## Source Code:

```
import pandas as pd
import csv

traindata=pd.read_csv("trainingdata.csv")

cols=traindata.shape[1]
rows=traindata.shape[0]
print(cols)
print(rows)

print("The training data is as follows ")
print(traindata)
print("\nThe most general hypothesis : ['?','?','?','?','?','?']\n")
h=['0','0','0','0','0','0']
print("The most specific hypothesis : ",h,"\n")

positivesample=[]
negativesample=[]

for i in range(rows):
    trainsample=[]
    trainsample.append(traindata.sky[i])
    trainsample.append(traindata.airtemp[i])
    trainsample.append(traindata.humidity[i])
    trainsample.append(traindata.wind[i])
    trainsample.append(traindata.water[i])
    trainsample.append(traindata.forecast[i])
    if(traindata.enjoyspot[i]!='No'):
        positivesample.append(trainsample)
    else:
        negativesample.append(trainsample)

print("Positive samples are \n",positivesample)
print("Negative samples are \n",negativesample)
```

```
for i in range(len(positivesample)):
   for j in range(cols-1):
        if h[j]=='0':
           h[j]=positivesample[i][j]
        if h[j]!=positivesample[i][j]:
           h[j]='?'
        else:
           h[j]=positivesample[i][j]
    print("\nFor Training example ", i ,"the hypothesis is",h)
print("\nThe Maximally Specific hypothesis for a given Training Example:\n")
print(h)
```

## Sample Dataset:

```
sky,airtemp,humidity,wind,water,forecast,enjoyspot
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,Cold,Change,Yes
```

## Output:

```
7
4
The training data is as follows
     sky airtemp humidity    wind water forecast enjoyspot
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  Cold   Change       Yes

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

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

Positive  samples are
 [['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same'], ['Sunny',
'Warm', 'High', 'Strong', 'Warm', 'Same'], ['Sunny', 'Warm',
'High', 'Strong', 'Cold', 'Change']]
```

```
Negative samples are
[['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change']]

For Training example 0 the hypothesis is ['Sunny', 'Warm', 'Normal',
'Strong', 'Warm', 'Same']

For Training example 1 the hypothesis is ['Sunny', 'Warm', '?',
'Strong', 'Warm', 'Same']

For Training example 2 the hypothesis is ['Sunny', 'Warm', '?',
'Strong', '?', '?']

The Maximally Specific hypothesis for a given Training Example:

['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

## Source Code:

```
import csv
with open('wsce.csv') as csvFile:
   examples=[tuple(line) for line in
   csv.reader(csvFile)] print(examples)

def more_general(h1,h2):
   more_general_parts=[]
   for x,y in zip(h1,h2):
      mg=x == '?' or (x!='0' and (x == y or y=='0'))
      more_general_parts.append(mg)
   return all(more_general_parts)

def fulfills(examples,hypothesis):
   return more_general(hypothesis,examples)

def min_generalizations(h,x):
   h_new=list(h)
   for i in range(len(h)):
      if not fulfills(x[i:i+1],h[i:i+1]):
         h_new[i]='?' if h[i]!='0' else x[i]
   return [tuple(h_new)]

def min_specialization(h,domains,x):
   results=[]
   for i in range(len(h)):
      if h[i]=='?':
         for val in domains[i]:
            if x[i]!= val:
               h_new=h[:i]+(val,)+h[i+1:]
               results.append(h_new)
      elif h[i]!="0":
         h_new=h[:i]+('0',)+h[i+1:]
         results.append(h_new)
   return results

min_generalizations(h=('0','0','Sunny'),x=('Rainy','Windy','Cloudy'))
```

```
min_specialization(h=('?','x',),domains=[['a','b','c'],['x','y']],x=('b','x'))

def get_domains(examples):
   d=[set() for i in examples[0]]
   for x in examples:
      for  i,xi in enumerate(x):
         d[i].add(xi)
   return [list(sorted(x)) for x in d]

get_domains(examples)


def generalize_S(x,G,S):
   S_prev=list(S)
   for s in S_prev:
      if s not in S:
         continue
      if not fulfills(x,s):#not(s must be true for x)
         S.remove(s)
         Splus=min_generalizations(s,x)
         #keep only generalizations that have a counter part in G
         S.update([h for h in Splus if any([more_general(g,h) for g in G])])
         ## remove hypothesis less specific than any other in S
         S.difference_update([h for h in S if any([more_general(h,h1) for h1 in S if
h!=h1])])
   return S

def specialize_G(x,domains,G,S):
   G_prev=list(G)
   for g in G_prev:
      if g not in G:
         continue
      if fulfills(x,g):#g is true for x=> hypothesis is incorrect
         G.remove(g)
         Gminus=min_specialization(g,domains,x)
         ##keep only specializations that have a counterpart in S G.update([h
         for h in Gminus if any([more_general(h,s) for s in S])])

         ##remove hypothesis less general than sny other in G
         G.difference_update([h for h in G if any([more_general(g1,h) for g1 in G
if h!=g1])])
   return G
```

```
def candidate_elimination(examples):
    domains=get_domains(examples)[:-1]

    G= set([('?',)*len(domains)])
    S= set([('0',)*len(domains)])
    i=0
    print("\n G[{0}]: ".format(i),G)
    print("\n S[{0}]: ".format(i),S)
    for xcx in examples:
        i=i+1
        x,cx=xcx[:-1],xcx[-1]
        if cx=='Y':
            G={g for g in G if fulfills(x,g)}
            S=generalize_S(x,G,S)
        else:
            S={s for s in S if not fulfills(x,s)}
            G=specialize_G(x,domains,G,S)
        print("\n G[{0}]: ".format(i),G)
        print("\n S[{0}]: ".format(i),S)
    enumerateVersionSpace(S,G)
    return S,G

def enumerateHypothesesBetween_s_g(s,g):
    hypotheses=[]
    for i,constraint in enumerate(g):
        if constraint != s[i]:
            hypothesis=g[:]
            hypothesis[i]=s[i]
            hypotheses.append(hypothesis)
    return hypotheses

def enumerateVersionSpace(S,G):
    #print("print S",S)
    #print("print G",G)
    hypotheses=[]
    hypotheses+=S
    hypotheses+=G
    print("Initial Hypothesis ",hypotheses)
```

```
    s=hypotheses[0]
    for i in range(1,len(hypotheses)):
inBetweenhypotheses=enumerateHypothesesBetween_s_g(list(s),list(hypotheses[i
]))
        hypotheses.extend(inBetweenhypotheses)
    setH=set()
    for h in hypotheses:
        setH.add(tuple(h))
    ans=[list(x) for x in setH]
    print("Version Space: ",ans)
```

## Sample Dataset:

*Sunny,Warm,Normal,Strong,Warm,Same,Y*
*Sunny,Warm,High,Strong,Warm,Same,Y*
*Rainy,Cold,High,Strong,Warm,Change,N*
*Sunny,Warm,High,Strong,Cold,Change,Y*

## Output:

```
[('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Y'),
('Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Y'),
('Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'N'),
('Sunny', 'Warm', 'High', 'Strong', 'Cold', 'Change', 'Y')]

[('Rainy', 'Windy', '?')]

[('a', 'x'), ('c', 'x'), ('?', '0')]


[['Rainy', 'Sunny'],
 ['Cold', 'Warm'],
 ['High', 'Normal'],
 ['Strong'],
 ['Cold', 'Warm'],
 ['Change', 'Same'],
 ['N', 'Y']]


G[0]:  {('?', '?', '?', '?', '?', '?')}

S[0]:  {('0', '0', '0', '0', '0', '0')}

G[1]:  {('?', '?', '?', '?', '?', '?')}

S[1]:  {('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same')}
```

```
G[2]:  {('?', '?', '?', '?', '?', '?')}

S[2]:  {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}

G[3]: {('?', '?', '?', '?', '?', 'Same'), ('?', 'Warm', '?', '?',
'?', '?'), ('Sunny', '?', '?', '?', '?', '?')}

S[3]:  {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}

G[4]: {('?', 'Warm', '?', '?', '?', '?'), ('Sunny', '?', '?', '?',
'?', '?')}

S[4]:  {('Sunny', 'Warm', '?', 'Strong', '?', '?')}
Initial Hypothesis [('Sunny', 'Warm', '?', 'Strong', '?', '?'),
('?', 'Warm', '?', '?', '?', '?'), ('Sunny', '?', '?', '?',
'?', '?')]
Version Space: [['Sunny', '?', '?', 'Strong', '?', '?'], ['Sunny',
'Warm', '?', 'Strong', '?', '?'], ['Sunny', '?', '?', '?', '?', '?'],
['?', 'Warm', '?', 'Strong', '?', '?'], ['?', 'Warm', '?', '?', '?',
'?'], ['Sunny', 'Warm', '?', '?', '?', '?']]
```

## Source Code:

```
import pandas as pd
import csv
from pandas import DataFrame
df_tennis=DataFrame.from_csv('id3dataset.csv')
print("\n Given play Tennis data set:\n\n",df_tennis)

df_tennis.keys()[0]

def entropy(probs):
   import math
   return sum([-prob*math.log(prob,2)for prob in probs])

def entropy_of_list(a_list):
   from collections import Counter
   cnt=Counter(x for x in a_list)
   print("\nClasses:",cnt)

   num_instances=len(a_list)
   print("\n Number of instances of the current sub class
is {0}:".format(num_instances))
   probs=[x / num_instances for x in cnt.values()]
   print(probs)
   print("\nClasses:",list(cnt.keys()))
   print("\n Probabilities of Class {0} is {1}".format(min(cnt),min(probs)))
   print("\n Probabilities of Class {0} is {1}".format(max(cnt),max(probs)))

   return entropy(probs)

print("\n INPUT DATA SET FOR ENTROPY
CALCULATION:\n",df_tennis['playtennis'])

total_entropy=entropy_of_list(df_tennis['playtennis'])
print("\n Total Entropy of PlayTennis Data Set:",total_entropy)

def information_gain(df,split_attribute_name,target_attribute_name,trace=0):
   print("Information Gain Calculation of ",split_attribute_name)

   df_split=df.groupby(split_attribute_name)
   print("Split :",type(df_split))
   for name,group in df_split:
       print("Name:\n",name)
       print("Group:\n",group)
```

```
    nobs=len(df.index)

    df_agg_ent=df_split.agg({target_attribute_name:[entropy_of_list,lambda x:
len(x)/nobs]})[target_attribute_name]
    print(df_agg_ent.columns)
    print("DFAGGENT",df_agg_ent)
    df_agg_ent.columns=['Entropy','PropObservations']

    new_entropy=sum(df_agg_ent['Entropy']*df_agg_ent['PropObservations'])
    old_entropy=entropy_of_list(df[target_attribute_name])
    return old_entropy-new_entropy

print('Info-gain for Outlook is :
'+str(information_gain(df_tennis,'outlook','playtennis')),"\n")
print('\nInfo-gain for humidity
is:'+str(information_gain(df_tennis,'humidity','playtennis')),'\n')
print('\nInfo_gain for wind
is:'+str(information_gain(df_tennis,'wind','playtennis')),'\n')
print('\nInfo-gain for temperature
is:'+str(information_gain(df_tennis,'temperature','playtennis')),'\n')

def id3(df,target_attribute_name,attribute_names,default_class=None):

    from collections import Counter
    cnt=Counter(x for x in df[target_attribute_name])

    if len(cnt)==1:
        return next(iter(cnt))

    elif df.empty or (not attribute_names):
        return default_class

    else:
        default_class=max(cnt.keys())
        gainz=[information_gain(df,attr,target_attribute_name) for attr
in attribute_names]
        index_of_max=gainz.index(max(gainz))
        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


attribute_names=list(df_tennis.columns)
print("List of Attributes:",attribute_names)
attribute_names.remove('playtennis')
print("Predicting Attributes:",attribute_names)

from pprint import pprint
tree=id3(df_tennis,'playtennis',attribute_names)
print("\n\nThe resultant decision tree is :\n")
pprint(tree)
attribute=next(iter(tree))
print("Best Attribute :\n",attribute)
print("Tree keys:\n",tree[attribute].keys())

def classify(instance,tree,default=None):
   attribute=next(iter(tree))
   print("Key:",tree.keys())
   print("Attribute:",attribute)

   if instance[attribute] in tree[attribute].keys():
      result=tree[attribute][instance[attribute]] print("Instance
      Attribute:",instance[attribute],"TreeKeys
:",tree[attribute].keys())
      if isinstance(result,dict):
         return classify(instance,result)
      else:
         return result
   else:
      return default
```

```
df_tennis['predicted']=df_tennis.apply(classify,axis=1,args=(tree,'no'))

print(df_tennis['predicted'])
print('\n Accuracy is:\n'+str( sum(df_tennis['playtennis']==df_tennis['predicted'])
/ (1.0*len(df_tennis.index))))

df_tennis[['playtennis','predicted']]

training_data=df_tennis.iloc[1:-4]
test_data=df_tennis.iloc[-4:]
train_tree=id3(training_data,'playtennis',attribute_names)

test_data['predicted2']=test_data.apply(classify,axis=1,args=(train_tree,'yes'))

print('\n\nAccuracy is : '+str( sum(test_data['playtennis']==test_data['predicted2'])
/(1.0*len(test_data.index))))
```

## Sample Dataset:

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

## Output:

```
Given play Tennis data set:

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

INPUT DATA SET FOR ENTROPY CALCULATION:
 0      no
1      no
2     yes
3     yes
4     yes
5      no
6     yes
7      no
8     yes
9     yes
10    yes
11    yes
12    yes
13     no
Name: playtennis, dtype: object

Classes: Counter({'yes': 9, 'no': 5})

 Number of instances of the current sub class is 14:
[0.35714285714285715, 0.6428571428571429]

Classes: ['no', 'yes']

 Probabilities of Class no is 0.35714285714285715

 Probabilities of Class yes is 0.6428571428571429

 Total Entropy of PlayTennis Data Set: 0.9402859586706309
```

```
Information Gain Calculation of  outlook
Split : <class 'pandas.core.groupby.generic.DataFrameGroupBy'>
Name:
 overcast
Group:
    playtennis   outlook temperature humidity     wind
2          yes  overcast         hot     high     weak
6          yes  overcast        cool   normal   strong
11         yes  overcast        mild     high   strong
12         yes  overcast         hot   normal     weak
Name:
 rain
Group:
    playtennis outlook temperature humidity     wind
3          yes    rain        mild      high    weak
4          yes    rain        cool    normal    weak
5           no    rain        cool    normal  strong
9          yes    rain        mild    normal    weak
13          no    rain        mild      high  strong
 Name:
  sunny
 Group:
     playtennis outlook temperature humidity     wind
 0           no   sunny          hot     high     weak
 1           no   sunny          hot     high   strong
 7           no   sunny         mild     high     weak
 8          yes   sunny         cool   normal     weak
 10         yes   sunny         mild   normal   strong

 Classes: Counter({'yes': 4})

  Number of instances of the current sub class is 4:
 [1.0]

 Classes: ['yes']

  Probabilities of Class yes is 1.0

  Probabilities of Class yes is 1.0

 Classes: Counter({'yes': 3, 'no': 2})

  Number of instances of the current sub class is 5:
 [0.6, 0.4]

 Classes: ['yes', 'no']

  Probabilities of Class no is 0.4

  Probabilities of Class yes is 0.6
```

```
Classes: Counter({'no': 3, 'yes': 2})

 Number of instances of the current sub class is 5:
[0.6, 0.4]

Classes: ['no', 'yes']

 Probabilities of Class no is 0.4

 Probabilities of Class yes is 0.6

Index(['entropy_of_list', '<lambda>'], dtype='object')
DFAGGENT entropy_of_list <lambda> temperature

cool                 0.811278  0.285714
hot                  1.000000  0.285714
mild                 0.918296  0.428571

Classes: Counter({'yes': 9, 'no': 5})

 Number of instances of the current sub class is 14:
[0.35714285714285715, 0.6428571428571429]

Classes: ['no', 'yes']

 Probabilities of Class no is 0.35714285714285715

 Probabilities of Class yes is 0.6428571428571429

Info-gain for temperature is:0.029222565658954647

List of Attributes: ['playtennis', 'outlook',
'temperature', 'humidity', 'wind']
Predicting Attributes: ['outlook', 'temperature',
'humidity', 'wind']

Key: dict_keys(['outlook'])
Attribute: outlook
Instance Attribute: sunny TreeKeys : dict_keys(['overcast',
'rain', 'sunny'])
Key: dict_keys(['temperature'])
Attribute: temperature
Instance Attribute: mild TreeKeys : dict_keys(['cool', 'hot', 'mild'])
Key: dict_keys(['outlook'])
Attribute: outlook
Instance Attribute: overcast TreeKeys : dict_keys(['overcast',
'rain', 'sunny'])
```

```
Key: dict_keys(['outlook'])
Attribute: outlook
Instance Attribute: overcast TreeKeys : dict_keys(['overcast',
'rain', 'sunny'])
Key: dict_keys(['outlook'])
Attribute: outlook
Instance Attribute: rain TreeKeys : dict_keys(['overcast',
'rain', 'sunny'])
Key: dict_keys(['wind'])
Attribute: wind
Instance Attribute: strong TreeKeys : dict_keys(['strong', 'weak'])

Accuracy is : 0.75
```

## Source Code:

```
import numpy as np
X=np.array(([2,9],[1,5],[3,6]),dtype=float)#Features
y=np.array(([92],[86],[89]),dtype=float)#Labels(Marks obtained)
c=np.amax(X,axis=0)#Normalize
print(c)
X=X/c#Normalize
y=y/100
print(X)
print(y)

def sigmoid(x):
   return 1/(1+np.exp(-x))

def sigmoid_grad(x):
   return x*(1-x)

#variable declaration
epoch=1000#Setting training iteration
eta=0.1#Setting learning rate(eta)
input_neurons=2#number of features in data set
hidden_neurons=3#number of hidden layers neurons
output_neurons=1#number of neurons at output layer

#weight and Random Initialization
wh=np.random.uniform(size=(input_neurons,hidden_neurons))#2x3
print(wh)
bh=np.random.uniform(size=(1,hidden_neurons))#1x3
print(bh)
wout=np.random.uniform(size=(hidden_neurons,output_neurons))#3x1
print(wout)
bout=np.random.uniform(size=(1,output_neurons))
print(bout)

for i in range(epoch):
   #forward Propagation
   h_ip=np.dot(X,wh)+bh#Dot product +bais
   print(h_ip)
   h_act=sigmoid(h_ip)
   h_act=sigmoid(h_ip)#Activation function
   o_ip=np.dot(h_act,wout)+bout
   output=sigmoid(o_ip)
```

```
#Backpropagation
#Error at the output layer
Eo=y-output#Error at o/p
outgrad=sigmoid_grad(output)
d_output=Eo*outgrad#Errj=0j(1-0j)(Tj=0j)
print("The d_output is ",d_output)

#Error at hidden layer
Eh=d_output.dot(wout.T)#T means transpose
hiddengrad=sigmoid_grad(h_act)#How much hidden layer wts
d_hidden=Eh+hiddengrad
wout+=h_act.T.dot(d_output)*eta#dataproduct of nextlayer error
wh+=X.T.dot(d_hidden)*eta

print("Normalized Input :\n"+str(X))
print("Actual Output:\n"+str(y))
print("Predicted Output: \n",output)
```

## Output:

```
[3. 9.]
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
[[0.92]
 [0.86]
 [0.89]]
[[0.6240161  0.22801234 0.74083257]
 [0.47176461 0.62605224 0.71910698]]
[[0.70122927 0.49799934 0.52608325]]
[[0.69084241]
 [0.70017139]
 [0.73341769]]
[[0.18875553]]
[[1.58900461 1.2760598  1.73907861]
 [1.17132608 0.92181025 1.17253132]
 [1.63975511 1.14337984 1.74632048]]
The d_output is  [[0.00511539]
 [0.00057706]
 [0.00195331]]
Normalized Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
```

```
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.87365679]
 [0.85533638]
 [0.87244738]]
[[1.64225627 1.34170621 1.78910423]
 [1.19982146 0.95691466 1.19930319]
 [1.69158941 1.20763141 1.79497289]]
The d_output is  [[0.00481775]
 [0.00036899]
 [0.00169713]]

[[0.66666667 1.         ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.87573014]
 [0.85698931]
 [0.87453293]]
[[1.69389144 1.40531851 1.83765336]
 [1.22745313 0.99093244 1.22528616]
 [1.74183088 1.26987548 1.84217063]]
The d_output is   [[0.00454882]
 [0.00017629]
 [0.00146631]]

[[0.66666667 1.         ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.877641  ]
 [0.85854839]
 [0.87645804]]
[[1.74398192 1.46695559 1.8847944 ]
 [1.25425943 1.02389506 1.25051673]
 [1.79055068 1.33016981 1.88798142]]
The d_output is   [[ 4.30532569e-03]
 [-2.40097486e-06]
 [ 1.25793208e-03]]
```

```
Normalized Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.8794039 ]
 [0.86001994]
 [0.8782367 ]]
[[1.7925971  1.52668143 1.93059259]
 [1.28027741 1.05583673 1.27502975]
 [1.83781753 1.38857735 1.93246986]]
The d_output is   [[ 0.00408441]
 [-0.00016831]
 [ 0.0010694 ]]
Normalized Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.88103208]
 [0.86140987]
 [0.8798817 ]]
[[1.83980373 1.58456336 1.97511008]
 [1.30554272 1.08679342 1.2988584 ]
 [1.88369755 1.44516439 1.97569745]]
The d_output is   [[ 0.00280457]
 [-0.00279779]
 [-0.0001014 ]]
Normalized Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89109928]
 [0.88816787]
 [0.89104441]]
[[7.0654943  7.18340166 7.16189728]
 [4.11146528 4.09158508 4.0844143 ]
 [6.82607438 6.76283411 6.87344369]]
```

```
The d_output is   [[ 0.00280471]
 [-0.00279789]
 [-0.00010128]]
Normalized Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89109804]
 [0.88816906]
 [0.89104323]]
[[7.06710336 7.1850321   7.16352858]
 [4.11233159 4.09246283 4.08529269]
 [6.82756134 6.76434189 6.87494959]]
The d_output is   [[ 0.00280486]
 [-0.00279798]
 [-0.00010117]]
Normalized Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.8910968 ]
 [0.88817025]
 [0.89104206]]
```

## Source Code:

```
import csv
import random
import math

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):
   trainSize=int(len(dataset)*splitRatio)
   trainSet=[]
   copy=list(dataset)

   while len(trainSet)<trainSize:
      index=random.randrange(len(copy))
      trainSet.append(copy.pop(index))
   return [trainSet,copy]

def separateByClass(dataset):
   separated={}
   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):
    summaries=[(mean(attribute),stdev(attribute)) for attribute in zip(*dataset)]
    del summaries[-1]
    return summaries

def summarizeByClass(dataset):
    separated=separateByClass(dataset)
    summaries={}
    for classValue,instances in separated.items():
        summaries[classValue]=summarize(instances)
    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={}
    for classValue,classSummaries in summaries.items():
        probabilities[classValue]=1
        for i in range(len(classSummaries)):
            mean,stdev=classSummaries[i]
            x=inputVector[i]
            probabilities[classValue]*=calculateProbability(x,mean,stdev)
    return probabilities

def predict(summaries,inputVector):
    probabilities=calculateClassProbabilities(summaries,inputVector)
    bestLabel,bestProb=None,-1
    for classValue,probability in probabilities.items():
        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='pima-indians-diabetes.csv'
    splitRatio=0.80
    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)

    #test model
    predictions=getPredictions(summaries,testSet)
    accuracy=getAccuracy(testSet,predictions)
    print('Accuracy:{0}%'.format(accuracy))

main()
```

## Sample Dataset:

```
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1
5,116,74,0,0,25.6,0.201,30,0
3,78,50,32,88,31,0.248,26,1
10,115,0,0,0,35.3,0.134,29,0
2,197,70,45,543,30.5,0.158,53,1
8,125,96,0,0,0,0.232,54,1
4,110,92,0,0,37.6,0.191,30,0
10,168,74,0,0,38,0.537,34,1
10,139,80,0,0,27.1,1.441,57,0
1,189,60,23,846,30.1,0.398,59,1
5,166,72,19,175,25.8,0.587,51,1
```

*7,100,0,0,0,30,0.484,32,1*
*0,118,84,47,230,45.8,0.551,31,1*
*7,107,74,0,0,29.6,0.254,31,1*
*1,103,30,38,83,43.3,0.183,33,0*
*1,115,70,30,96,34.6,0.529,32,1*
*3,126,88,41,235,39.3,0.704,27,0*
*8,99,84,0,0,35.4,0.388,50,0*
*7,196,90,0,0,39.8,0.451,41,1*
*9,119,80,35,0,29,0.263,29,1*
*11,143,94,33,146,36.6,0.254,51,1*
*10,125,70,26,115,31.1,0.205,41,1*
*7,147,76,0,0,39.4,0.257,43,1*
*1,97,66,15,140,23.2,0.487,22,0*
*13,145,82,19,110,22.2,0.245,57,0*
*5,117,92,0,0,34.1,0.337,38,0*
*5,109,75,26,0,36,0.546,60,0*
*3,158,76,36,245,31.6,0.851,28,1*
*3,88,58,11,54,24.8,0.267,22,0*
*6,92,92,0,0,19.9,0.188,28,0*
*10,122,78,31,0,27.6,0.512,45,0*
*4,103,60,33,192,24,0.966,33,0*
*11,138,76,0,0,33.2,0.42,35,0*
*9,102,76,37,0,32.9,0.665,46,1*
*2,90,68,42,0,38.2,0.503,27,1*
*4,111,72,47,207,37.1,1.39,56,1*
*3,180,64,25,70,34,0.271,26,0*
*7,133,84,0,0,40.2,0.696,37,0*
*7,106,92,18,0,22.7,0.235,48,0*
*9,171,110,24,240,45.4,0.721,54,1*
*7,159,64,0,0,27.4,0.294,40,0*
*0,180,66,39,0,42,1.893,25,1*
*1,146,56,0,0,29.7,0.564,29,0*
*2,71,70,27,0,28,0.586,22,0*
*7,103,66,32,0,39.1,0.344,31,1*
*7,105,0,0,0,0,0.305,24,0*
*...Similarly 700 entries*

## Output:

```
Split 768 rows into train=614 and test=154
rows Accuracy:72.07792207792207%
```