List of Experiments:

1. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

Introduction:

Find-S is a concept learning algorithm used in machine learning for finding the most specific hypothesis that fits all the positive training examples. It is a simple and efficient algorithm used for learning from labelled training data in a hypothesis space.

Algorithm:

1. Initialize h (hypothesis) with the most specific hypothesis h0 = (?, ?, ..., ?) (where each feature is unknown).
2. For each **positive** training example:
   - Compare the instance attributes with the current hypothesis.
   - If an attribute differs, **generalize** the hypothesis by replacing the specific value with ?.
3. Ignore **negative** training examples.
4. Return the final hypothesis h.

Training and Testing Data Split:

- **Training Data:** A subset of the dataset used to fit the model (usually **80%**).
- **Testing Data:** A separate subset to evaluate the model's accuracy (usually **20%**).
- **Validation Data (optional):** Sometimes used to fine-tune parameters (**10-20%**).

Advantages and Disadvantages:

Advantages:

- **Fast and Simple:** Requires only a single scan of the data.
- **Guaranteed Most Specific Hypothesis:** Ensures that no unnecessary generalizations are made.

Disadvantages:

- **Cannot Handle Negative Examples:** Ignores them, which can lead to incorrect hypotheses.
- **Sensitive to Noise:** Errors in training data lead to incorrect generalizations.

Suitable Data Sets and Justification:

Find-S works best with **Boolean or Categorical Data** because it learns the most specific hypothesis using discrete attributes.

Suitable Data Types:

1. **Loan Approval Dataset (Binary)**
   - Attributes: Income (High/Low), Credit Score (Good/Bad), Collateral (Yes/No).
   - Justification: Find-S can generalize from **"Approved"** cases without needing negative examples.
2. **Disease Diagnosis Dataset (Binary or Nominal)**
   - Attributes: Symptoms like Fever (Yes/No), Cough (Yes/No), Fatigue (Yes/No).
   - Justification: Find-S can learn to predict disease occurrence based on symptoms.
3. **Spam Email Classification (Categorical)**
   - Attributes: Contains "Free" (Yes/No), Clickable Links (Yes/No), Length of Email (Short/Long).
   - Justification: Works well for categorization-based learning.

Purpose of Libraries:

Library	Purpose
`pandas`	Load and manipulate dataset easily.
`numpy`	Perform efficient array operations.
`sklearn.model_selection`	Split data into training and testing.

Sample Dataset:

Dataset link:

https://drive.google.com/file/d/1tzcYiZrSwnOLSzucT0YU8HELGjoAo6JO/view?usp=sharing

Dataset SnippetOfLoan_data.csv:

Income	Credit Scor	Collateral	Loan Approved
Medium	Average	Yes	Yes
Low	Average	Yes	Yes
High	Average	Yes	Yes
Low	Good	No	No
Medium	Good	No	No
Low	Good	No	No
High	Good	No	Yes
Low	Average	Yes	No
High	Good	Yes	Yes
High	Average	Yes	Yes
High	Average	Yes	Yes
Medium	Average	Yes	Yes
Low	Good	No	Yes
High	Good	Yes	Yes
Medium	Bad	No	No
High	Good	No	Yes

Program:

With Libraries:

```
import pandas as pd
import numpy as np

df = pd.read_csv("loan_data.csv")

positive_data = df[df["Loan Approved"] == "Yes"].iloc[:, :-1].values

def find_s(data):
    hypothesis = data[0].copy()
    for instance in data:
        for i in range(len(hypothesis)):
            if hypothesis[i] != instance[i]:
                hypothesis[i] = "?"

    return hypothesis

final_hypothesis = find_s(positive_data)
print("Final Hypothesis:", final_hypothesis)
```

Output:

```
Final Hypothesis: ['?', '?', '?']
```

Without Libraries:

```
import csv
def load_csv(filename):
    dataset = []
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        next(reader)
        for row in reader:
            dataset.append(row)
    return dataset

def find_s_algorithm(data):
    hypothesis = None

    for row in data:
        if row[-1] == "Yes":
            if hypothesis is None:
                hypothesis = row[:-1]
            else:
                for i in range(len(hypothesis)):
                    if hypothesis[i] != row[i]:
                        hypothesis[i] = "?"

    return hypothesis
filename = "loan_data.csv"
dataset = load_csv(filename)
hypothesis = find_s_algorithm(dataset)
print("Final Hypothesis:", hypothesis)
```

Output:

```
Final Hypothesis: ['?', '?', '?']
```

Real Time Applications:

1. **Network Connectivity**

   - **Real-time applications:** Detecting connectivity in computer networks or social networks.

   - **Example:** When a new device joins a network, the Find algorithm helps check if two devices are in the same network segment.

2. **Image Processing and Computer Vision**

   - **Real-time applications:** Connected component labelling in binary images.

   - **Example:** In optical character recognition (OCR) and medical imaging, the Find algorithm is used to group pixels belonging to the same object.

3. **Kruskal's Algorithm for Minimum Spanning Tree (MST)**

   - **Real-time applications:** Finding the optimal way to lay down cables, road networks, or optimize power grids.

2. 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 Disadvantagesistent with the training examples.

Introduction:

Candidate Elimination is a concept learning algorithm used to learn the hypothesis space from a given training dataset. Unlike the Find-S algorithm, which finds only the most specific hypothesis, Candidate Elimination maintains a version space that Disadvantagesists of both:

1. Specific Hypothesis (S) → The most specific concept that fits the positive examples.
2. General Hypothesis (G) → The most general concept that eliminates negative examples.

By iterating through training examples, the algorithm refines both S and G, leading to a hypothesis space that best describes the data.

Algorithm:

1. Initialize
   - Set S (Specific Hypothesis) to the most specific hypothesis (all Φ or "null").
   - Set G (General Hypothesis) to the most general hypothesis (all ?).
2. For each training example:
   - If positive, generalize S minimally (only where necessary) to be Disadvantagesistent with the example.
   - If negative, specialize G to exclude that example while keeping it as general as possible.
3. Continue until all examples are processed.
4. Return S and G as the final version space.

Training and Testing Data Split:

- **Training Data**: 80% of the dataset is used to train the model.
- **Testing Data**: 20% of the dataset is used to evaluate how well the algorithm generalizes to unseen data.

Advantages and Disadvantages:

✅ Pros	❌ Cons
Learns both **specific** and **general** hypotheses	Computationally expensive for large datasets
More **robust** than Find-S (handles both positive and negative examples)	Does not handle noisy data well
Works well for **small to medium categorical datasets**	Requires **discrete/categorical attributes**

Purpose of Libraries:

Library	Purpose
`pandas`	Load and manipulate CSV dataset easily.
`numpy`	Efficient array operations for hypothesis handling.

Sample Dataset:

Dataset link:

https://drive.google.com/file/d/10KEFfmQR136pgd8qqVoTUtRcVJGAZsA6/view?usp=sharing

Dataset Snippet Of Weather_prediction_data.csv:

Sky	Temperatu	Humidity	Wind	PlayTennis
Sunny	Mild	Low	Strong	Yes
Cloudy	Mild	Low	Strong	Yes
Cloudy	Cold	High	Weak	No
Cloudy	Mild	High	Weak	Yes
Cloudy	Hot	Low	Weak	Yes
Rainy	Hot	Low	Strong	No
Cloudy	Hot	High	Strong	No
Rainy	Cold	Low	Weak	No
Rainy	Hot	Low	Strong	No
Cloudy	Mild	High	Strong	No
Sunny	Mild	Low	Strong	No

Program:

With Libraries:

```
import pandas as pd
import numpy as np
df = pd.read_csv("weather_prediction_data.csv")

data = df.values

def candidate_elimination(data):
    num_attributes = data.shape[1] - 1
    S = np.array(["Φ"] * num_attributes)
    G = np.array([["?"] * num_attributes])

    for row in data:
        example = row[:-1]
        label = row[-1]

        if label == "Yes":
            for i in range(num_attributes):
                if S[i] == "Φ":
                 S[i] = example[i]
                elif S[i] != example[i]:
                    S[i] = "?"


            G = [g for g in G if all(g[i] == '?' or g[i] == example[i] for i in range(num_attributes))]

        else:
                G_new = []
            for g in G:
                for i in range(num_attributes):
                    if example[i] != S[i] and S[i] != "?":
                        new_g = g.copy()
                        new_g[i] = S[i]
                        G_new.append(new_g)

            G = np.array(G_new)

    return S, G

S, G = candidate_elimination(data)
```

```
print("Final Specific Hypothesis:", S)
print("Final General Hypothesis:", G)
```

Output:

```
Final Specific Hypothesis: ['?' '?' '?' '?']
Final General Hypothesis: []
```

Without Libraries:

```
import csv

def load_csv(filename):
    dataset = []
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        next(reader)
        for row in reader:
            dataset.append(row)
    return dataset

def candidate_elimination(data):
    num_attributes = len(data[0]) - 1
    S = ["Φ"] * num_attributes
    G = [["?"] * num_attributes]

    for row in data:
        example = row[:-1]
        label = row[-1]

        if label == "Yes":
            for i in range(num_attributes):
                if S[i] == "Φ":
                    S[i] = example[i]
                elif S[i] != example[i]:
                    S[i] = "?"


            G = [g for g in G if all(g[i] == '?' or g[i] == example[i] for i in range(num_attributes))]

        else:
            G_new = []
            for g in G:
```

```
            for i in range(num_attributes):
                if example[i] != S[i] and S[i] != "?":
                    new_g = g[:]
                    new_g[i] = S[i]
                    G_new.append(new_g)

        G = G_new

    return S, G

filename = "weather_prediction_data.csv"
dataset = load_csv(filename)

S, G = candidate_elimination(dataset)
print("Final Specific Hypothesis:", S)
print("Final General Hypothesis:", G)
```

Output:

```
Final Specific Hypothesis: ['?' '?' '?' '?']
Final General Hypothesis: []
```

Real Time Applications:

1. **Speech Recognition**
   - **Application:** Learns acceptable speech patterns to improve voice assistants like Alexa, Siri, and Google Assistant.
   - **Example:** If a system learns correct pronunciations, it refines its understanding of spoken words.

2. **Fraud Detection in Banking**
   - **Application:** Identifies fraudulent transactions by continuously refining criteria based on user behaviour and anomalies.
   - **Example:** Credit card fraud detection systems use real-time data to filter out suspicious transactions.

3. Quality Control in Manufacturing

- **Application:** Detects defective products by analysing features such as weight, colour, or texture.
- **Example:** In real-time production lines, machines reject defective products by learning from previous inspections.

4. Personalized Recommendations

- **Application:** Refines user preferences for movie recommendations, product suggestions, and content filtering.
- **Example:** Netflix and Amazon refine their recommendation models using user feedback.

3. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Introduction:

The ID3 (Iterative Dichotomiser 3) algorithm is a classic algorithm used to build a decision tree based on information gain. It is used for supervised classification tasks and builds a tree structure where each node represents an attribute and each branch represents a decision rule.

Algorithm:

1. **Select Attribute**:
   - Select the attribute that maximizes **information gain**.
   - Information Gain is calculated as the difference between the **entropy** of the dataset before and after the split based on an attribute.
2. **Split Data**:
   - Partition the dataset into subsets based on the selected attribute.
3. **Recursion**:
   - For each subset, apply the ID3 algorithm recursively until one of the stopping conditions is met (e.g., all instances in the subset belong to the same class, no more attributes, or no remaining data).
4. **Stopping Condition**:
   - All examples are classified as a single class.
   - No attributes are left for splitting.
   - There are no more instances to split.

Information Gain Calculation:

- **Entropy** measures the uncertainty in the dataset. The formula for entropy is:

$$H(S) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

Where:

- p_i = probability of class *i* in the set S.

- **Information Gain** is calculated as:

$$\text{Gain}(S, A) = H(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} H(S_v)$$

Where:

- S_v is the subset of S where the attribute A has value *v*.

Training and Testing Data Split:

- **Training Data:** Typically, 80% of the data is used to train the model.
- **Testing Data:** The remaining 20% of the data is used to evaluate the model's performance.

Advantages and Disadvantages:

✅ Pros	❌ Cons
Simple to understand and implement	Can lead to overfitting if the tree is too deep
Works well for **categorical data**	Does not handle continuous variables without discretization
Efficient for small datasets	Sensitive to noisy data
Easily interpretable, easy to visualize	Prone to biased splits if one attribute has many levels

Suitable Data Sets and Justification:

ID3 is best suited for datasets that are:

- **Categorical** in nature (attributes have distinct categories).
- **Small to medium** in size, as it can become inefficient for large datasets.
- **Classification problems**, where the goal is to predict a class label (e.g., whether a loan will be approved or not, diagnosing diseases based on symptoms).

Example Datasets:

1. **Weather Dataset**: Predicting whether to play tennis based on weather conditions (e.g., Sky, Temperature, Humidity).
2. **Iris Dataset**: Classifying flowers into different species based on measurements like sepal length, sepal width, petal length, and petal width.

3. **Loan Approval Dataset**: Classifying whether a loan will be approved based on features like credit score, income, etc.

Justification:

- These datasets have **discrete categories** which make them ideal for ID3 since it works best with **categorical features**.
- They provide a **clear classification task** that is easy to implement and interpret.

Purpose of Libraries:

Library	Purpose
`pandas`	To handle and manipulate data (CSV files, data frames).
`numpy`	For numerical operations (handling arrays).
`sklearn`	For using pre-built datasets and splitting the data (also for testing model accuracy).

Sample Dataset:

Sepal_Leng	Sepal_Widt	Petal_Leng	Petal_Widt	Feature_Names
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa
4.8	3.4	1.6	0.2	Iris-setosa
4.8	3	1.4	0.1	Iris-setosa
4.3	3	1.1	0.1	Iris-setosa
5.8	4	1.2	0.2	Iris-setosa
5.7	4.4	1.5	0.4	Iris-setosa
5.4	3.9	1.3	0.4	Iris-setosa
5.1	3.5	1.4	0.3	Iris-setosa

Dataset link:

https://drive.google.com/file/d/1E9qQumsCoNTCfz2tpodVMgexreh_2xfG/view?usp=sharing

Program:

With Libraries:

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
iris = load_iris()
X = iris.data
y = iris.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

clf = DecisionTreeClassifier(criterion="entropy")
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy of ID3 on test set:", accuracy)

from sklearn.tree import export_text
tree_rules = export_text(clf, feature_names=iris.feature_names)
print("Decision Tree (ID3) with libraries:\n", tree_rules)
```

Output:

```
Accuracy of ID3 on test set: 0.9333333333333333
Decision Tree (ID3) with libraries:
 |--- petal length (cm) <= 2.60
|   |--- class: 0
|--- petal length (cm) >  2.60
|   |--- petal width (cm) <= 1.75
|   |   |--- petal length (cm) <= 5.05
|   |   |   |--- sepal length (cm) <= 4.95
|   |   |   |   |--- petal length (cm) <= 3.90
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- petal length (cm) >  3.90
|   |   |   |   |   |--- class: 2
|   |   |   |--- sepal length (cm) >  4.95
|   |   |   |   |--- class: 1
|   |   |--- petal length (cm) >  5.05
|   |   |   |--- class: 2
|   |--- petal width (cm) >  1.75
|   |   |--- petal length (cm) <= 4.85
|   |   |   |--- sepal width (cm) <= 3.10
|   |   |   |   |--- class: 2
|   |   |   |--- sepal width (cm) >  3.10
|   |   |   |   |--- class: 1
|   |   |--- petal length (cm) >  4.85
|   |   |   |--- class: 2
```

Without Libraries:

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris

def entropy(data):
    labels = data[:, -1]
    _, counts = np.unique(labels, return_counts=True)
    probs = counts / len(labels)
    return -np.sum(probs * np.log2(probs))

def information_gain(data, attribute_index):
    total_entropy = entropy(data)
    values, counts = np.unique(data[:, attribute_index], return_counts=True)
    weighted_entropy = 0
    for i, value in enumerate(values):
        subset = data[data[:, attribute_index] == value]
        weighted_entropy += (counts[i] / len(data)) * entropy(subset)
    return total_entropy - weighted_entropy

def id3(data, attributes, target_attribute_index):
      if len(np.unique(data[:, target_attribute_index])) == 1:
        return np.unique(data[:, target_attribute_index])[0]

    if len(attributes) == 0:
        return np.unique(data[:, target_attribute_index])[0]

    gains = [information_gain(data, i) for i in range(len(attributes))]
    best_attribute_index = np.argmax(gains)
    best_attribute = attributes[best_attribute_index]

    tree = {best_attribute: {}}
    values = np.unique(data[:, best_attribute_index])


    for value in values:
        subset = data[data[:, best_attribute_index] == value]
        new_attributes = attributes[:best_attribute_index] + attributes[best_attribute_index + 1:]
        tree[best_attribute][value] = id3(subset, new_attributes, target_attribute_index)

    return tree
  iris = load_iris()
```

```
X = iris.data
y = iris.target

data = np.column_stack((X, y))
attributes = iris.feature_names

tree = id3(data, list(attributes), target_attribute_index=-1)
print("Decision Tree (ID3) without libraries:", tree)
```

Output:

Real Time Applications:

1. **Loan and Credit Risk Assessment**

   - **Application:**Predicts whether a loan applicant is high-risk or low-risk based on financial history.
   - **Example:**Banks and fintech companies use decision trees to approve or reject loan applications.

2. **Customer Churn Prediction**

   - **Application:** Predicts whether a customer is likely to leave a service based on past interactions.
   - **Example:**Telecom companies (e.g., ATandT, Verizon) use decision trees to reduce churn by offering personalized retention plans.

3. **Real-Time Stock Market Prediction**

   - **Application:**Classifies stocks as buy, sell, or hold based on historical data and trends.

4. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

Introduction:

Back propagation is a supervised learning algorithm used for training artificial neural networks (ANNs). It is primarily used for optimizing weights in neural networks through a gradient descent method, adjusting weights by minimizing the error between the predicted and actual output.

Algorithm:

The backpropagation algorithm works by passing the data forward through the network and then calculating the error backward through the network. The basic steps for the backpropagation algorithm are:

1. **Initialization:** Initialize the weights randomly.
2. **Forward Pass:**
   - Input the data into the network and calculate the output using the current weights.
3. **Error Calculation:**
   - Compute the error (loss function) by comparing the predicted output and the actual output.
4. **Backward Pass:**
   - Propagate the error backward through the network to compute the gradients of the error with respect to each weight.
   - Update the weights using gradient descent, which moves the weights in the direction that reduces the error.
5. **Repeat:** Repeat the process for each epoch (iteration) until the error is minimized.

Training and Testing Data Split:

- **Training Data:** Used to train the neural network by adjusting weights and biases.
- **Testing Data:** Used to evaluate the model's generalization ability (how well it performs on unseen data).

  A typical split is 70% for training and 30% for testing, or 80%/20%.

Advantages and Disadvantages:

Advantages:

- **Versatile**: Can be applied to a wide range of problems, including classification and regression.
- **Convergence**: Efficient optimization technique using gradient descent.
- **Incremental Learning**: Can learn from data iteratively.

Disadvantages:

- **Slow Convergence**: The learning process can be slow and requires careful tuning of learning rates and epochs.
- **Overfitting**: Without regularization, backpropagation can overfit the model to training data.
- **Local Minima**: The algorithm might get stuck in local minima instead of finding the global minimum of the loss function.

Suitable Data Sets and Justification:

Classification Tasks:

- MNIST (handwritten digits), Iris (flower species classification), CIFAR-10 (image classification).
- These datasets have labelled data with features that help the network learn to classify inputs effectively.

Regression Tasks:

- Boston Housing Dataset (predicting house prices), Stock Price Prediction.
- Regression datasets have continuous target values, which backpropagation handles through a different loss function (e.g., Mean Squared Error).

Image Processing Tasks:

- Datasets like CIFAR-10 or Fashion-MNIST are ideal as they contain pixel data, which a neural network can effectively learn through backpropagation.

Purpose of Libraries:

- **NumPy:** For numerical computations, such as matrix operations and activation functions.
- **Scikit-learn:** For datasets, splitting data into training/testing sets, and performance evaluation.
- **Keras/TensorFlow (if using advanced neural networks):** For high-level neural network API and optimization algorithms.

Sample Dataset:

Dataset link:

https://drive.google.com/file/d/1lTc3HX-3he17n_2jiC6ozkJFv2Aq1bbV/view?usp=sharing

Dataset Snippet OfHousing.csv:

longitude	latitude	housing_m	total_room	total_bedr	population	households	median_inc	median_ho	ocean_proximity
-122.23	37.88	41	880	129	322	126	8.3252	452600	NEAR BAY
-122.22	37.86	21	7099	1106	2401	1138	8.3014	358500	NEAR BAY
-122.24	37.85	52	1467	190	496	177	7.2574	352100	NEAR BAY
-122.25	37.85	52	1274	235	558	219	5.6431	341300	NEAR BAY
-122.25	37.85	52	1627	280	565	259	3.8462	342200	NEAR BAY
-122.25	37.85	52	919	213	413	193	4.0368	269700	NEAR BAY
-122.25	37.84	52	2535	489	1094	514	3.6591	299200	NEAR BAY
-122.25	37.84	52	3104	687	1157	647	3.12	241400	NEAR BAY
-122.26	37.84	42	2555	665	1206	595	2.0804	226700	NEAR BAY
-122.25	37.84	52	3549	707	1551	714	3.6912	261100	NEAR BAY
-122.26	37.85	52	2202	434	910	402	3.2031	281500	NEAR BAY
-122.26	37.85	52	3503	752	1504	734	3.2705	241800	NEAR BAY
-122.26	37.85	52	2491	474	1098	468	3.075	213500	NEAR BAY
-122.26	37.84	52	696	191	345	174	2.6736	191300	NEAR BAY
-122.26	37.85	52	2643	626	1212	620	1.9167	159200	NEAR BAY
-122.26	37.85	50	1120	283	697	264	2.125	140000	NEAR BAY
-122.27	37.85	52	1966	347	793	331	2.775	152500	NEAR BAY

Program:

With Libraries:

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
```

```
data = fetch_california_housing()
X = data.data
y = data.target

scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

mlp = MLPRegressor(hidden_layer_sizes=(5,), max_iter=10000, learning_rate_init=0.01, random_state=42)

mlp.fit(X_train, y_train)

train_score = mlp.score(X_train, y_train)
test_score = mlp.score(X_test, y_test)

print(f"Train Score (R^2): {train_score}")
print(f"Test Score (R^2): {test_score}")
```

Output:

```
Train Score (R^2): 0.7379807041786632
Test Score (R^2): 0.7235686677898822
```

Without Libraries:

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

data = fetch_california_housing()
X = data.data
y = data.target.reshape(-1, 1)

scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

input_size = X_train.shape[1]
```

```
hidden_size = 5
output_size = 1
learning_rate = 0.001
epochs = 10000
clip_value = 1.0

np.random.seed(42)
W1 = np.random.randn(input_size, hidden_size) * np.sqrt(2.0 / (input_size + hidden_size))
b1 = np.zeros((1, hidden_size))
W2 = np.random.randn(hidden_size, output_size) * np.sqrt(2.0 / (hidden_size + output_size))
b2 = np.zeros((1, output_size))

def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return (x > 0).astype(float)

for epoch in range(epochs):

    Z1 = np.dot(X_train, W1) + b1
    A1 = relu(Z1)
    Z2 = np.dot(A1, W2) + b2
    A2 = Z2
    loss = np.mean((y_train - A2) ** 2)

    dA2 = A2 - y_train      dZ2 = dA2
    dW2 = np.dot(A1.T, dZ2)
    db2 = np.sum(dZ2, axis=0, keepdims=True)

    dA1 = np.dot(dZ2, W2.T)
    dZ1 = dA1 * relu_derivative(Z1)
    dW1 = np.dot(X_train.T, dZ1)
    db1 = np.sum(dZ1, axis=0, keepdims=True)

    dW1 = np.clip(dW1, -clip_value, clip_value)
    db1 = np.clip(db1, -clip_value, clip_value)
    dW2 = np.clip(dW2, -clip_value, clip_value)
    db2 = np.clip(db2, -clip_value, clip_value)


    W1 -= learning_rate * dW1
    b1 -= learning_rate * db1
```

```
    W2 -= learning_rate * dW2
    b2 -= learning_rate * db2

    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.6f}")

Z1_test = np.dot(X_test, W1) + b1
A1_test = relu(Z1_test)
Z2_test = np.dot(A1_test, W2) + b2
A2_test = Z2_test

test_loss = np.mean((y_test - A2_test) ** 2)
print(f"Test Loss (MSE): {test_loss:.6f}")
```

Output:

```
Epoch 0, Loss: 6.458786
Epoch 1000, Loss: 0.423795
Epoch 2000, Loss: 0.371037
Epoch 3000, Loss: 0.355205
Epoch 4000, Loss: 0.355134
Epoch 5000, Loss: 0.355134
Epoch 6000, Loss: 0.355134
Epoch 7000, Loss: 0.355134
Epoch 8000, Loss: 0.355134
Epoch 9000, Loss: 0.355134
Test Loss (MSE): 0.372126
```

Real Time Applications:

1. **Autonomous Vehicles (Self-Driving Cars)**

   - **Application**: Neural networks in autonomous vehicles use backpropagation for real-time object detection, path planning, and decision-making.
   - **Impact of Delay**: A high delay in backpropagation can slow down model training, delaying the deployment of improved models for real-time navigation.
   - **Optimization**: Hardware acceleration using GPUs, TPUs, and specialized AI chips (like NVIDIA's Drive AI platform) helps reduce delay.

2. **Real-Time Speech Recognition (Voice Assistants)**

   - **Application**: Systems like Google Assistant, Siri, and Alexa use deep learning models trained via backpropagation to recognize speech and generate responses.
   - **Impact of Delay**: Delays in training can impact the quality and accuracy of real-time speech-to-text conversion.
   - **Optimization**: Techniques like quantization and pruning reduce model complexity and speed up inference.

3. **Financial Market Predictions**

   - **Application**: Stock market prediction models rely on neural networks trained via backpropagation to analyse market trends.
   - **Impact of Delay**: A slow training process results in outdated models that fail to predict market movements effectively.
   - **Optimization**: Cloud-based distributed training speeds up backpropagation using multiple GPUs.

5. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, Disadvantagesidering few test data sets.

Introduction:

The Naïve Bayes classifier is a probabilistic machine learning algorithm based on Bayes' theorem. It is used for classification tasks and assumes that the features are conditionally independent given the class.

Bayes' Theorem:

Bayes' theorem is given by:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

where:

- P(Y|X) P(Y | X)P(Y|X) is the **posterior probability** (probability of class Y given input X).
- P(X|Y) P(X | Y)P(X|Y) is the **likelihood** (probability of input X given class Y).
- P(Y)P(Y)P(Y) is the **prior probability** (probability of class Y occurring).
- P(X)P(X)P(X) is the **evidence** (Disadvantages of probability of input X).

Algorithm:

1. Load the dataset and split it into training and testing sets.
2. Compute prior probabilities of each class in the training set.
3. Compute likelihood (conditional probability) of each feature given a class.
4. Apply Bayes' theorem to calculate the posterior probability for each class.
5. Assign the class label with the highest posterior probability.

Training and Testing Data Split:

- The dataset is split into training (80%) and testing (20%).
- Training data is used to estimate probabilities.
- Testing data is used to validate the model.

Advantages and Disadvantages:

Advantages:

- **Simple and Fast:** It is computationally efficient and requires less training time compared to other classifiers, making it suitable for real-time applications.
- **Works Well with Small Data:** Performs well even with a small dataset, making it useful for cases where limited data is available.
- **Handles High-Dimensional Data:** It works efficiently in scenarios with many features (e.g., text classification and spam filtering).

Disadvantages:

- **Strong Independence Assumption:** Assumes that all features are independent, which is rarely true in real-world applications and can lead to inaccurate predictions.
- **Poor Performance with Correlated Features:**If features are highly correlated, the model can give misleading probability estimates.
- **Zero Probability Issue:** If a category is missing in training data, Naïve Bayes assigns zero probability, causing issues in classification (solved using Laplace smoothing).

Suitable Data Sets and Justification:

Dataset	Reason
Spam Detection (Email Dataset)	Emails have independent features (words)
Sentiment Analysis (Twitter Dataset)	Text classification works well with Naïve Bayes
Loan Approval Dataset	Predicting approval based on independent financial features
Medical Diagnosis (Cancer Prediction)	Medical symptoms are often treated as independent

Justification:

- Naïve Bayes assumes independence of features → Suitable for text classification, medical data, spam filtering, and recommendation systems.

- Works well for categorical and numerical features → Ideal for structured tabular datasets.

Purpose of Libraries:

Library	Purpose
`numpy`	Numerical computations
`pandas`	Data handling and CSV file reading
`scikit-learn (sklearn)`	Implements `GaussianNB()` Naïve Bayes model
`train_test_split`	Splits dataset into training and testing sets
`accuracy_score`	Evaluates the model accuracy

Sample Dataset:

Dataset link:

https://drive.google.com/file/d/14j1wC5JBlay-CVZdh_0d9UZdGSlXxupk/view?usp=drive_link

Dataset Snippet Of Sentiment_Ananlysis_Twitter.csv:

Tweet_ID	Username	Tweet_Tex	Sentiment
tweet_1	user_43	This place	Positive
tweet_2	user_17	The produ	Neutral
tweet_3	user_32	The new d	Neutral
tweet_4	user_43	This song i	Neutral
tweet_5	user_16	I hate whe	Negative
tweet_6	user_4	Can't belie	Negative
tweet_7	user_26	Why do th	Neutral
tweet_8	user_27	The new d	Neutral
tweet_9	user_14	Can't belie	Negative
tweet_10	user_24	The produ	Neutral
tweet_11	user_23	The movie	Positive
tweet_12	user_6	The new d	Neutral
tweet_13	user_35	What a be	Neutral
tweet_14	user_22	The food a	Neutral
tweet_15	user_25	I am so tire	Neutral
tweet_16	user_10	This place	Positive
tweet_17	user_41	The food a	Neutral
tweet_18	user_39	I hate whe	Negative

Program:

With Libraries:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

df = pd.read_csv("sentiment_analysis_twitter.csv")

X_train, X_test, y_train, y_test = train_test_split(df["Tweet_Text"], df["Sentiment"], test_size=0.2, random_state=42)

vectorizer = TfidfVectorizer(stop_words="english")
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

model = MultinomialNB()
model.fit(X_train_tfidf, y_train)

y_pred = model.predict(X_test_tfidf)

accuracy = accuracy_score(y_test, y_pred)
print(f"Naïve Bayes Accuracy (Using Libraries): {accuracy * 100:.2f}%")
```

Output:

```
Naive Bayes Accuracy (Using Libraries): 100.00%
```

Without Libraries:

```
import pandas as pd
import re
import numpy as np
from collections import defaultdict

df = pd.read_csv("sentiment_analysis_twitter.csv")

def preprocess_text(text):
```

```
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    return text

df["Cleaned_Tweet"] = df["Tweet_Text"].apply(preprocess_text)

train_size = int(0.8 * len(df))
train_data = df.iloc[:train_size]
test_data = df.iloc[train_size:]

classes = train_data["Sentiment"].unique()

class_probabilities = {cls: len(train_data[train_data["Sentiment"] == cls]) / len(train_data) for
cls in classes}


word_counts = {cls: defaultdict(int) for cls in classes}
total_words = {cls: 0 for cls in classes}

for _, row in train_data.iterrows():
    words = row["Cleaned_Tweet"].split()
    sentiment = row["Sentiment"]
    for word in words:
        word_counts[sentiment][word] += 1
        total_words[sentiment] += 1

def predict_naive_bayes(tweet):
    words = preprocess_text(tweet).split()
    probs = {}

    for cls in classes:
        probs[cls] = class_probabilities[cls]
        for word in words:
            probs[cls] *= (word_counts[cls][word] + 1) / (total_words[cls] +
len(word_counts[cls]))

    return max(probs, key=probs.get)
test_data = test_data.copy()
test_data.loc[:, "Predicted_Sentiment"] =test_data["Tweet_Text"].apply(predict_naive_bayes)
accuracy = np.mean(test_data["Predicted_Sentiment"] == test_data["Sentiment"])
print(f"Manual Naïve Bayes Accuracy: {accuracy * 100:.2f}%")
```

Output:

```
Manual Na?ve Bayes Accuracy: 91.67%
```

Real Time Applications:

1. **Spam Filtering**

   - Used in email filtering systems (e.g., Gmail, Outlook) to classify emails as spam or non-spam based on words, frequency, and sender behaviour.
   - Example: Spam Assassin, a popular open-source spam filtering system, uses Naïve Bayes.

2. **Text Classification and Sentiment Analysis**

   - Used in news categorization, document classification, and sentiment analysis of social media posts, reviews, and comments.
   - Example: Twitter sentiment analysis for classifying tweets as positive, negative, or neutral.
   - Used in automated customer support (e.g., chatbots detecting user sentiment).

3. **Recommendation Systems**

   - Helps recommend books, movies, products, or advertisements based on user preferences.
   - Example: Netflix and Amazon use Naïve Bayes to suggest movies or products based on customer behaviour.

4. **Medical Diagnosis and Disease Prediction**

   - Used in predicting diseases based on symptoms, lab results, and patient history.
   - Example: Detecting cancer, heart disease, or diabetes using historical patient data.
   - Often applied in AI-powered medical chatbots for preliminary diagnosis.

6. Assuming a set of documents that need to be classified, use the naïve 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.

Introduction:

The Naïve Bayes classifier is a probabilistic machine learning algorithm based on Bayes' theorem. It is used for classification tasks and assumes that the features are conditionally independent given the class.

Bayes' Theorem:

Bayes' theorem is given by:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

where:

- P(Y|X) P(Y | X)P(Y|X) is the **posterior probability** (probability of class Y given input X).
- P(X|Y) P(X | Y)P(X|Y) is the **likelihood** (probability of input X given class Y).
- P(Y)P(Y)P(Y) is the **prior probability** (probability of class Y occurring).
- P(X)P(X)P(X) is the **evidence** (Disadvantagestant probability of input X).

Algorithm Steps:

We will implement Naive Bayes Classification in Java for the Spam Detection dataset using built-in Java classes and libraries.

1. **Load Dataset** (spam_detection_emails.csv)
2. **Preprocess the Text** (Remove special characters, lowercase conversion)
3. **Train the Naïve Bayes Model**:
   - Calculate **prior probabilities** for "Spam" and "Ham".
   - Compute **likelihood probabilities** for each word.
4. **Classify Emails** using Bayes' Theorem.
5. **Evaluate Performance**:

- Accuracy = $\frac{\text{Correct Predictions}}{\text{Total Predictions}}$
- Precision = $\frac{\text{True Positives}}{\text{True Positives}+\text{False Positives}}$
- Recall = $\frac{\text{True Positives}}{\text{True Positives}+\text{False Negatives}}$

Sample Dataset:

Dataset link:

https://drive.google.com/file/d/1f1yMNzFpmdztoySjD8fRiXZ7f8-YQiLW/view?usp=drive_link

Dataset Snippet Of Spam_Detection_Emails.csv:

email_2	user_26	Limited-tin	Spam
email_3	user_8	Limited-tin	Spam
email_4	user_20	Join thousa	Spam
email_5	user_2	Get rich qu	Spam
email_6	user_36	Exclusive a	Spam
email_7	user_41	Meeting re	Ham
email_8	user_50	Join thousa	Spam
email_9	user_22	Limited-tin	Spam
email_10	user_1	Join thousa	Spam
email_11	user_22	Earn mone	Spam
email_12	user_4	Meeting re	Ham
email_13	user_29	Your invoic	Ham
email_14	user_38	Exclusive o	Spam
email_15	user_25	Dinner plar	Ham
email_16	user_43	Earn mone	Spam
email_17	user_23	Get rich qu	Spam
email_18	user_10	Exclusive o	Spam
email_19	user_44	Earn mone	Spam

Program:

```
import java.io.*;
import java.util.*;

public class NaiveBayesWithoutLibraries {
    private static Map<String, Integer>spamWordCounts = new HashMap<>();
    private static Map<String, Integer>hamWordCounts = new HashMap<>();
    private static int spamEmails = 0, hamEmails = 0;
    private static int totalSpamWords = 0, totalHamWords = 0;
    private static Set<String> vocabulary = new HashSet<>();

    public static void main(String[] args) {
        String filePath = "spam_detection_emails.csv";
        List<String[]> dataset = loadCSV(filePath);

        int trainSize = (int) (dataset.size() * 0.8);
        List<String[]>trainData = dataset.subList(0, trainSize);
        List<String[]>testData = dataset.subList(trainSize, dataset.size());

        trainNaiveBayes(trainData);
        evaluateNaiveBayes(testData);
    }
```

```
private static List<String[]>loadCSV(String filePath) {
   List<String[]> data = new ArrayList<>();
   try (BufferedReaderbr = new BufferedReader(new FileReader(filePath))) {
      String line;
      br.readLine();
      while ((line = br.readLine()) != null) {
         String[] values = line.split(",", 4);
         data.add(new String[]{values[2], values[3]});
      }
   } catch (IOException e) {
      e.printStackTrace();
   }
   return data;
}

private static void trainNaiveBayes(List<String[]>trainData) {
   for (String[] entry : trainData) {
      String emailText = preprocess(entry[0]);
      String label = entry[1];

      String[] words = emailText.split("\\s+");
      if (label.equalsIgnoreCase("Spam")) {
         spamEmails++;
         for (String word : words) {
            spamWordCounts.put(word, spamWordCounts.getOrDefault(word, 0) + 1);
            totalSpamWords++;
            vocabulary.add(word);
         }
      } else {
         hamEmails++;
         for (String word : words) {
            hamWordCounts.put(word, hamWordCounts.getOrDefault(word, 0) + 1);
            totalHamWords++;
            vocabulary.add(word);
         }
      }
   }
}

private static String predictNaiveBayes(String emailText) {
   String[] words = preprocess(emailText).split("\\s+");
   double spamProb = Math.log((double) spamEmails / (spamEmails + hamEmails));
```

```
        double hamProb = Math.log((double) hamEmails / (spamEmails + hamEmails));

        for (String word : words) {
                spamProb += Math.log((spamWordCounts.getOrDefault(word, 0) + 1.0) /
(totalSpamWords + vocabulary.size()));
                 hamProb += Math.log((hamWordCounts.getOrDefault(word, 0) + 1.0) /
(totalHamWords + vocabulary.size()));
        }
        return spamProb>hamProb ? "Spam" : "Ham";
    }

    private static void evaluateNaiveBayes(List<String[]>testData) {
        int correct = 0, total = testData.size();
        int tp = 0, fp = 0, tn = 0, fn = 0;

        for (String[] entry : testData) {
            String actual = entry[1];
            String predicted = predictNaiveBayes(entry[0]);

            if (predicted.equals(actual)) {
                correct++;
                if (actual.equals("Spam")) tp++; else tn++;
            } else {
                if (predicted.equals("Spam")) fp++; else fn++;
            }
        }

        double accuracy = (double) correct / total;
        double precision = tp / (double) (tp + fp);
        double recall = tp / (double) (tp + fn);

        System.out.printf("Accuracy: %.2f%%\n", accuracy * 100);
        System.out.printf("Precision: %.2f\n", precision);
        System.out.printf("Recall: %.2f\n", recall);
    }

    private static String preprocess(String text) {
        return text.toLowerCase().replaceAll("[^a-z\\s]", "");
    }
}
```

Output:

```
Accuracy: 95.83%
Precision: 1.00
Recall: 0.93
```

Real Time Applications:

1. **Legal Document Classification**
   - Used in law firms to automatically classify legal documents based on their content (e.g., contracts, case laws).
   - Helps in e-discovery, where relevant legal documents are retrieved based on queries.

2. **Intrusion Detection Systems (IDS) and Cybersecurity**
   - Helps detect malware, phishing, and network intrusions by classifying incoming traffic patterns.
   - Example: Naïve Bayes is used in IDS tools to flag abnormal network behaviour.

3. **Customer Support and Chatbots**
   - Used in automated customer service bots to classify and respond to user queries efficiently.
   - Example: E-commerce platforms like Amazon and eBay use Naïve Bayes for support ticket classification.

4. **Stock Market and Financial Forecasting**
   - Helps classify market trends, stock price movements, and investment risks.
   - Example: Predicting whether a stock will rise or fall based on historical data and news sentiment.

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

Introduction:

A Naïve Bayesian Network is a probabilistic graphical model based on Bayes' Theorem. It assumes conditional independence between features given the class label, making it simple yet effective for classification tasks.

Algorithm:

1. **Preprocessing:** Load dataset, clean missing values, and split into training/testing sets.
2. **Feature Selection**: Identify important features influencing the target variable (e.g., heart disease presence).
3. **Probability Estimation:** Compute:
    - Prior probability: P(Y)P(Y)P(Y) (Probability of class occurrence)
    - Likelihood: P(X|Y)P(X|Y)P(X|Y) (Probability of features given a class)
    - Evidence: P(X)P(X)P(X) (Overall probability of features)
4. **Bayes' Theorem Calculation:**

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

   where:
    - Y = Class label (Heart Disease: Yes/No)
    - X = Features (e.g., cholesterol, age, blood pressure)
5. **Prediction:** Assign the class with the highest probability.
6. **Model Evaluation:** Compute accuracy, precision, recall, F1-score.

Training and Testing Data Split:

- 80-20 split: 80% training, 20% testing
- Justification: Ensures sufficient data for training while keeping unseen data for evaluation.

Advantages and Disadvantages:

Advantages:

- Fast and scalable (even with large datasets).
- Works well with categorical and numerical data.
- Performs well even with small datasets.

Disadvantages:

- Assumes feature independence, which is often unrealistic.
- Struggles with highly correlated features.
- Performs poorly on complex relationships.

Suitable Data Sets and Justification:

Dataset	Reason for Suitability
Heart Disease Dataset	Predicts disease based on risk factors (age, cholesterol, etc.).
Spam Detection Dataset	Classifies emails as spam/ham using word probabilities.
Iris Dataset	Simple classification task (setosa, versicolor, virginica).
Customer Churn Dataset	Predicts customer retention based on demographics.

Why Heart Disease Dataset?

- Contains categorical and numerical features.
- Medical diagnosis is probabilistic, making Naïve Bayes useful.

Purpose of Libraries:

Library	Purpose
`pgmpy`	Builds **Bayesian Networks**
`pandas`	Handles **CSV data processing**
`numpy`	Performs **numerical computations**
`sklearn`	Splits dataset, applies ML models

Sample Dataset:

Dataset link:

https://drive.google.com/file/d/1eD7qGZ0vAlLcDOwiUwhppI1ARIpgfPy0/view?usp=sharing

Dataset Snippet Of Heart.csv:

age	sex	chest_pain	resting_blo	cholestoral	fasting_blo	rest_ecg	Max_heart	exercise_in	oldpeak	slope	vessels_col	thalassemia	target
52	Male	Typical ang	125	212	Lower than	ST-T wave a	168	No	1	Downslopir	Two	Reversable	0
53	Male	Typical ang	140	203	Greater tha	Normal	155	Yes	3.1	Upsloping	Zero	Reversable	0
70	Male	Typical ang	145	174	Lower than	ST-T wave a	125	Yes	2.6	Upsloping	Zero	Reversable	0
61	Male	Typical ang	148	203	Lower than	ST-T wave a	161	No	0	Downslopir	One	Reversable	0
62	Female	Typical ang	138	294	Greater tha	ST-T wave a	106	No	1.9	Flat	Three	Fixed Defec	0
58	Female	Typical ang	100	248	Lower than	Normal	122	No	1	Flat	Zero	Fixed Defec	1
58	Male	Typical ang	114	318	Lower than	Left ventric	140	No	4.4	Upsloping	Three	Normal	0
55	Male	Typical ang	160	289	Lower than	Normal	145	Yes	0.8	Flat	One	Reversable	0
46	Male	Typical ang	120	249	Lower than	Normal	144	No	0.8	Downslopir	Zero	Reversable	0
54	Male	Typical ang	122	286	Lower than	Normal	116	Yes	3.2	Flat	Two	Fixed Defec	0
71	Female	Typical ang	112	149	Lower than	ST-T wave a	125	No	1.6	Flat	Zero	Fixed Defec	1
43	Female	Typical ang	132	341	Greater tha	Normal	136	Yes	3	Flat	Zero	Reversable	0
34	Female	Atypical an	118	210	Lower than	ST-T wave a	192	No	0.7	Downslopir	Zero	Fixed Defec	1
51	Male	Typical ang	140	298	Lower than	ST-T wave a	122	Yes	4.2	Flat	Three	Reversable	0
52	Male	Typical ang	128	204	Greater tha	ST-T wave a	156	Yes	1	Flat	Zero	No	0

Program:

With Libraries:

```
import numpy as np
import pandas as pd
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import KBinsDiscretizer

url = "heart.csv"
columns = ['age', 'sex', 'chest_pain_type', 'resting_blood_pressure', 'cholesterol',
        'fasting_blood_sugar', 'rest_ecg', 'max_heart_rate', 'exercise_induced_angina',
        'oldpeak', 'slope', 'vessels_colored_by_flourosopy', 'thalassemia', 'target']

df = pd.read_csv(url, header=None, names=columns)

le = LabelEncoder()

df['sex'] = le.fit_transform(df['sex'])
df['chest_pain_type'] = le.fit_transform(df['chest_pain_type'])
df['fasting_blood_sugar'] = df['fasting_blood_sugar'].map({True: 1, False: 0})
df['rest_ecg'] = le.fit_transform(df['rest_ecg'])
```

```
df['exercise_induced_angina'] = df['exercise_induced_angina'].map({True: 1, False: 0})
df['slope'] = le.fit_transform(df['slope'])
df['vessels_colored_by_flourosopy'] = le.fit_transform(df['vessels_colored_by_flourosopy'
df['thalassemia'] = le.fit_transform(df['thalassemia'])

continuous_columns = ['age', 'resting_blood_pressure', 'cholesterol', 'oldpeak',
'max_heart_rate']
df[continuous_columns] = df[continuous_columns].apply(pd.to_numeric, errors='coerce')

df[continuous_columns] = df[continuous_columns].fillna(df[continuous_columns].median())

kbins = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')

df[continuous_columns] = kbins.fit_transform(df[continuous_columns])

X = df.drop(columns=['target'])
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

model = BayesianNetwork([
    ('age', 'target'),
    ('sex', 'target'),
    ('chest_pain_type', 'target'),
    ('resting_blood_pressure', 'target'),
    ('cholesterol', 'target'),
    ('fasting_blood_sugar', 'target'),
    ('rest_ecg', 'target'),
    ('max_heart_rate', 'target'),
    ('exercise_induced_angina', 'target'),
    ('oldpeak', 'target'),
    ('slope', 'target'),
    ('vessels_colored_by_flourosopy', 'target'),
    ('thalassemia', 'target')
])

cpd_age = TabularCPD(variable='age', variable_card=3, values=[[0.5], [0.3], [0.2]])
cpd_sex = TabularCPD(variable='sex', variable_card=2, values=[[0.7], [0.3]])
cpd_chest_pain_type = TabularCPD(variable='chest_pain_type', variable_card=4,
values=[[0.4], [0.2], [0.3], [0.1]])
cpd_resting_blood_pressure = TabularCPD(variable='resting_blood_pressure',
variable_card=3, values=[[0.4], [0.3], [0.3]])
```

```
cpd_cholesterol = TabularCPD(variable='cholesterol', variable_card=3, values=[[0.5], [0.3], [0.2]])
cpd_fasting_blood_sugar = TabularCPD(variable='fasting_blood_sugar', variable_card=2, values=[[0.8], [0.2]])
cpd_rest_ecg = TabularCPD(variable='rest_ecg', variable_card=3, values=[[0.5], [0.3], [0.2]])
cpd_max_heart_rate = TabularCPD(variable='max_heart_rate', variable_card=4, values=[[0.4], [0.3], [0.2], [0.1]])
cpd_exercise_induced_angina = TabularCPD(variable='exercise_induced_angina', variable_card=2, values=[[0.6], [0.4]])
cpd_oldpeak = TabularCPD(variable='oldpeak', variable_card=3, values=[[0.4], [0.3], [0.3]])
cpd_slope = TabularCPD(variable='slope', variable_card=3, values=[[0.4], [0.3], [0.3]])
cpd_vessels_colored_by_flourosopy = TabularCPD(variable='vessels_colored_by_flourosopy', variable_card=4, values=[[0.5], [0.3], [0.1], [0.1]])
cpd_thalassemia = TabularCPD(variable='thalassemia', variable_card=3, values=[[0.5], [0.3], [0.2]])

num_combinations = np.prod([3, 2, 4, 3, 3, 2, 3, 4, 2, 3, 3, 4, 3])

target_values = np.random.rand(2, num_combinations)

target_values /= target_values.sum(axis=0)

cpd_target = TabularCPD(
    variable='target',
    variable_card=2,
    0 = No heart disease, 1 = Heart disease
    values=target_values,
    evidence=['age', 'sex', 'chest_pain_type', 'resting_blood_pressure', 'cholesterol',
              'fasting_blood_sugar', 'rest_ecg', 'max_heart_rate', 'exercise_induced_angina',
              'oldpeak', 'slope', 'vessels_colored_by_flourosopy', 'thalassemia'],
    evidence_card=[3, 2, 4, 3, 3, 2, 3, 4, 2, 3, 3, 4, 3]
)

model.add_cpds(cpd_age, cpd_sex, cpd_chest_pain_type, cpd_resting_blood_pressure, cpd_cholesterol, cpd_fasting_blood_sugar, cpd_rest_ecg, cpd_max_heart_rate, cpd_exercise_induced_angina,  cpd_oldpeak, cpd_slope,cpd_vessels_colored_by_flourosopy, cpd_thalassemia, cpd_target)

inference = VariableElimination(model)

sample_patient = {
    'age': 0,
```

```
    'sex': 1,
    'chest_pain_type': 0,
    'resting_blood_pressure': 1,
    'cholesterol': 1,
    'fasting_blood_sugar': 0,
    'rest_ecg': 0,
    'max_heart_rate': 2,
    'exercise_induced_angina': 0,
    'oldpeak': 1,
    'slope': 1,
    'vessels_colored_by_flourosopy': 2,
    'thalassemia': 0
}

query = inference.query(variables=['target'], evidence=sample_patient)
print(query)
```

Output:

```
+-----------+---------------+
| target    |   phi(target) |
+===========+===============+
| target(0) |        0.4922 |
+-----------+---------------+
| target(1) |        0.5078 |
+-----------+---------------+
```

Without Libraries:

```
import pandas as pd

file_path = "heart.csv"
df = pd.read_csv(file_path)

p_heart_disease = len(df[df['target'] == 1]) / len(df)

cp_values = df['chest_pain_type'].unique()
p_cp_given_hd = {}

for cp in cp_values:
    p_cp_given_hd[cp] = len(df[(df['chest_pain_type'] == cp) and (df['target'] == 1)]) / len(df[df['target'] == 1])
```

```
print("Probability of Heart Disease:", p_heart_disease)
print("Conditional Probabilities of Chest Pain Type given Heart Disease:")
print(p_cp_given_hd)
```

Output:

```
Probability of Heart Disease: 0.5131707317073171
Conditional Probabilities of Chest Pain Type given Heart Disease:
{'Typical angina': 0.23193916349809887, 'Atypical angina': 0.25475285171102663, 'Non-anginal pain': 0.41634980988593157, 'Asymptomatic': 0.
09695817490494296}
```

Real Time Applications:

1. **Sentiment Analysis and Text Classification**
   - Used to analyse the tone and sentiment in social media posts, product reviews, and customer feedback.
   - Example: Twitter Sentiment Analysis to classify tweets as positive, negative, or neutral.
   - Applied in news categorization, where articles are classified into topics like politics, sports, or entertainment.

2. **Medical Diagnosis and Disease Prediction**
   - Used in AI-powered medical diagnosis to predict diseases based on symptoms and patient history.
   - Example: Predicting diabetes, cancer, or heart disease based on patient medical records.
   - Helps doctors make early diagnoses and provide better treatment recommendations.

8. 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.

Introduction:

Clustering:

Clustering is an unsupervised learning technique that groups data points into clusters based on similarities. It helps in pattern recognition, data segmentation, and anomaly detection.

k-Means Clustering:

k-Means is a partition-based clustering algorithm that groups data points into KKK clusters by minimizing the sum of squared distances between points and their assigned centroids.

Expectation-Maximization (EM):

Expectation-Maximization (EM) is an iterative optimization algorithm that finds maximum likelihood estimates of parameters when data is incomplete or has latent variables.

Silhouette Score (Clustering Quality Measure):

The silhouette score measures how well each data point fits within its assigned cluster. It ranges from -1 to 1:

- 1: Perfect clustering
- 0: Overlapping clusters
- -1: Poor clustering

$$S(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

where:

- a(i)a(i)a(i) = Average intra-cluster distance (distance within the same cluster)
- b(i)b(i)b(i) = Average nearest-cluster distance (distance to the closest other cluster)

Algorithm:

For k-Means Clustering:

1. Choose KKK initial centroids randomly.
2. Assign each data point to the closest centroid.
3. Recalculate the centroids by computing the mean of assigned points.
4. Repeat until convergence (i.e., centroids do not change).

For Expectation-Maximization (EM):

1. Initialize parameters (e.g., mean, covariance, probabilities).
2. Expectation Step (E-Step): Calculate probability distribution over latent variables.
3. Maximization Step (M-Step): Update parameters based on computed probabilities.
4. Repeat until convergence (i.e., parameter changes are minimal).

Training and Testing Data Split:

- Training Data: Used to train the clustering model.
- Testing Data: Used to evaluate model performance.

Common Splits:

Training	Testing
80%	20%
70%	30%
90%	10%

Advantages and Disadvantages:

For k-Means Clustering:

Pros	Cons
Fast & easy to implement	Assumes spherical clusters
Works well with large datasets	Sensitive to outliers & cluster initialization
Converges quickly	Requires specifying K in advance

For Expectation-Maximization (EM):

Pros	Cons
Handles overlapping clusters	Computationally expensive
Soft clustering (probabilistic assignments)	Requires good initialization
Works well with complex distributions	Sensitive to local optima

Suitable Data Sets and Justification:

Dataset Type	Suitable Algorithm	Reason
Well-separated clusters	k-Means	Works best for spherical clusters
Overlapping clusters	EM (GMM)	Models complex distributions
Non-linear clusters	EM (GMM)	k-Means fails on non-spherical clusters

Justification:

- Provides controlled testing conditions.
- Helps compare algorithm behaviour without real-world noise.
- Ensures equal cluster distribution for fair evaluation.

Purpose of Libraries:

Library	Purpose
NumPy & Pandas	Data manipulation
Scikit-learn (sklearn)	Machine learning models (k-Means, EM-GMM)
Matplotlib	Visualization

Sample Dataset:

Dataset link:

https://drive.google.com/file/d/1_wqsvUTZRyMcPppIlcqS8E4a5n_ebQW_/view?usp=sharing

Dataset Snippet OfCustomers.csv:

CustomerID	Gender	Age	Annual Inc	Spending S	Profession	Work Expe	Family Size
1	Male	19	15000	39	Healthcare	1	4
2	Male	21	35000	81	Engineer	3	3
3	Female	20	86000	6	Engineer	1	1
4	Female	23	59000	77	Lawyer	0	2
5	Female	31	38000	40	Entertainm	2	6
6	Female	22	58000	76	Artist	0	2
7	Female	35	31000	6	Healthcare	1	3
8	Female	23	84000	94	Healthcare	1	3
9	Male	64	97000	3	Engineer	0	3
10	Female	30	98000	72	Artist	1	4
11	Male	67	7000	14	Engineer	1	3
12	Female	35	93000	99	Healthcare	4	4
13	Female	58	80000	15	Executive	0	5
14	Female	24	91000	77	Lawyer	1	1
15	Male	37	19000	13	Doctor	0	1
16	Male	22	51000	79	Healthcare	1	2
17	Female	35	29000	35	Homemake	9	5
18	Male	20	89000	66	Healthcare	1	6
19	Male	52	20000	29	Entertainm	1	4
20	Female	35	62000	98	Artist	0	1

Program:

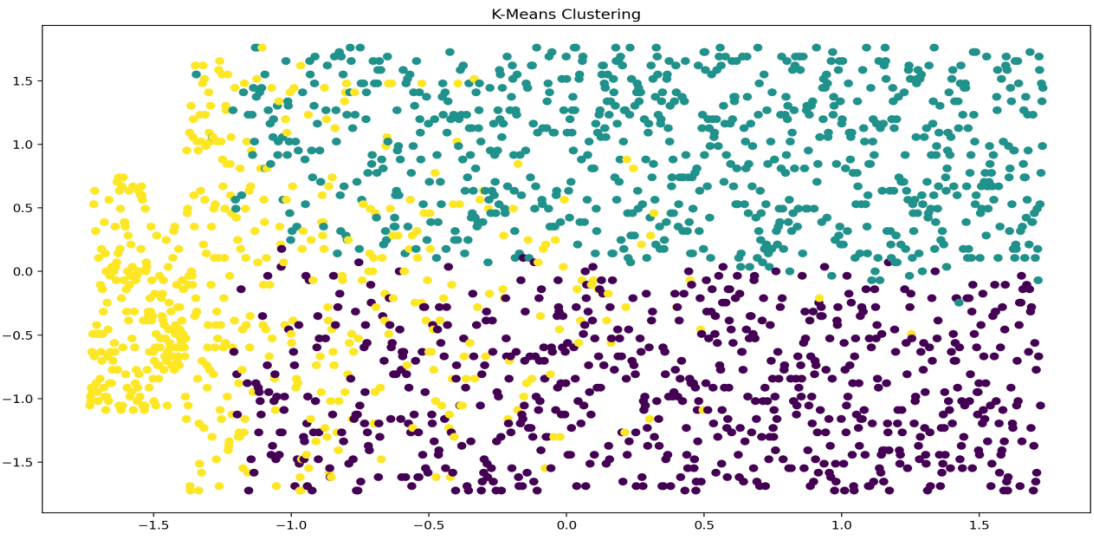
With Libraries:

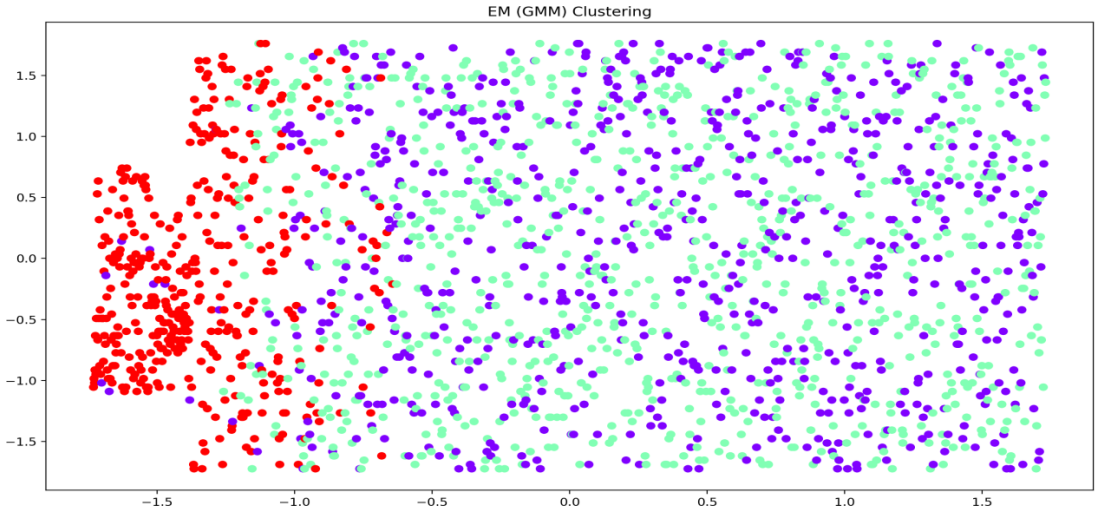
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score

file_path = "Customers.csv"
df = pd.read_csv(file_path)

df_numeric = df.select_dtypes(include=[np.number])

df_numeric.fillna(df_numeric.mean(), inplace=True)

scaler = StandardScaler()
```

```
df_scaled = scaler.fit_transform(df_numeric)

kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
kmeans_labels = kmeans.fit_predict(df_scaled)
kmeans_silhouette = silhouette_score(df_scaled, kmeans_labels)

gmm = GaussianMixture(n_components=3, random_state=42)
gmm_labels = gmm.fit_predict(df_scaled)
gmm_silhouette = silhouette_score(df_scaled, gmm_labels)

print("\nComparison of Clustering Quality:")
print(f"K-Means Silhouette Score: {kmeans_silhouette:.4f}")
print(f"EM (GMM) Silhouette Score: {gmm_silhouette:.4f}")

plt.figure(figsize=(10, 5))
plt.scatter(df_scaled[:, 0], df_scaled[:, 1], c=kmeans_labels, cmap='viridis')
plt.title("K-Means Clustering")
plt.show()

plt.figure(figsize=(10, 5))
plt.scatter(df_scaled[:, 0], df_scaled[:, 1], c=gmm_labels, cmap='rainbow')
plt.title("EM (GMM) Clustering")
plt.show()
```

Output:

```
Comparison of Clustering Quality:
K-Means Silhouette Score: 0.1527
EM (GMM) Silhouette Score: 0.1303
```





Without Libraries:

```
import pandas as pd
import numpy as np
import random
import math

file_path = "Customers.csv"
df = pd.read_csv(file_path)


df_numeric = df.select_dtypes(include=[np.number])
df_numeric.fillna(df_numeric.mean(), inplace=True)

data = df_numeric.to_numpy()

def min_max_scale(data):
    return (data - np.min(data, axis=0)) / (np.max(data, axis=0) - np.min(data, axis=0))

data = min_max_scale(data)

def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2) ** 2))

def k_means(data, k=3, max_iters=100):
    centroids = data[random.sample(range(data.shape[0]), k)]

    for _ in range(max_iters):
```

```
        clusters = [[] for _ in range(k)]

        for point in data:
            distances = [euclidean_distance(point, centroid) for centroid in centroids]
            cluster_index = np.argmin(distances)
            clusters[cluster_index].append(point)

         new_centroids = [np.mean(cluster, axis=0) if cluster else centroids[i] for i, cluster in enumerate(clusters)]

        if np.allclose(centroids, new_centroids):
            break
        centroids = new_centroids

    return np.array([np.argmin([euclidean_distance(point, centroid) for centroid in centroids]) for point in data])

kmeans_labels = k_means(data, k=3)

def gaussian_pdf(x, mean, cov):
    dim = len(mean)
    det = np.linalg.det(cov)
    norm_Disadvantagest = 1.0 / (np.power((2 * np.pi), float(dim) / 2) * np.sqrt(det))
    x_mu = x - mean
     return norm_Disadvantagest * np.exp(-0.5 * np.dot(np.dot(x_mu.T, np.linalg.inv(cov)), x_mu))

def expectation_maximization(data, k=3, max_iters=100):
    n, d = data.shape
    weights = np.ones((n, k)) / k
    means = data[random.sample(range(n), k)]
    covariances = [np.cov(data.T) for _ in range(k)]
    priors = np.ones(k) / k

    for _ in range(max_iters):
            for i in range(n):
            for j in range(k):
                weights[i, j] = priors[j] * gaussian_pdf(data[i], means[j], covariances[j])
            weights[i] /= np.sum(weights[i])

        for j in range(k):
            responsibility = weights[:, j]
            sum_resp = np.sum(responsibility)
```

```
            means[j] = np.dot(responsibility, data) / sum_resp
            covariances[j] = np.cov(data.T, aweights=responsibility, bias=True)
            priors[j] = sum_resp / n

    return np.argmax(weights, axis=1)

gmm_labels = expectation_maximization(data, k=3)
print("\nManual Clustering Results:")
print("K-Means Cluster Labels:", kmeans_labels[:10])
print("EM Cluster Labels:", gmm_labels[:10])
```

Output:

```
Manual Clustering Results:
K-Means Cluster Labels: [1 1 1 1 1 1 1 1 1 1]
EM Cluster Labels: [1 1 1 1 1 1 1 1 1 1]
```

Real Time Applications:

For Expectation-Maximization (EM):

The EM algorithm is primarily used for probabilistic clustering, density estimation, and hidden variable problems. Some key applications include:

1. **Speech Recognition**
   - Used in Hidden Markov Models (HMMs) to estimate parameters for speech signals.
   - Helps improve speech-to-text conversion in virtual assistants like Google Assistant, Siri, and Alexa.

2. **Natural Language Processing (NLP)**
   - Used in text classification and topic modelling, such as Latent Dirichlet Allocation (LDA).
   - Helps in unsupervised machine translation models.

For k-Means Clustering:

The k-Means algorithm is used for unsupervised clustering in real-world problems. Some key applications include:

1. **Customer Segmentation (Marketing and Retail)**
   - Used by e-commerce platforms like Amazon, Netflix, and Spotify to segment users based on behaviour.
   - Helps in personalized marketing and recommendation systems.

2. **Image Compression**
   - Used in reducing image size by clustering pixels into fewer colors.
   - Applied in JPEG compression techniques.

9. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

Introduction:

k-Nearest Neighbours (k-NN) is a supervised machine learning algorithm used for classification and regression tasks. It classifies a data point based on how its neighbours are classified. The majority vote of the k-nearest data points determines the classification of a new instance.

Example: If k=3 and the three nearest neighbours belong to the class "Iris Setosa", the new data point is classified as Iris Setose.

Algorithm:

1. Load the dataset (Training and Testing).
2. Choose the number of neighbours (k).
3. For each test data point:
    - Compute the Euclidean distance (or other distance metrics) from each training point.
    - Sort distances and select k-nearest neighbours.
    - Perform a majority vote among neighbours.
4. Assign the most common class label to the test data.
5. Evaluate model performance using accuracy, precision, recall, etc.

Training and Testing Data Split:

The dataset is split into:

- Training Set (80%): Used to train the k-NN model.
- Testing Set (20%): Used to evaluate performance.

A train-test split prevents over fitting and ensures the model generalizes well to unseen data.

Advantages and Disadvantages:

Advantages:

- Simple and easy to understand.
- Works well with small datasets.

- No training phase (instance-based learning).

Disadvantages:

- Computationally expensive for large datasets.
- Sensitive to irrelevant or redundant features.
- Needs feature scaling (e.g., Min-Max Scaling).

Suitable Data Sets and Justification:

1. **Iris Dataset (Classification):**
   - **Why?** Well-balanced, small dataset with clear class separations.
   - **Features:** Sepal Length, Petal Length, etc.

2. **MNIST Handwritten Digits (Classification):**
   - **Why?** Image classification problem where k-NN works well with proper feature extraction.

3. **Breast Cancer Dataset (Medical Classification):**
   - **Why?** Used to detect malignancy; k-NN performs well on structured medical data.

4. **Customer Segmentation Dataset (Clustering):**
   - **Why?** Used in marketing to group customers based on spending behaviour.

Purpose of Libraries:

Library	Purpose
`pandas`	Load and process datasets
`numpy`	Perform numerical computations
`scikit-learn`	Preprocessing, model training, evaluation
`matplotlib/seaborn`	Visualization (optional)

Sample Dataset:

Dataset link:

https://drive.google.com/file/d/1E9qQumsCoNTCfz2tpodVMgexreh_2xfG/view?usp=sharing

Dataset Snippet Of Iris_data.csv:

Sepal_Leng	Sepal_Widt	Petal_Leng	Petal_Widt	Feature_Names
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa
4.8	3.4	1.6	0.2	Iris-setosa
4.8	3	1.4	0.1	Iris-setosa
4.3	3	1.1	0.1	Iris-setosa
5.8	4	1.2	0.2	Iris-setosa
5.7	4.4	1.5	0.4	Iris-setosa
5.4	3.9	1.3	0.4	Iris-setosa
5.1	3.5	1.4	0.3	Iris-setosa

Program:

With Libraries:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score


from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

```
y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

correct = np.sum(y_pred == y_test)
wrong = np.sum(y_pred != y_test)

print(f"Correct Predictions: {correct}")
print(f"Wrong Predictions: {wrong}")
print(f"Accuracy: {accuracy:.2f}")

print("\nActual vs Predicted:")
for i in range(len(y_test)):
    print(f"Actual: {iris.target_names[y_test[i]]}, Predicted: {iris.target_names[y_pred[i]]}")
```

Output:

```
Correct Predictions: 30
Wrong Predictions: 0
Accuracy: 1.00

Actual vs Predicted:
Actual: versicolor, Predicted: versicolor
Actual: setosa, Predicted: setosa
Actual: virginica, Predicted: virginica
Actual: versicolor, Predicted: versicolor
Actual: versicolor, Predicted: versicolor
Actual: setosa, Predicted: setosa
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: virginica
Actual: versicolor, Predicted: versicolor
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: virginica
Actual: setosa, Predicted: setosa
Actual: setosa, Predicted: setosa
Actual: setosa, Predicted: setosa
Actual: setosa, Predicted: setosa
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: virginica
Actual: versicolor, Predicted: versicolor
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: virginica
Actual: setosa, Predicted: setosa
Actual: virginica, Predicted: virginica
Actual: setosa, Predicted: setosa
Actual: virginica, Predicted: virginica
Actual: virginica, Predicted: virginica
Actual: virginica, Predicted: virginica
Actual: virginica, Predicted: virginica
Actual: virginica, Predicted: virginica
Actual: setosa, Predicted: setosa
Actual: setosa, Predicted: setosa
```

Without Libraries:

```
import numpy as np
import pandas as pd
```

```
from collections import Counter

from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target

np.random.seed(42)
indices = np.random.permutation(len(X))
split = int(0.8 * len(X))

X_train, X_test = X[indices[:split]], X[indices[split:]]
y_train, y_test = y[indices[:split]], y[indices[split:]]

def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))

def knn_predict(X_train, y_train, test_sample, k=3):
    distances = []
    for i, train_sample in enumerate(X_train):
        dist = euclidean_distance(test_sample, train_sample)
        distances.append((dist, y_train[i]))
    distances.sort(key=lambda x: x[0])
    k_nearest_labels = [label for _, label in distances[:k]]
    return Counter(k_nearest_labels).most_common(1)[0][0]

y_pred = np.array([knn_predict(X_train, y_train, sample, k=3) for sample in X_test])

correct = np.sum(y_pred == y_test)
wrong = np.sum(y_pred != y_test)
accuracy = correct / len(y_test)

print(f"Correct Predictions: {correct}")
print(f"Wrong Predictions: {wrong}")
print(f"Accuracy: {accuracy:.2f}")

print("\nActual vs Predicted:")
for i in range(len(y_test)):
    print(f"Actual: {iris.target_names[y_test[i]]}, Predicted: {iris.target_names[y_pred[i]]}")
```

Output:

```
Correct Predictions: 29
Wrong Predictions: 1
Accuracy: 0.97

Actual vs Predicted:
Actual: versicolor, Predicted: versicolor
Actual: setosa, Predicted: setosa
Actual: versicolor, Predicted: versicolor
Actual: versicolor, Predicted: versicolor
Actual: setosa, Predicted: setosa
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: virginica
Actual: virginica, Predicted: virginica
Actual: setosa, Predicted: setosa
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: virginica
Actual: virginica, Predicted: virginica
Actual: setosa, Predicted: setosa
Actual: virginica, Predicted: virginica
Actual: setosa, Predicted: setosa
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: virginica
Actual: virginica, Predicted: virginica
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: virginica
Actual: versicolor, Predicted: versicolor
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: virginica
Actual: virginica, Predicted: virginica
Actual: setosa, Predicted: setosa
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: versicolor
Actual: setosa, Predicted: setosa
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: virginica
```

Real Time Applications:

The k-Nearest Neighbor (k-NN) algorithm is widely used in real-time applications due to its simplicity and effectiveness. Here are some key applications:

1. **Healthcare and Medical Diagnosis**
   - Disease prediction (e.g., diabetes, cancer detection).
   - Patient classification based on symptoms and medical history.

2. **Recommendation Systems**
   - Used in Netflix, YouTube, and Amazon for movie, video, and product recommendations.
   - Suggesting similar users or items based on past preferences.

3. **Fraud Detection and Cybersecurity**
   - Identifying fraudulent transactions in credit card processing.
   - Detecting anomalies in network traffic and intrusion detection.

10. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Introduction:

Locally Weighted Regression (LWR) is a non-parametric regression algorithm that makes predictions using only nearby data points rather than fitting a single global function to all data. It assigns weights to training data points based on their proximity to the query point, emphasizing local patterns.

Algorithm:

1. Choose a query point $x_q$where we want to predict y_q
2. Compute weights for all training points using a kernel function:

$$w_i = \exp\left(\frac{-(x_i - x_q)^2}{2\tau^2}\right)$$

   - w_i is the weight for the i^{th} training point.
   - τ (bandwidth) controls how much influence distant points have.
3. Compute the locally weighted regression model for the query point:
   - Fit a weighted linear regression model:

$$\theta = (X^T W X)^{-1} X^T W y$$

   where W is a diagonal matrix of weights.

4. Predict $y_q$for x_q using the learned parameters.
5. Repeat for multiple test points to Disadvantagestruct the regression curve.

Training and Testing Data Split:

LWR doesn't require explicit training; it computes weights dynamically based on the test point. However, we can still evaluate it using:

- 80% training data (used for making predictions)
- 20% test data (used to evaluate performance)

Advantages and Disadvantages:

Advantages:

- Captures local trends better than global models.
- No assumption of data distribution, unlike linear regression.
- More flexible than parametric models.

Disadvantages:

- Slow for large datasets (since it computes regression separately for each test point).
- Requires careful selection of τ(bandwidth affects smoothness).
- Not interpretable as a global model (no single equation for all data).

Suitable Data Sets and Justification:

LWR works well on datasets with non-linear relationships. Suitable datasets include:

1. **Stock Market Data**
   - **Justification:** Stock prices fluctuate in non-linear patterns. LWR captures these local variations better than a simple linear model.
2. **House Price Prediction**
   - **Justification:** Prices depend on location, size, and other factors with complex interactions.
3. **Medical Data (Patient Diagnoses)**
   - **Justification:** Different health factors influence diagnosis non-linearly.
4. **Weather Forecasting**
   - **Justification:** Temperature, humidity, and pressure exhibit local variations.

Why Stock Market Data in Our Case?

- Highly Non-Linear: Prices fluctuate based on multiple unpredictable factors.
- Short-Term Trends Matter: LWR effectively captures local short-term price movements.

Purpose of Libraries:

Library	Purpose
NumPy	For efficient matrix operations, essential for regression calculations.
Pandas	To load and manipulate the stock dataset easily.
Matplotlib	To visualize LWR predictions against real stock prices.
SciPy (cdist)	Efficient distance calculations for weighting functions.

Sample Dataset:

Dataset link:

https://drive.google.com/file/d/19eIhft8p-P7oQ01DEphOKsVR3rRizQ9V/view?usp=sharing

Dataset Snippet OfTITAN.NS.csv:

Date	Open	High	Low	Close	Adj Close	Volume
29-08-2022	2457.25	2540.9	2457.25	2531.7	2514.938	1149676
30-08-2022	2545	2623.1	2543	2604.65	2587.405	1794736
01-09-2022	2592.95	2647.15	2565.65	2622	2604.64	1438313
02-09-2022	2631	2645	2607.05	2612.6	2595.302	800453
05-09-2022	2625	2640	2610.85	2627.35	2609.955	733481
06-09-2022	2631.7	2646.9	2611	2631.65	2614.226	680071
07-09-2022	2607	2641.4	2600.4	2637.75	2620.286	438635
08-09-2022	2649	2659.9	2616	2626.7	2609.309	754150
09-09-2022	2652	2652.15	2593	2603.45	2586.213	777186
12-09-2022	2616.8	2669.7	2610	2661.15	2643.531	808360
13-09-2022	2680	2723	2672	2705.55	2687.637	1536444
14-09-2022	2645	2712	2645	2700.6	2682.72	915281
15-09-2022	2714.15	2717	2648	2656.15	2638.564	753648
16-09-2022	2648	2677.65	2604.25	2611.8	2594.508	1425731
19-09-2022	2611.75	2632	2552.55	2619.8	2602.455	808421
20-09-2022	2642.2	2693.55	2635.1	2684	2666.23	844553
21-09-2022	2688	2700	2656.65	2666.65	2648.994	1052572
22-09-2022	2660	2744.95	2650	2737.6	2719.475	1352472

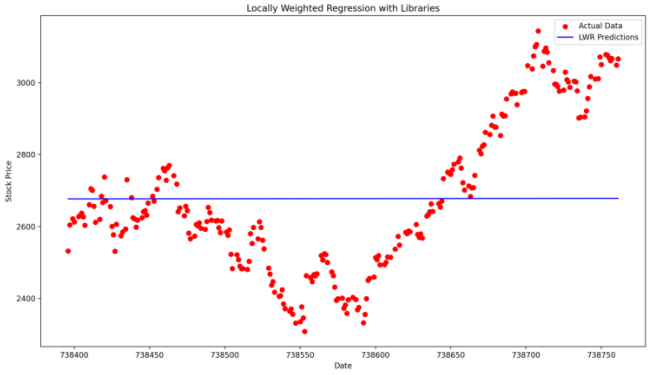
Program:

With Libraries:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
```

```
file_path = "TITAN.NS.csv"
df = pd.read_csv(file_path)

df['Date'] = pd.to_datetime(df['Date'])
df['Date'] = df['Date'].map(lambda x: x.toordinal())
X = df[['Date']].values
y = df['Close'].values

def locally_weighted_regression(X_train, y_train, X_test, tau=0.1):
    m = X_train.shape[0]
    y_pred = np.zeros(len(X_test))

    for i, x in enumerate(X_test):
        W = np.exp(-cdist(X_train, [x], 'sqeuclidean') / (2 * tau ** 2))
        W = np.diag(W.flatten())

        theta = np.linalg.pinv(X_train.T @ W @ X_train) @ X_train.T @ W @ y_train
        y_pred[i] = np.dot(x, theta)

    return y_pred

X_test = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
y_pred = locally_weighted_regression(X, y, X_test, tau=10000)

plt.scatter(X, y, color='red', label="Actual Data")
plt.plot(X_test, y_pred, color='blue', label="LWR Predictions")
plt.xlabel("Date")
plt.ylabel("Stock Price")
plt.legend()
plt.title("Locally Weighted Regression with Libraries")
plt.show()
```

Output:

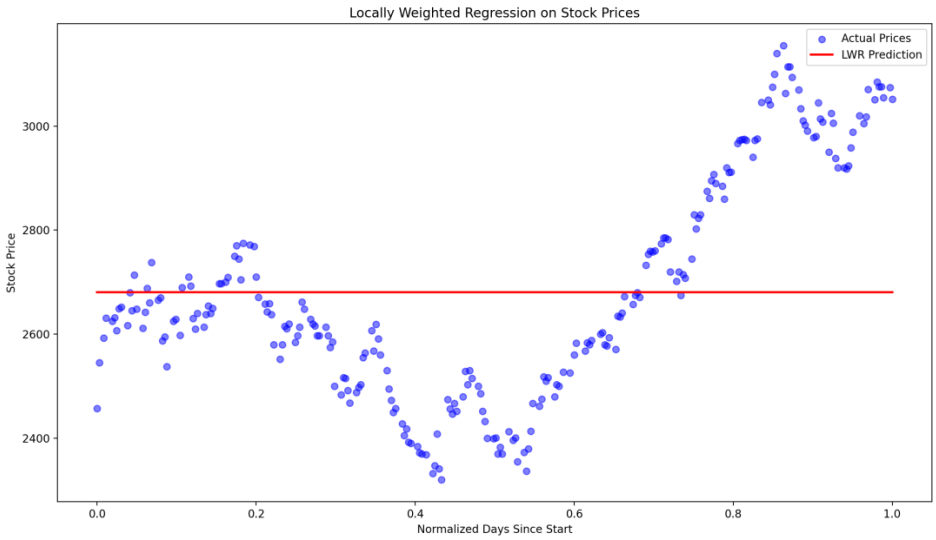


Without Libraries:

```
import csv
import math
import matplotlib.pyplot as plt
from datetime import datetime

file_path = "TITAN.NS.csv"

X, y = [], []
with open(file_path, 'r') as file:
    reader = csv.reader(file)
    next(reader)
    first_date = None
    for row in reader:
        try:
            date = datetime.strptime(row[0], "%Y-%m-%d")
            if first_date is None:
                first_date = date
            days_since_start = (date - first_date).days

            X.append(float(days_since_start))
            y.append(float(row[1]))
             except ValueError as e:
            print(f"Skipping row {row} due to error: {e}")

X_min, X_max = min(X), max(X)
X = [(x - X_min) / (X_max - X_min) for x in X]

def weighted_matrix(X_train, x_query, tau):
    """Computes weight matrix W for a given x_query."""
    return [math.exp(-((xi - x_query) ** 2) / (2 * tau ** 2)) for xi in X_train]

def locally_weighted_regression(x_query, X_train, y_train, tau):
    """Compute LWR prediction for a single query point x_query without NumPy."""
    weights = weighted_matrix(X_train, x_query, tau)
    W = [[weights[i] if i == j else 0 for j in range(len(weights))] for i in range(len(weights))]

    X_b = [[1, xi] for xi in X_train]


    XT_W = [[sum(X_b[i][j] * W[i][i] for i in range(len(X_train))) for j in range(2)] for k in
range(2)]
```

```
    XT_W_Y = [sum(X_b[i][j] * W[i][i] * y_train[i] for i in range(len(X_train))) for j in range(2)]


    det = XT_W[0][0] * XT_W[1][1] - XT_W[0][1] * XT_W[1][0]
    if det == 0:
        return sum(y_train) / len(y_train)
        XT_W_inv = [[XT_W[1][1] / det, -XT_W[0][1] / det], [-XT_W[1][0] / det, XT_W[0][0] / det]]

    theta = [sum(XT_W_inv[i][j] * XT_W_Y[j] for j in range(2)) for i in range(2)]

    return theta[0] + theta[1] * x_query

tau = 0.1
y_pred = [locally_weighted_regression(x, X, y, tau) for x in X]

plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label="Actual Prices", alpha=0.5)
plt.plot(X, y_pred, color='red', label="LWR Prediction", linewidth=2)
plt.xlabel("Normalized Days Since Start")
plt.ylabel("Stock Price")
plt.title("Locally Weighted Regression on Stock Prices")
plt.legend()
plt.show()
```

Output:



Real Time Applications:

The Locally Weighted Regression (LWR) algorithm is a non-parametric regression technique that is effective for real-time applications requiring adaptability to non-linear and dynamic data. Here are some key real-time applications:

1. **Autonomous Vehicles and Robotics**
   - Path planning and obstacle avoidance in self-driving cars.
   - Used in robotic arm movement prediction and control.

2. **Stock Market and Financial Predictions**
   - Predicting stock price trends by adapting to market fluctuations.
   - Used in risk assessment and portfolio management.

3. **Healthcare and Medical Diagnosis**
   - Personalized treatment recommendations based on patient data.
   - Disease progression modelling (e.g., diabetes and cancer growth patterns).