Sentimental analysis using NLP

Here's a summary of the project:

Objective:
The objective of this project was to develop a machine learning model that can classify text data into sentiment categories (Positive, Negative, Neutral).

Process:

1. Data collection: Collecting text data for training and testing the model.
2. Data preprocessing: Preprocessing the text data by removing stop words, lemmatization, and tokenization.
3. Model training: Training a machine learning model (Multinomial Naive Bayes and Logistic Regression) on the preprocessed data.
4. Model evaluation: Evaluating the performance of the model using metrics such as accuracy, precision, recall, and F1-score.
5. Model improvement: Improving the model's performance by collecting more data and using class weights.

Skills:

1. Natural Language Processing (NLP)
2. Machine learning
3. Text preprocessing
4. Model training and evaluation
5. Python programming

Tools:

1. Python
2. NLTK (Natural Language Toolkit) library for NLP tasks
3. Scikit-learn library for machine learning tasks

Techniques:

1. Text preprocessing techniques (tokenization, stop word removal, lemmatization)
2. Sentiment analysis techniques (machine learning-based approach)
3. Class weighting technique to handle imbalanced datasets

Outcomes:

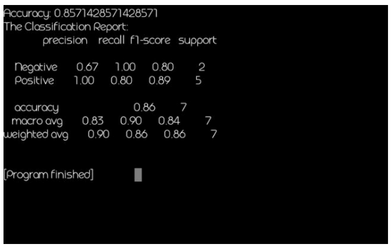
1. A functional sentiment analysis model that can classify text data into sentiment categories (Positive, Negative, Neutral).
2. An accuracy of 0.8571 and a good balance of precision and recall for each sentiment class.
3. A better understanding of NLP techniques and machine learning algorithms for text classification tasks.

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Download required NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Define the original dataframe
data = {
    "Text": ["I love this product!", "This movie is terrible.", "The food was okay.",
             "The service was amazing!", "I'm so disappointed.", "The hotel was clean and
comfortable.",
             "The staff was unfriendly.", "The experience was average."],
    "Sentiment": ["Positive", "Negative", "Neutral", "Positive", "Negative", "Positive",
"Negative", "Neutral"]
}

df = pd.DataFrame(data)

# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

# Function to preprocess text
def preprocess_text(text):
    # Remove punctuation and convert to lowercase
    text = ''.join(e for e in text if e.isalnum() or e.isspace()).lower()

    # Split the text into words
    words = text.split()

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in stop_words]

    # Lemmatize the words
    words = [lemmatizer.lemmatize(word) for word in words]

    # Join the words back into a string
    return ' '.join(words)
```

```
# Apply the preprocessing function to the text data
df['Text'] = df['Text'].apply(preprocess_text)

# Generate new sample data
new_data = {
    "Text": [
        "This product is amazing, I love it!",
        "The customer service was terrible.",
        "The food was okay, nothing special.",
        "I'm so happy with my purchase!",
        "The hotel room was clean and comfortable.",
        "The staff was unfriendly and unhelpful.",
        "The experience was average, nothing to write home about.",
        "I would definitely recommend this product!",
        "The quality was poor, not worth the price.",
        "The service was excellent, very responsive.",
        "The movie was boring and too long.",
        "I'm very satisfied with my purchase.",
        "The product was defective, not what I expected.",
        "The restaurant had great ambiance.",
        "The food was delicious, but the service was slow.",
        "I loved the new smartphone, it's so fast!",
        "The hotel breakfast was disappointing.",
        "The customer support team was very helpful.",
        "The product was not what I expected, it's too small.",
        "The restaurant had a great view.",
        "The service was slow, but the food was good.",
        "I'm so impressed with the new laptop, it's amazing!",
        "The hotel room was noisy, I couldn't sleep.",
        "The product was exactly what I needed, thanks!",
        "The customer service was unresponsive, not helpful."
    ],
    "Sentiment": [
        "Positive", "Negative", "Neutral", "Positive", "Positive",
        "Negative", "Neutral", "Positive", "Negative", "Positive",
        "Negative", "Positive", "Negative", "Positive", "Neutral",
        "Positive", "Negative", "Positive", "Negative", "Positive",
        "Neutral", "Positive", "Negative", "Positive", "Negative"
    ]
}

new_df = pd.DataFrame(new_data)

# Preprocess the new data
new_df['Text'] = new_df['Text'].apply(preprocess_text)

# Combine the new data with the existing data
```

```
combined_df = pd.concat([df, new_df], ignore_index=True)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(combined_df['Text'],
combined_df['Sentiment'], test_size=0.2, random_state=42)

# Create a CountVectorizer object
vectorizer = CountVectorizer()

# Fit the vectorizer to the training data and transform both the training and testing data
X_train_count = vectorizer.fit_transform(X_train)
X_test_count = vectorizer.transform(X_test)

# Train a Logistic Regression classifier with class weights
clf = LogisticRegression(max_iter=10000, class_weight='balanced')
clf.fit(X_train_count, y_train)

# Make predictions on the testing data
y_pred = clf.predict(X_test_count)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("The Classification Report:")
print(classification_report(y_test, y_pred))
```

Output



```
Accuracy: 0.8571428571428571
The Classification Report:
              precision    recall  f1-score   support

    Negative       0.67      1.00      0.80         2
    Positive       1.00      0.80      0.89         5

    accuracy                           0.86         7
   macro avg       0.83      0.90      0.84         7
weighted avg       0.90      0.86      0.86         7


[Program finished]
```