Assignment No. 4

Problem Statement: Understand and implement the Naïve Bayes classification algorithm.

Objective:

1. Understand the Theory – Learn the mathematical background of the Naïve Bayes algorithm, including Bayes' Theorem and conditional probabilities.
2. Implement Naïve Bayes – Apply the algorithm to a dataset and analyze its performance.
3. Evaluate Performance – Measure the accuracy, precision, recall, and F1-score of the classifier.

Prerequisite :

1. A Python environment with essential libraries like pandas, numpy, matplotlib, seaborn, and scikit-learn.
2. Basic knowledge of Python, statistics, and machine learning principles.
3. Statistics Knowledge – Understanding of probability, conditional probability, and Bayes' Theorem.
4. Machine Learning Principles – Familiarity with classification techniques and model evaluation metrics.

Theory :

Naïve Bayes is a classification algorithm based on **Bayes' Theorem**, which calculates the probability of a class given certain features. It is called "naïve" because it assumes that all features are **independent** of each other, which may not always be true in real-world datasets. Despite this simplification, it performs well in many applications.

1. Bayes' Theorem

The foundation of Naïve Bayes is **Bayes' Theorem**, which states:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Where:

- P(A|B) = Probability of event A occurring given that event B has occurred (posterior probability).
- P(B|A) = Probability of event B occurring given that event A has occurred (likelihood).
- P(A) = Prior probability of event A occurring.
- P(B) = Total probability of event B occurring.

2. Assumptions of Naïve Bayes

1. **Feature Independence** – Each feature contributes **independently** to the probability of a class label.
2. **Equal Importance of Features** – All features are given equal importance in predicting the output.
3. **Conditional Independence** – Given the class label, the features do not depend on each other.

3. Types of Naïve Bayes Classifiers

There are three main types of Naïve Bayes classifiers:

1. **Gaussian Naïve Bayes (GNB)** – Assumes that features follow a normal (Gaussian) distribution. Used for continuous numerical data.
2. **Multinomial Naïve Bayes (MNB)** – Suitable for classification with **discrete count data**, commonly used in text classification.
3. **Bernoulli Naïve Bayes (BNB)** – Works with **binary/boolean features**, mainly used in spam detection or sentiment analysis.

4. Steps in Naïve Bayes Classification

1. **Data Preprocessing** – Load the dataset, clean missing values, and prepare feature-target variables.
2. **Calculate Prior Probabilities** – Compute P(A) for each class in the dataset.
3. **Compute Likelihood** – Calculate P(B|A) based on the type of Naïve Bayes model used (Gaussian, Multinomial, or Bernoulli).
4. **Apply Bayes' Theorem** – Compute the **posterior probability** P(A|B) for each class and assign the highest probability class to the data point.
5. **Evaluate Performance** – Use accuracy, precision, recall, and F1-score to assess the classifier's performance.

5. Advantages of Naïve Bayes

1. **Fast and Efficient** – Works well with large datasets and high-dimensional data.
2. **Handles Missing Data** – Can work well even if some features have missing values.
3. **Performs Well with Small Data** – Requires less training data compared to other classifiers.
4. **Interpretable and Simple** – Easy to implement and understand.
5. **Works Well in Text Classification** – Commonly used in spam detection, sentiment analysis, and document classification.

6. Disadvantages of Naïve Bayes

1. **Strong Feature Independence Assumption** – Often unrealistic in real-world data.
2. **Zero Probability Issue** – If a category is missing in the training dataset, it assigns zero probability (solved using **Laplace smoothing**).
3. **Poor Performance on Highly Correlated Features** – If features are dependent, it may give incorrect classifications.

4. **Limited for Complex Datasets** – Not ideal for datasets where relationships between features are important.

7. Applications of Naïve Bayes

1. **Spam Detection** – Classifies emails as spam or not spam based on word frequency.
2. **Sentiment Analysis** – Determines the sentiment of text (positive, negative, neutral).
3. **Medical Diagnosis** – Predicts diseases based on patient symptoms.
4. **Credit Scoring** – Assesses credit risk in finance.
5. **Recommendation Systems** – Suggests products or content based on user behavior.

1. Code & Output

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('C:/Users/dnyan/ML Assignments/Dataset/adult.csv', header=None, sep=',\s')
```

```
df.shape
```

```
(32561, 15)
```

```
df.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

```
#Renaming the columns
col_names = ['age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_status', 'occupation', 'relationship',
             'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'native_country', 'income']

df.columns = col_names

df.columns
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
       'marital_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'income'],
      dtype='object')
```

```
df.head()
```

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week	native_country
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba

```python
# find numerical variables

numerical = [var for var in df.columns if df[var].dtype!='O']

print('There are {} numerical variables\n'.format(len(numerical)))

print('The numerical variables are :', numerical)
```

```
There are 6 numerical variables

The numerical variables are : ['age', 'fnlwgt', 'education_num', 'capital_gain', 'capital_loss', 'hours_per_week']
```

```python
# view the numerical variables

df[numerical].head()
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week
0	39	77516	13	2174	0	40
1	50	83311	13	0	0	13
2	38	215646	9	0	0	40
3	53	234721	7	0	0	40
4	28	338409	13	0	0	40

```python
X = df.drop(['income'], axis=1)

y = df['income']
```

```python
# split X and y into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

```python
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

```
((22792, 14), (9769, 14))
```

```python
# check data types in X_train

X_train.dtypes
```

```
age                int64
workclass         object
fnlwgt             int64
education         object
education_num      int64
marital_status    object
occupation        object
relationship      object
race              object
sex               object
capital_gain       int64
capital_loss       int64
hours_per_week     int64
native_country    object
dtype: object
```

```python
# display categorical variables
categorical = [col for col in X_train.columns if X_train[col].dtypes == 'O']
categorical
```

```
['workclass',
 'education',
 'marital_status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native_country']
```

```python
# display numerical variables
numerical = [col for col in X_train.columns if X_train[col].dtypes != 'O']
numerical
```

```
['age',
 'fnlwgt',
 'education_num',
 'capital_gain',
 'capital_loss',
 'hours_per_week']
```

```python
# print percentage of missing values in the categorical variables in training set
X_train[categorical].isnull().mean()
```

```
workclass         0.055985
education         0.000000
marital_status    0.000000
occupation        0.056072
relationship      0.000000
race              0.000000
sex               0.000000
native_country    0.018164
dtype: float64
```

```python
# print categorical variables with missing data

for col in categorical:
    if X_train[col].isnull().mean()>0:
        print(col, (X_train[col].isnull().mean()))
```

```
workclass 0.055984555984555984
occupation 0.05607230607230607
native_country 0.018164268164268166
```

```python
# impute missing categorical variables with most frequent value

for df2 in [X_train, X_test]:
    df2['workclass'].fillna(X_train['workclass'].mode()[0], inplace=True)
    df2['occupation'].fillna(X_train['occupation'].mode()[0], inplace=True)
    df2['native_country'].fillna(X_train['native_country'].mode()[0], inplace=True)
```

```python
# check missing values in categorical variables in X_train

X_train[categorical].isnull().sum()
```

```
workclass         0
education         0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
native_country    0
dtype: int64
```

```python
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Define categorical and numerical features
categorical_features = ['workclass', 'education', 'marital_status', 'occupation',
                        'relationship', 'race', 'sex', 'native_country']
numerical_features = ['age', 'fnlwgt', 'education_num', 'capital_gain', 'capital_loss', 'hours_per_week']

# Create preprocessing pipeline
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numerical_features),  # Scale numerical features
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)  # OneHot encode categorical features
])

# Fit and transform training & test data
X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)
```

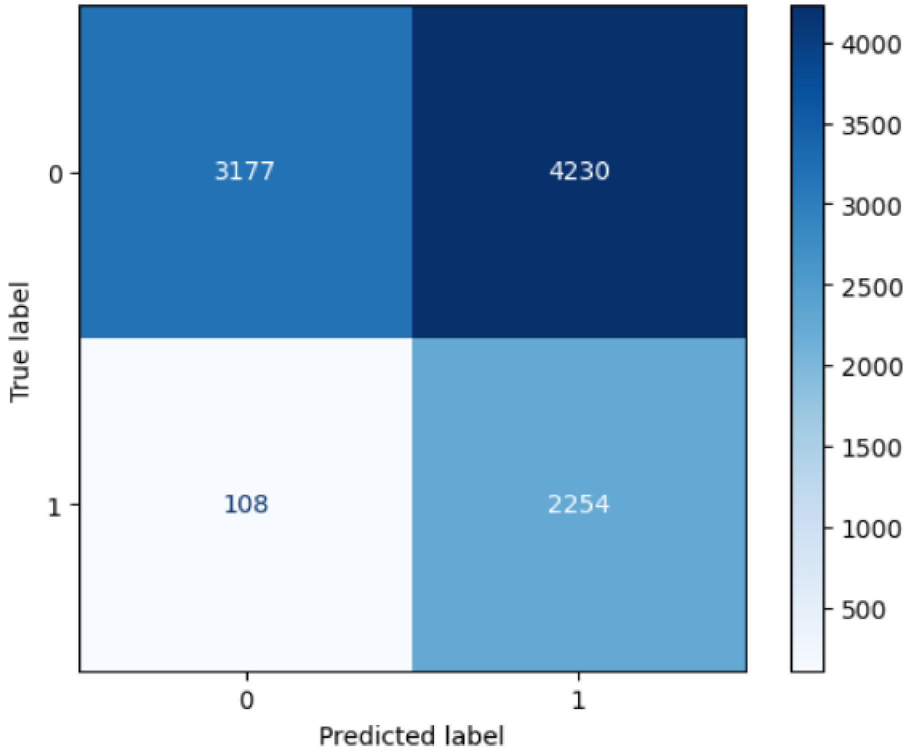
```python
# Modify ColumnTransformer to return a dense array
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numerical_features),  # Scale numerical features
    ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False), categorical_features)  # OneHot encode categorical features
])
```

```python
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Convert the sparse matrix to dense array
X_train_dense = X_train.toarray()
X_test_dense = X_test.toarray()

# Train Naïve Bayes Model
nb_model = GaussianNB()
nb_model.fit(X_train_dense, y_train)

y_pred_nb = nb_model.predict(X_test_dense)

# Evaluate Naïve Bayes
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print(f"Naïve Bayes Accuracy: {accuracy_nb:.4f}")
```

```
Naïve Bayes Accuracy: 0.5559
```

```python
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
# Convert the sparse matrix to a dense array for Naive Bayes model
X_train_dense = X_train.toarray()
X_test_dense = X_test.toarray()

# Initialize the Naive Bayes model
gnb = GaussianNB()

# Train the Naive Bayes model on the training data
gnb.fit(X_train_dense, y_train)

# Make predictions on the test and training sets
y_pred = gnb.predict(X_test_dense)
y_pred_train = gnb.predict(X_train_dense)

# Model accuracy on test set
print('Model accuracy score: {0:0.4f}'.format(accuracy_score(y_test, y_pred)))

# Model accuracy on training set
print('Training-set accuracy score: {0:0.4f}'.format(accuracy_score(y_train, y_pred_train)))

# Model score on training set
print('Training set score: {:.4f}'.format(gnb.score(X_train_dense, y_train)))

# Model score on test set
print('Test set score: {:.4f}'.format(gnb.score(X_test_dense, y_test)))
# Check class distribution in test set
print("\nClass distribution in test set:")
print(y_test.value_counts())

# Calculate null accuracy score (accuracy by predicting the majority class only)
null_accuracy = max(y_test.value_counts()) / len(y_test)
print('Null accuracy score: {0:0.4f}'.format(null_accuracy))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
```

```
Model accuracy score: 0.5559
Training-set accuracy score: 0.5552
Training set score: 0.5552
Test set score: 0.5559

Class distribution in test set:
income
<=50K    7407
>50K     2362
Name: count, dtype: int64
Null accuracy score: 0.7582
```




```python
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_nb))
```

```
              precision    recall  f1-score   support

       <=50K       0.97      0.43      0.59      7407
        >50K       0.35      0.95      0.51      2362

    accuracy                           0.56      9769
   macro avg       0.66      0.69      0.55      9769
weighted avg       0.82      0.56      0.57      9769
```

```python
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
```

```python
# print classification accuracy

classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)
print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
```

```
Classification accuracy : 0.5559
```

```python
# print classification error

classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('Classification error : {0:0.4f}'.format(classification_error))
```

```
Classification error : 0.4441
```

```
# print precision score

precision = TP / float(TP + FP)
print('Precision : {0:0.4f}'.format(precision))
```

```
Precision : 0.4289
```

```
recall = TP / float(TP + FN)
print('Recall or Sensitivity : {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity : 0.9671
```

```
true_positive_rate = TP / float(TP + FN)
print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))
```

```
True Positive Rate : 0.9671
```

```
false_positive_rate = FP / float(FP + TN)
print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))
```

```
False Positive Rate : 0.6524
```

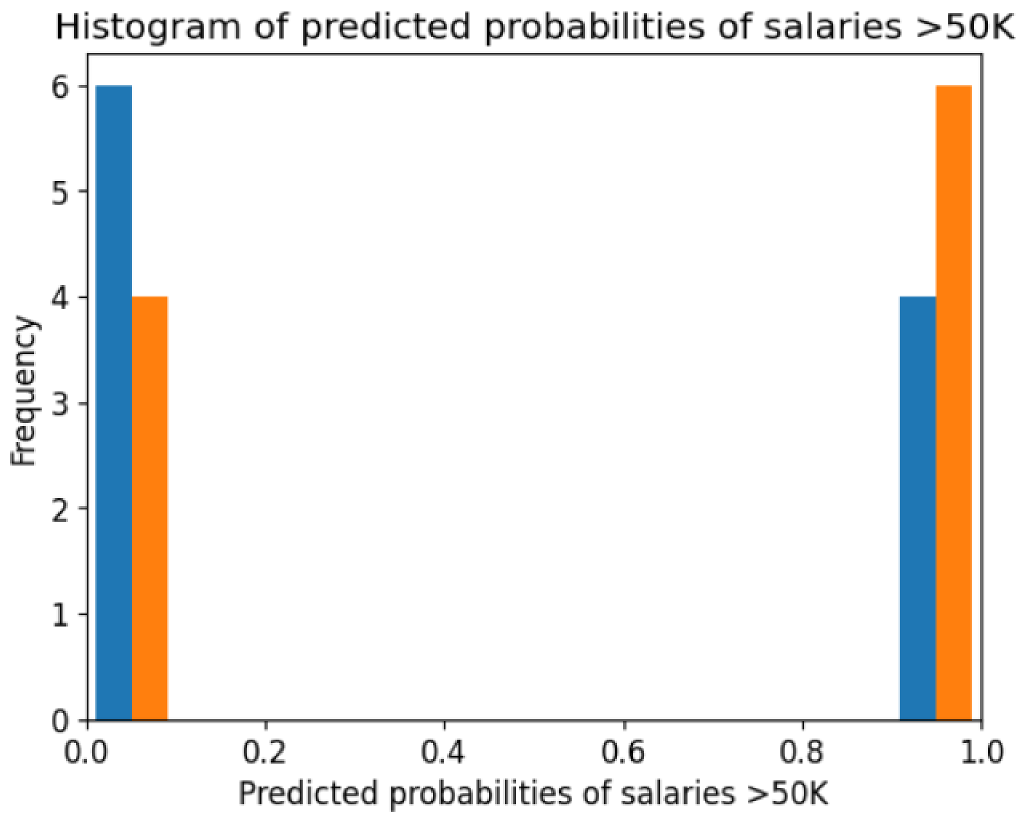
```
specificity = TN / (TN + FP)
print('Specificity : {0:0.4f}'.format(specificity))
```

```
Specificity : 0.3476
```

```
# Convert sparse matrix to dense format
X_test_dense = X_test.toarray()

# Get the first 10 predicted probabilities
y_pred_prob = gnb.predict_proba(X_test_dense)[0:10]
print(y_pred_prob)
```

```
[[1.08049317e-007 9.99999892e-001]
 [8.38320515e-017 1.00000000e+000]
 [4.67108205e-024 1.00000000e+000]
 [1.00000000e+000 3.95523741e-108]
 [9.90996122e-001 9.00387825e-003]
 [2.56885513e-002 9.74311449e-001]
 [1.00000000e+000 6.46567254e-015]
 [5.01665964e-013 1.00000000e+000]
 [8.82209566e-016 1.00000000e+000]
 [1.00000000e+000 6.02960314e-012]]
```

```
# plot histogram of predicted probabilities

# adjust the font size
plt.rcParams['font.size'] = 12

# plot histogram with 10 bins
plt.hist(y_pred_prob, bins = 10)

# set the title of predicted probabilities
plt.title('Histogram of predicted probabilities of salaries >50K')

# set the x-axis limit
plt.xlim(0,1)

# set the title
plt.xlabel('Predicted probabilities of salaries >50K')
plt.ylabel('Frequency')
```



Github: https://github.com/dnyaneshwardhere/ML

Conclusion:

Gaussian Naïve Bayes is an effective and simple classification algorithm for numerical datasets. Despite its **assumption of feature independence**, it performs well in many real-world applications, especially when the data is normally distributed. While it has limitations, such as sensitivity to non-Gaussian distributions and the independence assumption, its efficiency, simplicity, and effectiveness make it a valuable tool in machine learning.