Experiment 3.2

Student Name: Sumant Rao **UID:21BCS7561**

Branch: CSE **Section/Group: 911 "B"**

Semester: 5th **Date of Performance: 18/10/23**

Subject Name: AIML **Subject Code:21CSH-316**

1. Aim:

Implement Naïve Bayes theorem to classify the English text

2. Objective:

How to calculate the probabilities required by the Naive Bayes algorithm.

How to implement the Naive Bayes algorithm from scratch.

How to apply Naive Bayes to a real-world predictive modeling problem.

3. Algorithm:

Step 1: Separate By Class.

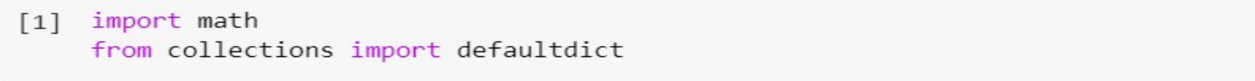
Step 2: Summarize Dataset.

Step 3: Summarize Data By Class.

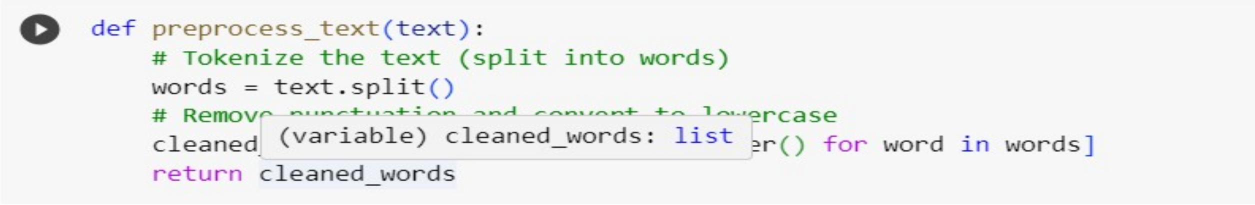
Step 4: Gaussian Probability Density Function.

Step 5: Class Probabilities.

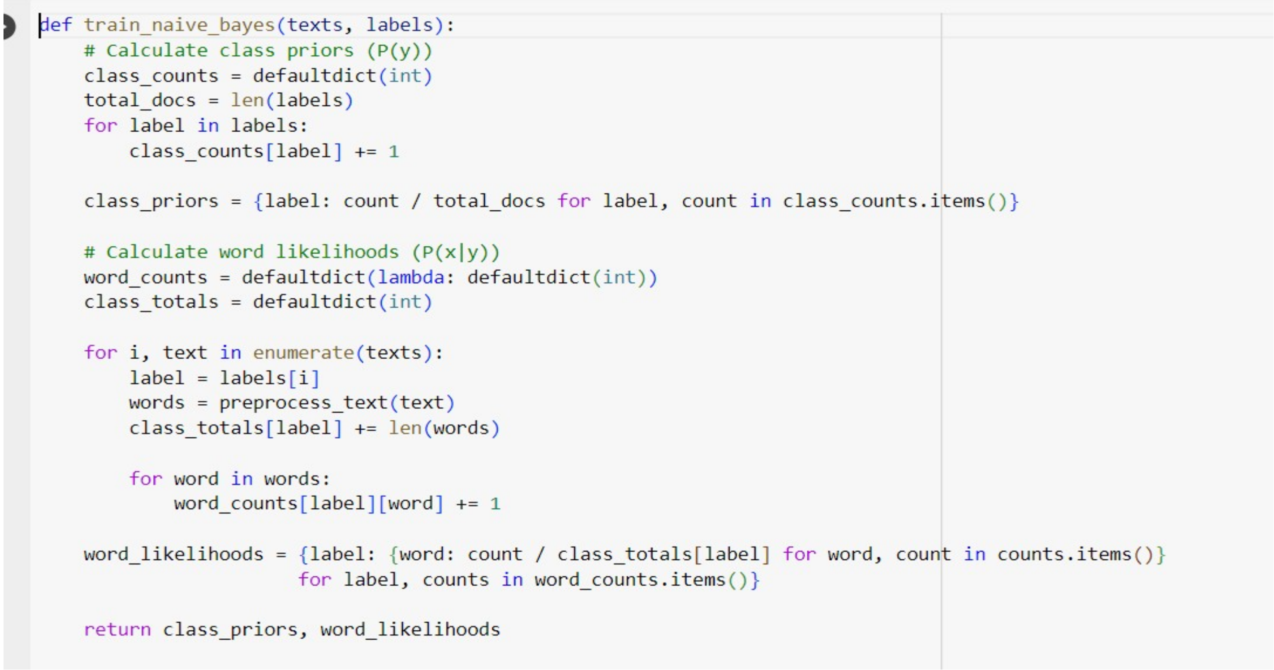
4. Code:



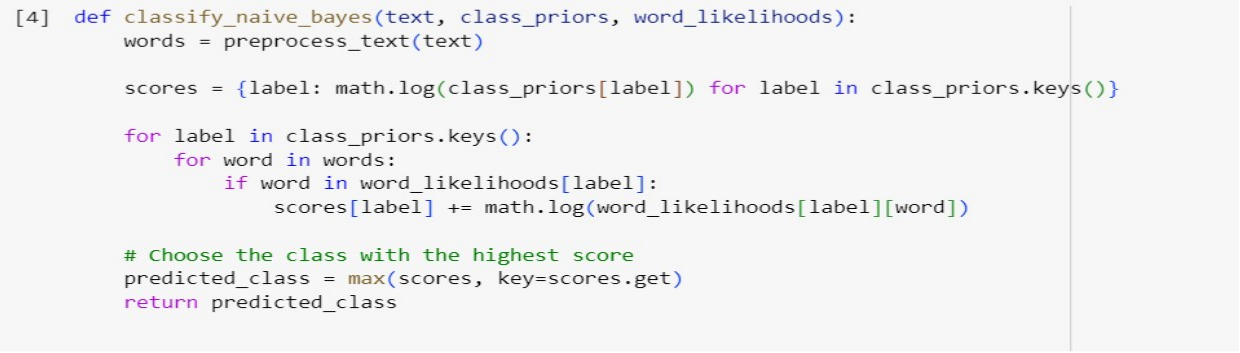
```
[1]  import math
     from collections import defaultdict
```



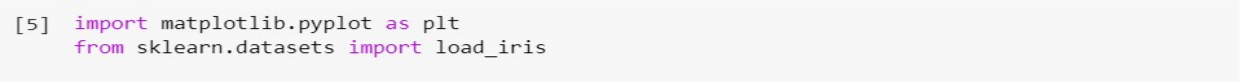
```
def preprocess_text(text):
    # Tokenize the text (split into words)
    words = text.split()
    # Remove punctuation and convert to lowercase
    cleaned (variable) cleaned_words: list er() for word in words]
    return cleaned_words
```



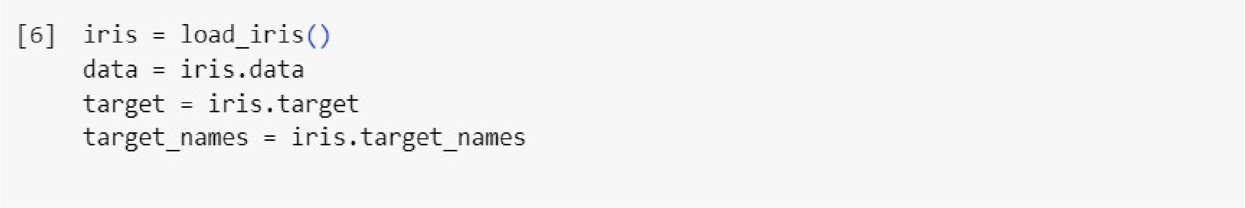
```
def train_naive_bayes(texts, labels):
    # Calculate class priors (P(y))
    class_counts = defaultdict(int)
    total_docs = len(labels)
    for label in labels:
        class_counts[label] += 1

    class_priors = {label: count / total_docs for label, count in class_counts.items()}

    # Calculate word likelihoods (P(x|y))
    word_counts = defaultdict(lambda: defaultdict(int))
    class_totals = defaultdict(int)

    for i, text in enumerate(texts):
        label = labels[i]
        words = preprocess_text(text)
        class_totals[label] += len(words)

        for word in words:
            word_counts[label][word] += 1

    word_likelihoods = {label: {word: count / class_totals[label] for word, count in counts.items()}
                       for label, counts in word_counts.items()}

    return class_priors, word_likelihoods
```



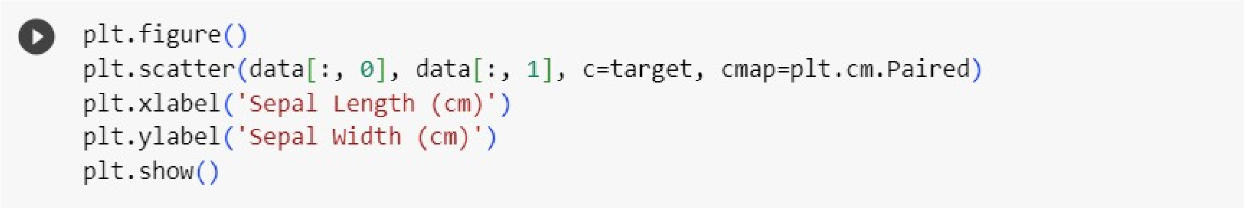
```
[4]  def classify_naive_bayes(text, class_priors, word_likelihoods):
         words = preprocess_text(text)

         scores = {label: math.log(class_priors[label]) for label in class_priors.keys()}

         for label in class_priors.keys():
             for word in words:
                 if word in word_likelihoods[label]:
                     scores[label] += math.log(word_likelihoods[label][word])

         # Choose the class with the highest score
         predicted_class = max(scores, key=scores.get)
         return predicted_class
```



```
[5]  import matplotlib.pyplot as plt
     from sklearn.datasets import load_iris
```

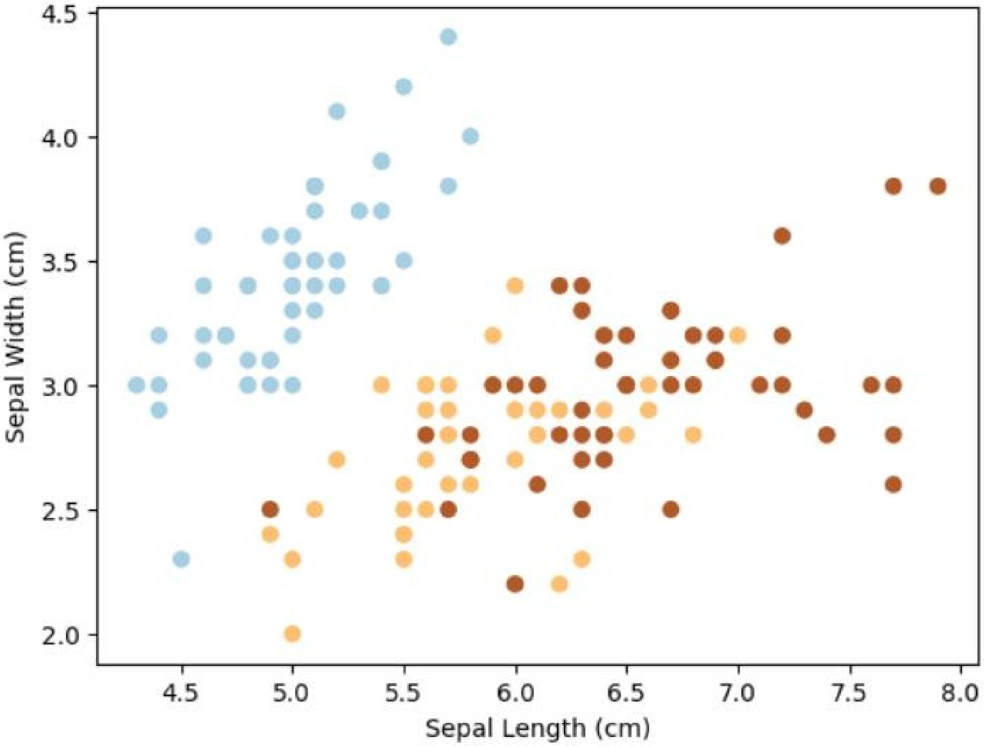


```
[6] iris = load_iris()
    data = iris.data
    target = iris.target
    target_names = iris.target_names
```



```
plt.figure()
plt.scatter(data[:, 0], data[:, 1], c=target, cmap=plt.cm.Paired)
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.show()
```

5. Output


Sepal Width (cm)
Sepal Length (cm)