Text Classification Using Naive Bayes

Overview

Text classification is a fundamental task in Natural Language Processing (NLP) that involves assigning predefined categories to textual data. Applications include spam detection, sentiment analysis, and topic labeling. One of the most effective and straightforward algorithms for text classification is the **Naive Bayes classifier**.

The Naive Bayes classifier is a probabilistic model based on Bayes' Theorem, operating under the "naive" assumption that features are conditionally independent given the class label. Despite this simplification, it performs remarkably well in various scenarios, especially in text classification tasks. ?cite?turn0search0?

Why Use Naive Bayes for Text Classification?

- **Simplicity**: Easy to implement and computationally efficient, making it suitable for large datasets.
- **Performance**: Despite its simplicity, it often outperforms more complex models, particularly when the independence assumption holds.
- **Robustness**: Performs well even with limited training data and can handle irrelevant features effectively.

Prerequisites

Ensure you have the following installed:

- **Python Environment**: Python 3.x
- **Libraries**:
  - `scikit-learn` : For machine learning algorithms and utilities
  - `numpy` : For numerical operations
  - `pandas` : For data manipulation

Install the required libraries using pip:

```
pip install scikit-learn numpy pandas
```

Files Included

- **text_classification.py**: Contains the implementation of the Naive Bayes text classification model.
- **data.csv**: A sample dataset with text samples and their corresponding labels.

Code Description

The following code demonstrates how to implement a Naive Bayes classifier for text classification:

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Sample data
texts = ["I love this movie", "This film was terrible", "Amazing acting", "Horrible plo
labels = ["positive", "negative", "positive", "negative"]

# Vectorize the text data
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.25, random_s

# Initialize the Multinomial Naive Bayes model
model = MultinomialNB()

# Train the model
model.fit(X_train, y_train)

# Test the model
test_text = ["Great movie!"]
test_vector = vectorizer.transform(test_text)
prediction = model.predict(test_vector)
print(f"Prediction: {prediction[0]}")

# Evaluate the model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

Explanation:

1. **Data Preparation**: We start with a list of text samples (`texts`) and their corresponding sentiment labels (`labels`).

2. **Text Vectorization**: The `CountVectorizer` from `scikit-learn` is used to convert the text data into a matrix of token counts, which serves as input features for the model.

3. **Data Splitting**: The dataset is split into training and testing sets using `train_test_split` to evaluate the model's performance on unseen data.

4. **Model Initialization and Training**: A `MultinomialNB` model is initialized and trained on the training data.

5. **Prediction**: The trained model predicts the sentiment of a new text sample (`test_text`).

6. **Evaluation**: The model's performance is assessed using accuracy, classification report, and confusion matrix metrics.

Expected Output

For the provided sample data, the output will include the predicted sentiment for the `test_text` and the evaluation metrics for the model:

```
 Prediction: positive
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

    negative       1.00      1.00      1.00         1
    positive       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2

Confusion Matrix:
[[1 0]
 [0 1]]
```

Use Cases

- **Spam Detection**: Classifying emails or messages as spam or not spam.

- **Sentiment Analysis**: Determining the sentiment (positive, negative, neutral) of text data, such as product reviews or social media posts.

- **Document Categorization**: Assigning documents to predefined categories based on their content.

Advantages

- **Efficiency**: Fast training and prediction, suitable for large-scale applications.

- **Simplicity**: Easy to implement and interpret.

- **Scalability**: Performs well with a large number of features, common in text data.

Future Enhancements

- **Incorporate TF-IDF Vectorization**: Using `TfidfVectorizer` instead of `CountVectorizer` can improve performance by considering the importance of words across the dataset.

- **Hyperparameter Tuning**: Experimenting with different smoothing parameters (e.g., `alpha` in `MultinomialNB`) to optimize