Assignment No. 4

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

Objective:

1. Grasp the Concept – Gain a solid understanding of the mathematical foundation behind the Naïve Bayes algorithm, focusing on Bayes' Theorem and conditional probability principles.
2. Apply Naïve Bayes – Implement the algorithm on a dataset and assess its effectiveness.
3. Assess Performance – Evaluate the classifier's accuracy, precision, recall, and F1-score to determine its efficiency.

Prerequisite :

1. Python Environment – Set up a Python workspace with key libraries such as `pandas`, `numpy`, `matplotlib`, `seaborn`, and `scikit-learn`.
2. Foundational Knowledge – Have a basic grasp of Python programming, statistics, and core machine learning concepts.
3. Statistical Understanding – Comprehend probability, conditional probability, and Bayes' Theorem.
4. Machine Learning Basics – Be familiar with classification methods and model evaluation metrics.

Theory :

Naïve Bayes is a probabilistic classification algorithm based on **Bayes' Theorem**, used to determine the likelihood of a class given specific features. The term *"naïve"* comes from the assumption that all features are **independent**, which may not always hold true in real-world datasets. Despite this simplification, it remains highly effective for various applications.

Naïve Bayes is built upon **Bayes' Theorem**

Where:

1. P(A∣B)P(A∣B) = Probability of event A occurring given that event B has occurred (**posterior probability**).
2. P(B∣A)P(B∣A) = Probability of event B occurring given that event A has occurred (**likelihood**).
3. P(A)P(A) = Prior probability of event A occurring.
4. P(B)P(B) = Total probability of event B occurring.

Feature Independence – Each feature is assumed to contribute independently to the probability of a class label.

Equal Feature Importance – All features are considered equally significant in making predictions.

Conditional Independence – Given the class label, features are assumed not to be dependent on each other.

5. Steps in Naïve Bayes Classification

1. Data Preprocessing – Load the dataset, clean missing values, and prepare the feature-target variables.
2. Compute Prior Probabilities – Calculate P(A)P(A) for each class.
3. Calculate Likelihood – Determine P(B∣A)P(B∣A) based on the selected Naïve Bayes model (Gaussian, Multinomial, or Bernoulli).
4. Apply Bayes' Theorem – Compute the posterior probability P(A∣B)P(A∣B) for each class and classify based on the highest probability.
5. Evaluate Performance – Assess model accuracy using metrics like precision, recall, F1-score, and overall accuracy.

Advantages of Naïve Bayes

- Efficient and Fast – Works well with large and high-dimensional datasets.
- Handles Missing Data – Can still perform well even if some features are missing.
- Effective on Small Datasets – Requires less training data than many other classifiers.
- Simple and Interpretable – Easy to understand and implement.
- Excels in Text Classification – Widely used for spam filtering, sentiment analysis, and document categorization.

Limitations of Naïve Bayes

- Unrealistic Feature Independence Assumption – Often, real-world features are correlated, which can affect accuracy.
- Zero Probability Problem – If a feature value is missing in the training set, the model assigns zero probability to it (solved using Laplace Smoothing).
- Poor Performance with Highly Correlated Features – When features are dependent, the model may produce incorrect classifications.

- Limited for Complex Datasets – Not ideal for problems requiring feature interactions.

5. Code & Output

```
[1]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     %matplotlib inline
```

```
[3]: df = pd.read_csv('/Users/pranavashokdivekar/this_mac/Machine Learning/adult.csv', header=None, sep=',\s')
```

```
[5]: df.head()
```

[5]:

	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

```
[6]: #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
```

```
[6]: 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')
```

```
[9]: categorical = [var for var in df.columns if df[var].dtype=='O']

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

     print('The categorical variables are :\n\n', categorical)
```

```
There are 9 categorical variables

The categorical variables are :

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

```
: df[categorical].isnull().sum()
```

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

```
: for var in categorical:

      print(df[var].value_counts())
```

```
workclass
Private             22696
Self-emp-not-inc     2541
Local-gov            2093
?                    1836
State-gov            1298
Self-emp-inc         1116
Federal-gov           960
Without-pay            14
Never-worked            7
Name: count, dtype: int64
education
HS-grad         10501
Some-college     7291
Bachelors        5355
Masters          1723
Assoc-voc        1382
11th             1175
Assoc-acdm       1067
10th              933
7th-8th           646
Prof-school       576
9th               514
12th              433
```

[13]:
```
df.workclass.unique()
```

[13]:
```
array(['State-gov', 'Self-emp-not-inc', 'Private', 'Federal-gov',
       'Local-gov', '?', 'Self-emp-inc', 'Without-pay', 'Never-worked'],
      dtype=object)
```

[14]:
```
df.workclass.value_counts()
```

[14]:
```
workclass
Private             22696
Self-emp-not-inc     2541
Local-gov            2093
?                    1836
State-gov            1298
Self-emp-inc         1116
Federal-gov           960
Without-pay            14
Never-worked            7
Name: count, dtype: int64
```

[16]:
```
df['workclass'].replace('?', np.nan, inplace=True)
```

[17]:
```
df.workclass.value_counts()
```

[17]:
```
workclass
Private             22696
Self-emp-not-inc     2541
Local-gov            2093
State-gov            1298
Self-emp-inc         1116
Federal-gov           960
Without-pay            14
Never-worked            7
Name: count, dtype: int64
```

[18]:
```
# check labels in occupation variable

df.occupation.unique()
```

[18]:
```
array(['Adm-clerical', 'Exec-managerial', 'Handlers-cleaners',
       'Prof-specialty', 'Other-service', 'Sales', 'Craft-repair',
       'Transport-moving', 'Farming-fishing', 'Machine-op-inspct',
       'Tech-support', '?', 'Protective-serv', 'Armed-Forces',
       'Priv-house-serv'], dtype=object)
```

[21]:
```
# replace '?' values in occupation variable with `NaN`
df['occupation'].replace('?', np.nan, inplace=True)
```

[22]:
```
# again check the frequency distribution of values in occupation variable

df.occupation.value_counts()
```

[22]:
```
occupation
Prof-specialty       4140
Craft-repair         4099
Exec-managerial      4066
Adm-clerical         3770
Sales                3650
Other-service        3295
Machine-op-inspct    2002
Transport-moving     1597
Handlers-cleaners    1370
Farming-fishing       994
Tech-support          928
Protective-serv       649
Priv-house-serv       149
Armed-Forces            9
Name: count, dtype: int64
```

[23]:
```
# check labels in native_country variable

df.native_country.unique()
```

[23]:
```
array(['United-States', 'Cuba', 'Jamaica', 'India', '?', 'Mexico',
       'South', 'Puerto-Rico', 'Honduras', 'England', 'Canada', 'Germany',
       'Iran', 'Philippines', 'Italy', 'Poland', 'Columbia', 'Cambodia',
       'Thailand', 'Ecuador', 'Laos', 'Taiwan', 'Haiti', 'Portugal',
       'Dominican-Republic', 'El-Salvador', 'France', 'Guatemala',
       'China', 'Japan', 'Yugoslavia', 'Peru',
       'Outlying-US(Guam-USVI-etc)', 'Scotland', 'Trinadad&Tobago',
       'Greece', 'Nicaragua', 'Vietnam', 'Hong', 'Ireland', 'Hungary',
       'Holand-Netherlands'], dtype=object)
```

```
[27]: # again check the frequency distribution of values in native_country variable

df.native_country.value_counts()
```

```
[27]: native_country
United-States                 29170
Mexico                          643
Philippines                     198
Germany                         137
Canada                          121
Puerto-Rico                     114
El-Salvador                     106
India                           100
Cuba                             95
England                          90
Jamaica                          81
South                            80
China                            75
```

```
[33]: # 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)
```

```
[34]: # check the shape of X_train and X_test

X_train.shape, X_test.shape
```

```
[34]: ((22792, 14), (9769, 14))
```

```
[35]: # check data types in X_train

X_train.dtypes
```

```
[35]: 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
```

```
[36]:
categorical = [col for col in X_train.columns if X_train[col].dtypes == 'O']
categorical
```

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

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

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

```
[42]: X_test[categorical].isnull().sum()
```

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

```
[43]: # check missing values in X_train

X_train.isnull().sum()
```

```
[43]: age               0
workclass         0
fnlwgt            0
education         0
education_num     0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
capital_gain      0
capital_loss      0
hours_per_week    0
native_country    0
dtype: int64
```

```
[44]: # check missing values in X_test

X_test.isnull().sum()
```

```
[44]: age               0
workclass         0
fnlwgt            0
education         0
education_num     0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
capital_gain      0
capital_loss      0
hours_per_week    0
native_country    0
dtype: int64
```

```
[45]: # print categorical variables
categorical
```

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

```
[46]: X_train[categorical].head()
```

[46]:

	workclass	education	marital_status	occupation	relationship	race	sex	native_country
32098	Private	HS-grad	Married-civ-spouse	Craft-repair	Husband	White	Male	United-States
25206	State-gov	HS-grad	Divorced	Adm-clerical	Unmarried	White	Female	United-States
23491	Private	Some-college	Married-civ-spouse	Sales	Husband	White	Male	United-States
12367	Private	HS-grad	Never-married	Craft-repair	Not-in-family	White	Male	Guatemala
7054	Private	7th-8th	Never-married	Craft-repair	Not-in-family	White	Male	Germany

```
[47]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       30725 non-null  object
 2   fnlwgt          32561 non-null  int64
```

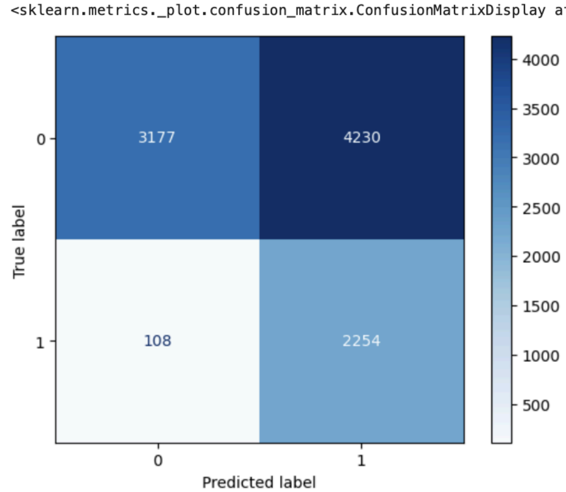
```
[49]: # Modify ColumnTransformer to return a dense array
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numerical_features),  # Scale numerical features
    ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False), categorical_features)  # OneHot encode categorical features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

```
[50]: 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
```

```
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
```

```
[51]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1e7b161a420>
```



Github :- https://github.com/Pranav-Divekar/Machine-learning-

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.