|  |  |
| --- | --- |
| **Name:** | **Soham Ajit Dahanukar** |
| **Roll No:** | **13** |
| **Class/Sem:** | TE/V |
| **Experiment No.:** | 7 |
| **Title:** | Implementation of any one classifier using languages like JAVA/ python. |
| **Date of Performance:** |  |
| **Date of Submission:** |  |
| **Marks:** |  |
| **Sign of Faculty:** |  |

**Aim:** To implement Naïve Bayesian classification

**Objective:**  Develop a program to implement Bayesian classification.

**Theory:**

The Naive Bayes is a classification algorithm that is suitable for binary and multiclass classification. Naïve Bayes performs well in cases of categorical input variables compared to numerical variables. It is useful for making predictions and forecasting data based on historical results.

The naïve Bayesian classifier, or simple Bayesian classifier, works as follows:

1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, X = (x1, x2,...,xn), depicting n measurements made on the tuple from n attributes, respectively, A1, A2,..., An.
2. Suppose that there are m classes, C1,C2,...,Cm. Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X. That is, the naïve Bayesian classifier predicts that tuple X belongs to the class Ci if and only if

P(Ci|X) > P(Cj|X) Thus we maximize P(Ci|X).The class Ci for which P(Ci|X) is maximized is called the maximum posteriori hypothesis.

By Bayes’ theorem,

P(Ci|X) = P(X|Ci)\*P(Ci)/ P(X)

1. As P(X) is constant for all classes, only P(X|Ci)P(Ci) need be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is, P(C1) = P(C2) = ··· = P(Cm), and we would therefore maximize P(X|Ci). Otherwise, we maximize P(X|Ci)P(Ci). Note that the class prior probabilities may be estimated by P(Ci)=|Ci,D|/|D|, where |Ci,D| is the number of training tuples of class Ci in D.

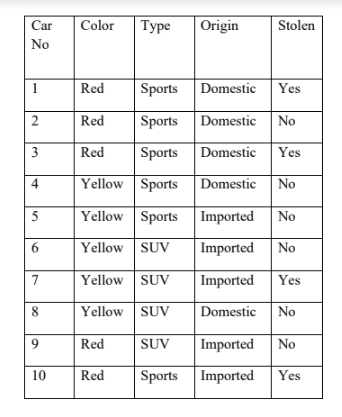
The equation: Posterior = Prior x (Likelihood over Marginal probability)

There are four parts:

* Posterior probability (updated probability after the evidence is considered)
* Prior probability (the probability before the evidence is considered)
* Likelihood (probability of the evidence, given the belief is true)
* Marginal probability (probability of the evidence, under any circumstance)

Bayes' Rule can answer a variety of probability questions, which help us (and machines) understand the complex world we live in.

**Example:**



P (yes) =5/10

P (No)=5/10

-Color:

P(Red/Y)=3/5 P(yellow/Y)=2/5

P(Red/N)=2/5 P(yellow/N)=3/5

-Type:

P(SUV/Y)=1/5 P(Sports/Y)=4/5

P(SUV/N)=3/5 P(Sports/N)=2/5

-Origin:

P(Domentic/Y)=2/5 P(Imported/Y)=3/5

P(Domentic /N)=3/5 P(Imported/N)=2/5

P(x|Yes).P(Yes)= 0.024

P(x|No).P(No)=0.072

So, Bayesian Classification Predicts the class “NO”

**Code:**

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score, classification\_report

# Sample data

data = [

{'text': 'I love machine learning', 'label': 'positive'},

{'text': 'I hate bugs', 'label': 'negative'},

{'text': 'Machine learning is fascinating', 'label': 'positive'},

{'text': 'Bugs are annoying', 'label': 'negative'},

# Add more data as needed

]

# Extract features and labels

texts = [item['text'] for item in data]

labels = [item['label'] for item in data]

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(texts, labels, test\_size=0.2, random\_state=42)

# Vectorize the text data using CountVectorizer

vectorizer = CountVectorizer()

X\_train\_vectorized = vectorizer.fit\_transform(X\_train)

X\_test\_vectorized = vectorizer.transform(X\_test)

# Create and train a Naive Bayes classifier

classifier = MultinomialNB()

classifier.fit(X\_train\_vectorized, y\_train)

# Make predictions on the test set

predictions = classifier.predict(X\_test\_vectorized)

# Evaluate the classifier

accuracy = accuracy\_score(y\_test, predictions)

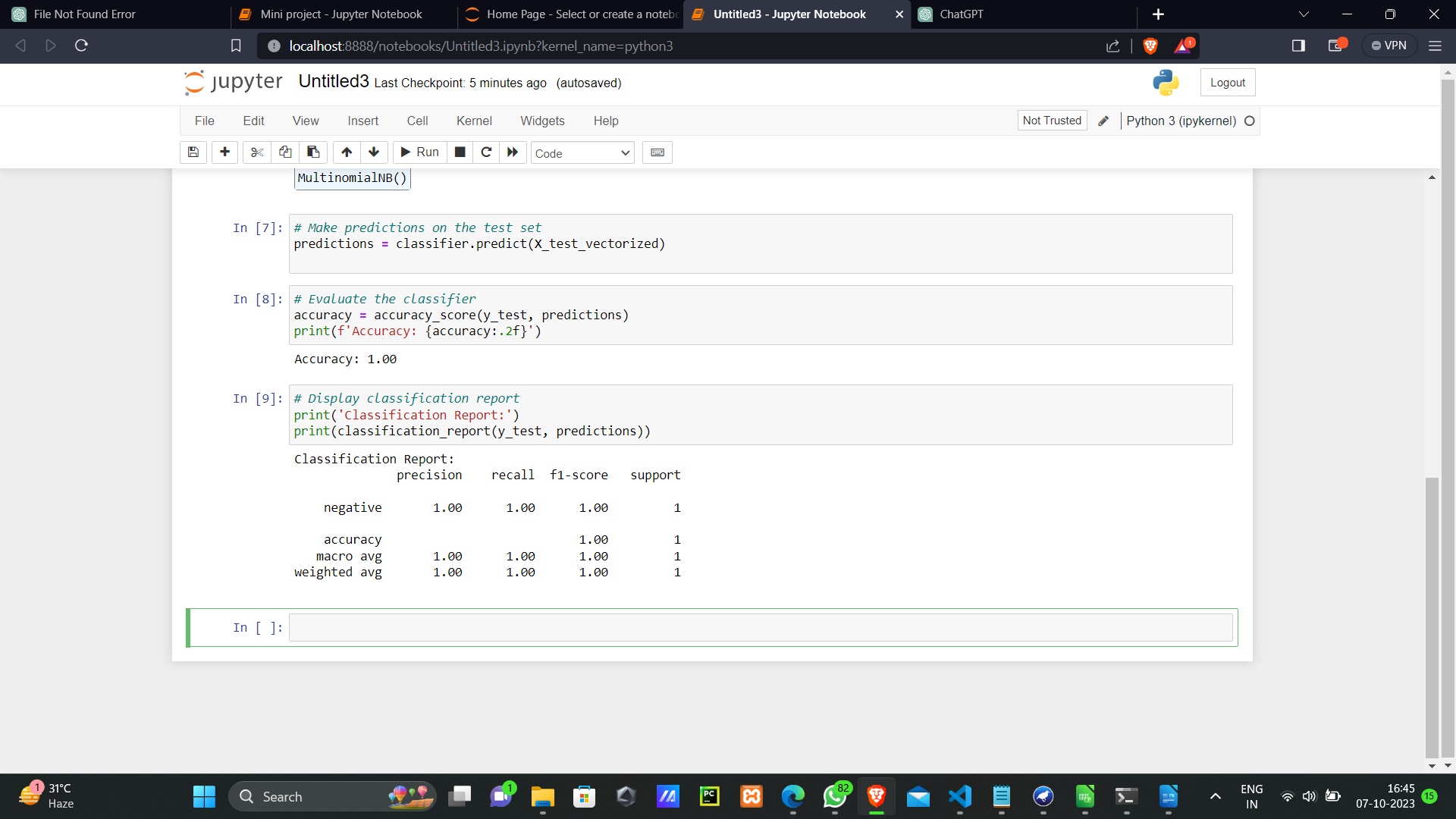
print(f'Accuracy: {accuracy:.2f}')

# Display classification report

print('Classification Report:')

print(classification\_report(y\_test, predictions))

**Output:**



**Conclusion:**