Sentiment Analyzer Documentation by Luo Xin Yi

Objective and Overview

Objective

The *sentiment_analyzer.py* script is designed to perform sentiment analysis on user-provided restaurant reviews using the Hugging Face Inference API with the *distilbert-base-uncased-finetuned-sst-2-english model* (Hugging Face, 2025a). The primary objective is to classify text as positive or negative, providing a confidence score, to assist local businesses in understanding customer feedback. The tool incorporates robust input validation, configuration management, and result logging to ensure reliability and usability in real-world applications.

Overview

The script is a Python-based command-line application that uses natural language processing (NLP) to analyze sentiment in text inputs. Key functionalities include:

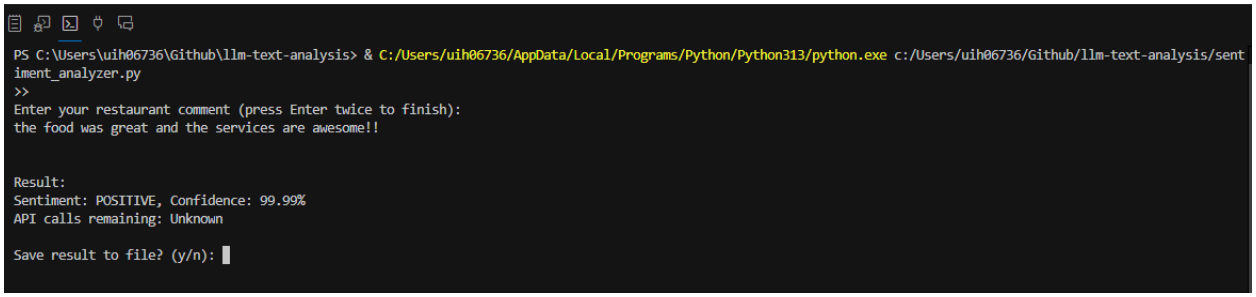
- **Input Collection**: Users enter users' review via the console, with input terminated by two consecutive Enter presses.
- **Input Validation**: The script employs the Natural Language Toolkit (NLTK) word list (Bird et al., 2009) to ensure inputs are meaningful, requiring at least 60% of words to be valid English words and filtering out repetitive or gibberish text.
- **API Integration**: The Hugging Face Inference API (Hugging Face, 2025b) is used to query the DistilBERT model for sentiment classification.
- **Result Processing**: The script parses API responses to extract sentiment labels and confidence scores, handling errors such as rate limits or network issues.
- **Result Storage**: Users can save results to timestamped files in a designated directory.
- **Configuration Management**: A *config.ini* file, parsed using Python's configparser (Python Software Foundation, 2025), stores API and validation parameters for modularity.

The tool is designed for accessibility, requiring only a free Hugging Face API key and standard Python libraries *(requests, nltk, configparser, python-dotenv*). It serves as an educational resource for learning NLP and a practical tool for small-scale sentiment analysis tasks.

Sample Outputs of Program in Use

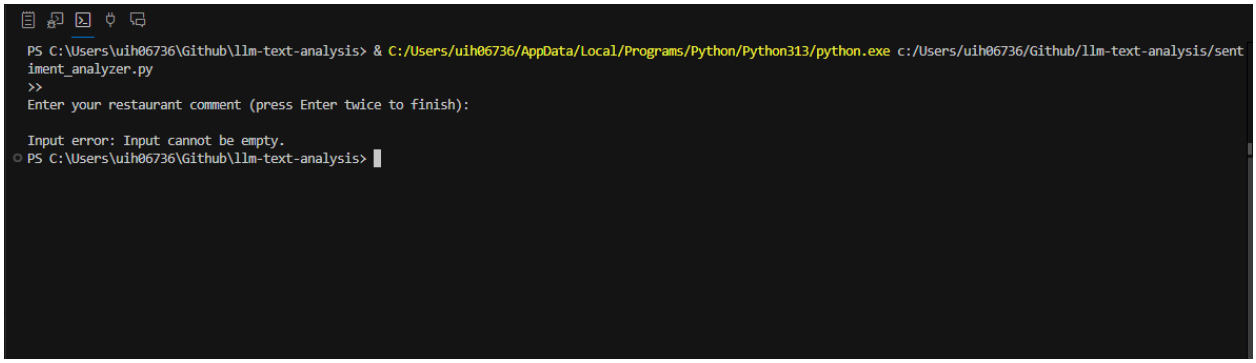
The first example illustrates a valid input processed by the DistilBERT model, yielding a high-confidence positive sentiment. The user opts to save the result, which is written to a file in the ***results*** directory. The second example shows a semi-coherent input rejected due to failing the 60% meaningful word ratio threshold, highlighting the script's validation mechanism.

Example of Accepted Input:

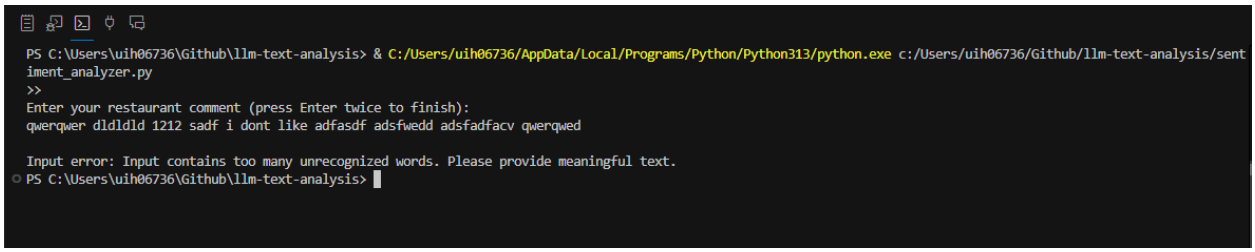


Example of Rejected Input:

1. Empty string case



2. Gibberish case



Reflections on API Usage and Limitations

API Usage

The *sentiment_analyzer.py* script utilizes the Hugging Face Inference API to perform sentiment analysis with the **distilbert-base-uncased-finetuned-sst-2-english** model, a distilled version of BERT optimized for binary sentiment classification (Sanh et al., 2019). The API's free tier (Hugging Face, 2025b) enables rapid integration without local model hosting, requiring only an API key stored securely in a .env file using *python-dotenv* (Saurabh, 2025). The script sends text inputs to the API endpoint. (**https://api-inference.huggingface.co/models/distilbert/distilbert-base-uncased-finetuned-sst-2-english**) and processes responses containing sentiment labels and confidence scores. Robust error handling addresses HTTP errors (e.g., 429 for rate limits) and network issues, while the *X-RateLimit-Remaining* header informs users of available API calls, typically around 1,000 per day for free accounts.

The choice of DistilBERT balances performance and efficiency, making it suitable for lightweight applications. The script's configuration management via *config.ini* allows users to modify API endpoints or validation parameters without altering the code, enhancing maintainability (Python Software Foundation, 2025).

Limitations

Despite its effectiveness, the script and the Hugging Face Inference API have notable limitations:

1. **Handling of Gibberish Inputs**:
    - **Issue**: The DistilBERT model assigns sentiment to any input, including semi-coherent or gibberish text. This stems from the model's training on the SST-2 dataset, which assumes coherent English sentences (Socher et al., 2013).
    - **Mitigation**: The script enforces a minimum 60% meaningful word ratio using NLTK's word corpus (Bird et al., 2009) and filters repetitive or overly long unrecognized words. However, inputs with valid but contextually incoherent words may still pass validation.
    - **Reflection**: This limitation highlights the need for additional coherence checks, such as sentence structure analysis, which were explored but rolled back due to time constraints. Future work could integrate a language model to assess semantic coherence (Devlin et al., 2018).

2. **API Rate Limits**:
    - **Issue**: The free Inference API imposes rate limits (approximately 1,000 requests per day), restricting high-volume usage (Hugging Face, 2025b).
    - **Mitigation**: The script handles HTTP 429 errors and displays remaining calls, but users must monitor usage or consider paid options for larger datasets.
    - **Reflection**: For small-scale applications, the free tier is adequate, but production environments may require local model deployment using Hugging Face Transformers (Wolf et al., 2020).
3. **Binary Sentiment Classification**:
    - **Issue**: The DistilBERT model supports only binary (positive/negative) classification, limiting its ability to capture neutral or nuanced sentiments (Sanh et al., 2019).
    - **Mitigation**: The script focuses on clear positive or negative reviews, with validation ensuring meaningful inputs to reduce ambiguity.
    - **Reflection**: For applications requiring multi-class or aspect-based sentiment analysis, alternative models like RoBERTa or domain-specific fine-tuning would be necessary (Liu et al., 2019).
4. **External API Dependency**:
    - **Issue**: The script relies on the Hugging Face Inference API, requiring an internet connection and being subject to potential API downtime or changes.
    - **Mitigation**: Secure API key storage and comprehensive error handling mitigate some risks, but offline functionality is not supported.
    - **Reflection**: Local deployment of the model using Hugging Face Transformers could eliminate this dependency, though it requires additional computational resources (Wolf et al., 2020).
5. **Validation Strictness**:
    - **Issue**: The 60% meaningful word ratio may reject valid but informal inputs (e.g., reviews with slang or typos).
    - **Mitigation**: Configurable parameters in config.ini allow users to adjust validation thresholds.
    - **Reflection**: Balancing strict validation with flexibility is critical. A custom word list for domain-specific terms (e.g., restaurant slang) could improve robustness.

Conclusion

The *sentiment_analyzer.py* script is an accessible tool for sentiment analysis, by using the Hugging Face Inference API to provide actionable insights for restaurant reviews. Iterative development addressed challenges such as API response parsing, gibberish inputs, and configuration management, resulting in a modular and user-friendly application. While limitations like rate limits and binary classification exist, the script serves as an effective educational and practical tool for NLP applications. Future enhancements could include batch processing, visualization, or local model deployment to overcome current constraints.

References

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python: Analyzing text with the Natural Language Toolkit*. O'Reilly Media. Retrieved from https://www.nltk.org/

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv*. https://arxiv.org/abs/1810.04805

Hugging Face. (2025a). DistilBERT base uncased finetuned SST-2. Retrieved from https://huggingface.co/distilbert/distilbert-base-uncased-finetuned-sst-2-english

Hugging Face. (2025b). Inference API providers: Hugging Face inference. Retrieved from https://huggingface.co/docs/inference-providers/providers/hf-inference

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A robustly optimized BERT pretraining approach. *arXiv*. https://arxiv.org/abs/1907.11692

Python Software Foundation. (2025). Configparser — Configuration file parser. Retrieved from https://docs.python.org/3/library/configparser.html

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *arXiv*. https://arxiv.org/abs/1910.01108

Saurabh, K. (2025). Python-dotenv: Read key-value pairs from a .env file. Retrieved from https://github.com/theskumar/python-dotenv

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Retrieved from https://nlp.stanford.edu/sentiment/

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., & Brew, J. (2020). Transformers: State-of-the-art natural language processing. *arXiv*. https://huggingface.co/docs/transformers/index

APPENDIX:

a. Quickstart manual: Lecturer may kindly access the github repository of this assignment to **clone the project to access the codebase and test the script**(https://github.com/luoxinyi2611/llm-text-analysis.git)

b. Python Script in raw text:

```
import requests
import os
import re
import configparser
from datetime import datetime
from dotenv import load_dotenv
import nltk
from nltk.corpus import words

# Download NLTK words dataset (run once)
nltk.download('words', quiet=True)
ENGLISH_WORDS = set(words.words())

# Load configuration from config.ini
config = configparser.ConfigParser()
config.read('config.ini')

# API configuration
API_URL = config['API']['url']
API_KEY_ENV = config['API']['key_env_variable']
OUTPUT_DIR = config['Output']['directory']
MIN_WORDS = int(config['InputValidation']['min_words'])
MAX_WORDS = int(config['InputValidation']['max_words'])

# Load API key from .env file
load_dotenv()
API_KEY = os.getenv(API_KEY_ENV)
if not API_KEY:
    raise ValueError(f"Hugging Face API key not found. Set
{API_KEY_ENV} in .env file.")
```

```
def validate_input(text):
    """Validate and clean user input."""
    text = re.sub(r'\s+', ' ', text.strip())  # Collapse multiple spaces
    text = re.sub(r'[.!?]+', '.', text)  # Normalize punctuation to single periods
    if not text:
        raise ValueError("Input cannot be empty.")

    words = text.split()
    if len(words) < MIN_WORDS:
        raise ValueError(f"Input too short. Please provide at least {MIN_WORDS} words.")
    if len(words) > MAX_WORDS:
        raise ValueError(f"Input too long. Please limit to {MAX_WORDS} words.")

    # Check for meaningful words
    meaningful_words = [word.lower() for word in words if word.lower() in ENGLISH_WORDS]
    meaningful_ratio = len(meaningful_words) / len(words) if words else 0
    if meaningful_ratio < 0.6:  # Increased from 0.5 to 0.8
        raise ValueError("Input contains too many unrecognized words. Please provide meaningful text.")

    # Check for repetitive or gibberish words
    for word in words:
        if len(word) > 10 and len(set(word.lower())) < 4:
            raise ValueError(f"Input contains repetitive gibberish: '{word}'.")
        if len(word) > 15 and not word.lower() in ENGLISH_WORDS:
            raise ValueError(f"Input contains likely gibberish: '{word}'.")
```

```
lvi.
lvii.       return text
lviii.
lix.    def get_user_input():
lx.         """Get text input from user via console."""
lxi.        print("Enter your restaurant comment (press Enter twice to
        finish):")
lxii.       lines = []
lxiii.      while True:
lxiv.           line = input()
lxv.            if line == "":
lxvi.               break
lxvii.          lines.append(line)
lxviii.
lxix.       text = " ".join(lines)
lxx.        return validate_input(text)
lxxi.
lxxii.  def call_huggingface_api(text):
lxxiii.     """Send text to Hugging Face API and return result with
        rate limit info."""
lxxiv.      headers = {"Authorization": f"Bearer {API_KEY}"}
lxxv.       payload = {"inputs": text}
lxxvi.
lxxvii.     try:
lxxviii.        response = requests.post(API_URL, headers=headers,
        json=payload)
lxxix.          response.raise_for_status()
lxxx.           result = response.json()
lxxxi.          remaining =
        response.headers.get("X-RateLimit-Remaining", "Unknown")
lxxxii.         return result, remaining
lxxxiii.    except requests.exceptions.HTTPError as e:
lxxxiv.         if response.status_code == 429:
lxxxv.              raise Exception("API rate limit reached. Please
        try again later.")
lxxxvi.         raise Exception(f"API request failed: {str(e)}")
```

```
        except requests.exceptions.RequestException as e:
            raise Exception(f"Network error: {str(e)}")

def process_response(response):
    """Parse API response for sentiment analysis."""
    if isinstance(response, list) and len(response) > 0 and isinstance(response[0], list):
        predictions = response[0]
        if not predictions:
            raise ValueError("Empty predictions in API response.")
        best_prediction = max(predictions, key=lambda x: x.get("score", 0))
        label = best_prediction.get("label")
        score = best_prediction.get("score")
        if label and score:
            return f"Sentiment: {label}, Confidence: {score:.2%}"
    elif isinstance(response, dict):
        if "error" in response:
            raise ValueError(f"API error: {response['error']}")
        raise ValueError(f"Unexpected response format: {response}")
    raise ValueError(f"Unexpected API response format: {response}")

def save_results(text, result):
    """Save input and result to a text file in results directory."""
    os.makedirs(OUTPUT_DIR, exist_ok=True)
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = os.path.join(OUTPUT_DIR, f"results_{timestamp}.txt")

    with open(filename, "w", encoding="utf-8") as f:
```

```
            f.write("Task: Sentiment Analysis\n")
            f.write(f"Input Text:\n{text}\n\n")
            f.write(f"Result:\n{result}\n")

        return filename

    def main():
        """Main function to run the sentiment analysis tool."""
        try:
            text = get_user_input()
            response, remaining_calls = call_huggingface_api(text)
            result = process_response(response)
            print("\nResult:")
            print(result)
            print(f"API calls remaining: {remaining_calls}")

            save_choice = input("\nSave result to file? (y/n):
    ").strip().lower()
            if save_choice == "y":
                filename = save_results(text, result)
                print(f"Results saved to {filename}")

        except ValueError as e:
            print(f"Input error: {str(e)}")
        except Exception as e:
            print(f"Error: {str(e)}")

    if __name__ == "__main__":
        main()
```