Experiment 4

Build a naïve Bayes classifier using supervised dataset.

Aim: Build a naïve Bayes classifier using supervised dataset.

Theory:

Naïve Bayes Classifier is a probabilistic machine learning model used for classification tasks. It is based on Bayes' theorem, which describes the probability of an event based on prior knowledge of conditions related to the event.

1. Bayes' Theorem:

Bayes' theorem provides a mathematical framework to update the probability estimate for a hypothesis as more evidence or information becomes available. It is expressed as:

$P(C|X) = P(X|C) * P(C) / P(X)$

Where:

- P(C|X) is the posterior probability: the probability of class C given the feature vector X.
- P(X|C) is the likelihood: the probability of feature vector X given class C.
- P(C) is the prior probability of class C, i.e., the frequency of class C in the dataset.
- P(X) is the evidence or normalizing constant, which is the overall probability of feature vector X.

2. Naïve Assumption:

The "naïve" part of Naïve Bayes assumes that all the features (attributes) in the dataset are conditionally independent of each other given the class label. This simplification reduces the complexity of calculating P(X|C) to the product of individual feature probabilities:

$$P(X|C) = P(x_1, x_2, ..., x_n|C) = \prod_{i=1}^{n} P(x_i|C)$$

This assumption often doesn't hold in real-world data, but Naïve Bayes still performs surprisingly well in many cases.

3. Working of Naïve Bayes Classifier:

- During the training phase, the classifier calculates the prior probabilities P(C) for each class based on the frequency of each class in the training dataset.
- It also calculates the likelihood probabilities P(xi|C) for each feature value conditioned on each class.

- When a new data instance X comes in, the classifier computes the posterior probability P(C|X) for each class using Bayes' theorem.
- The predicted class is the one with the highest posterior probability.

4. Types of Naïve Bayes Classifiers:

- Gaussian Naïve Bayes: Assumes features follow a normal (Gaussian) distribution; used with continuous data.
- Multinomial Naïve Bayes: Used for discrete count data like word counts in text classification.
- Bernoulli Naïve Bayes: Used for binary/boolean features (e.g., word presence/absence in documents).

Conclusion:

The Naïve Bayes Classifier is a simple yet powerful probabilistic model that applies Bayes' theorem with the assumption of feature independence. Despite this assumption, it delivers effective performance, especially in text classification and spam detection. Its efficiency and ease of implementation make it a popular choice for classification tasks. The Gaussian, Multinomial, and Bernoulli variants allow it to handle different data types effectively.

Viva Questions:

1. What is the main assumption behind Naïve Bayes?
2. How does Naïve Bayes apply Bayes' theorem in classification?
3. Why is it called "naïve"?
4. What are the types of Naïve Bayes classifiers?
5. What are the advantages and disadvantages of Naïve Bayes?

Program:

```
# Import necessary libraries

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.metrics import accuracy_score, classification_report


# Load dataset into pandas DataFrame
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong'],
    'Play': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}


df = pd.DataFrame(data)


# Encode categorical variables to numeric
label_encoders = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le


# Separate features and target variable
X = df.drop('Play', axis=1)
y = df['Play']
```

```
# Split data into training and testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create and train the Naïve Bayes classifier
model = GaussianNB()
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy*100:.2f}%")

# Detailed classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Example prediction: Predict whether to play given new weather data
# New data: Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong
# Encode new data using label encoders
new_data = {
    'Outlook': 'Sunny',
    'Temperature': 'Cool',
    'Humidity': 'High',
    'Wind': 'Strong'
```

```
}

# Convert new data to numeric
new_data_encoded = [label_encoders[col].transform([new_data[col]])[0] for col in ['Outlook', 'Temperature', 'Humidity', 'Wind']]

# Predict class
prediction_encoded = model.predict([new_data_encoded])[0]
prediction = label_encoders['Play'].inverse_transform([prediction_encoded])[0]

print(f"\nPrediction for new data {new_data}: Play = {prediction}")
```