Practical Assignment No. 2	
Title:	Naive Bayes model
Problem Statement:	Use Naive Bayes model, to classify a dataset from UCI repository. Present the Confusion matrix for the classifier. For measuring performance use at least five metrics such as accuracy, precision, recall, F-measure etc.
Objective:	To use Scikit Learn for implementing data science operations.
Outcome:	CO606.1: Make use of Scikit Learn for implementing data science operations
Software or Hardware Requirements:	Anaconda/Java/GCC
Theory:	**Classification:** Classification is a supervised machine learning task that involves assigning a label or category to a given input based on its features. The model is trained using a labeled dataset and then used to predict the class of new or unseen data. Naïve Bayes is one of the most popular and efficient classification algorithms, especially for text classification, spam detection, and medical diagnosis. **Naïve Bayes Classifier:** The Naïve Bayes classifier is a probabilistic model based on Bayes' Theorem, assuming that all features are independent given the class label. Despite this "naïve" assumption, it performs remarkably well in many real-world scenarios. **Bayes' Theorem:** $$P(C\|X) = \frac{P(X\|C) \cdot P(C)}{P(X)}$$ **Where:** • P(C | X) = Posterior probability (probability of class *C* given features *X*) • P(X | C) = Likelihood (probability of features *X* given class *C*) • P(C) = Prior probability of class *C* • P(X) = Probability of features *X* **The classifier predicts the class with the highest posterior probability:** $$\hat{C} = \arg\max_{C} P(C\|X)$$

Types of Naïve Bayes Classifiers

Depending on the nature and distribution of data, there are different variants of Naïve Bayes classifiers, each optimized for specific data types.

1) Gaussian Naïve Bayes

- **Used for:** Continuous numerical features.
- **Assumption:** The continuous values associated with each class are distributed according to a **Gaussian (Normal) distribution**.
- **Example:** Iris dataset, where features like sepal length and petal width are continuous.

Formula for probability density function:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Where:

- μ_y = mean of feature x_i for class y
- σ_y = standard deviation of feature x_i for class y

Example in Python:

```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
```

Applications:

- Medical diagnosis
- Sensor data classification
- Continuous-valued datasets

2) Multinomial Naïve Bayes

- **Used for:** Discrete features, especially count data.
- **Assumption:** Features represent counts or frequencies (e.g., number of times a word appears).
- Commonly used for **text classification** and **document categorization**.

Formula:

$$P(x_i|y) = \frac{(N_{y,i} + \alpha)}{(\sum_i N_{y,i} + \alpha n)}$$

Where:

- $N_{y,i}$: Number of times feature i appears in class y
- α: Smoothing parameter (Laplace smoothing)
- n: Number of features

Example in Python:

```
from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB()
```

Applications:

- Spam email detection
- Sentiment analysis
- News article classification

3) Bernoulli Naïve Bayes

- **Used for:** Binary or Boolean features.
- **Assumption:** Each feature is binary (e.g., present/absent, yes/no, 0/1).

- Suitable when features indicate the presence or absence of a characteristic.

Formula:

$$P(x_i|y) = P_i^x(1 - P_i)^{(1-x)}$$

Where:

- P_i: Probability that feature i is present in class y
- x: Binary value (0 or 1)

Example in Python:

```
from sklearn.naive_bayes import BernoulliNB

model = BernoulliNB()
```

Applications:

- Document classification (presence/absence of words)
- Recommendation systems
- Binary attribute datasets

4) Complement Naïve Bayes

- **Used for:** Imbalanced datasets and text classification with skewed class distributions.
- It is a variant of the Multinomial Naïve Bayes that **uses statistics from all classes except the target class**, improving performance on imbalanced data.

Example in Python:

```
from sklearn.naive_bayes import ComplementNB
```

```
model = ComplementNB()
```

Applications:

- Text classification with unequal class frequencies (e.g., spam vs. non-spam)

5) Categorical Naïve Bayes (for nominal features)

- **Used for:** Categorical features with discrete values (not counts or binary).
- Handles categorical variables directly without converting them to numeric values.

Example in Python:

```
from sklearn.naive_bayes import CategoricalNB

model = CategoricalNB()
```

Applications:

- Demographic classification (e.g., gender, region, occupation)
- Customer segmentation

Confusion Matrix:

A confusion matrix is a performance measurement tool used to evaluate the accuracy of a classification model. It compares actual vs. predicted classifications.

Actual \ Predicted	**Class 1**	**Class 2**
Class 1	**TP**	**FP**
Class 2	**FN**	**TN**

Where:

- TP (True Positive): Correctly predicted positive class
- TN (True Negative): Correctly predicted negative class
- FP (False Positive): Incorrectly predicted positive class
- FN (False Negative): Incorrectly predicted negative class

Performance Metrics:
To measure the performance of the classifier, the following metrics are used:

1. **Accuracy:**
   The ratio of correctly predicted observations to total observations.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2. Precision:

The ratio of correctly predicted positive observations to total predicted positives.

$$Precision = \frac{TP}{TP + FP}$$

3. Recall (Sensitivity):

The ratio of correctly predicted positive observations to all actual positives.

$$Recall = \frac{TP}{TP + FN}$$

4. **F1-Score:**

The harmonic mean of precision and recall.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

5. **Specificity:**

Measures the proportion of actual negatives correctly identified.

$$Specificity = \frac{TN}{TN + FP}$$

These metrics together provide a comprehensive evaluation of the model's predictive ability.

Steps Involved

1. **Import Libraries:** Load required Python libraries (`pandas`, `sklearn`, `numpy`, `matplotlib`).
2. **Load Dataset:** Import dataset from UCI repository or `sklearn.datasets`.
3. **Preprocess Data:** Handle missing values, encode categorical variables, and split dataset.
4. **Train the Model:** Fit the Naïve Bayes classifier on training data.
5. **Predict Output:** Use the trained model to predict class labels on test data.
6. **Evaluate Performance:** Generate confusion matrix and calculate performance metrics.

Advantages of Naïve Bayes

- Simple and fast to implement.
- Performs well with high-dimensional data.
- Requires less training data.
- Works efficiently for text classification and real-time predictions.

	Limitations ● Assumes feature independence, which may not hold true in real-world data. ● Poor performance when features are highly correlated. ● Continuous features need distributional assumptions (e.g., Gaussian).
Input/Datasets/Test Cases:	Dataset- from UCI Machine Learning Repository Name of the Dataset: Description of the Dataset: Dataset Characteristics: Subject Area: Associated Tasks: Feature Type: # Instances: # Features:
Results:	Execute code to use any two appropriate UCI dataset and any two types of NB. Take a print of this code with output for submission as part of results.
Analysis and conclusion:	Write your own analysis of output and conclusion(Minimum 2 statements of Analysis for two types of NBimplementation each, Minimum 1 Statement Conclusion)
References:	Reference /Links(min Any 2, include dataset ref.). Write references in IEEE format.