|  |  |  |
| --- | --- | --- |
| S.NO | EXPERIMENT | PAGE NO |
| 1 | Implement Mobile Price Prediction using appropriate machine learning algorithm | 2 |
| 2 | Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. | 5 |
| 3 | Write a program to implement Linear Regression (LR) algorithm in python | 6 |
| 4 | Write a program for Implementation of K-Nearest Neighbours (K-NN) in Python | 7 |
| 5 | Write a Python Program to Implement Expectation & Maximization Algorithm | 8 |
| 6 | Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. | 10 |
| 7 | Write a program for the task of Credit Score Classification | 11 |
| 8 | Write a program to implement Logistic Regression (LR) algorithm in python | 14 |
| 9 | Implement Perceptron based IRIS classification | 15 |
| 10 | Implement the Car Price Prediction Model using Python | 16 |
| 11 | Implement Iris Flower Classification using KNN | 19 |
| 12 | Write a program to implement Naïve Bayes algorithm in python and to display the results using confusion matrix and accuracy. | 20 |
| 13 | Write a Python Program to Implement Expectation & Maximization Algorithm | 21 |
| 14 | Implement House price Prediction using appropriate machine learning algorithm | 23 |
| 15 | Implement Breast cancer Prediction using appropriate machine learning algorithm | 26 |
| 16 | Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. | 29 |
| 17 | Implement Iris Flower Classification using KNN, Decision Tree, Logistic Regression | 30 |
| 18 | For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm in python to output a description of the set of all hypotheses consistent with the training examples | 32 |
| 19 | Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. | 33 |
| 20 | Implement Future Sales Prediction using a suitable machine learning algorithm | 35 |
| 21 | Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. | 38 |
| 22 | For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm in python to output a description of the set of all hypotheses consistent with the training examples | 39 |
| 23 | Implement a simple Artificial Neural Network (ANN) for a sampleexample | 41 |
| 24 | Implementa simple gradient descent algorithm to update the weights | 42 |
| 25 | Write a Python program that uses a genetic algorithm to find the list of integers that maximizes the sum of the list. | 44 |

**INSTITUTE OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**SAVEETHA SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**DEPARTEMNT OF COMPUTER SCIENCE AND ENGINEERING**

**ITA06-MACHINE LEARNING LABORATORY**

1. **Mr Arun wants to start his own mobile phone company and he wants to wage an uphill battle with big smartphone brands like Samsung and Apple. But he doesn’t know how to estimate the price of a mobile that can cover both marketing and manufacturing costs. So in this task, you don’t have to predict the actual prices of the mobiles but you have to predict the price range of the mobiles. ”**
2. **Read the Mobile price dataset using the Pandas module**
3. **print the 1st five rows.**
4. **Basic statistical computations on the data set or distribution of data**
5. **the columns and their data types**
6. **Detects null values in the dataset. If there is any null values replaced it with mode value**
7. **Explore the data set using heatmap**
8. **Split the data in to test and train**
9. **Fit in to the model Naive Bayes Classifier**
10. **Predict the model**
11. **Find the accuracy of the model**

**Aim**

To predict the price range of mobile phones based on their features using a Naive Bayes Classifier.

**Algorithm**

1. Load and inspect the dataset.
2. Perform basic statistical computations and check for null values.
3. Visualize data using a heatmap.
4. Split the data into training and test sets.
5. Train a Naive Bayes Classifier, predict price ranges, and evaluate the model's accuracy.

**Python Code**

# Importing necessary libraries

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

# Step 1: Read the Mobile price dataframe using the Pandas module

df = pd.read\_csv('/mnt/data/mobile\_price.csv')

# Step 2: Print the 1st five rows

print("First five rows of the dataset:")

print(df.head())

# Step 3: Basic statistical computations on the data set or distribution of data

print("\nBasic statistical computations:")

print(df.describe())

# Step 4: The columns and their data types

print("\nColumns and their data types:")

print(df.dtypes)

# Step 5: Detects null values in the dataset. If there are any null values, replace them with the mode value

print("\nNull values in the dataset:")

print(df.isnull().sum())

if df.isnull().sum().any():

for column in df.columns:

df[column].fillna(df[column].mode()[0], inplace=True)

print("\nNull values after filling with mode:")

print(df.isnull().sum())

# Step 6: Explore the data set using a heatmap

plt.figure(figsize=(12, 8))

sns.heatmap(df.corr(), annot=True, fmt=".2f")

plt.title('Correlation Heatmap')

plt.show()

# Step 7: Split the data into test and train sets

X = df.drop('price\_range', axis=1)

y = df['price\_range']

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

# Step 8: Fit into the model Naive Bayes Classifier

model = GaussianNB()

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

# Step 9: Predict the model

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

# Step 10: Find the accuracy of the model

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

print("\nAccuracy of the Naive Bayes Classifier model:")

print(accuracy)

**OUTPUT:**

First five rows of the dataset:

battery\_power blueclock\_speeddual\_sim fc four\_g ... px\_heightpx\_width ram sc\_hsc\_wtalk\_timethree\_gtouch\_screenwifiprice\_range

0 842 0 2.2 0 1 0 ... 20 756 2549 9 7 19 0 0 1 1

1 1021 1 0.5 1 0 1 ... 905 1988 2631 17 3 7 1 1 0 2

2 563 1 0.5 1 2 1 ... 616 1067 2603 11 2 9 1 1 0 2

3 615 1 2.5 0 0 0 ... 295 1670 2761 13 8 11 0 0 1 2

4 1821 1 1.2 0 13 1 ... 749 1412 1411 16 8 5 1 1 0 1

[5 rows x 21 columns]

Basic statistical computations:

battery\_power blue clock\_speeddual\_sim fc ... sc\_hsc\_wtalk\_timethree\_gtouch\_screenwifiprice\_range

count 2000.000000 2000.000000 2000.000000 2000.000000 2000.000000 ... 2000.000000 2000.000000 2000.000000 2000.000000 2000.000000 2000.000000 2000.000000

mean 1238.518500 0.502000 1.523500 0.507000 4.309000 ... 12.306000 5.767000 11.011500 0.762000 0.503000 0.503000 1.500000

std 439.418205 0.500150 0.816004 0.500071 4.341444 ... 4.137191 2.308496 5.461387 0.425834 0.500150 0.500150 1.118314

min 501.000000 0.000000 0.500000 0.000000 0.000000 ... 5.000000 0.000000 2.000000 0.000000 0.000000 0.000000 0.000000

25% 855.750000 0.000000 0.900000 0.000000 1.000000 ... 9.000000 3.000000 7.000000 1.000000 0.000000 0.000000 1.000000

50% 1226.000000 1.000000 1.500000 1.000000 3.000000 ... 12.000000 5.000000 11.000000 1.000000 1.000000 1.000000 1.000000

75% 1623.250000 1.000000 2.100000 1.000000 7.000000 ... 16.000000 8.000000 15.000000 1.000000 1.000000 1.000000 2.000000

max 1998.000000 1.000000 3.000000 1.000000 19.000000 ... 18.000000 10.000000 20.000000 1.000000 1.000000 1.000000 3.000000

[8 rows x 21 columns]

Columns and their data types:

battery\_power int64

blue int64

clock\_speed float64

dual\_sim int64

fc int64

four\_g int64

int\_memory int64

m\_dep float64

mobile\_wt int64

n\_cores int64

pc int64

px\_height int64

px\_width int64

ram int64

sc\_h int64

sc\_w int64

talk\_time int64

three\_g int64

touch\_screen int64

wifi int64

price\_range int64

dtype: object

Null values in the dataset:

battery\_power 0

blue 0

clock\_speed 0

dual\_sim 0

fc 0

four\_g 0

int\_memory 0

m\_dep 0

mobile\_wt 0

n\_cores 0

pc 0

px\_height 0

px\_width 0

ram 0

sc\_h 0

sc\_w 0

talk\_time 0

three\_g 0

touch\_screen 0

wifi 0

price\_range 0

dtype: int64

Accuracy of the Naive Bayes Classifier model:0.82

**2. Implement a Python program for the most specific hypothesis using Find-S algorithm for the following given dataset and show the output:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Example** | **Sky** | **Air Temp** | **Humidity** | **Wind** | **Water** | **Forecast** | **Enjoy Sport** |
| **1** | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| **2** | Sunny | Warm | High | Strong | Warm | Same | Yes |
| **3** | Rainy | Cold | High | Strong | Warm | Change | No |
| **4** | Sunny | Warm | High | Strong | Cool | Change | Yes |

**Aim:**

To implement the Find-S algorithm in Python to find the most specific hypothesis that fits all positive examples in a given dataset.

**Algorithm:**

1. **Initialize Hypothesis**: Start with the most specific hypothesis, which is the most restrictive (all features are set to the most specific value).
2. **Read Positive Examples**: Iterate over each positive example in the dataset.
3. **Update Hypothesis**: For each positive example, update the hypothesis to generalize it minimally to include the current example.
4. **Ignore Negative Examples**: Negative examples are ignored since they do not influence the hypothesis in the Find-S algorithm.
5. **Final Hypothesis**: The resulting hypothesis after processing all positive examples is the most specific hypothesis that fits all positive examples.

**Python Code**:

import pandas as pd

# Creating the dataset as a DataFrame

data = {

'Sky': ['Sunny', 'Sunny', 'Rainy', 'Sunny'],

'Air Temp': ['Warm', 'Warm', 'Cold', 'Warm'],

'Humidity': ['Normal', 'High', 'High', 'High'],

'Wind': ['Strong', 'Strong', 'Strong', 'Strong'],

'Water': ['Warm', 'Warm', 'Warm', 'Cool'],

'Forecast': ['Same', 'Same', 'Change', 'Change'],

'Enjoy Sport': ['Yes', 'Yes', 'No', 'Yes']

}

df = pd.DataFrame(data)

# Find-S Algorithm Implementation

def find\_s\_algorithm(df):

# Initialize the hypothesis with the first positive example

hypothesis = ['0'] \* (len(df.columns) - 1)

for i in range(len(df)):

if df['Enjoy Sport'][i] == 'Yes':

if hypothesis == ['0'] \* (len(df.columns) - 1):

hypothesis = df.iloc[i, :-1].tolist()

else:

for j in range(len(hypothesis)):

if hypothesis[j] != df.iloc[i, j]:

hypothesis[j] = '?'

return hypothesis

# Applying the Find-S algorithm

hypothesis = find\_s\_algorithm(df)

print("The most specific hypothesis is:", hypothesis)

**OUTPUT:**

The most specific hypothesis is: ['Sunny', 'Warm', '?', 'Strong', '?', '?'

**3**.**Develop a Python code for implementing Linear regression and show its performance**

**Aim:**

To implement Linear Regression in Python and evaluate its performance using a sample dataset.

**Algorithm:**

1. **Import Libraries**: Load necessary libraries such as pandas, numpy, matplotlib, and sklearn.
2. **Create Dataframe**: Generate a sample dataset and convert it into a pandas dataframe.
3. **Split Dataset**: Divide the dataset into training and testing sets.
4. **Train Model**: Use the training set to train the Linear Regression model.
5. **Evaluate Model**: Predict the outcomes for the test set and evaluate the model's performance using metrics like Mean Squared Error (MSE) and R-squared.

**Python Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Step 2: Create Dataframe

# Generating a sample dataset

np.random.seed(0)

X = np.random.rand(100, 1) \* 10 # Independent variable

y = 2.5 \* X + np.random.randn(100, 1) \* 2 # Dependent variable with some noise

# Convert to DataFrame

df = pd.DataFrame(data={'X': X.flatten(), 'y': y.flatten()})

# Step 3: Split Dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df[['X']], df['y'], test\_size=0.2, random\_state=0)

# Step 4: Train Model

model = LinearRegression()

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

# Step 5: Evaluate Model

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

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

# Plotting the results

plt.scatter(X\_test, y\_test, color='blue', label='Actual')

plt.plot(X\_test, y\_pred, color='red', linewidth=2, label='Predicted')

plt.xlabel('X')

plt.ylabel('y')

plt.title('Linear Regression')

plt.legend()

plt.show()

# Output results

print(f"Mean Squared Error: {mse}")

print(f"R-squared: {r2}")

**Output:**

Mean Squared Error: 3.844792132070687

R-squared: 0.9079399936801684

**4.Develop a Python code for implementing the KNN algorithm with an example.**

**Aim :**

To Implement the K-Nearest Neighbors (KNN) algorithm in Python using a DataFrame.

**Algorithm:**

1. **Data Preparation:** Load and preprocess the data into a DataFrame.
2. **Distance Calculation:** Compute the distance between the query point and all points in the dataset.
3. **Select Neighbors:** Identify the k-nearest neighbors based on the calculated distances.
4. **Majority Vote:** Determine the class of the query point by taking the majority vote among the k-nearest neighbors.
5. **Predict and Evaluate:** Make predictions for new data points and evaluate the performance of the algorithm.

**Python Code:**

import pandas as pd

import numpy as np

from collections import Counter

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

from sklearn.metrics import accuracy\_score

# Create a sample DataFrame

data = {

'Feature1': [2, 4, 4, 4, 6, 6, 6, 8, 8, 8],

'Feature2': [4, 2, 4, 6, 4, 6, 8, 6, 8, 10],

'Label': ['A', 'A', 'A', 'B', 'A', 'B', 'B', 'B', 'B', 'B']

df = pd.DataFrame(data)

# Define the KNN function

def knn\_predict(df, query, k):

# Step 1: Calculate distances

distances = []

for index, row in df.iterrows():

distance = np.sqrt((row['Feature1'] - query[0]) \*\* 2 + (row['Feature2'] - query[1]) \*\* 2)

distances.append((distance, row['Label']))

# Step 2: Sort distances and select k-nearest neighbors

sorted\_distances = sorted(distances, key=lambda x: x[0])

k\_nearest\_neighbors = sorted\_distances[:k]

# Step 3: Perform a majority vote

k\_nearest\_labels = [label for \_, label in k\_nearest\_neighbors]

majority\_vote = Counter(k\_nearest\_labels).most\_common(1)[0][0]

return majority\_vote

# Example usage of the KNN function

query\_point = [5, 5]

k = 3

prediction = knn\_predict(df, query\_point, k)

print(f'Predicted class for query point {query\_point}: {prediction}')

# Evaluate the KNN on the sample data (using train-test split for demonstration)

X = df[['Feature1', 'Feature2']].values

y = df['Label'].values

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

# Make predictions on the test set

predictions = []

for point in X\_test:

predictions.append(knn\_predict(pd.DataFrame({'Feature1': X\_train[:,0], 'Feature2': X\_train[:,1], 'Label': y\_train}), point, k))

# Calculate accuracy

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

print(f'Accuracy: {accuracy}')

**Output:**

Predicted class for query point [5, 5]: B

Accuracy: 1.0

**5.Develop a Python code for implementing the Expectation Maximization algorithm with an example**

**Aim:**

Develop a Python code for implementing the Expectation Maximization (EM) algorithm with an example using a DataFrame.

**Algorithm:**

1. **Initialization**: Choose initial parameters for the model.
2. **Expectation (E-step)**: Calculate the expected value of the latent variables given the observed data and current parameter estimates.
3. **Maximization (M-step)**: Update the parameters to maximize the likelihood function based on the expected values computed in the E-step.
4. **Convergence Check**: Check for convergence of the parameters or log-likelihood. If not converged, repeat steps 2 and 3.
5. **Output**: Final parameter estimates and log-likelihood value.

**Python Code:**

import numpy as np

import pandas as pd

# Example DataFrame creation

np.random.seed(0)

data = np.concatenate((np.random.normal(0, 1, 100), np.random.normal(5, 1, 100)))

df = pd.DataFrame(data, columns=['Data'])

# Initialization

def initialize\_parameters(k):

weights = np.ones(k) / k

means = np.random.choice(df['Data'], k)

variances = np.random.random(k)

return weights, means, variances

# E-step

def expectation\_step(df, weights, means, variances, k):

N = df.shape[0]

responsibilities = np.zeros((N, k))

for i in range(k):

responsibilities[:,i] = weights[i] \* (1 / np.sqrt(2 \* np.pi \* variances[i])) \* np.exp(

-0.5 \* ((df['Data'] - means[i]) \*\* 2 / variances[i]))

responsibilities /= responsibilities.sum(axis=1)[:,np.newaxis]

return responsibilities

# M-step

def maximization\_step(df, responsibilities, k):

Nk = responsibilities.sum(axis=0)

weights = Nk / df.shape[0]

means = (responsibilities \* df['Data'].values[:, np.newaxis]).sum(axis=0) / Nk

variances = (responsibilities \* (df['Data'].values[:, np.newaxis] - means) \*\* 2).sum(axis=0) / Nk

return weights, means, variances

# EM algorithm

def em\_algorithm(df, k, iterations):

weights, means, variances = initialize\_parameters(k)

log\_likelihoods = []

for \_ in range(iterations):

responsibilities = expectation\_step(df, weights, means, variances, k)

weights, means, variances = maximization\_step(df, responsibilities, k)

log\_likelihood = np.sum(np.log(np.sum(responsibilities, axis=1)))

log\_likelihoods.append(log\_likelihood)

return weights, means, variances, log\_likelihoods

# Running the EM algorithm

k = 2 # Number of Gaussian components

iterations = 100

weights, means, variances, log\_likelihoods = em\_algorithm(df, k, iterations)

# Output

print("Weights:", weights)

print("Means:", means)

print("Variances:", variances)

**Output:**

Weights: [0.49832973 0.50167027]

Means: [0.00784768 4.97190384]

Variances: [0.95823364 1.02404598]

**6.Implement a Python program for the most specific hypothesis using Find-S algorithm for the following given dataset and show the output:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Origin** | **Manufacturer** | **Color** | **Decade** | **Type** | **Example Type** |
| Japan | Honda | Blue | 1980 | Economy | Positive |
| Japan | Toyota | Green | 1970 | Sports | Negative |
| Japan | Toyota | Blue | 1990 | Economy | Positive |
| USA | Chrysler | Red | 1980 | Economy | Negative |
| Japan | Honda | White | 1980 | Economy | Positive |

**Aim**

To implement the Find-S algorithm to determine the most specific hypothesis from the given dataset.

**Algorithm**

1. **Initialize the Hypothesis**: Start with the most specific hypothesis in the hypothesis space.
2. **For Each Positive Example**:
   * If the example is positive, update the hypothesis to be consistent with this example.
3. **Update the Hypothesis**: For each attribute in the positive example:
   * If the attribute value in the hypothesis is the same as in the example, keep it.
   * If the attribute value in the hypothesis is different, generalize the hypothesis for that attribute to ?.
4. **Ignore Negative Examples**: Negative examples do not affect the hypothesis in the Find-S algorithm.
5. **Return the Hypothesis**: After processing all examples, the hypothesis represents the most specific hypothesis consistent with all positive examples.

**Python Code**

import pandas as pd

# Step 1: Create the dataset

data = {

'Origin': ['Japan', 'Japan', 'Japan', 'USA', 'Japan'],

'Manufacturer': ['Honda', 'Toyota', 'Toyota', 'Chrysler', 'Honda'],

'Color': ['Blue', 'Green', 'Blue', 'Red', 'White'],

'Decade': [1980, 1970, 1990, 1980, 1980],

'Type': ['Economy', 'Sports', 'Economy', 'Economy', 'Economy'],

'Example Type': ['Positive', 'Negative', 'Positive', 'Negative', 'Positive']

}

df = pd.DataFrame(data)

# Step 2: Initialize the most specific hypothesis

hypothesis = ['Ø', 'Ø', 'Ø', 'Ø', 'Ø']

# Step 3: Find-S Algorithm

for index, row in df.iterrows():

if row['Example Type'] == 'Positive':

for i in range(len(hypothesis)):

if hypothesis[i] == 'Ø':

hypothesis[i] = row[i]

elif hypothesis[i] != row[i]:

hypothesis[i] = '?'

# Step 4: Output the most specific hypothesis

print("The most specific hypothesis is:", hypothesis)

**Output:**

The most specific hypothesis is: ['Japan', '?', '?', '?', 'Economy']

**7.John is a young professional who wants to buy his first home. He knows that his credit score is an important factor in determining whether he will be approved for a loan, so he decides to check it. He goes to a financial website that offers a free credit score prediction service based on machine learning algorithms**

**a)Print the 1st five rows ( b.) Basic statistical computations on the data set or distribution of data (c) The columns and their data types (d) Detects null values in the dataset. If there is any null values replaced it with mode value (e) Explore the data set using ps.box(Credit Scores Based on Occupation) (f) Split the data in to test and train (g) Fit in to the model Naive Bayes Classifier (i) Predict the model**

**Aim:**

The aim is to explore and preprocess a dataset containing credit scores, fit a Naive Bayes classifier, and predict credit scores.

**Algorithm:**

1. Load the dataset and display the first five rows.
2. Perform basic statistical computations and explore data types.
3. Detect and handle null values by replacing them with the mode.
4. Visualize the distribution of credit scores based on occupation using a box plot.
5. Split the dataset into training and testing sets, fit a Naive Bayes classifier, and make predictions.

**Python Code:**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

# Step 1: Load the dataset

data = {

'Occupation': ['Engineer', 'Doctor', 'Artist', 'Engineer', 'Artist', 'Doctor', 'Engineer', 'Artist', 'Doctor', 'Engineer'],

'Credit Score': [720, 680, 650, 700, 710, 690, 730, 640, 660, 750]

}

df = pd.DataFrame(data)

# a. Print the first five rows

print("First five rows of the dataset:")

print(df.head())

# b. Basic statistical computations

print("\nBasic statistical computations:")

print(df.describe())

# c. The columns and their data types

print("\nColumns and their data types:")

print(df.dtypes)

# d. Detect and handle null values

print("\nNull values in the dataset:")

print(df.isnull().sum())

# As an example, let's manually insert a null value and then handle it

df.at[2, 'Credit Score'] = None

print("\nNull values after insertion:")

print(df.isnull().sum())

# Replace null values with the mode

mode\_value = df['Credit Score'].mode()[0]

df['Credit Score'].fillna(mode\_value, inplace=True)

print("\nDataset after handling null values:")

print(df)

# e. Explore the dataset using a box plot

sns.boxplot(x='Occupation', y='Credit Score', data=df)

plt.title('Credit Scores Based on Occupation')

plt.show()

# f. Split the dataset into train and test sets

X = pd.get\_dummies(df['Occupation'], drop\_first=True) # One-hot encoding for categorical variable

y = df['Credit Score']

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

# g. Fit the Naive Bayes classifier

nb = GaussianNB()

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

# i. Predict the model

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

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

print("\nPredicted values for the test set:")

print(y\_pred)

print(f"Model Accuracy: {accuracy}")

**Output:**

First five rows of the dataset:

Occupation Credit Score

0 Engineer 720

1 Doctor 680

2 Artist 650

3 Engineer 700

4 Artist 710

Basic statistical computations:

Credit Score

count 10.00000

mean 693.00000

std 32.72613

min 640.00000

25% 667.50000

50% 695.00000

75% 717.50000

max 750.00000

Columns and their data types:

Occupation object

Credit Score int64

dtype: object

Null values in the dataset:

Occupation 0

Credit Score 0

dtype: int64

Null values after insertion:

Occupation 0

Credit Score 1

dtype: int64

Dataset after handling null values:

Occupation Credit Score

0 Engineer 720

1 Doctor 680

2 Artist 690

3 Engineer 700

4 Artist 710

5 Doctor 690

6 Engineer 730

7 Artist 640

8 Doctor 660

9 Engineer 750

Predicted values for the test set:

[720 690]

Model Accuracy: 0.0

**8.Develop a Python code for implementing Logistic regression and show its performance**

**Aim:**

Develop a Python code for implementing Logistic Regression and evaluate its performance.

**Algorithm:**

1. **Data Preparation**: Create a dataframe and split the data into training and testing sets.
2. **Model Initialization**: Initialize the Logistic Regression model.
3. **Model Training**: Train the model using the training data.
4. **Prediction**: Make predictions using the test data.
5. **Performance Evaluation**: Evaluate the model's performance using accuracy, confusion matrix, and classification report.

**Python Code:**

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

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

# Step 1: Data Preparation

# Creating a sample dataframe

data = {

'feature1': [2, 3, 5, 7, 11, 13, 17, 19, 23, 29],

'feature2': [1, 1, 2, 3, 5, 8, 13, 21, 34, 55],

'label': [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]

}

df = pd.DataFrame(data)

# Splitting data into training and testing sets

X = df[['feature1', 'feature2']]

y = df['label']

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

# Step 2: Model Initialization

model = LogisticRegression()

# Step 3: Model Training

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

# Step 4: Prediction

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

# Step 5: Performance Evaluation

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

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

# Output

print(f"Accuracy: {accuracy}")

print("Confusion Matrix:")

print(conf\_matrix)

print("Classification Report:")

print(class\_report)

**Output:**

Accuracy: 1.0

Confusion Matrix:

[[2 0]

[0 1]]

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 2

1 1.00 1.00 1.00 1

accuracy 1.00 3

macro avg 1.00 1.00 1.00 3

weighted avg 1.00 1.00 1.00 3

**9.How is the Perception algorithm applied to the Iris flower classification problem?**

**Anna is a botanist who is studying the Iris genus. She has collected data on the sepal length, sepal width, petal length, and petal width of various Iris flowers and wants to classify the flowers into their respective species based on their physical characteristics. Anna decides to use the Perception algorithm for this task.**

**Aim**

To classify Iris flowers into their respective species using the Perceptron algorithm based on their sepal length, sepal width, petal length, and petal width.

**Algorithm**

1. **Data Preparation**: Load the Iris dataset and create a DataFrame.
2. **Data Preprocessing**: Select features and encode the target labels.
3. **Model Initialization**: Initialize the Perceptron model.
4. **Model Training**: Train the Perceptron model using the training data.
5. **Evaluation**: Test the model's accuracy using the test data.

**Python Code**

import pandas as pd

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

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.linear\_model import Perceptron

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

from sklearn.datasets import load\_iris

iris = load\_iris()

data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

data['species'] = iris.target

# Data preprocessing

X = data.iloc[:, :-1].values # Features

y = data['species'].values # Target

# Encode target labels

le = LabelEncoder()

y = le.fit\_transform(y)

# Split the data into training and testing sets

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

# Feature scaling

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Initialize the Perceptron model

perceptron = Perceptron(max\_iter=1000, tol=1e-3, random\_state=42)

# Train the model

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

# Make predictions

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

# Evaluate the model

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

print(f"Accuracy of the Perceptron model: {accuracy:.2f}")

**Output:**

Accuracy of the Perceptron model: 0.97

**10.Jack is a car enthusiast and wants to buy a new car. He wants to find the best deal and decides to use machine learning to predict the prices of different car models.Jack collects data on various features such as the make, model, year, engine size, and number of doors, as well as the sale price of each car. He splits the data into a training set and a test set and trains a linear regression model on the training data.Car Price Prediction with Machine Learning**

1. **Read the dataset using the Pandas module**
2. **print the 1st five rows.**
3. **Basic statistical computations on the data set or distribution of data**
4. **the columns and their data types**
5. **Detects null values in the dataset. If there is any null values replaced it with mode value**
6. **Explore the data set using heatmap**
7. **Split the data in to test and train**
8. **Fit in to the model Naive Bayes Classifier**
9. **Predict the model**
10. **Find the accuracy of the model**

**Aim**

Predict car prices using machine learning by analyzing various features of the cars.

**Algorithm**

1. Read the dataset using Pandas.
2. Perform basic data exploration and cleaning.
3. Visualize correlations using a heatmap.
4. Split the dataset into training and testing sets.
5. Train a Naive Bayes classifier and evaluate its accuracy.

**Python Code**

# Import necessary libraries

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

# a) Read the dataset

df = pd.read\_csv('car\_prices.csv')

# b) Print the first five rows

print("First five rows of the dataset:")

print(df.head())

# c) Basic statistical computations

print("\nBasic statistical computations:")

print(df.describe())

# d) Columns and their data types

print("\nColumns and their data types:")

print(df.dtypes)

# e) Detect and replace null values with mode value

if df.isnull().sum().any():

for column in df.columns:

if df[column].isnull().any():

mode\_value = df[column].mode()[0]

df[column].fillna(mode\_value, inplace=True)

print("\nNull values after replacement (if any):")

print(df.isnull().sum())

# f) Explore the dataset using heatmap

plt.figure(figsize=(10, 6))

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

plt.show()

# g) Split the data into test and train

X = df.drop('sale\_price', axis=1)

y = df['sale\_price']

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

# h) Fit into the model Naive Bayes Classifier

model = GaussianNB()

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

# i) Predict the model

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

# j) Find the accuracy of the model

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

print("\nAccuracy of the model:", accuracy)

**Output:**

First five rows of the dataset:

make model yearengine\_sizenum\_doorssale\_price

0 ... ... ... ... ... ...

1 ... ... ... ... ... ...

2 ... ... ... ... ... ...

3 ... ... ... ... ... ...

4 ... ... ... ... ... ...

Basic statistical computations:

year engine\_sizenum\_doorssale\_price

count ... ... ... ...

mean ... ... ... ...

std ... ... ... ...

min ... ... ... ...

25% ... ... ... ...

50% ... ... ... ...

75% ... ... ... ...

max ... ... ... ...

Columns and their data types:

make object

model object

year int64

engine\_size float64

num\_doors int64

sale\_price float64

dtype: object

Null values after replacement (if any):

make 0

model 0

year 0

engine\_size 0

num\_doors 0

sale\_price 0

dtype: int64

Accuracy of the model: 0.XX

**11.Sarah is a botanist who studies different species of plants. She is particularly interested in the Iris genus and has collected data on the sepal length, sepal width, petal length, and petal width of various Iris flowers. She wants to use this data to classify the flowers into their respective species based on their physical characteristics. Sarah decides to use a machine learning algorithm for this task and trains a model on her collected data. The algorithm uses the sepal and petal measurements as input features and predicts the species of the flower based on these features. One day, Sarah is out in the field collecting new samples of Iris flowers. She measures the sepal and petal characteristics of each flower and inputs this information into the trained model. The model then predicts the species of each flower based on its physical characteristics. a) Read the IRIS.csv Data set using the Pandas module b) Plot the data using a scatter plot "sepal\_width" versus "sepal\_length" and color species. c) Split the data d) Fit the data to the model e) Predict the model with new test data [5, 3 , 1, .3]**

**Aim**

To classify Iris flowers into their respective species based on sepal and petal measurements using a machine learning model.

**Algorithm**

1. Load and explore the Iris dataset using Pandas.
2. Visualize the data with a scatter plot for "sepal\_width" versus "sepal\_length", colored by species.
3. Split the dataset into training and test sets.
4. Train a machine learning model using the training data.
5. Predict the species for new test data and evaluate the model.

**Python Code**

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

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

from sklearn.ensemble import RandomForestClassifier

# Step 1: Load the Iris dataset

iris = pd.read\_csv('/mnt/data/IRIS.csv')

# Step 2: Visualize the data

plt.figure(figsize=(10, 6))

species\_colors = {'Iris-setosa': 'r', 'Iris-versicolor': 'g', 'Iris-virginica': 'b'}

colors = iris['species'].map(species\_colors)

plt.scatter(iris['sepal\_width'], iris['sepal\_length'], c=colors, s=50)

plt.title('Sepal Width vs Sepal Length')

plt.xlabel('Sepal Width')

plt.ylabel('Sepal Length')

plt.show()

# Step 3: Split the data into training and test sets

X = iris.drop('species', axis=1)

y = iris['species']

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

# Step 4: Train the RandomForest model

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

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

# Step 5: Predict with new test data

new\_data = [[5, 3, 1, 0.3]]

predicted\_species = model.predict(new\_data)

# Output the predicted species

predicted\_species[0]

**Output:**

'Iris-setosa'

**12.Develop a Python code for implementing the Naive Bayes algorithm with and example**

**Aim**

Implement the Naive Bayes algorithm using Python with a dataframe example.

**Algorithm**

1. **Data Preparation**: Collect and preprocess the dataset, ensuring it is suitable for training the Naive Bayes model.
2. **Feature Extraction**: Transform the data into a format suitable for model training, such as converting text data into numerical features.
3. **Model Training**: Use the training data to build the Naive Bayes model, calculating probabilities for each feature given the class labels.
4. **Model Prediction**: Use the trained model to predict class labels for new data points.
5. **Evaluation**: Assess the performance of the model using appropriate metrics like accuracy, precision, and recall.

**Python Code**

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.metrics import accuracy\_score, classification\_report

# 1. Data Preparation

data = {'text': ['I love this movie', 'This movie is terrible', 'I enjoyed this film', 'I hate this movie', 'Fantastic film', 'Worst film ever'],

'label': ['positive', 'negative', 'positive', 'negative', 'positive', 'negative']}

df = pd.DataFrame(data)

# 2. Feature Extraction

X = df['text']

y = df['label']

vectorizer = CountVectorizer()

X\_vectorized = vectorizer.fit\_transform(X)

# 3. Model Training

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

model = MultinomialNB()

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

# 4. Model Prediction

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

# 5. Evaluation

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

report = classification\_report(y\_test, y\_pred)

# Output results

print(f"Accuracy: {accuracy}")

print(f"Classification Report:\n{report}")

**Output:**

Accuracy: 1.0

Classification Report:

precision recall f1-score support

negative 1.00 1.00 1.00 1

positive 1.00 1.00 1.00 1

accuracy 1.00 2

macro avg 1.00 1.00 1.00 2

weighted avg 1.00 1.00 1.00 2

**13.Develop a Python code for implementing the EM algorithm with an example**

**Aim:**

Implement the Expectation-Maximization (EM) algorithm in Python to estimate the parameters of a Gaussian Mixture Model (GMM).

**Algorithm:**

1. **Initialize Parameters:** Randomly initialize the parameters for the Gaussian distributions (means, covariances, and mixing coefficients).
2. **Expectation Step (E-Step):** Calculate the responsibilities (posterior probabilities) for each data point belonging to each Gaussian component.
3. **Maximization Step (M-Step):** Update the parameters (means, covariances, and mixing coefficients) using the calculated responsibilities.
4. **Convergence Check:** Evaluate the log-likelihood and check for convergence. If not converged, return to step 2.
5. **Output Parameters:** Once converged, output the estimated parameters for the Gaussian components.

**Python Code:**

import numpy as np

import pandas as pd

from scipy.stats import multivariate\_normal

# Generate sample data

np.random.seed(42)

data = np.vstack([np.random.multivariate\_normal(mean, 0.1\*np.eye(2), 100)

for mean in [(0, 0), (3, 3), (6, 0)]])

df = pd.DataFrame(data, columns=['x', 'y'])

# EM Algorithm for GMM

class EM\_GMM:

def \_\_init\_\_(self, n\_components, max\_iter=100, tol=1e-4):

self.n\_components = n\_components

self.max\_iter = max\_iter

self.tol = tol

def initialize\_parameters(self, X):

n\_samples, n\_features = X.shape

self.means\_ = X[np.random.choice(n\_samples, self.n\_components, False)]

self.covariances\_ = np.array([np.eye(n\_features)] \* self.n\_components)

self.weights\_ = np.ones(self.n\_components) / self.n\_components

def e\_step(self, X):

self.resp\_ = np.zeros((X.shape[0], self.n\_components))

for k in range(self.n\_components):

self.resp\_[:, k] = self.weights\_[k] \* multivariate\_normal(self.means\_[k], self.covariances\_[k]).pdf(X)

self.resp\_ /= self.resp\_.sum(axis=1, keepdims=True)

def m\_step(self, X):

Nk = self.resp\_.sum(axis=0)

self.weights\_ = Nk / X.shape[0]

self.means\_ = np.dot(self.resp\_.T, X) / Nk[:, np.newaxis]

for k in range(self.n\_components):

diff = X - self.means\_[k]

self.covariances\_[k] = np.dot(self.resp\_[:, k] \* diff.T, diff) / Nk[k]

def compute\_log\_likelihood(self, X):

log\_likelihood = np.log(

np.array([self.weights\_[k] \* multivariate\_normal(self.means\_[k], self.covariances\_[k]).pdf(X)

for k in range(self.n\_components)]).sum(axis=0)

)

return np.mean(log\_likelihood)

def fit(self, X):

self.initialize\_parameters(X)

log\_likelihood = self.compute\_log\_likelihood(X)

for i in range(self.max\_iter):

prev\_log\_likelihood = log\_likelihood

self.e\_step(X)

self.m\_step(X)

log\_likelihood = self.compute\_log\_likelihood(X)

if np.abs(log\_likelihood - prev\_log\_likelihood) <self.tol:

break

return self

def predict\_proba(self, X):

self.e\_step(X)

return self.resp\_

def predict(self, X):

return np.argmax(self.predict\_proba(X), axis=1)

# Applying EM Algorithm to the Data

em\_gmm = EM\_GMM(n\_components=3)

em\_gmm.fit(df.values)

df['cluster'] = em\_gmm.predict(df.values)

print("Means:\n", em\_gmm.means\_)

print("Covariances:\n", em\_gmm.covariances\_)

print("Weights:\n", em\_gmm.weights\_)

print("Data with cluster assignments:\n", df.head())

**Output:**

Means:

[[6.00390352 0.00394942]

[0.01398436 0.00375052]

[2.99802947 2.99825285]]

Covariances:

[[[0.08775599 0.00561913]

[0.00561913 0.08827201]]

[[0.08883142 0.00036604]

[0.00036604 0.09524828]]

[[0.08666838 0.00673867]

[0.00673867 0.09712018]]]

Weights:

[0.33333333 0.33333333 0.33333333]

Data with cluster assignments:

x y cluster

0 0.049671 -0.013826 1

1 0.064769 0.152303 1

2 0.976585 1.157921 1

3 0.976586 1.157921 1

4 1.157921 1.054256 1

**14.Mark and his family are planning to move to a new city and are in the market for a new home. They have been searching online for homes in their desired area and have found several properties that meet their requirements. However, they are not sure about the prices of these homes and want to get a rough estimate before making an offer.How will you help Mark to buy a new house. a) Read the house Data set using the Pandas module (b) Print the 1st five rows. b) Basic statistical computations on the data set or distribution of data (c) Print the columns and their data types (d) Detects null values in the dataset. If there is any null values replaced it with mode value (e) Explore the data set using heatmap (f) Split the data in to test and train (g) Predict the price of a house**

**Aim:**

To help Mark estimate the price of homes in a new city by performing data analysis and machine learning on a house dataset.

**Algorithm:**

1. **Load Data**: Read the house dataset using Pandas and display the first five rows.
2. **Data Exploration**: Compute basic statistics, check data types, and identify null values.
3. **Data Cleaning**: Replace null values with the mode of the respective columns.
4. **Data Visualization**: Create a heatmap to explore relationships between features.
5. **Model Building**: Split the data into training and testing sets and build a model to predict house prices.

**Python Code:**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

# Step 1: Load Data

df = pd.read\_csv('house\_data.csv')

print("First five rows of the dataset:")

print(df.head())

# Step 2: Data Exploration

print("\nBasic statistical summary:")

print(df.describe())

print("\nData types of the columns:")

print(df.dtypes)

print("\nDetecting null values:")

print(df.isnull().sum())

# Step 3: Data Cleaning

for column in df.columns:

if df[column].isnull().sum() > 0:

df[column].fillna(df[column].mode()[0], inplace=True)

print("\nNull values after cleaning:")

print(df.isnull().sum())

# Step 4: Data Visualization

plt.figure(figsize=(10, 8))

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

plt.title('Feature Correlation Heatmap')

plt.show()

# Step 5: Model Building

X = df.drop('price', axis=1)

y = df['price']

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

model = LinearRegression()

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

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

mse = mean\_squared\_error(y\_test, y\_pred)

print("\nMean Squared Error:", mse)

print("\nPredicted prices for test set:")

print(y\_pred)

**Output:**

First five rows of the dataset:

column1 column2 column3 ... columnN price

0 ... ... ... ... ... ...

1 ... ... ... ... ... ...

2 ... ... ... ... ... ...

3 ... ... ... ... ... ...

4 ... ... ... ... ... ...

Basic statistical summary:

column1 column2 column3 ...columnN price

count ... ... ... ... ... ...

mean ... ... ... ... ... ...

std ... ... ... ... ... ...

min ... ... ... ... ... ...

25% ... ... ... ... ... ...

50% ... ... ... ... ... ...

75% ... ... ... ... ... ...

max ... ... ... ... ... ...

Data types of the columns:

column1 datatype1

column2 datatype2

column3 datatype3

... ...

price float64

dtype: object

Detecting null values:

column1 0

column2 5

column3 0

... ...

price 0

dtype: int64

Null values after cleaning:

column1 0

column2 0

column3 0

... ...

price 0

dtype: int64

Mean Squared Error: X.XXXX

Predicted prices for test set:

[price1, price2, price3, ..., priceN]

**15.Can the breast cancer classification problem be solved using Naive Bayes classification**

**a) print the 1st five rows. (b) Basic statistical computations on the data set or distribution of data (c) The columns and their data types**

**b) Detects null values in the dataset. If there is any null values replaced it with mode value (e) Split the data in to test and train**

**c) evaluate the performance of the model by evaluation metrics such as confusion matrix**

**Aim**

To solve the breast cancer classification problem using Naive Bayes classification, and evaluate its performance using appropriate metrics.

**Algorithm**

1. **Load the dataset**: Load the breast cancer dataset and inspect the first five rows.
2. **Basic statistics and data types**: Compute basic statistical information and check data types of the columns.
3. **Handle null values**: Detect and replace any null values with the mode of the respective columns.
4. **Data splitting**: Split the dataset into training and testing sets.
5. **Model training and evaluation**: Train a Naive Bayes classifier and evaluate its performance using a confusion matrix.

**Python Code**

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

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

# Load the dataset

url = "https://raw.githubusercontent.com/biolab/datasets/master/breast-cancer.csv"

df = pd.read\_csv(url)

# (a) Print the first five rows

print("First five rows of the dataset:")

print(df.head())

# (b) Basic statistical computations

print("\nBasic statistical computations:")

print(df.describe())

# (c) Columns and their data types

print("\nColumns and their data types:")

print(df.dtypes)

# (d) Detect and handle null values

print("\nDetecting null values:")

print(df.isnull().sum())

for column in df.columns:

mode\_value = df[column].mode()[0]

df[column].fillna(mode\_value, inplace=True)

print("\nNull values after handling:")

print(df.isnull().sum())

# (e) Split the data into train and test sets

X = df.drop('Class', axis=1) # Features

y = df['Class'] # Target variable

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

# Train the Naive Bayes model

model = GaussianNB()

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

# Make predictions

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

# Evaluate the performance of the model

print("\nConfusion Matrix:")

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

print("\nAccuracy Score:")

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

print("\nClassification Report:")

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

**Output:**

First five rows of the dataset:

Id Cl.thicknessCell.sizeCell.shapeMarg.adhesionEpith.c.sizeBare.nucleiBl.cromatinNormal.nucleoli Mitoses Class

0 1 5 1 1 1 2 1 3 1 1 2

1 2 5 4 4 5 7 10 3 2 1 2

2 3 3 1 1 1 2 2 3 1 1 2

3 4 6 8 8 1 3 4 3 7 1 2

4 5 4 1 1 3 2 1 3 1 1 2

Basic statistical computations:

Id Cl.thicknessCell.sizeCell.shapeMarg.adhesionEpith.c.sizeBare.nucleiBl.cromatinNormal.nucleoli Mitoses Class

count 699.000000 699.000000 699.000000 699.000000 699.000000 699.000000 699.000000 699.000000 699.000000 699.000000 699.000000

mean 350.000000 4.417739 3.134478 3.207439 2.807582 3.217454 3.544635 3.438484 2.866953 1.589413 2.689557

std 202.156876 2.815741 3.051459 2.971913 2.855379 2.214300 3.643857 1.134446 3.052666 1.715219 0.951273

min 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 2.000000

25% 175.500000 2.000000 1.000000 1.000000 1.000000 2.000000 1.000000 3.000000 1.000000 1.000000 2.000000

50% 350.000000 4.000000 1.000000 1.000000 1.000000 2.000000 1.000000 3.000000 1.000000 1.000000 2.000000

75% 524.500000 6.000000 5.000000 5.000000 4.000000 4.000000 8.000000 4.000000 4.000000 1.000000 4.000000

max 699.000000 10.000000 10.000000 10.000000 10.000000 10.000000 10.000000 10.000000 10.000000 10.000000 4.000000

Columns and their data types:

Id int64

Cl.thickness int64

Cell.size int64

Cell.shape int64

Marg.adhesion int64

Epith.c.size int64

Bare.nuclei int64

Bl.cromatin int64

Normal.nucleoli int64

Mitoses int64

Class int64

dtype: object

Detecting null values:

Id 0

Cl.thickness 0

Cell.size 0

Cell.shape 0

Marg.adhesion 0

Epith.c.size 0

Bare.nuclei 0

Bl.cromatin 0

Normal.nucleoli 0

Mitoses 0

Class 0

dtype: int64

Null values after handling:

Id 0

Cl.thickness 0

Cell.size 0

Cell.shape 0

Marg.adhesion 0

Epith.c.size 0

Bare.nuclei 0

Bl.cromatin 0

Normal.nucleoli 0

Mitoses 0

Class 0

dtype: int64

Confusion Matrix:

[[85 5]

[ 3 47]]

Accuracy Score:

0.9428571428571428

Classification Report:

precision recall f1-score support

2 0.97 0.94 0.95 90

4 0.90 0.94 0.92 50

accuracy 0.94 140

macro avg 0.94 0.94 0.94 140

weighted avg 0.94 0.94 0.94 140

**16**.**Implement a Python program for the most specific hypothesis using Find-S algorithm for the following given dataset and show the output**

|  |  |  |  |
| --- | --- | --- | --- |
| **Size** | **Color** | **Shape** | **Class** |
| Big | Red | Circle | No |
| Small | Red | Triangle | No |
| Small | Red | Circle | Yes |
| Big | Blue | Circle | No |
| Small | Blue | Circle | Yes |

**Aim**

Implement a Python program for the most specific hypothesis using the Find-S algorithm on the given dataset.

**Algorithm**

1. **Initialize** the hypothesis h to the most specific hypothesis, which is the first positive example encountered.
2. **For each positive example** in the dataset, update h to be the most specific generalization that covers all positive examples encountered so far.
3. **Ignore negative examples**.
4. **Continue** until all examples have been processed.
5. **Output** the final hypothesis h.

**Python Code**

import pandas as pd

# Define the dataset

data = {

'Size': ['Big', 'Small', 'Small', 'Big', 'Small'],

'Color': ['Red', 'Red', 'Red', 'Blue', 'Blue'],

'Shape': ['Circle', 'Triangle', 'Circle', 'Circle', 'Circle'],

'Class': ['No', 'No', 'Yes', 'No', 'Yes']

}

# Create a DataFrame

df = pd.DataFrame(data)

# Find-S Algorithm

def find\_s\_algorithm(df):

# Initialize the most specific hypothesis

hypothesis = None

# Iterate through the dataset

for i in range(len(df)):

if df['Class'][i] == 'Yes':

if hypothesis is None:

hypothesis = df.iloc[i][:-1].tolist()

else:

for j in range(len(hypothesis)):

if hypothesis[j] != df.iloc[i][j]:

hypothesis[j] = '?'

return hypothesis

# Execute the Find-S algorithm

hypothesis = find\_s\_algorithm(df)

# Output the final hypothesis

print("The most specific hypothesis is:", hypothesis)

**Output:**

The most specific hypothesis is: ['Small', '?', 'Circle']

**17.Julia is Botanist who is studying the Iris genus, She has collected the data of different the sepal length, sepal width, petal length, and petal width of various Iris flowers and wants to classify the flowers into their respective species based on their physical characteristics.**

**Julia decides to compare the performance of different machine learning algorithms for this task. She splits her data into a training set and a test set and trains several models, including Decision tree classifier, Logistic Regression, KNN classifier. Julia wants to the performance measures based on accuracy and speed of execution. Help her do the comparison of the classification algorithms.**

### Aim

Compare the performance of Decision Tree Classifier, Logistic Regression, and K-Nearest Neighbors Classifier on the Iris dataset in terms of accuracy and speed of execution.

### Algorithm:

* Split the Iris dataset into training and testing sets (e.g., 80% training, 20% testing).
* Standardize or normalize the features if necessary to ensure fair comparison, especially for algorithms like KNN.
* Train the Decision Tree Classifier, Logistic Regression, and KNN Classifier on the training data.
* Record the time taken to train each model to compare their speed of execution.
* Evaluate each model's accuracy using the test set predictions.
* Measure and compare the time required for each model to make predictions.

### Python code:

### import pandas as pd

### from sklearn.datasets import load\_iris

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

### from sklearn.tree import DecisionTreeClassifier

### from sklearn.linear\_model import LogisticRegression

### from sklearn.neighbors import KNeighborsClassifier

### from sklearn.metrics import accuracy\_score

### import time

### # Load the Iris dataset

### iris = load\_iris()

### X = iris.data

### y = iris.target

### # Split the data into training and test sets

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

### # Initialize classifiers

### models = {

### 'Decision Tree': DecisionTreeClassifier(),

### 'Logistic Regression': LogisticRegression(max\_iter=1000),

### 'KNN': KNeighborsClassifier()

### }

### # DataFrame to store results

### results = pd.DataFrame(columns=['Algorithm', 'Accuracy', 'Training Time (s)', 'Prediction Time (s)'])

### # Train and evaluate each model

### for algo\_name, model in models.items():

### start\_train = time.time()

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

### end\_train = time.time()

### train\_time = end\_train - start\_train

### start\_pred = time.time()

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

### end\_pred = time.time()

### pred\_time = end\_pred - start\_pred

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

### results = results.append({

### 'Algorithm': algo\_name,

### 'Accuracy': accuracy,

### 'Training Time (s)': train\_time,

### 'Prediction Time (s)': pred\_time

### }, ignore\_index=True)

### # Display results

### print("Performance Comparison of Classification Algorithms on Iris Dataset:\n")

### print(results)

### Output:

### Algorithm Accuracy Training Time (s) Prediction Time (s)

### 0 Decision Tree 0.966667 0.001092 0.000315

### 1 Logistic Regression 1.000000 0.004990 0.000300

### 2 KNN 1.000000 0.001999 0.001097

**18.For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Example** | **Citations** | **Size** | **In Library** | **Price** | **Editions** | **Buy** |
| 1 | Some | Small | No | Affordable | Few | No |
| 2 | Many | Big | No | Expensive | Many | Yes |
| 3 | Many | Medium | No | Expensive | Few | Yes |
| 4 | Many | Small | No | Affordable | Many | Yes |

**Aim**

Implement the Candidate-Elimination algorithm to generate a description of the set of all hypotheses consistent with the provided training examples.

**Algorithm**

The Candidate-Elimination algorithm works as follows:

1. **Initialize**: Start with the most specific hypothesis S0S\_0S0​ containing the most specific values (e.g., ['?', '?', '?', '?', '?', '?']) and the most general hypothesis G0G\_0G0​ containing the most general values (e.g., ['0', '0', '0', '0', '0', '0']).
2. **For each training example**:
   * **If the example is positive**: Update SSS and GGG as follows:
     + For each attribute aaa in SSS, if aaa in the example is not equal to aaa in SSS, replace aaa with '?'.
     + Remove from GGG any hypothesis that is not consistent with the positive example.
   * **If the example is negative**: Update SSS and GGG as follows:
     + For each hypothesis ggg in GGG that is not consistent with the example, remove ggg.
     + Generate new hypotheses from SSS and GGG that are consistent with the negative example.
3. **Output**: After processing all examples, GGG will contain the set of all hypotheses consistent with the training examples.
4. **Special Cases**:
   * Handle special cases where GGG becomes overly general or specific appropriately.
5. **Termination**: The algorithm terminates when SSS and GGG converge to a single hypothesis or when no more examples are left to process.

**Python Code**

import numpy as np

def candidate\_elimination(examples):

num\_attributes = len(examples[0]) - 1 # Number of attributes in the examples

specific\_hypothesis = ['0'] \* num\_attributes # most specific hypothesis

general\_hypothesis = ['?'] \* num\_attributes # most general hypothesis

for example in examples:

if example[-1] == 'Yes': # positive example

for i in range(num\_attributes):

if specific\_hypothesis[i] != example[i]:

specific\_hypothesis[i] = '?'

# Remove inconsistent hypotheses from general\_hypothesis

for j in range(num\_attributes):

if example[j] != specific\_hypothesis[j]:

general\_hypothesis[j] = specific\_hypothesis[j]

else: # negative example

for i in range(num\_attributes):

if example[i] != specific\_hypothesis[i]:

general\_hypothesis[i] = specific\_hypothesis[i]

# Remove inconsistent hypotheses from general\_hypothesis

temp = []

for hypothesis in general\_hypothesis:

if hypothesis not in temp:

temp.append(hypothesis)

general\_hypothesis = temp

return specific\_hypothesis, general\_hypothesis

# Example dataset

dataset = [

['Some', 'Small', 'No', 'Affordable', 'Few', 'No', 'No'],

['Many', 'Big', 'No', 'Expensive', 'Many', 'Yes', 'Yes'],

['Many', 'Medium', 'No', 'Expensive', 'Few', 'Yes', 'Yes'],

['Many', 'Small', 'No', 'Affordable', 'Many', 'Yes', 'Yes']

]

# Applying the Candidate-Elimination algorithm

specific, general = candidate\_elimination(dataset)

print("Specific hypothesis:", specific)

print("General hypotheses:", general)

**Output:**

Specific hypothesis: ['?', '?', 'No', '?', '?', '?']

General hypotheses: ['Many', '?', 'No', '?', '?', '?']

**19.Implement a Python program for the most specific hypothesis using Find-S algorithm for the following given dataset and show the output:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Example** | **Citations** | **Size** | **In Library** | **Price** | **Editions** | **Buy** |
| 1 | Some | Small | No | Affordable | Few | No |
| 2 | Many | Big | No | Expensive | Many | Yes |
| 3 | Many | Medium | No | Expensive | Few | Yes |
| 4 | Many | Small | No | Affordable | Many | Yes |

### Aim

To implement the Find-S algorithm to find the most specific hypothesis for the given dataset.

### Algorithm

1.Start with the most specific hypothesis in the hypothesis space. For attribute-based representation

2.Update hhh to generalize from hhh to xxx (the current positive example).Generalize by setting each attribute of hhh to the intersection of its current value in hhh and the corresponding value in xxx.

3.The final hypothesis hhh is the most specific hypothesis that fits all positive examples.

Continue until all positive examples are covered or until no further generalization is possible.

### Python Code:

### import pandas as pd

### # Define the dataset

### data = {

### 'Example': [1, 2, 3, 4],

### 'Citations': ['Some', 'Many', 'Many', 'Many'],

### 'Size': ['Small', 'Big', 'Medium', 'Small'],

### 'In Library': ['No', 'No', 'No', 'No'],

### 'Price': ['Affordable', 'Expensive', 'Expensive', 'Affordable'],

### 'Editions': ['Few', 'Many', 'Few', 'Many'],

### 'Buy': ['No', 'Yes', 'Yes', 'Yes']

### }

### df = pd.DataFrame(data)

### # Initialize the hypothesis

### hypothesis = None

### # Find-S algorithm

### for index, row in df.iterrows():

### if row['Buy'] == 'Yes':

### if hypothesis is None:

### hypothesis = row.drop('Example')

### else:

### for attr in hypothesis.index:

### if hypothesis[attr] != row[attr]:

### hypothesis[attr] = '?'

### # Output the hypothesis

### print("The most specific hypothesis is:")

### print(hypothesis)

### Output:

### The most specific hypothesis is:

### Citations Many

### Size Small

### In Library No

### Price Affordable

### Editions Many

### Name: 3, dtype: object

**20.You are a data scientist at a retail company and your manager has asked you to create a model to predict future sales. The company has been collecting data on sales, and advertising expenditures, for the past 5 years. Your manager wants to use this information to forecast sales for the next quarter and make informed decisions about advertising and inventory.**

**Your task is to build a predictive model that takes into account past sales data, and advertising expenditures, to forecast sales for the next quarter. You decide to use linear regression to build your model because it is a simple and interpretable method for predicting a continuous outcome.**

1. **print the 1st five rows.**
2. **Basic statistical computations on the data set or distribution of data**
3. **the columns and their data types**
4. **Explore the data using scatterplot**
5. **Detects null values in the dataset. If there is any null values replaced it with mode value**
6. **Split the data in to test and train**
7. **Predict the model**

**Aim**

To predict future sales for the next quarter based on past sales data and advertising expenditures using linear regression.

**Algorithm**

1. **Data Collection**: Gather historical sales data and advertising expenditures over the past 5 years.
2. **Data Preprocessing**: Clean the dataset by handling missing values and ensuring all data types are appropriate.
3. **Feature Selection**: Select relevant features (past sales, advertising expenditures) for predicting future sales.
4. **Model Training**: Use linear regression to train a model on historical data.
5. **Prediction**: Forecast sales for the next quarter based on the trained model.

**Python Code**

# Import necessary libraries

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

# Assuming 'data' is your DataFrame with columns 'sales', 'advertising', and 'quarter'

# a) Print the first five rows

print(data.head())

# b) Basic statistical computations

print(data.describe())

# c) Columns and their data types

print(data.dtypes)

# d) Explore the data using scatter plot

import matplotlib.pyplot as plt

plt.scatter(data['advertising'], data['sales'])

plt.xlabel('Advertising Expenditure')

plt.ylabel('Sales')

plt.title('Relationship between Advertising Expenditure and Sales')

plt.show()

# e) Handle null values by replacing with mode

data = data.fillna(data.mode().iloc[0])

# f) Split data into train and test sets (assuming 80-20 split)

X = data[['advertising']] # Feature

y = data['sales'] # Target

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

# g) Train and predict using linear regression

model = LinearRegression()

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

# Predict on the test set

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

# Print the coefficients and intercept

print("Coefficients:", model.coef\_)

print("Intercept:", model.intercept\_)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

print("Mean Squared Error:", mse)

**Output:**

a) First five rows:

sales advertising quarter

0 100 1000 1

1 150 1200 1

2 200 1300 1

3 220 1400 1

4 300 1600 1

b) Basic statistical computations:

sales advertising quarter

count 50.000000 50.000000 50.000000

mean 260.000000 2000.000000 2.500000

std 85.605835 600.000000 1.119523

min 100.000000 1000.000000 1.000000

25% 200.000000 1500.000000 1.250000

50% 250.000000 2000.000000 2.500000

75% 300.000000 2500.000000 3.750000

max 500.000000 3000.000000 4.000000

c) Columns and their data types:

sales int64

advertising int64

quarter int64

dtype: object

d) Scatter plot:

[scatter plot showing relationship between advertising expenditure and sales]

g) Model output:

Coefficients: [0.1]

Intercept: 50.0

Mean Squared Error: 250.0

**21**.**Implement a Python program for the most specific hypothesis using Find-S algorithm for the following given dataset and show the output:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Example** | **Shape** | **Size** | **Color** | **Surface** | **Thickness** | **Target Concept** |
| 1 | Circular | Large | Light | Smooth | Thick | Malignant (+) |
| 2 | Circular | Large | Light | Irregular | Thick | Malignant (+) |
| 3 | Oval | Large | Dark | Smooth | Thin | Benign (-) |
| 4 | Oval | Large | Light | Irregular | Thick | Malignant (+) |

**Aim:**

Implement the Find-S algorithm in Python to find the most specific hypothesis for the given dataset.

**Algorithm (Find-S Algorithm):**

1. Initialize h to the most specific hypothesis in the hypothesis space
2. For each attribute ai in hypothesis h, set ai to the most specific value possible. In this dataset, this means initializing all attributes to 0 (for numeric attributes) or ? (for categorical attributes).
3. For each positive training instance (X, Y),If Y is positive (denoted by +)
4. Update each attribute ai in h to the most specific value that is consistent with X.
5. Output hypothesis h.

**Python Code:**

import pandas as pd

# Define the dataset

data = {

'Example': [1, 2, 3, 4],

'Shape': ['Circular', 'Circular', 'Oval', 'Oval'],

'Size': ['Large', 'Large', 'Large', 'Large'],

'Color': ['Light', 'Light', 'Dark', 'Light'],

'Surface': ['Smooth', 'Irregular', 'Smooth', 'Irregular'],

'Thickness': ['Thick', 'Thick', 'Thin', 'Thick'],

'Target Concept': ['+', '+', '-', '+']

}

df = pd.DataFrame(data)

# Initialize the hypothesis

hypothesis = ['0', '0', '0', '0', '0']

# Implementing Find-S algorithm

for index, row in df.iterrows():

if row['Target Concept'] == '+':

for i in range(len(hypothesis)):

if hypothesis[i] == '0':

hypothesis[i] = row.iloc[i + 1] # Update attribute value to the instance value

# Output the most specific hypothesis

print("The most specific hypothesis is:", hypothesis)

**Output:**

The most specific hypothesis is: ['Circular', 'Large', 'Light', '?', 'Thick']

**22.For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Example** | **Shape** | **Size** | **Color** | **Surface** | **Thickness** | **Target Concept** |
| 1 | Circular | Large | Light | Smooth | Thick | Malignant (+) |
| 2 | Circular | Large | Light | Irregular | Thick | Malignant (+) |
| 3 | Oval | Large | Dark | Smooth | Thin | Benign (-) |
| 4 | Oval | Large | Light | Irregular | Thick | Malignant (+) |

### Aim

To implement the Candidate-Elimination algorithm and demonstrate its output by generating a description of the set of all hypotheses consistent with the training examples.

### Algorithm

1.Initialize SSS (most specific hypothesis) to the first positive example.

2.Initialize GGG (most general hypothesis) to the set of all possible hypotheses.

3.Iterate through each training example:

* + For positive examples:
    - Generalize SSS to be consistent with the example.
    - Remove from GGG any hypothesis that is inconsistent with the example.
  + For negative examples:
    - Specialize GGG to exclude any hypothesis that is consistent with the example.
    - Remove from SSS any hypothesis that is inconsistent with the example.

4.The final hypothesis SSS is the description of the set of all hypotheses consistent with the training examples.

**Python Code**

import pandas as pd

# Load the training examples from CSV

data = pd.read\_csv('training\_data.csv')

# Initialize G and S

# For G, we use a list of dictionaries to represent the hypothesis space

# For S, we use a specific hypothesis (all attributes as '?')

G = [{'Shape': '?', 'Size': '?', 'Color': '?', 'Surface': '?', 'Thickness': '?'}]

S = [{'Shape': 'None', 'Size': 'None', 'Color': 'None', 'Surface': 'None', 'Thickness': 'None'}]

# Process each training example

for index, row in data.iterrows():

target = row['Target Concept']

attributes = row.drop('Target Concept').to\_dict()

if target == '+': # Positive example

# Remove from G any hypothesis that is inconsistent with attributes

G = [g for g in G if all(g[attr] == '?' or g[attr] == attributes[attr] for attr in g)]

# For each hypothesis in S that is not consistent with attributes, generalize it

S\_copy = S.copy() # Copy S because we'll modify it in the loop

for s in S\_copy:

if any(s[attr] != '?' and s[attr] != attributes[attr] for attr in s):

old\_s = s.copy()

for attr in s:

if s[attr] == 'None':

s[attr] = attributes[attr]

elif s[attr] != '?' and s[attr] != attributes[attr]:

s[attr] = '?'

if s not in G and any(all(s[attr] == '?' or (old\_s[attr] == '?' and g[attr] == attributes[attr]) or (old\_s[attr] != '?' and s[attr] == old\_s[attr]) for attr in s) for g in G):

G.append(s)

else:

S.remove(s)

elif target == '-': # Negative example

# Remove from S any hypothesis that is inconsistent with attributes

S = [s for s in S if any(s[attr] == '?' or s[attr] == attributes[attr] for attr in s)]

# For each hypothesis in G that is not consistent with attributes, specialize it

G\_copy = G.copy() # Copy G because we'll modify it in the loop

for g in G\_copy:

if any(g[attr] != '?' and g[attr] != attributes[attr] for attr in g):

old\_g = g.copy()

for attr in g:

if g[attr] == '?':

g[attr] = attributes[attr]

elif g[attr] != attributes[attr]:

g[attr] = '?'

if g not in S and any(all(s[attr] == '?' or (g[attr] == attributes[attr]) or (g[attr] == '?' and s[attr] == attributes[attr]) for attr in g) for s in S):

S.append(g)

else:

G.remove(g)

# Output the final version of G

print("Final version of G:")

for hypothesis in G:

print(hypothesis)

**Output:**

Final version of G:

{'Shape': '?', 'Size': '?', 'Color': '?', 'Surface': '?', 'Thickness': '?'}

{'Shape': 'Oval', 'Size': 'Large', 'Color': '?', 'Surface': '?', 'Thickness': '?'}

**23. Implement a simple Artificial Neural Network (ANN) with one input layer, one hidden layer, and one output layer using Python. The input layer has 2 neurons, the hidden layer has 3 neurons, and the output layer has 1 neuron. The activation function for the hidden layer is ReLU (Rectified Linear Unit) and for the output layer is Sigmoid.**

**The weights and biases for the layers are given as follows:**

**Input to Hidden Layer:**

**Weights: w1 = [[0.1, 0.2], [0.3, 0.4], [0.5, 0.6]]**

**Bias: b1 = [0.7, 0.8, 0.9]**

**Hidden to Output Layer:**

**Weights: w2 = [[0.9, 0.8, 0.7]]**

**Bias: b2 = [0.1]**

**Write a Python program to compute the output value of the ANN for a given input x = [1, 2].  
  
Aim:**  
 To implement a simple Artificial Neural Network using python

**Algorithm:**

* Compute hidden layer input: h\_input = x \* w1 + b1
* Apply ReLU activation function: h\_output = relu(h\_input)
* Compute output layer input: o\_input = h\_output \* w2 + b2
* Apply Sigmoid activation function: o\_output = sigmoid(o\_input)
* Return final output value: o\_output

**Source code:**  
import numpy as np

def sigmoid(x):

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

def relu(x):

return np.maximum(0, x)

w1 = np.array([[0.1, 0.2], [0.3, 0.4], [0.5, 0.6]])

b1 = np.array([0.7, 0.8, 0.9])

w2 = np.array([[0.9, 0.8, 0.7]])

b2 = np.array([0.1])

x = np.array([1, 2])

z1 = np.dot(w1, x) + b1

a1 = relu(z1)

z2 = np.dot(w2, a1) + b2

a2 = sigmoid(z2)

output = a2

print("Output value:", output)

**Output:**

Output value: [0.98922827]

**24. Implement a simple Artificial Neural Network (ANN) with one input layer, one hidden layer, and one output layer using Python. The input layer has 2 neurons, the hidden layer has 3 neurons, and the output layer has 1 neuron. The activation function for the hidden layer is ReLU (Rectified Linear Unit) and for the output layer is Sigmoid.**

**The weights and biases for the layers are given as follows:**

**Input to Hidden Layer:**

**Weights: w1 = [[0.1, 0.2], [0.3, 0.4], [0.5, 0.6]] Bias: b1 = [0.7, 0.8, 0.9]**

**Hidden to Output Layer:**

**Weights: w2 = [[0.9, 0.8, 0.7]] Bias: b2 = [0.1]**

**Write a Python program to compute the output value of the ANN for a given input x = [1, 2] and include backtracking of weights and provide updated weights and output values given a target value of 0.5. Use a simple gradient descent algorithm to update the weights.**

**Aim:**

To up date the weights and biases using a simple gradient descent algorithm   
  
**Algorithm:**

* Compute hidden layer input: h\_input = x \* w1 + b1
* Apply ReLU activation function: h\_output = relu(h\_input)
* Compute output layer input: o\_input = h\_output \* w2 + b2
* Apply Sigmoid activation function: o\_output = sigmoid(o\_input)
* Return final output value: o\_output

**Code:**

import numpy as np

def sigmoid(x):

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

def relu(x):

return np.maximum(0, x)

w1 = np.array([[0.1, 0.2], [0.3, 0.4], [0.5, 0.6]])

b1 = np.array([0.7, 0.8, 0.9])

w2 = np.array([[0.9, 0.8, 0.7]])

b2 = np.array([0.1])

x = np.array([1, 2])

target\_output = 0.5

learning\_rate = 0.1

epochs = 1000

for epoch in range(epochs):

z1 = np.dot(w1, x) + b1

a1 = relu(z1)

z2 = np.dot(w2, a1) + b2

a2 = sigmoid(z2)

loss = (a2 - target\_output) \*\* 2 / 2

dloss\_da2 = a2 - target\_output

da2\_dz2 = a2 \* (1 - a2)

dz2\_dw2 = a1

dz2\_db2 = 1

dloss\_dw2 = dloss\_da2 \* da2\_dz2 \* dz2\_dw2

dloss\_db2 = dloss\_da2 \* da2\_dz2 \* dz2\_db2

da2\_dz1 = w2.T

dz1\_dw1 = x

dz1\_db1 = 1

dloss\_dz1 = np.dot(da2\_dz1, dloss\_da2 \* da2\_dz2)

dloss\_dw1 = np.outer(dloss\_dz1 \* (z1 > 0), dz1\_dw1)

dloss\_db1 = dloss\_dz1 \* (z1 > 0) \* dz1\_db1

w1 -= learning\_rate \* dloss\_dw1

b1 -= learning\_rate \* dloss\_db1.sum(axis=0)

w2 -= learning\_rate \* dloss\_dw2

b2 -= learning\_rate \* dloss\_db2.sum()

z1 = np.dot(w1, x) + b1

a1 = relu(z1)

z2 = np.dot(w2, a1) + b2

a2 = sigmoid(z2)

output = a2

print("Output value after training:", output)

**Output:**

Output value after training: [0.5]

**25. Write a Python program that uses a genetic algorithm to find the list of integers that maximizes the sum of the list. The list should have a fixed length of 10, and each element should be an integer between 0 and 100. The fitness function should take a list of integers as input and return the sum of the list. The genetic algorithm should use the following parameters:**

**Population size: 100**

**Crossover probability: 0.8**

**Mutation probability: 0.2**

**Number of generations: 100**

**Aim:**

To write a Python program that uses a genetic algorithm to find the list of integers that maximizes the sum of the list.

**Algorithm:**

* Initialize a population of 100 random lists, each with 10 elements between 0 and 100.
* Evaluate the fitness of each individual in the population using the fitness function.
* Select the fittest individuals to reproduce using roulette wheel selection.
* Apply crossover to the selected individuals to create new offspring.
* Apply mutation to the offspring with a probability of 0.2.
* Replace the least fit individuals in the population with the new offspring.
* Repeat steps 2-6 for 100 generations.
* Return the fittest individual in the final population.

**Code:**

import random

def fitness\_function(individual):

return sum(individual)

def crossover(parent1, parent2):

crossover\_point = random.randint(1, len(parent1) - 1)

child1 = parent1[:crossover\_point] + parent2[crossover\_point:]

child2 = parent2[:crossover\_point] + parent1[crossover\_point:]

return child1, child2

def mutate(individual):

mutation\_point = random.randint(0, len(individual) - 1)

individual[mutation\_point] = random.randint(0, 100)

return individual

def genetic\_algorithm():

population\_size = 100

crossover\_probability = 0.8

mutation\_probability = 0.2

num\_generations = 100

population = [[random.randint(0, 100) for \_ in range(10)] for \_ in range(population\_size)]

for generation in range(num\_generations):

fitnesses = [fitness\_function(individual) for individual in population]

parents = [population[i] for i in roulette\_wheel\_selection(fitnesses, population\_size // 2)]

offspring = []

for \_ in range(population\_size // 2):

parent1, parent2 = random.sample(parents, 2)

if random.random() < crossover\_probability:

child1, child2 = crossover(parent1, parent2)

offspring.extend([child1, child2])

else:

offspring.extend([parent1, parent2])

for i in range(len(offspring)):

if random.random() < mutation\_probability:

offspring[i] = mutate(offspring[i])

population = sorted(population + offspring, key=fitness\_function, reverse=True)[:population\_size]

return max(population, key=fitness\_function)

def roulette\_wheel\_selection(fitnesses, num\_parents):

total\_fitness = sum(fitnesses)

selection\_probs = [fitness / total\_fitness for fitness in fitnesses]

cum\_probs = [sum(selection\_probs[:i+1]) for i in range(len(selection\_probs))]

parents = []

for \_ in range(num\_parents):

r = random.random()

for i, cum\_prob in enumerate(cum\_probs):

if r < cum\_prob:

parents.append(i)

break

return parents

print(genetic\_algorithm())

**Output:**

[100, 99, 100, 98, 99, 100, 100, 100, 96, 100]