----------1----------

```
import csv
a=[]
with open('enjoysport.csv')as trainData:
   for row in csv.reader(trainData):
      a.append(row)
      print(row)
n=len(a[0])-1
S=['0']*n
print("Initial hypothesis ",S)
print("FIND S ALGORITHM")
S=a[0][:-1]
for i in range(len(a)):
   if a[i][n]=="yes":
      for j in range(n):
         if a[i][j]!=S[j]:
            S[j]='?'
   print("\nTraining example no {0}, Hypothesis is".format(i+1),S)
print("\nMaximally specific hypothesis is ", S )
```

----------2----------

```
import csv
a = []
with open('enjoySport.csv', 'r') as trainData:
   for row in csv.reader(trainData):
      a.append(row)
      print(row)
n=len(a[0])-1

print("\n The initial value of hypothesis: ")
s = ['0'] * n
g = ['?'] * n
print ("\n The most specific hypothesis S0 :",s)
print (" \n The most general hypothesis G0 :",g)

s=a[0][:-1]
temp=[]
print("\n Candidate Elimination algorithm\n")

for i in range(len(a)):
   if a[i][n]=="yes": #Use Positive for manufacture.csv
      for j in range(n):
         if a[i][j]!=s[j]:
            s[j]='?'
      for j in range(n):
         for k in range(len(temp)):  #Use len(temp)-1 for manufacture.csv
            if temp[k][j]!='?' and temp[k][j]!=s[j]:
               del temp[k]

   if a[i][n]=="no": #Use Negative for manufacture.csv
      for j in range(n):
          if s[j]!=a[i][j] and s[j]!='?':
             g[j]=s[j]
             if g not in temp:
              temp.append(g)
             g= ['?']*n

   print("\n For Training Example No :{0} the hypothesis is S{0} ".format(i+1),s)
   if (len(temp)==0):
         print(" For Training Example No :{0} the hypothesis is G{0} ".format(i+1),g)
   else:
         print(" For Training Example No :{0} the hypothesis is G{0}".format(i+1),temp)
```

----------3----------

```
import pandas as pd
from math import log
from pprint import pprint

df = pd.read_csv('tennis.csv')
data = df.values.tolist()
attr_names = df.columns.values.tolist()
print(df)

def entropy(pos, neg):
    if pos == 0 or neg == 0:
        return 0
    tot = pos + neg
    return -pos / tot * log(pos / tot, 2) - neg / tot * log(neg / tot, 2)

def gain(data, attr, pos, neg):
    d, E, acu = {}, entropy(pos, neg), 0
    for i in data:
        if i[attr] not in d:
            d[i[attr]] = {}
        d[i[attr]][i[-1]] = 1 + d[i[attr]].get(i[-1], 0)
    for i in d:
        tot = d[i].get('Yes', 0) + d[i].get('No', 0)
        acu += tot / (pos + neg) * entropy(d[i].get('Yes', 0), d[i].get('No', 0))
    return E - acu


def build(data, attr_names):
    pos, sz = len([x for x in data if x[-1] == 'Yes']), len(data[0]) - 1
    neg = len(data) - pos
    if neg == 0 or pos == 0:
        return 'Yes' if neg == 0 else 'No'

    root = max([[gain(data, i, pos, neg), i] for i in range(sz)])[1]
    fin, res = {}, {}
    uniq_attr = set([x[root] for x in data])
    for i in uniq_attr:
        res[i] = build([x[:root] + x[root + 1:] for x in data if x[root] == i], attr_names[:root] +
attr_names[root+1:])
    fin[attr_names[root]] = res
    return fin

tree = build(data, attr_names)
pprint(tree)
```

----------4----------

```
import numpy as np
inputNeurons=10
hiddenlayerNeurons=5
outputNeurons=2

input  = np.random.randint(1,100,inputNeurons)
output = np.array([1.0,0.0])
hidden_layer=np.random.rand(1,hiddenlayerNeurons)
print(input)
hidden_biass=np.random.rand(1,hiddenlayerNeurons)
output_bias=np.random.rand(1,outputNeurons)
hidden_weights=np.random.rand(inputNeurons,hiddenlayerNeurons)
output_weights=np.random.rand(hiddenlayerNeurons,outputNeurons)

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

def gradient(layer):
   return layer*(1-layer)

for i in range(1000):

   hidden_layer=np.dot(input,hidden_weights)
   hidden_layer=sigmoid(hidden_layer+hidden_biass)

   output_layer=np.dot(hidden_layer,output_weights)
   output_layer=sigmoid(output_layer+output_bias)

   error = (output-output_layer)
   gradient_outputLayer=gradient(output_layer)

   error_terms_output=gradient_outputLayer * error

   error_terms_hidden=gradient(hidden_layer)*np.dot(error_terms_output,output_weights.T)

   gradient_hidden_weights =
np.dot(input.reshape(inputNeurons,1),error_terms_hidden.reshape(1,hiddenlayerNeurons))
   gradient_ouput_weights =
np.dot(hidden_layer.reshape(hiddenlayerNeurons,1),error_terms_output.reshape(1,outputNeurons))

   hidden_weights = hidden_weights + 0.05*gradient_hidden_weights
   output_weights = output_weights + 0.05*gradient_ouput_weights
   print("**********************")
   print("Iteration:",i,"::::",error)
   print("Output::::",output_layer)
```

----------5----------

```
from pprint import pprint
import pandas as pd
df_golf = pd.read_csv("golf.csv")
print(df_golf)
attribute_names = list(df_golf.columns)
print("List of Attributes:", attribute_names)
attribute_names.remove('label')
print("Predicting Attributes:", attribute_names)
table=dict()
priorProb=dict()
train=df_golf.sample(frac=0.6,random_state=100) #random state is a seed value
test=df_golf.drop(train.index)
print("---------------------------------")
print(train)
print("---------------------------------")
print(test)
print("---------------------------------")
for attr_val, data_subset in train.groupby("label"):
   from collections import Counter
   valueCount = dict()
   count=0
   for attr_value in attribute_names:
      cnt = Counter(x for x in data_subset[attr_value])
      count=sum(cnt.values())
      valueCount[attr_value]=dict(cnt)
      print("value count", valueCount.values())
      print("counter:-",cnt)
   table[attr_val]=valueCount
   priorProb[attr_val]=count
print("*****************************************")
print("\n\nThe Resultant table is :\n")
pprint(table)
pprint(priorProb)
totalSize=test['label'].count()
correctPridictions=0
for k, row in test.iterrows():
   rowTuple=dict(row)
   print("print row tuple")
   pprint(rowTuple)
   postioriList=list()
   labelList=list()
   for label in table.keys():
      posteriori = 1.0
      print("RowTuple",rowTuple.keys())
      print("RowValues",rowTuple.values())
      for key in [x for x in rowTuple.keys() if x != 'label']:
         print(key, "label:",label)
```

```
            attributeValue=rowTuple.get(key)
            if attributeValue in table[label][key].keys():
                countList=table[label][key].values()
                attributeCount=table[label][key][attributeValue]
                posteriori=1.0*attributeCount/sum(countList)*posteriori
        posteriori=posteriori*priorProb[label]
        labelList.append(label)
        postioriList.append(posteriori)
        print(labelList)
        print(postioriList)
    maxProbInd = postioriList.index(max(postioriList))
    print(rowTuple['label'], "::::", labelList[maxProbInd])
    if rowTuple['label'] == labelList[maxProbInd]:
        print(rowTuple['label'],"::::",labelList[maxProbInd])
        correctPridictions=correctPridictions+1
        print("POSTERIORI OF:",label,"is:",posteriori)
print("Number of Correct Predictions : Number of Samples",correctPridictions,":",totalSize)
print("Accuracy:",100.0*correctPridictions/totalSize)
```

----------6----------

```
import pandas as pd
df_imdb = pd.read_csv("imdb_labelled.txt",sep='\t',index_col=None)
print(df_imdb.keys())
print(df_imdb)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_imdb['Text'],
df_imdb['Label'],train_size=0.8,test_size=0.2,random_state=100)

from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
X_train_cv = cv.fit_transform(X_train)
X_test_cv = cv.transform(X_test)

from sklearn.naive_bayes import MultinomialNB
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train_cv, y_train)
predictions = naive_bayes.predict(X_test_cv)

from sklearn.metrics import accuracy_score, precision_score, recall_score
print('Accuracy score: ', accuracy_score(y_test, predictions))
print('Precision score: ', precision_score(y_test, predictions))
print('Recall score: ', recall_score(y_test, predictions))
```

----------7----------

```
names = "A,B,C,D,E,F,G,H,I,J,K,L,M,RESULT"
names = names.split(",")
print(names)

import pandas as pd
data = pd.read_csv("datasetheart.csv",names = names)
print(data.head(5))
print(data.tail(5))

from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
model = BayesianModel([("A","B"),("B","C"),("C","D"),("D","RESULT")])
model.fit(data,estimator=MaximumLikelihoodEstimator)

from pgmpy.inference import VariableElimination
infer = VariableElimination(model)
q = infer.query(variables=['RESULT'],evidence={"B":1})
print(q['RESULT'])
```

----------8----------

```
from sklearn import datasets,metrics
iris = datasets.load_iris()

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(iris.data,iris.target)

from sklearn.cluster import KMeans
model1 =KMeans(n_clusters=3)
model1.fit(X_train,y_train)
print("K-means: ",metrics.accuracy_score(y_test,model1.predict(X_test)))

from sklearn.mixture import GaussianMixture
model2 = GaussianMixture(n_components=3)
model2.fit(X_train,y_train)
print("EM: ",metrics.accuracy_score(y_test,model2.predict(X_test)))
```

----------9----------

```
from sklearn import datasets
iris = datasets.load_iris()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(iris.data,iris.target,train_size=0.8,test_size=0.2,random_state=100)

from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train,y_train)
predicted= model.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, predicted))
print(classification_report(y_test, predicted))
```

----------10----------

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 1000)
y = np.log(np.abs((x ** 2) - 1) + 0.5)
x = x + np.random.normal(scale=0.05, size=1000)
plt.scatter(x, y, alpha=0.3)

def local_regression(x0, x, y, tau):
   x0 = np.r_[1, x0]
   x = np.c_[np.ones(len(x)), x]
   xw = x.T * radial_kernel(x0, x, tau)
   beta = np.linalg.pinv(xw @ x) @ xw @ y
   return x0 @ beta

def radial_kernel(x0, x, tau):
   return np.exp(np.sum((x - x0) ** 2, axis=1) / (-2 * tau ** 2))

def plot_lr(tau):
   domain = np.linspace(-5, 5, num=300)
   pred = [local_regression(x0, x, y, tau) for x0 in domain]
   plt.scatter(x, y, alpha=0.3)
   plt.plot(domain, pred, color="blue")
   return plt

print(plot_lr(3).show())
```
