Offensive Language and Sentiment Analysis for User Comments

Table of Contents

1. Background . . . 1
2. Requirements . . . 1
3. Method . . . 2
4. Implementation . . . 2
5. Milestones . . . 3
6. Gathering Results . . . 3

1. Background

This project aims to implement a system for analyzing user comments, focusing on sentiment analysis, offensive language detection, and categorization. The objective is to provide a nuanced classification of comments, identifying offensive content while refining overall sentiment scoring to allow for better moderation and filtering of inappropriate or harmful messages.

The initial implementation used a basic sentiment analysis model and iteratively introduced various offensive language detection techniques. These enhancements aimed to provide contextually aware categorizations and accommodate varying degrees of offensiveness.

2. Requirements

- **Must Have**:
  - Load and preprocess user comments in a standardized format.
  - Perform sentiment analysis on user comments using a pre-trained model.
  - Detect and classify comments containing offensive language.
  - Implement a weighted sentiment adjustment for comments flagged as offensive.
- **Should Have**:
  - Handle different comment lengths, ensuring compatibility with model input size constraints.
  - Provide granular categorization for comments based on offensive word density and patterns.
  - Maintain a modular design for easy modification and integration of additional features.
- **Could Have**:
  - Support multi-language sentiment and offensive language analysis.

    - Use context-aware models to further enhance classification accuracy.
    - Implement a sarcasm detection mechanism for refining sentiment scores.

3. Method

The project is divided into two primary phases:

1. **Sentiment Analysis Integration**: The `distilbert-base-uncased-finetuned-sst-2-english` model was used for initial sentiment classification. Key steps include:
    - Environment setup and import of necessary libraries (`transformers`, `torch`, `pandas`).
    - Data loading and preprocessing with error handling for malformed entries.
    - Batch processing for efficient handling of large datasets.
2. **Offensive Language Detection and Weight Adjustment**: Offensive language detection is performed in multiple stages, focusing on:
    - Flagging highly offensive words using a predefined list.
    - Implementing word density-based weighting adjustments.
    - Fine-tuning offensive categories with context-based severity modifiers.

4. Implementation

1. **Data Preparation and Setup (Cells 1-2)**:
    - **Environment Setup and Imports**: Initialize libraries and define helper functions (`refine_labels`).
    - **Dataset Loading**: Handle file errors and verify the presence of required columns.
    - Extract comments from the dataset and convert them to a standardized string format for analysis.
2. **Sentiment Analysis (Cell 3)**:
    - Use the pre-trained `distilbert-base-uncased-finetuned-sst-2-english` model.
    - Process comments in batches to optimize memory usage.
    - Store the results in a structured format (`df_sentiments` DataFrame) for further analysis.
3. **Sentiment Distribution Analysis (Cell 4)**:
    - Analyze the distribution of positive, negative, and neutral sentiments.
    - Calculate average sentiment scores for each category.
4. **Extreme Sentiment Extraction (Cell 5)**:
    - Identify and display the top 5 positive and negative comments based on sentiment scores.
5. **Offensive Language Detection (Cell 6)**:
    - Implement a word frequency analysis using regex-based tokenization.
    - Identify the top 50 most common words for reference.

6. **Flagging Highly Offensive Comments (Cell 7)**:
    - Create a custom list for words of interest (e.g., racial slurs).
    - Add an `Offensive_Flag` column to flag comments with highly offensive terms.
    - Update the `Sentiment Label` for flagged comments to `Highly Offensive`.
7. **Sentiment Label Adjustment (Cell 8)**:
    - Modify sentiment labels for flagged comments.
    - Adjust sentiment scores to indicate the intensity of offensive language.
8. **Summary Report (Cell 9)**:
    - Provide a text-based summary of sentiment distribution and offensive language identification.
    - Highlight key insights and findings from the sample analysis.

5. Milestones

1. **Milestone 1: Environment Setup and Data Loading**:
    - Complete the environment configuration.
    - Verify dataset integrity and perform initial extraction.
2. **Milestone 2: Sentiment Analysis Integration**:
    - Implement sentiment analysis using `transformers` pipelines.
    - Validate the model output for a sample of comments.
3. **Milestone 3: Offensive Language Detection**:
    - Develop and test offensive language detection using predefined word lists.
    - Implement and validate the offensive language flagging mechanism.
4. **Milestone 4: Sentiment Label Refinement and Weight Adjustments**:
    - Refine labels based on detected offensive language.
    - Finalize scoring mechanisms and evaluate output consistency.

6. Gathering Results

The evaluation phase includes running the detection on a comprehensive set of sample comments and validating the output accuracy. Metrics for evaluation include:

- **Accuracy of Sentiment Classification**: Measure the consistency between predicted and actual sentiment.
- **Precision of Offensive Language Detection**: Evaluate the percentage of correctly flagged comments.
- **Weighted Sentiment Score Impact**: Analyze how offensive flags influence overall sentiment scoring.

The refined model demonstrates a balanced approach to comment classification, accounting for both sentiment intensity and the presence of harmful language. Future enhancements could include context-aware analysis and support for a broader range of languages.