Lab 10

Naive Bayes:

It is a classification technique based on Bayes' Theorem with an independence assumption among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Bayes theorem provides a way of computing posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

$$\underbrace{P(c \mid x)}_{\text{Posterior Probability}} = \frac{\overbrace{P(x \mid c)}^{\text{Likelihood}}\,\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Above,

- P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
- P(c) is the prior probability of class.
- P(x|c) is the likelihood which is the probability of the predictor given class.
- P(x) is the prior probability of the predictor.

How Do Naive Bayes Algorithms Work?

Let's understand it using an example. Below we have a training data set of weather and corresponding target variable 'Play' (suggesting possibilities of playing). Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.

- **Convert the data set into a frequency table**
  In this first step data set is converted into a frequency table
- **Create Likelihood table by finding the probabilities**

Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

- **Use Naive Bayesian equation to calculate the posterior probability**

  Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of the prediction.

Problem: Players will play if the weather is sunny. Is this statement correct?

- We can solve it using the above-discussed method of posterior probability.
- P(Yes | Sunny) = P(Sunny | Yes) * P(Yes) / P (Sunny)
- Here P(Sunny | Yes) * P(Yes) is in the numerator, and P (Sunny) is in the denominator.
- Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64
- Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

Question 1: Naïve bayse code from Scratch:

- These lines import necessary libraries. numpy is used for numerical operations, train_test_split is from scikit-learn and is used for splitting the dataset into training and testing sets, and accuracy_score is used to calculate the accuracy of the classifier.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Self-created dataset
data = np.array([
    [1, 'Sunny', 'Hot', 'High', 'No'],
    [2, 'Sunny', 'Hot', 'High', 'No'],
    [3, 'Overcast', 'Hot', 'High', 'Yes'],
    [4, 'Rainy', 'Mild', 'High', 'Yes'],
    [5, 'Rainy', 'Cool', 'Normal', 'Yes'],
    [6, 'Rainy', 'Cool', 'Normal', 'No'],
    [7, 'Overcast', 'Cool', 'Normal', 'Yes'],
    [8, 'Sunny', 'Mild', 'High', 'No'],
    [9, 'Sunny', 'Cool', 'Normal', 'Yes'],
    [10, 'Rainy', 'Mild', 'Normal', 'Yes']
])
```

- This separates the dataset into features (X) and labels (y). Features are columns 1 to 3 (indexing starts from 0), and labels are in column .
- **train_test_split:** This is a function from the scikit-learn library that splits a dataset into random train and test subsets.

```python
# Split data into features and labels
X = data[:, 1:4]  # Features
y = data[:, 4]    # Labels

# 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=0)
```

- Function to calculates the probabilities of different labels in the dataset.

```python
# Helper function to calculate probabilities
def calculate_probabilities(data, label_column):
    probabilities = {}
    labels, counts = np.unique(data[:, label_column], return_counts=True)
    total_samples = len(data)

    for label, count in zip(labels, counts):
        probabilities[label] = count / total_samples

    return probabilities
```

- Function to trains the Naive Bayes classifier by calculating probabilities for each label and feature value.

```python
# Helper function to train Naive Bayes classifier
def train_naive_bayes(X, y):
    num_features = X.shape[1]
    unique_labels = np.unique(y)
    probabilities = {}

    for label in unique_labels:
        label_indices = np.where(y == label)[0]
        label_data = X[label_indices]

        probabilities[label] = []
        for i in range(num_features):
            feature_values, counts = np.unique(label_data[:, i], return_counts=True)
            feature_probabilities = dict(zip(feature_values, counts / len(label_data)))
            probabilities[label].append(feature_probabilities)

    return probabilities
```

- Function predicts the label for a given instance using the trained Naive Bayes classifier.

```python
# function to predict using Naive Bayes classifier
def predict_naive_bayes(instance, probabilities):
    predicted_label = None
    max_probability = -1

    for label, label_probabilities in probabilities.items():
        instance_probability = 1.0
        for i, value in enumerate(instance):
            if value in label_probabilities[i]:
                instance_probability *= label_probabilities[i][value]
            else:
                instance_probability = 0  # If the value is unseen in training data, probability is 0

        if instance_probability > max_probability:
            max_probability = instance_probability
            predicted_label = label

    return predicted_label
```

- Prediction based on test instance

```python
# Train the Naive Bayes classifier
probabilities = train_naive_bayes(X, y)

# Example instance for prediction
test_instance = ['Sunny', 'Hot', 'High']

# Make a prediction
prediction = predict_naive_bayes(test_instance, probabilities)

print(f'The predicted label for the instance {test_instance} is: {prediction}')
```

In case your dataset is a CVS file:

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

# Read data from CSV file
filename = 'play_data.csv'
df = pd.read_csv(filename)

# Display the loaded data
print("Loaded Data:")
print(df)

# Split data into features and labels
X = df.iloc[:, :-1].values  # Features (all columns except the last one)
y = df.iloc[:, -1].values   # Labels (last column)

# ... code…(rest of the Naive Bayes code remains the same)
```

Question 2: Naïve bayse code using built-in library:

Sklearn

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python.

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

# Reading CSV files
df = pd.read_csv('E:\\playsheet_dataset.csv')
print(df)

# Encoding the strings to Numericals
Numerics = LabelEncoder()

inputs = df.drop('Play', axis='columns')
target = df['Play']
print(target)

# Creating the new dataframe
inputs['outlook_n'] = Numerics.fit_transform(inputs['Outlook'])
inputs['Temp_n'] = Numerics.fit_transform(inputs['Temp'])
inputs['Humidity_n'] = Numerics.fit_transform(inputs['Humidity'])
inputs['windy_n'] = Numerics.fit_transform(inputs['Windy'])
print(inputs)

# Dropping the string values
inputs_n = inputs.drop(['Outlook', 'Temp', 'Humidity', 'Windy'], axis='columns')
print(inputs_n)

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(inputs_n, target, test_size=0.2, random_state=42)

# Applying the Gaussian Naive Bayes to the training set
Classifier = GaussianNB()
Classifier.fit(X_train, y_train)

# Checking the accuracy on the training set
accuracy_train = Classifier.score(X_train, y_train)
print(f'Training Set Accuracy: {accuracy_train * 100:.2f}%')

# Checking the accuracy on the testing set
accuracy_test = Classifier.score(X_test, y_test)
print(f'Testing Set Accuracy: {accuracy_test * 100:.2f}%')

# Making a prediction
prediction = Classifier.predict([[0, 0, 0, 1]])
print(f'Prediction: {prediction}')
```

K-means:

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. The objective of K-means is simple: group similar data points together and discover underlying patterns. To achieve this objective, K-means looks for a fixed number (k) of clusters in a dataset. A cluster refers to a collection of data points aggregated together because of certain similarities.

Question 3: K-means from scratch:

```python
import numpy as np
import matplotlib.pyplot as plt

def kmeans(X, k, max_iterations=100):
    # Randomly initialize centroids
    centroids = X[np.random.choice(X.shape[0], k, replace=False)]

    for _ in range(max_iterations):
        # Assign each data point to the nearest centroid
        distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)
        labels = np.argmin(distances, axis=1)

        # Update centroids based on the mean of data points in each cluster
        for i in range(k):
            centroids[i] = np.mean(X[labels == i], axis=0)

    return labels, centroids

# Create a self-created dataset
np.random.seed(42)
data = np.concatenate([np.random.normal(loc=5, scale=1, size=(50, 2)),
                       np.random.normal(loc=10, scale=1, size=(50, 2))])

# Run K-means clustering
k = 2
labels, centroids = kmeans(data, k)

# Plot the data points and centroids
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis', marker='o')
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', label='Centroids')
plt.title('K-means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

In case your dataset is a CVS file:

```
//... kmean code...

# Load data from CSV file
df = pd.read_csv('data.csv')
# Extract features from the DataFrame
X = df[['Feature1', 'Feature2']].values

//... plot graph...
```

Question 4: Kmeans code using built-in library:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Generate random data
np.random.seed(42)
X = np.concatenate([np.random.normal(loc=5, scale=1, size=(50, 2)),
                    np.random.normal(loc=10, scale=1, size=(50, 2))])

# Run K-means clustering using scikit-learn
k = 2
kmeans = KMeans(n_clusters=k, random_state=42)
labels = kmeans.fit_predict(X)
centroids = kmeans.cluster_centers_

# Plot the data points and centroids
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o')
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', label='Centroids')
plt.title('K-means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```