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

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

data = pd.read\_csv('D:/Machine Learning/iris1.csv')

print(data.head())

X = data.iloc[:, :-1]

y = data.iloc[:, -1]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = GaussianNB()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy \* 100:.2f}%')

**OUTPUT:**

Accuracy: 91.30%

1. **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 data set.**

import pandas as pd

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

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn import metrics

# Load the dataset, skipping the first row if it contains headers

file\_path = 'text\_labels.csv'

msg = pd.read\_csv(file\_path, skiprows=1, names=['message', 'label'])

# Check for missing values

missing\_values = msg.isnull().sum()

print(f"Missing values:\n{missing\_values}")

# Map labels to numerical values

msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})

# Extract features and labels

X = msg.message

y = msg.labelnum

# Split the dataset into train and test data

xtrain, xtest, ytrain, ytest = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

print('The total number of Training Data:', ytrain.shape)

print('The total number of Test Data:', ytest.shape)

# Vectorize the text data

count\_vect = CountVectorizer()

xtrain\_dtm = count\_vect.fit\_transform(xtrain)

xtest\_dtm = count\_vect.transform(xtest)

print('\nThe words or Tokens in the text documents\n')

print(count\_vect.get\_feature\_names\_out())

# Train the Naive Bayes classifier

clf = MultinomialNB().fit(xtrain\_dtm, ytrain)

predicted = clf.predict(xtest\_dtm)

# Evaluate the classifier

print('\nAccuracy of the classifier is', metrics.accuracy\_score(ytest, predicted))

print('\nConfusion matrix')

print(metrics.confusion\_matrix(ytest, predicted))

print('\nThe value of Precision', metrics.precision\_score(ytest, predicted))

print('\nThe value of Recall', metrics.recall\_score(ytest, predicted))

**OUTPUT**

Accuracy of the classifier is 1.0

Confusion matrix

[[2 0]

[0 2]]

The value of Precision 1.0

The value of Recall 1.0

1. **Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.**

import numpy as np

import pandas as pd

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

# Read Cleveland Heart Disease data

heartDisease = pd.read\_csv('heart.csv')

heartDisease = heartDisease.replace('?', np.nan)

# Display the data

print('Sample instances from the dataset are given below')

print(heartDisease.head())

# Display the attribute names and data types

print('\nAttributes and datatypes')

print(heartDisease.dtypes)

# Create Bayesian Network model

model = BayesianModel([

('age', 'heartdisease'),

('sex', 'heartdisease'),

('exang', 'heartdisease'),

('cp', 'heartdisease'),

('heartdisease', 'restecg'),

('heartdisease', 'chol')

])

# Learning CPDs using Maximum Likelihood Estimators

print('\nLearning CPD using Maximum likelihood estimators')

model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

# Inferencing with Bayesian Network

print('\nInferencing with Bayesian Network:')

HeartDiseasetest\_infer = VariableElimination(model)

# Computing the Probability of HeartDisease given restecg

print('\n1. Probability of HeartDisease given evidence= restecg: 1')

q1 = HeartDiseasetest\_infer.query(variables=['heartdisease'], evidence={'restecg': 1})

print(q1)

# Computing the Probability of HeartDisease given cp

print('\n2. Probability of HeartDisease given evidence= cp: 2')

q2 = HeartDiseasetest\_infer.query(variables=['heartdisease'], evidence={'cp': 2})

print(q2)

**OUTPUT**

1. Probability of HeartDisease given evidence= restecg: 1

+-----------------+---------------------+

| heartdisease | phi(heartdisease) |

+=================+=====================+

| heartdisease(0) | 0.1016 |

+-----------------+---------------------+

| heartdisease(1) | 0.0000 |

+-----------------+---------------------+

| heartdisease(2) | 0.2361 |

+-----------------+---------------------+

| heartdisease(3) | 0.2017 |

+-----------------+---------------------+

| heartdisease(4) | 0.4605 |

+-----------------+---------------------+

2. Probability of HeartDisease given evidence= cp: 2

+-----------------+---------------------+

| heartdisease | phi(heartdisease) |

+=================+=====================+

| heartdisease(0) | 0.3742 |

+-----------------+---------------------+

| heartdisease(1) | 0.2018 |

+-----------------+---------------------+

| heartdisease(2) | 0.1375 |

+-----------------+---------------------+

| heartdisease(3) | 0.1541 |

+-----------------+---------------------+

| heartdisease(4) | 0.1323 |

+-----------------+---------------------+

1. **Implement an algorithm to demonstrate Polynomial Classifier.**

import pandas as pd

import numpy as np

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

from sklearn.pipeline import make\_pipeline

import matplotlib.pyplot as plt

# Load the data

file\_path = 'data (1).csv'

data = pd.read\_csv(file\_path)

# Display the first few rows of the dataset

print("Dataset Head:\n", data.head())

# Extract features and target variable

X = data[['Temperature']].values

y = data['Pressure'].values

# Define the polynomial degree

degree = 2

# Create a pipeline that transforms the features to polynomial features and then fits a

model = make\_pipeline(PolynomialFeatures(degree), LinearRegression())

# Fit the model

model.fit(X, y)

# Predict the pressure values

y\_pred = model.predict(X)

# Plot the original data and the polynomial regression curve

plt.scatter(X, y, color='blue', label='Original data')

plt.plot(X, y\_pred, color='red', label='Polynomial regression')

plt.xlabel('Temperature')

plt.ylabel('Pressure')

plt.title('Polynomial Regression of Pressure vs. Temperature')

plt.legend()

plt.show()

# Display the model coefficients and intercept

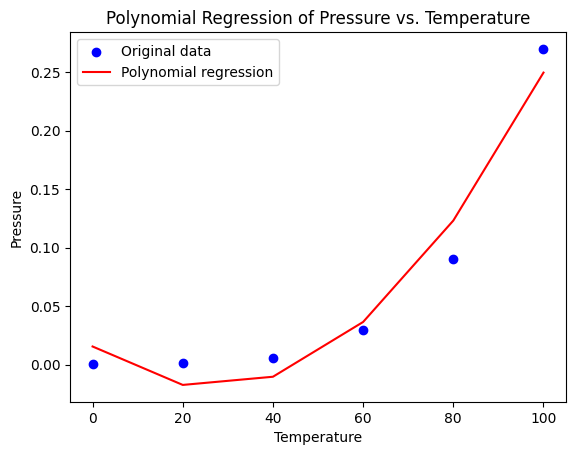
coef = model.named\_steps['linearregression'].coef\_

intercept = model.named\_steps['linearregression'].intercept\_

print(f"Model Coefficients: {coef}")

print(f"Model Intercept: {intercept}")

**OUTPUT**



Model Coefficients: [ 0.00000e+00 -2.63925e-03 4.98125e-05]

Model Intercept: 0.015550000000003172

1. **Build an Artificial Neural Network by implementing the Backpropagation**

**algorithm and test the same using appropriate data sets.**

import numpy as np

# Data

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) # Normalizing inputs

y = y / 100 # Normalizing outputs

# Sigmoid Function

def sigmoid(x):

return 1 / (1 + np.exp(-x))

# Derivative of Sigmoid Function

def derivatives\_sigmoid(x):

return x \* (1 - x)

# Variable initialization

epoch = 5000 # Setting training iterations

lr = 0.1 # Setting learning rate

inputlayer\_neurons = 2 # Number of features in dataset

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))

# Training algorithm

for i in range(epoch):

# Forward Propagation

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

# Updating weights and biases

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

wh += X.T.dot(d\_hiddenlayer) \* lr

# Results

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n", output)

**OUTPUT**

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.89510403]

[0.88368551]

[0.89148676]]

1. **Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.**

from sklearn.cluster import KMeans

from sklearn import preprocessing

from sklearn.mixture import GaussianMixture

from sklearn.datasets import load\_iris

import sklearn.metrics as sm

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

dataset=load\_iris()

# print(dataset)

X=pd.DataFrame(dataset.data)

X.columns=['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width']

y=pd.DataFrame(dataset.target)

y.columns=['Targets']

# print(X)

plt.figure(figsize=(14,7))

colormap=np.array(['red','lime','black'])

# REAL PLOT

plt.subplot(1,3,1)

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[y.Targets],s=40)

plt.title('Real')

# K-PLOT

plt.subplot(1,3,2)

model=KMeans(n\_clusters=3)

model.fit(X)

predY=np.choose(model.labels\_,[0,1,2]).astype(np.int64)

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[predY],s=40)

plt.title('KMeans')

# GMM PLOT

scaler=preprocessing.StandardScaler()

scaler.fit(X)

xsa=scaler.transform(X)

xs=pd.DataFrame(xsa,columns=X.columns)

gmm=GaussianMixture(n\_components=3)

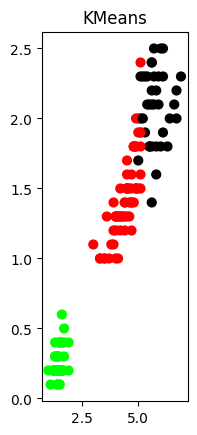
gmm.fit(xs)

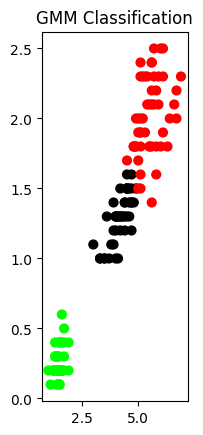
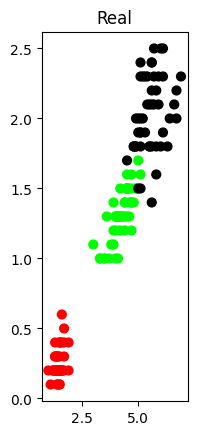
y\_cluster\_gmm=gmm.predict(xs)

plt.subplot(1,3,3)

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[y\_cluster\_gmm],s=40)

plt.title('GMM Classification')

**OUTPUT**



1. **Write a program to implement k-Nearest Neighbor algorithm to classify the iris**

**data set. 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.model\_selection import train\_test\_split

import numpy as np

dataset=load\_iris()

#print(dataset)

X\_train,X\_test,y\_train,y\_test=train\_test\_split(dataset["data"],dataset["target"],random\_state=0)

kn=KNeighborsClassifier(n\_neighbors=1)

kn.fit(X\_train,y\_train)

for i in range(len(X\_test)):

x=X\_test[i]

x\_new=np.array([x])

prediction=kn.predict(x\_new)

print("TARGET=",y\_test[i],dataset["target\_names"][y\_test[i]],"PREDICTED=",prediction,dataset["target\_names"][prediction])

print(kn.score(X\_test,y\_test))

**OUTPUT**

TARGET= 1 versicolor PREDICTED= [2] ['virginica']

0.9736842105263158

1. **Implement an algorithm to demonstrate Decision Tree Classifier.**

# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn import datasets

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import matplotlib.pyplot as plt

from sklearn import tree

# Load the Iris dataset

iris = datasets.load\_iris()

X = iris.data

y = iris.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Create and train the Decision Tree classifier

clf = DecisionTreeClassifier(random\_state=42)

clf.fit(X\_train, y\_train)

# Make predictions on the testing set

y\_pred = clf.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

print('Classification Report:')

print(classification\_report(y\_test, y\_pred))

print('Confusion Matrix:')

print(confusion\_matrix(y\_test, y\_pred))

# Visualize the Decision Tree

plt.figure(figsize=(20,10))

tree.plot\_tree(clf, filled=True, feature\_names=iris.feature\_names, class\_names=iris.target\_names)

plt.show()

**OUTPUT**

Accuracy: 1.00

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 19

1 1.00 1.00 1.00 13

2 1.00 1.00 1.00 13

accuracy 1.00 45

macro avg 1.00 1.00 1.00 45

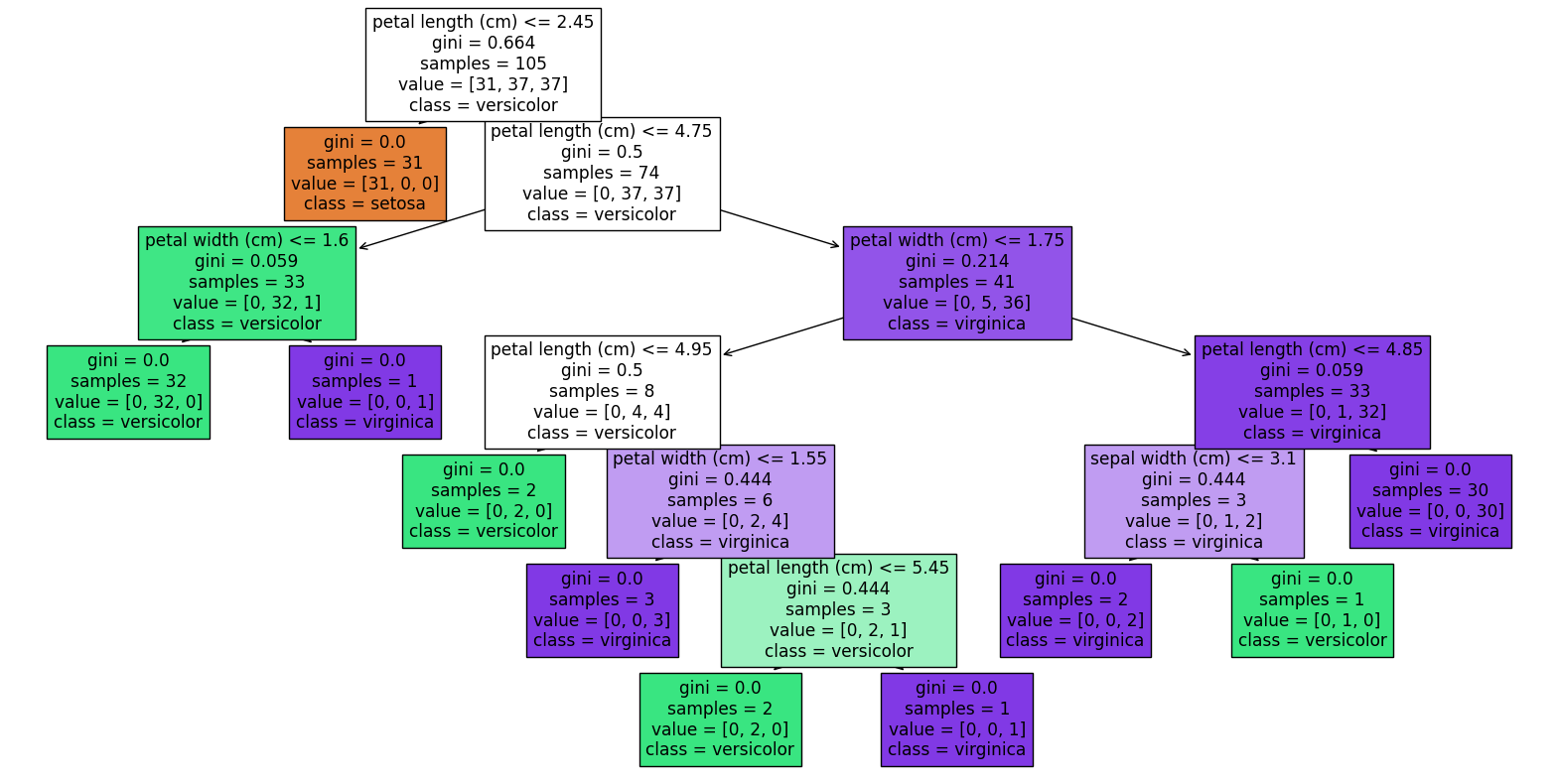
weighted avg 1.00 1.00 1.00 45

Confusion Matrix:

[[19 0 0]

[ 0 13 0]

[ 0 0 13]]



1. **Implement an algorithm to demonstrate the significance of Genetic Algorithm in python.**

import numpy as np

import random

import matplotlib.pyplot as plt

# Define the objective function

def objective\_function(x):

return x \* np.sin(10 \* np.pi \* x) + 1

# Generate initial population

def generate\_population(size, x\_min, x\_max):

population = np.random.uniform(x\_min, x\_max, size)

return population

# Calculate fitness of each individual

def calculate\_fitness(population):

fitness = objective\_function(population)

return fitness

# Select individuals based on fitness (roulette wheel selection)

def selection(population, fitness, num\_parents):

fitness\_sum = np.sum(fitness)

probabilities = fitness / fitness\_sum

parents = np.random.choice(population, size=num\_parents, p=probabilities)

return parents

# Crossover (single point)

def crossover(parents, offspring\_size):

offspring = np.empty(offspring\_size)

crossover\_point = np.uint8(offspring\_size[1]/2)

for k in range(offspring\_size[0]):

parent1\_idx = k % parents.shape[0]

parent2\_idx = (k+1) % parents.shape[0]

offspring[k, 0:crossover\_point] = parents[parent1\_idx, 0:crossover\_point]

offspring[k, crossover\_point:] = parents[parent2\_idx, crossover\_point:]

return offspring

# Mutation

def mutation(offspring, mutation\_rate):

for idx in range(offspring.shape[0]):

if np.random.rand() < mutation\_rate:

random\_value = np.random.uniform(-0.1, 0.1, 1)

offspring[idx] = offspring[idx] + random\_value

offspring[idx] = np.clip(offspring[idx], 0, 1)

return offspring

# Genetic Algorithm

def genetic\_algorithm(objective\_function, generations, population\_size, x\_min, x\_max, num\_parents, mutation\_rate):

population = generate\_population(population\_size, x\_min, x\_max)

for generation in range(generations):

fitness = calculate\_fitness(population)

parents = selection(population, fitness, num\_parents)

offspring\_size = (population\_size - parents.shape[0],)

offspring = crossover(parents.reshape(parents.shape[0], 1), (offspring\_size[0], 1)).flatten()

offspring = mutation(offspring, mutation\_rate)

population[:num\_parents] = parents

population[num\_parents:] = offspring

best\_fitness = np.max(calculate\_fitness(population))

print(f"Generation {generation}: Best Fitness = {best\_fitness}")

best\_solution\_idx = np.argmax(calculate\_fitness(population))

best\_solution = population[best\_solution\_idx]

return best\_solution

# Parameters

generations = 100

population\_size = 20

x\_min = 0

x\_max = 1

num\_parents = 10

mutation\_rate = 0.1

# Run Genetic Algorithm

best\_solution = genetic\_algorithm(objective\_function, generations, population\_size, x\_min, x\_max, num\_parents, mutation\_rate)

print(f"Best solution: x = {best\_solution}, f(x) = {objective\_function(best\_solution)}")

# Plot the objective function

x = np.linspace(0, 1, 1000)

y = objective\_function(x)

plt.plot(x, y, label="Objective Function")

plt.plot(best\_solution, objective\_function(best\_solution), 'ro', label="Best Solution")

plt.legend()

plt.xlabel("x")

plt.ylabel("f(x)")

plt.show()

**OUTPUT**

Generation 1: Best Fitness = 1.4507073452016166

Generation 2: Best Fitness = 1.4507073452016166

Generation 3: Best Fitness = 1.4507073452016166

Generation 4: Best Fitness = 1.4507073452016166

Generation 5: Best Fitness = 1.4507073452016166

Generation 6: Best Fitness = 1.4507073452016166

Generation 7: Best Fitness = 1.4507073452016166

Generation 8: Best Fitness = 1.4507073452016166

.

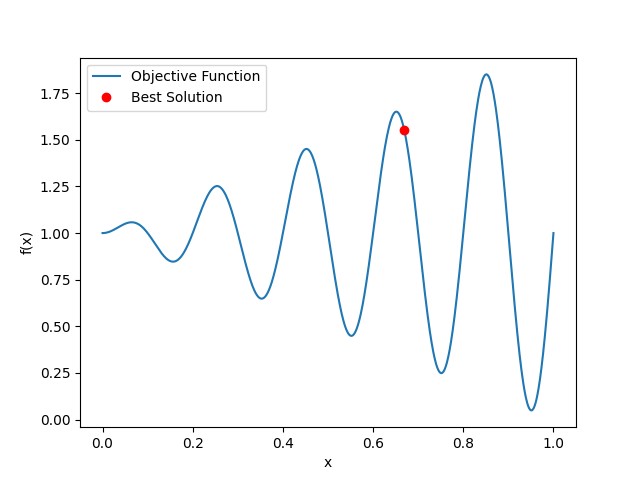
.

.

.

Generation 99: Best Fitness = 1.4499783217343818

Best solution: x = 0.45448766562581167, f(x) = 1.4499783217343818



1. **Case Study**

import pandas as pd

# Load the dataset

file\_path = 'iris1.csv'

data = pd.read\_csv(file\_path)

data.head()

from sklearn.naive\_bayes import GaussianNB, MultinomialNB, ComplementNB, BernoulliNB, CategoricalNB

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

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score

# Encode the target labels

le = LabelEncoder()

data['species'] = le.fit\_transform(data['species'])

# Split the data into training and test sets

X = data.drop(columns='species')

y = data['species']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize the classifiers

classifiers = {

'GaussianNB': GaussianNB(),

'MultinomialNB': MultinomialNB(),

'ComplementNB': ComplementNB(),

'BernoulliNB': BernoulliNB(),

'CategoricalNB': CategoricalNB()

}

# Train and evaluate each classifier

accuracies = {}

for name, clf in classifiers.items():

try:

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

accuracies[name] = accuracy\_score(y\_test, y\_pred) \* 100

except Exception as e:

accuracies[name] = str(e)

accuracies

sorted\_accuracies = dict(sorted(accuracies.items(), key=lambda item: item[1], reverse=True))

print(sorted\_accuracies)

best\_algorithm = max(accuracies, key=accuracies.get)

print(f'The best algorithm is {best\_algorithm} with an accuracy of {accuracies[best\_algorithm]:.2f}%')

**OUTPUT**

{'GaussianNB': 90.32258064516128,

'MultinomialNB': 80.64516129032258,

'ComplementNB': 64.51612903225806,

'BernoulliNB': 25.806451612903224,

'CategoricalNB': 93.54838709677419}

SORTED

{'CategoricalNB': 93.54838709677419, 'GaussianNB': 90.32258064516128, 'MultinomialNB': 80.64516129032258, 'ComplementNB': 64.51612903225806, 'BernoulliNB': 25.806451612903224}

The best algorithm is CategoricalNB with an accuracy of 93.55%