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%
```

2. **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
```

3. 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

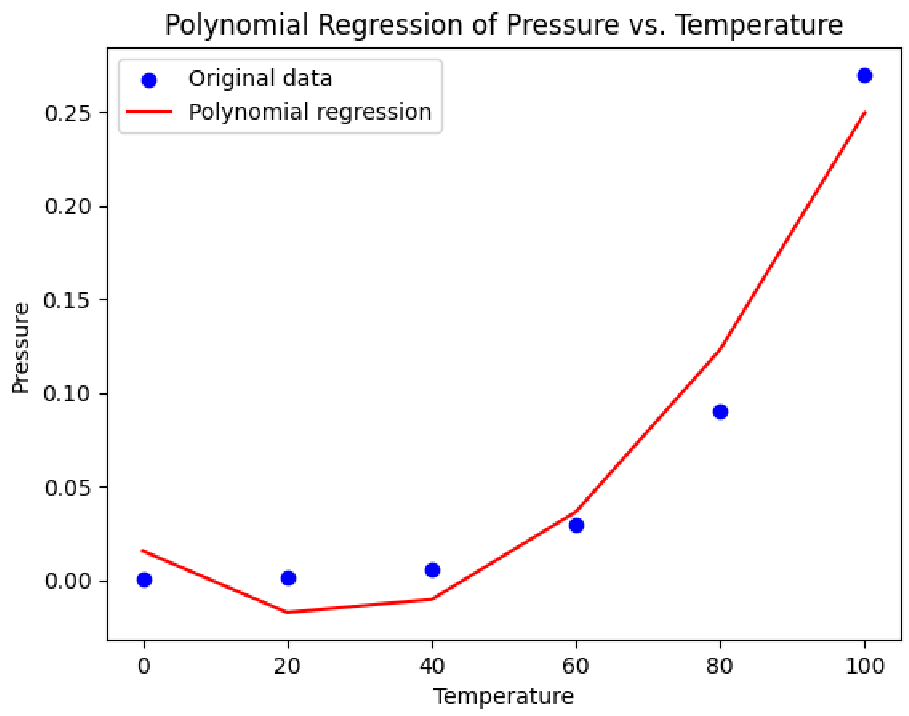
```
1. Probability of HeartDisease given evidence= restecg: 1
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1016 |
+-----------------+---------------------+
| heartdisease(1) |              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 |
+-----------------+---------------------+
```

4. 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
```

5. 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

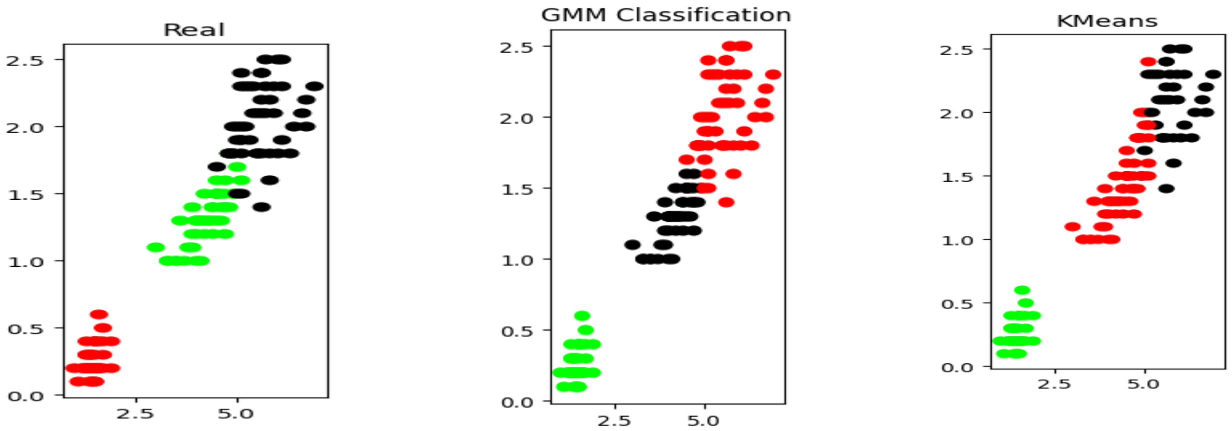
```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89510403]
 [0.88368551]
 [0.89148676]]
```

6. 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



7. 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_sta
te=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=",predictio
n,dataset["target_names"][prediction])
print(kn.score(X_test,y_test))
```

OUTPUT

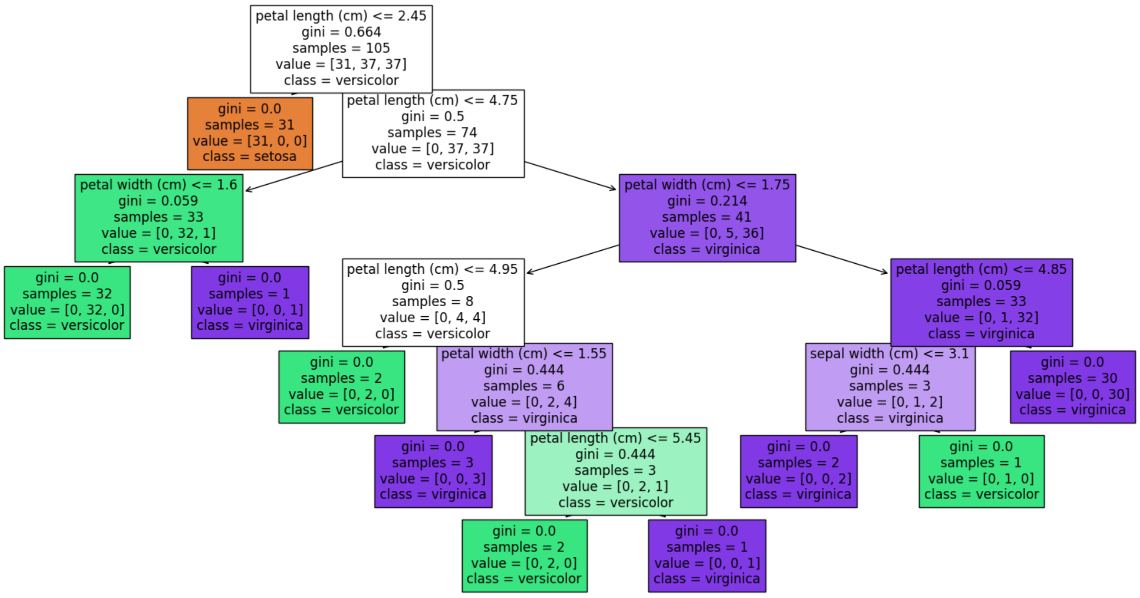
```
TARGET= 1 versicolor PREDICTED= [2] ['virginica']
0.9736842105263158
```

8. 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]]
```



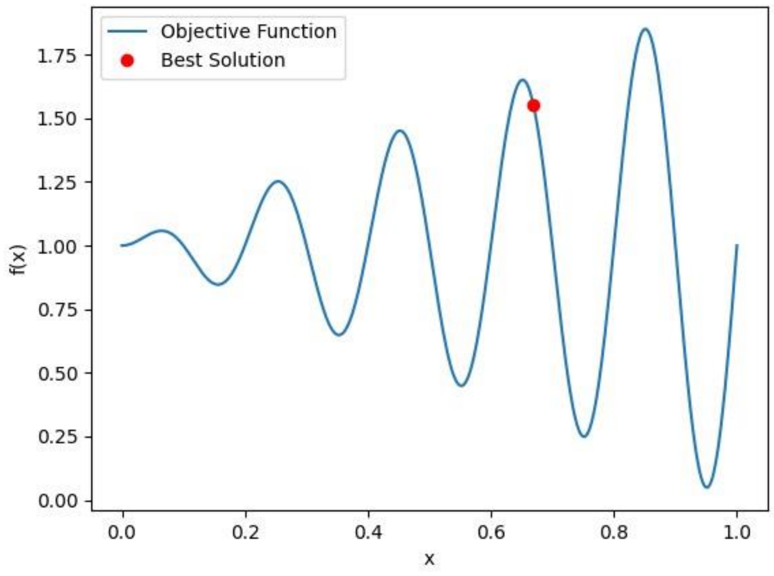
9. Implement an algorithm to demonstrate the significance of Genetic Algorithm in python.

```
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
```



10. 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%
```