Sentiment Analysis API Documentation

Introduction

This document provides an overview of a Sentiment Analysis API developed using Flask. The API processes customer reviews from XLSX or CSV files and performs sentiment analysis using the Groq API. It returns structured results indicating the sentiment distribution among the reviews.

Approach to Solving the Problem

The primary goal of this project was to create an API that could efficiently analyze customer reviews and return the sentiment scores. The approach involved the following steps:

1. **API Development:**
   - Developed a Flask-based API that accepts XLSX or CSV files containing customer reviews.
   - Utilized the Groq API to perform sentiment analysis on the extracted review text.
2. **File Handling:**
   - Implemented functionality to read both CSV and XLSX files.
   - Extracted review texts from the uploaded files, ensuring the presence of a "review" column.
3. **Sentiment Analysis:**
   - For each review, the sentiment was analyzed using the Groq API.
   - Collected results and classified them into positive, negative, or neutral categories.
4. **Structured Response:**
   - The API returns the sentiment distribution in a structured JSON format.

Implemented Structured Response

The API provides a structured JSON response in the following format:

```
{
  "positive": score,
  "negative": score,
  "neutral": score
}
```

Where:
- positive : Count of reviews classified as positive.
- negative: Count of reviews classified as negative.
- neutral: Count of reviews classified as neutral.

Model Used

- **Groq API**
  - Description of the Groq API and its capabilities for sentiment analysis.
  - Details about the specific model used:
    - **Model Name:** mixtral-8x7b-32768
    - **Purpose:** Designed for analyzing text data, the model is fine-tuned for sentiment classification, providing insights into whether a review is positive, negative, or neutral.
  - Discussion on the advantages of using this model, such as:
    - High accuracy in understanding context and nuances in language.
    - Ability to handle various review styles and sentiments.

Examples of API Usage

Example API Outputs

The following are the outputs obtained from testing the API with customer reviews:

1. **Using a CSV File:**
   - **Command:**

```
curl -X POST -F "file=@curl -X POST -F "file=@/path/to/
customer_reviews.csv" http://127.0.0.1:5000/analyze
```

Response:

```
{
  "negative": 24,
  "neutral": 0,
  "positive": 27
}
```

2.Using an XLSX File:

- **Command:**

```
curl -X POST -F "file=@curl -X POST -F "file=@/path/to/customer_reviews.xlsx"
http://127.0.0.1:5000/analyze
```

- **Response:**

```
{
  "negative": 24,
  "neutral": 0,
  "positive": 27
}
```

Analysis of Results

- **Consistency in Results:** The sentiment analysis returned consistent results for both file formats (CSV and XLSX), indicating that the API is robust in handling different input types.
- **Sentiment Distribution:** The output shows a slightly positive sentiment overall, with a majority of reviews being classified as positive (27) compared to negative (24) and neutral (0).
- **Limitations:** The API currently relies solely on the Groq API for sentiment analysis, which may not capture nuanced sentiments, especially in reviews with mixed feelings or sarcasm.

Potential Improvements

1. **Enhanced Sentiment Classification:** Implementing more granular sentiment classifications (e.g., very positive, somewhat positive) could provide deeper insights into customer sentiment.
2. **Emoji Analysis:** Incorporating analysis of emojis present in reviews could further refine sentiment detection, as emojis often convey emotions that words may not fully capture.

3. **Data Storage:** Storing analyzed results in a database for future reference could enhance data accessibility and allow for historical trend analysis.

Additional Insights

The API demonstrates the feasibility of integrating language models for sentiment analysis tasks. It provides a foundation for building more complex systems that can analyze customer feedback in real-time, enabling businesses to respond promptly to customer sentiments.

Implementation

Flask API Code

```
from flask import Flask, request, jsonify
import pandas as pd
import os
from groq import Groq
from dotenv import load_dotenv
load_dotenv()

app = Flask("senti")

# Initialize Groq client with the API key
client = Groq(api_key=os.environ.get("GROQ_API_KEY"))
@app.route('/analyze', methods=['POST'])
def analyze():
    # Check if a file was uploaded
    if 'file' not in request.files:
        return jsonify({"error": "No file uploaded"}), 400

    file = request.files['file']

    # Check if the file is a valid Excel or CSV file
    if not file.filename.endswith(('.xlsx', '.xls', '.csv')):
        return jsonify({"error": "File must be in XLSX, XLS, or CSV
format"}), 400

    # Read the uploaded file based on its format
    if file.filename.endswith('.csv'):
        df = pd.read_csv(file)
    else:
        df = pd.read_excel(file)
```

```
    if "review" not in df.columns and "Review" not in df.columns:
        return jsonify({"error": "File must contain 'review' column"}), 400

    # Extract reviews from the DataFrame
    reviews = df["review"].tolist() if "review" in df.columns else
df["Review"].tolist()

    results = []

    # Analyze each review using the Groq API and log each sentiment result
    for i, review in enumerate(reviews):
        chat_completion = client.chat.completions.create(
            messages=[
                {
                    "role": "user",
                    "content": f"Analyze the sentiment of this review:
{review}",
                }
            ],
            model="mixtral-8x7b-32768",
        )
        sentiment = chat_completion.choices[0].message.content
        results.append(sentiment)

        # Print the review and the sentiment result for debugging purposes
        print(f"Review {i+1}: {review}")
        print(f"Sentiment: {sentiment}")
        print("-" * 50)

    # Custom logic to classify sentiments based on results (only 1 sentiment
per review)
    positive_score = 0
    negative_score = 0
    neutral_score = 0

    for result in results:
        result_lower = result.lower()
        # Adjust positive/negative classifications to be more strict
        if 'positive' in result_lower or 'great' in result_lower or
'fantastic' in result_lower:
            positive_score += 1
```

```
        elif 'negative' in result_lower or 'bad' in result_lower or
'terrible' in result_lower:
            negative_score += 1
        else:
            # Expanded list of neutral indicators
            neutral_keywords = ["okay", "average", "nothing special",
"mediocre", "moderate", "no complaints", "fine", "not bad", "neutral",
"ordinary"]
            if any(keyword in result_lower for keyword in neutral_keywords):
                neutral_score += 1
            else:
                neutral_score += 1  # Default to neutral for any ambiguity


    # Ensure no review is counted multiple times and the total count adds up
to the number of reviews
    total_reviews = len(reviews)
    classified_reviews = positive_score + negative_score + neutral_score


    print(f"Total reviews: {total_reviews}")
    print(f"Classified reviews: {classified_reviews} (should match total
reviews)")


    return jsonify({
        "positive": positive_score,
        "negative": negative_score,
        "neutral": neutral_score
    })


if __name__ == '__main__':
    app .run(debug=True)
```

Outputs

Using a CSV File:
- **Command:**

```
curl -X POST -F "file=@/Users/souravbudke/Desktop/customer_reviews.csv"
http://127.0.0.1:5000/analyze
```

- **Response:**

```
{
  "negative": 24,
  "neutral": 0,
  "positive": 27
}
```

Using an XLSX File:
- **Command:**

```
curl -X POST -F "file=@/Users/souravbudke/Downloads/customer_reviews.xlsx"
http://127.0.0.1:5000/analyze
```

- **Response:**

```
{
  "negative": 24,
  "neutral": 0,
  "positive": 27
}
```

References

1. **Flask Documentation:** Flask is a lightweight WSGI web application framework in Python. Documentation can be found at Flask Documentation.
2. **Pandas Documentation:** A powerful data analysis and manipulation library for Python. Documentation is available at Pandas Documentation.
3. **Groq API Documentation:** Groq provides API services for large language models. For more information, visit Groq API Documentation.
4. **dotenv Documentation:** A Python library for managing environment variables. More details can be found at python-dotenv Documentation.