**Find-S**

H=[0,0,0,0,0,0]

import pandas as pd

import numpy as np

df=pd.read\_csv("finds.csv",sep=",",header=None)

print(df)

attribute=np.array(df)[:,:-1]

print(attribute)

target=np.array(df)[:,-1]

print(target)

for i in range(len(df)):

    for j in range(len(df.columns)-1):

        if df.iloc[i,-1]=="Yes":

            if H[j]==0:

                H[j]=df.iloc[i,j]

            elif df.iloc[i,j]!=H[j]:

                H[j]='?'

print(H)

**Candidate Elimination**

import pandas as pd

df=pd.read\_csv("finds.csv",sep=",",header=None)

#Intialize S and G

S=[0,0,0,0,0,0]

G=list()

for i in range(len(df.columns)-1):

    G.append(['?','?','?','?','?','?'])

#Read samples

for i in range(len(df)):

    for j in range(len(df.columns)-1):

        if df.iloc[i,-1]=="Yes":

            if S[j]==0:

                S[j]=df.iloc[i,j]

            elif df.iloc[i,j]!=S[j]:

                S[j]="?"

            if G[j][j]!='?' and S[j]=='?':

                G[j][j]='?'

        else:

            if df.iloc[i,j]!=S[j] and S[j]!='?':

                G[j][j]=S[j]

print(S)

print(G)

**Decision Tree**

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

from sklearn import tree

# Load the data

df = pd.read\_csv("decision\_tree1.csv", sep=",")

# Encode categorical features as numeric codes

for col in df.columns:

    if df[col].dtype == 'object':

        df[col] = df[col].astype('category')

        df[col] = df[col].cat.codes

# Prepare the feature matrix and target vector

x = df.iloc[:, :-1]

y = df.iloc[:, -1]

# Train the Decision Tree classifier

model = DecisionTreeClassifier(criterion="entropy")

clf = model.fit(x, y)

# Plot the tree

plt.figure(figsize=(6,4))

tree.plot\_tree(clf, feature\_names=x.columns.tolist(), filled=True)

plt.show()

# Verify the number of features in the training data

print(f"Number of features in training data: {x.shape[1]}")

# Adjust new\_data to have the same number of features

# Replace the list with the appropriate number of features for your dataset

new\_data = [[1, 0, 0, 1]]  # Example adjustment

# Predict using new data

ypred = clf.predict(new\_data)

print(ypred)

**Naive Bayes**

import pandas as pd

df=pd.read\_csv("iris1.csv")

x=df.iloc[:,:-1]

y=df.iloc[:,-1]

y=y.astype('category')

y=y.cat.codes

from sklearn.model\_selection import train\_test\_split

xtrain,xtest,ytrain,ytest=train\_test\_split(x,y,test\_size=0.3)

from sklearn.naive\_bayes import GaussianNB

g=GaussianNB()

g.fit(xtrain,ytrain)

ypred=g.predict(xtest)

from sklearn.metrics import accuracy\_score

print(accuracy\_score(ytest,ypred))

**Bayesian Belief Network**

import pandas as pd

import numpy as np

from pgmpy.inference import VariableElimination

from pgmpy.models import BayesianModel

from pgmpy.estimators import MaximumLikelihoodEstimator

df=pd.read\_csv("Medical Dataset.csv")

df=df.replace("?",np.nan)

model=BayesianModel([('age','heartdisease'),('sex','heartdisease'),

('exang','heartdisease'),('cp','heartdisease'),

('heartdisease','restecg'),('heartdisease','chol')])

model.fit(df,estimator=MaximumLikelihoodEstimator)

infer=VariableElimination(model)

q=infer.query(variables=['heartdisease'],evidence={'restecg':1})

print(q)

**E-M Algorithm**

import numpy as np

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

df=pd.read\_csv('Iris.csv')

x=df.iloc[:,:-1]

y=df.iloc[:,-1]

colormap=np.array(["red","green","blue"])

y=y.astype('category')

y=y.cat.codes

gm=GaussianMixture(n\_components=3)

gm.fit(x)

gmc=gm.predict(x)

km=KMeans(n\_clusters=3)

km.fit(x)

kmc=km.predict(x)

import matplotlib.pyplot as plt

plt.subplot(1,3,1)

plt.scatter(x.iloc[:,0],x.iloc[:,1],c=colormap[y],s=40)

plt.subplot(1,3,2)

plt.scatter(x.iloc[:,0],x.iloc[:,1],c=colormap[gmc],s=40)

plt.subplot(1,3,3)

plt.scatter(x.iloc[:,0],x.iloc[:,1],c=colormap[kmc],s=40)

plt.show()

**K-Neighbor/Nearest**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

df=pd.read\_csv('Iris.csv')

x=df.iloc[:,:-1]

y=df.iloc[:,-1]

y=y.astype('category')

y=y.cat.codes

xtrain,xtest,ytrain,ytest=train\_test\_split(x,y,test\_size=0.3)

kn=KNeighborsClassifier(n\_neighbors=3)

kn.fit(xtrain,ytrain)

ypred=kn.predict(xtest)

i=0

for label in ytest:

    if label==ypred[i]:

        print('Correct',label)

    else:

        print('Incorrect',label,ypred[i])

**Regression**

import numpy as np

import matplotlib.pyplot as plt

# Locally Weighted Regression (LWR) Function

def locally\_weighted\_regression(x\_query, X, y, tau=0.1):

    X = np.array(X)

    y = np.array(y)

    x\_query = np.array(x\_query)

    # Gaussian kernel function

    kernel\_weights = np.exp(-(X - x\_query)\*\*2 / (2 \* tau\*\*2))

    W = np.diag(kernel\_weights)

    # Add an intercept term to X for the constant coefficient

    X\_design = np.vstack([X, np.ones\_like(X)]).T

    # Perform locally weighted linear regression

    theta = np.linalg.inv(X\_design.T @ W @ X\_design) @ (X\_design.T @ W @ y)

    # Prediction at x\_query

    y\_query = np.array([x\_query, 1]).T @ theta

    return y\_query

# Generate Synthetic Data

np.random.seed(0)

X = np.linspace(0, 10, 100)

y = np.sin(X) + np.random.normal(0, 0.1, X.shape)

# Apply LWR and Predict

x\_queries = np.linspace(0, 10, 100)

y\_pred = [locally\_weighted\_regression(x, X, y, tau=0.5) for x in x\_queries]

# Plot Results

plt.figure(figsize=(6, 4))

plt.scatter(X, y, color='blue', label='Data Points')

plt.plot(x\_queries, y\_pred, color='red', label='LWR Fit (tau=0.5)')

plt.title('Locally Weighted Regression')

plt.xlabel('X')

plt.ylabel('Y')

plt.legend()

plt.grid(True)

plt.show()

**SVM**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.pipeline import make\_pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.svm import LinearSVC

from sklearn.metrics import accuracy\_score

iris=pd.read\_csv("Iris.csv")

x=iris.iloc[:,:-1]

y=iris.iloc[:,-1]

y=y.astype('category')

y=y.cat.codes

xtrain,xtest,ytrain,ytest=train\_test\_split(x,y,test\_size=0.3)

svmclf=make\_pipeline(StandardScaler(),LinearSVC(C=15))

svmclf.fit(xtrain,ytrain)

ypred=(svmclf.predict(xtest))

acc=accuracy\_score(ytest,ypred)

print(acc)

**Back Propogation**

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

def sigmoid (x):

    return 1/(1 + np.exp(-x))

def derivatives\_sigmoid(x):

    return x \* (1 - x)

epoch=5

lr=0.1

inputlayer\_neurons = 2

hiddenlayer\_neurons = 3

output\_neurons = 1

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))

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)

    d\_hiddenlayer = EH \* hiddengrad

    wout += hlayer\_act.T.dot(d\_output) \*lr

    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)