
return gain
def ID3(examples, attrs):
root = Node()
max_gain = 0
max_feat = ""
for feature in attrs:
#print ("\n",examples)
gain = info_gain(examples, feature)
if gain >max_gain:
max_gain = gain
max_feat = feature
root.value = max_feat
#print ("\nMax feature attr",max_feat)
uniq = np.unique(examples[max_feat])
#print ("\n",uniq)
for u in uniq:
#print ("\n",u)
subdata = examples[examples[max_feat] == u]
#print ("\n",subdata)
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
defprintTree(root: Node, depth=0):
for i in range(depth):
print("\t", end="")
print(root.value, end="")
ifroot.isLeaf:
print(" -> ", root.pred)
print()
for child in root.children:
printTree(child, depth + 1)
def classify(root: Node, new):
for child in root.children:
ifchild.value == new[root.value]:
ifchild.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)
print ("------------------")
new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"}
classify (root, new)
```

DATA SET

OUTPUT:

```
Decision Tree is:
outlook
overcast -> ['yes']
rain
wind
strong -> ['no']
weak -> ['yes']
sunny
humidity
high -> ['no']
normal -> ['yes']
Predicted Label for new example {'outlook': 'sunny', 'temperature': 'hot', 'humidity': 'normal', 'wind':
'strong'} is: ['yes']
```

Artificial Neural Network by implementing the Back-propagation algorithm

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) #maximum of X array longitudinally
y = y/100
#Sigmoid Function
def sigmoid (x):
return 1/(1 + np.exp(-x))
```

```
#Derivative of Sigmoid Function
defderivatives_sigmoid(x):
return x * (1 - x)
#Variable initialization
epoch=5 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
#Forward Propogation
hinp1=np.dot(X,wh)
hinp=hinp1 + bh
hlayer_act = sigmoid(hinp)
outinp1=np.dot(hlayer_act,wout)
outinp= outinp1+bout
output = sigmoid(outinp)
#Backpropagation
EO = y-output
outgrad = derivatives_sigmoid(output)
d_output = EO * outgrad
EH = d_output.dot(wout.T)
hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts contributed to error
d_hiddenlayer = EH * hiddengrad
wout += hlayer_act.T.dot(d_output) *lr # dotproduct of nextlayererror and currentlayerop
wh += X.T.dot(d_hiddenlayer) *lr
print ("-----------Epoch-", i+1, "Starts----------")
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
print ("-----------Epoch-", i+1, "Ends----------\n")
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

OUTPUT:

```
————Epoch- 1 Starts———-
Input:
[[0.66666667 1. ]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:
```

[[0.81951208]
[0.8007242]
[0.82485744]]
————Epoch- 1 Ends———-
————Epoch- 2 Starts———-
Input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:
[[0.82033938]
[0.80153634]
[0.82568134]]
————Epoch- 2 Ends———-
————Epoch- 3 Starts———-
Input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:
[[0.82115226]
[0.80233463]
[0.82649072]]
————Epoch- 3 Ends———-
————Epoch- 4 Starts———-
Input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]

Predicted Output:
[[0.82195108]
[0.80311943]
[0.82728598]]
————Epoch- 4 Ends———-
————Epoch- 5 Starts———-
Input:
[[0.66666667 1.]

[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:
[[0.8227362]
[0.80389106]
[0.82806747]]
———-Epoch- 5 Ends———-

Input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:
[[0.8227362]
[0.80389106]
[0.82806747]]
naïve Bayesian classifier

```
import csv
import random
import math
def loadcsv(filename):
lines = csv.reader(open(filename, "r"));
dataset = list(lines)
for i in range(len(dataset)):
#converting strings into numbers for processing
dataset[i] = [float(x) for x in dataset[i]]
return dataset
def splitdataset(dataset, splitratio):
#67% training size
trainsize = int(len(dataset) * splitratio);
trainset = []
copy = list(dataset);
while len(trainset) < trainsize:
#generate indices for the dataset list randomly to pick ele for training data
index = random.randrange(len(copy));
trainset.append(copy.pop(index))
return [trainset, copy]
def separatebyclass(dataset):
separated = {} #dictionary of classes 1 and 0
#creates a dictionary of classes 1 and 0 where the values are
#the instances belonging to each class
```

```
for i in range(len(dataset)):
vector = dataset[i]
if (vector[-1] not in separated):
separated[vector[-1]] = []
separated[vector[-1]].append(vector)
return separated
def mean(numbers):
return sum(numbers)/float(len(numbers))
def stdev(numbers):
avg = mean(numbers)
variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
return math.sqrt(variance)
def summarize(dataset): #creates a dictionary of classes
summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
del summaries[-1] #excluding labels +ve or -ve
return summaries
def summarizebyclass(dataset):
separated = separatebyclass(dataset);
#print(separated)
summaries = {}
for classvalue, instances in separated.items():
#for key,value in dic.items()
#summaries is a dic of tuples(mean,std) for each class value
summaries[classvalue] = summarize(instances) #summarize is used to cal to mean and std
return summaries
def calculateprobability(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):
probabilities = {} # probabilities contains the all prob of all class of test data
for classvalue, classsummaries in summaries.items():#class and attribute information as mean and sd
probabilities[classvalue] = 1
for i in range(len(classsummaries)):
mean, stdev = classsummaries[i] #take mean and sd of every attribute for class 0 and 1 seperaely
x = inputvector[i] #testvector's first attribute
probabilities[classvalue] *= calculateprobability(x, mean, stdev);#use normal dist
return probabilities
def predict(summaries, inputvector): #training and test data is passed
probabilities = calculateclassprobabilities(summaries, inputvector)
bestLabel, bestProb = None, -1
for classvalue, probability in probabilities.items():#assigns that class which has he highest prob
if bestLabel is None or probability > bestProb:
bestProb = probability
bestLabel = classvalue
```

```
return bestLabel
def getpredictions(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
def main():
filename = 'naivedata.csv'
splitratio = 0.67
```



```
dataset = loadcsv(filename);
trainingset, testset = splitdataset(dataset, splitratio)
print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingset),
len(testset)))
# prepare model
summaries = summarizebyclass(trainingset);
#print(summaries)
# test model
predictions = getpredictions(summaries, testset) #find the predictions of test data with the training
data
accuracy = getaccuracy(testset, predictions)
print('Accuracy of the classifier is : {0}%'.format(accuracy))
main()
```

OUTPUT:

Split 768 rows into train=514 and test=254

Rows Accuracy of the classifier is : 71.65354330708661%