Introduction to Sentiment Analysis

Sentiment analysis, also known as opinion mining, is a Natural Language Processing (NLP) technique used to determine the emotional tone behind a body of text. It's a powerful tool for understanding public opinion, monitoring brand reputation, and analyzing customer feedback. This notebook will guide you through the process of performing a sentiment analysis task from start to finish, using a supervised machine learning approach.
We will cover the following key stages:

1. **Data Acquisition:** Loading a suitable dataset for training and testing.
2. **Data Preprocessing:** Cleaning and transforming the raw text data into a format that a machine learning model can understand.
3. **Feature Extraction:** Converting the processed text into numerical features.
4. **Model Implementation:** Training a machine learning classifier on the data.
5. **Evaluation and Insights:** Assessing the model's performance and interpreting the results.

This guide will utilize popular Python libraries such as pandas for data manipulation, NLTK and scikit-learn for NLP and machine learning tasks, and matplotlib and seaborn for data visualization.

1. Data Acquisition

For this project, we will use a publicly available dataset of labeled tweets or movie reviews. A common choice is the NLTK movie_reviews corpus or a Kaggle dataset of tweets. This notebook will assume we are using a CSV file with two columns: text and sentiment (labeled as 'positive', 'negative', or 'neutral').

```
# Import necessary libraries
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
# Replace 'your_dataset.csv' with the actual path to your file
try:
    df = pd.read_csv('your_dataset.csv')
    print("Dataset loaded successfully.")
    print(df.head())
    print("\nDataset Info:")
```

```
    df.info()
except FileNotFoundError:
    print("Dataset not found. Please provide a valid path to your CSV
file.")
```

2. Data Preprocessing

Raw text data is messy and needs to be cleaned before it can be used to train a model. Our preprocessing function will perform the following steps:

- **Lowercasing:** Convert all text to lowercase to ensure consistency.
- **Punctuation and Special Character Removal:** Remove punctuation, numbers, and other special characters.
- **Tokenization:** Split sentences into individual words (tokens).
- **Stop-word Removal:** Eliminate common words (e.g., 'the', 'a', 'is') that don't add significant meaning.
- **Lemmatization:** Reduce words to their base or root form (e.g., 'running' becomes 'run').

<!-- end list -->

```
# Download NLTK resources (run this once)
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')

# Initialize NLTK tools
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # Check if the text is a string
    if not isinstance(text, str):
        return ""
    # Lowercase and remove special characters
    text = re.sub(r'[^a-zA-Z\s]', '', text.lower())
    # Tokenize
    tokens = text.split()
    # Remove stop words and lemmatize
    processed_tokens = [lemmatizer.lemmatize(word) for word in tokens
if word not in stop_words]
    # Join tokens back into a string
    return " ".join(processed_tokens)

# Apply the preprocessing function to the 'text' column
df['cleaned_text'] = df['text'].apply(preprocess_text)

print("\nOriginal vs. Cleaned Text:")
print(df[['text', 'cleaned_text']].head())
```

3. Feature Extraction

Machine learning models require numerical input. We'll use the **TF-IDF (Term Frequency-Inverse Document Frequency)** vectorizer to convert our cleaned text data into a matrix of numerical features. TF-IDF gives more weight to words that are important and less frequent across all documents.

```
# Define features (X) and target (y)
X = df['cleaned_text']
y = df['sentiment']

# 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, stratify=y)

# Initialize TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000) # You can adjust
max_features

# Fit and transform the training data, and transform the test data
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

print("\nShape of TF-IDF matrices:")
print(f"X_train_tfidf shape: {X_train_tfidf.shape}")
print(f"X_test_tfidf shape: {X_test_tfidf.shape}")
```

4. Model Implementation (Logistic Regression)

We will use **Logistic Regression**, a powerful and interpretable machine learning algorithm, to classify the sentiment of the text. It's a great choice for a baseline model due to its efficiency and effectiveness on text classification problems.

```
# Initialize the Logistic Regression model
model = LogisticRegression(max_iter=1000)

# Train the model
print("\nTraining the model...")
model.fit(X_train_tfidf, y_train)
print("Model training complete.")

# Make predictions on the test set
y_pred = model.predict(X_test_tfidf)
```

5. Evaluation and Insights

Finally, we'll evaluate our model's performance using standard metrics like accuracy, a classification report, and a confusion matrix. We will also visualize the results to gain a deeper understanding of the model's strengths and weaknesses.

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"\nModel Accuracy: {accuracy:.2f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Visualize the confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=model.classes_, yticklabels=model.classes_)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Sentiment Analysis')
plt.show()

# Display some example predictions to get a feel for the model
example_texts = [
    "This movie was absolutely fantastic, a masterpiece!",
    "I'm not happy with the service; it was a terrible experience.",
    "The weather is just okay today.",
    "This product is amazing, highly recommended."
]

# Preprocess and vectorize the example texts
cleaned_examples = [preprocess_text(text) for text in example_texts]
vectorized_examples = tfidf_vectorizer.transform(cleaned_examples)

# Predict sentiment for the examples
predictions = model.predict(vectorized_examples)

print("\nSentiment Predictions for Example Texts:")
for i, text in enumerate(example_texts):
    print(f"Text: \"{text}\" -> Predicted Sentiment:
{predictions[i]}")
```

This notebook provides a complete and reproducible workflow for performing sentiment analysis on textual data. It can be easily adapted to different datasets and classification models (e.g., Naive Bayes, SVM) to explore and compare various approaches.