Q: predict the category to which a piece of given text belongs

Text classification, which is a common task in Natural Language Processing (NLP). Text classification involves assigning predefined categories to text documents. This can be used for spam detection, sentiment analysis, and more.

1. Introduction to Text Classification

Text classification is the process of categorizing text into organized groups. By using machine learning algorithms, we can automate this process. Common applications include:

- **Spam Detection**: Classifying emails as spam or not spam.
- **Sentiment Analysis**: Determining if a review is positive or negative.
- **Topic Labeling**: Assigning topics to news articles.

2. Steps in Text Classification

1. **Data Collection**: Gather a dataset of text documents with labeled categories.
2. **Text Preprocessing**: Clean and prepare the text data.
    - Tokenization: Splitting text into words or tokens.
    - Removing Stop Words: Eliminating common words that do not carry significant meaning (e.g., “and”, “the”).
    - Stemming/Lemmatization: Reducing words to their base or root form.
3. **Feature Extraction**: Convert text into numerical features.
    - **Bag of Words (BoW)**: Representing text by the frequency of words.
    - **TF-IDF (Term Frequency-Inverse Document Frequency)**: Reflecting the importance of a word in a document relative to a collection of documents.
4. **Model Training**: Train a machine learning model on the processed data.
5. **Model Evaluation**: Assess the model’s performance using metrics like accuracy, precision, recall, and F1-score.
6. **Prediction**: Use the trained model to classify new text data.

Python Code Example

```
# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

# Sample dataset
data = {
    'text': [
        'I love this movie, it is fantastic!',
        'This film is terrible, I hate it.',
        'What a great performance by the actor!',
        'The plot was boring and predictable.',
        'An excellent movie with a wonderful story.',
        'I did not enjoy the film at all.'
    ],
    'label': ['positive', 'negative', 'positive', 'negative', 'positive', 'negative']
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Text preprocessing and feature extraction
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(df['text'])
y = df['label']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a Naive Bayes classifier
model = MultinomialNB()
```

```
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
print(classification_report(y_test, y_pred))

# Function to predict the label of a given sentence
def predict_label(sentence):
    sentence_transformed = vectorizer.transform([sentence])
    prediction = model.predict(sentence_transformed)
    return prediction[0]

# Prompt the user for a new sentence and classify it
new_sentence = input("Enter a sentence to classify as positive or negative: ")
predicted_label = predict_label(new_sentence)
print(f'The sentence "{new_sentence}" is classified as: {predicted_label}')
```

Explanation of the Code

1. **Data Preparation**: We create a sample dataset with text and corresponding labels.
2. **Text Preprocessing**: We use TfidfVectorizer to convert text into numerical features.
3. **Model Training**: We split the data into training and testing sets and train a Naive Bayes classifier.
4. **Model Evaluation**: We evaluate the model's performance using accuracy and a classification report.