Experiment3.2

Student Name:- Nishant Kumar Mehta
UID:- 21BCS3402
Branch:- B.E(CSE)
Section/Group:-602/B
Semester:- Vth
Date of Performance:- 26/10/23
Subject Name:- Artificial Intelligence & Machine Learning
Subject Code:- 21CSH-316

Aim:- Implement Naive Bayes theorem to classify the English text.

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.

Procedure/Algorithm/Code:-

First we will develop each piece of the algorithm, then we will tie all of the elements together into a working implementation applied to a real dataset.

This Naive Bayes tutorial is broken down into 5 parts:

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.

These steps will provide the foundation that you need to implement Naive Bayes from scratch and apply it to your own predictive modeling problems.

Source Code:-

```
from collections import defaultdict
import math

# Sample training data (text samples with corresponding labels)
training_data = [
    ("I love programming", "positive"),
    ("I hate bugs", "negative"), ("Coding
    is fun", "positive"), ("Bugs are
    annoying", "negative")
]

# Step 1: Data Preparation
vocabulary = set()
class_word_counts = defaultdict(lambda: defaultdict(int))
class_counts = defaultdict(int)

for text, label in training_data:
    words = text.lower().split()
    vocabulary.update(words)
    class_counts[label] += 1
    for word in words:
        class_word_counts[label][word] += 1

# Step 2: Calculate Class Priors

total_samples = len(training_data)
class_priors = {label: count / total_samples for label, count in class_counts.items()}
```

```
# Step 3: Calculate Likelihood Probabilities
word_likelihoods = defaultdict(lambda: defaultdict(float))
for label, word_counts in class_word_counts.items():
    total_words_in_class = sum(word_counts.values())
    for word in vocabulary:
        word_likelihoods[label][word] = (word_counts[word] + 1) /
(total_words_in_class + len(vocabulary))

# Step 4: Text Classification (Example for a new
text) new_text = "I love coding bugs"
tokenized_text = new_text.lower().split()

class_scores = defaultdict(float)
for label in class_priors:
    class_scores[label] = math.log(class_priors[label]) # Prior
    probability for word in tokenized_text:
        if word in vocabulary:
            class_scores[label] += math.log(word_likelihoods[label][word]) #
Likelihood probability
predicted_class = max(class_scores, key=class_scores.get)
print("Predicted Class:", predicted_class)
```

Output:-

```
    ("I love programming", "positive"),
    ("I hate bugs", "negative"),
    ("Coding is fun", "positive"),
    ("Bugs are annoying", "negative")
]

# Step 1: Data Preparation

vocabulary = set()
class_word_counts = defaultdict(lambda: defaultdict(int))
class_counts = defaultdict(int)

for text, label in training_data:
    words = text.lower().split()
    vocabulary.update(words)
    class_counts[label] += 1
    for word in words:
        class_word_counts[label][word] += 1

# Step 2: Calculate Class Priors

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

# Step 3: Calculate Likelihood Probabilities

word_likelihoods = defaultdict(lambda: defaultdict(float))

for label, word_counts in class_word_counts.items():
    total_words_in_class = sum(word_counts.values())
    for word in vocabulary:
        word_likelihoods[label][word] = (word_counts[word] + 1) / (total_words_in_class + len(vocabulary))

# Step 4: Text Classification (Example for a new text)
new_text = "I love coding bugs"
tokenized_text = new_text.lower().split()

class_scores = defaultdict(float)

for label in class_priors:
    class_scores[label] = math.log(class_priors[label])  # Prior probability
    for word in tokenized_text:
        if word in vocabulary:
            class_scores[label] += math.log(word_likelihoods[label][word])  # Likelihood probability

predicted_class = max(class_scores, key=class_scores.get)

print("Predicted Class:", predicted_class)
```

```
Predicted Class: positive
```

Learning Outcomes:-

- Learnt about Data Summary.
- Learnt about Dara Cleaning.
- Implementation of Data Visualization.