LLM-Based Workflow for Text Classification and Translation

David (Taekyu) Kang

March 9th 2025

Overview and Approach

This project implements an end-to-end pipeline for automated sentiment and toxicity detection on English text datasets. The primary goal of this project is to leverage large-language models (LLM) to classify and explain sentiment and toxicity in text. Future scope of this project include de-toxifying sentences and translating it back into the original language.

The overall architecture of the system is modular and follows fundamental Object-Oriented-Programming styles. This is noticeable in the way the different Python modules are interconnected and how the file structure helps keep data and code organized and modular.

The system is largely divided into:

1. Data ingestion and preparation
2. Machine Translation to convert text to English
3. Classification and Explanation of Sentiment Analysis and Toxicity Analysis and Reporting Results
4. Evaluation of LLM's Performance and Reporting Results

Pipeline

The overall project directory is organized into 5 main files (see below):

```
src/
├── __pycache__/
├── analysis/
│   └── analyze.py
├── data/
│   ├── input/
│   └── output/
├── evaluation/
│   └── base_evaluation.py
├── preprocessing/
│   └── translate.py
└── main.py
```

The main.py file is the root file that gets called to initiate the preprocessing -> analysis -> evaluation sequences on the dataset.

The /preprocessing folder currently contains 1 python module that is responsible for translating the input text into English. In the future, additional preprocessing steps can be added to fit the expanding needs of the project. The /analysis folder contains the related codes that pertain to analyzing the dataset and classifying text into the different sentiment and toxic categories. The /data folder contains both the /input (testing data) as well as the /output files (predicted/output data). The /analysis and /evaluation folders contain the python modules that handle analyzing the given data and evaluating the performance of the models.

Pipeline Components

- analyze_and_classify() (in analyze.py): This function handles both sentiment and toxicity predictions in a unified call. It processes the input data and returns prediction results with explanations.
- evaluate() (in base_evaluation.py): Conducts task-specific evaluation based on the returned predictions, supporting different metrics depending on whether the task is sentiment or toxicity detection.

Task Integration

Both tasks (sentiment and toxicity detection) are integrated into a single analysis pipeline. The main.py file specifies two DataFrame containers—one for sentiment and one for toxicity predictions. The script runs run_analysis_and_evaluation() twice:

- Once for the sentiment dataset.
- Once for the toxicity dataset.

By designing analyze_and_classify() to return a combined result, we allow for flexible task expansion in the future, such as adding emotion classification or hate speech detection.

Implementation Details

Key Libraries

- pandas: Data ingestion and manipulation.
- scikit-learn / TensorFlow / OpenAI API (presumed, pending confirmation from analyze.py): Likely used for classification tasks, depending on whether the system uses traditional ML or LLM APIs. I initially used OpenAI API as its performance was surprior but switched to a pretrained model from Hugging Face (Granite)
- Custom evaluation metrics (in base_evaluation.py): Implemented to assess model predictions in terms of accuracy, precision, recall, or task-specific metrics.

Environment Setup

- Python version: 3.9+
- Package management: pip or conda
- Required packages (requirements.txt). This file was autogenerated using pipreqs to identify missing packages from the project and automatically download them.

Evaluation Methods and Preliminary Results

The base_evaluation.py contains evaluate() function which currently supports two evaluation streams:

- Sentiment Evaluation
- Toxicity Evaluation

This module calculates various evaluation metrics including accuracy, precision, recall, and F1. The full results can be found under data/output/*_evaluation.txt but some of the key results for sentiment analysis are:

- Accuracy: 0.86 (i.e. 86% of all predictions were correct)
- F1 Score: 0.63 (F1 score tells us the harmonic mean of precision and balance)

Given the high Accuracy but mediocre F1 score, there may be some bias toward the majority class while struggling with minority cases.

On the other hand, results for toxicity analysis reached 100% accuracy. This appears to show that our model performs exceptionally well at determining toxic vs non-toxic text. However, it's important to keep in mind that our sample size is quite small and we may be susceptible to overfitting the model.

Challenges and Limitations

Challenges and Limitations:

- We currently have a very small sample size to work with when analyzing and evaluating the model's performance. We'll need to increase the dataset in the future to better evaluate the performance.
- The scalability of these models (especially throughput) depends heavily on the quantity and quality of the GPU. On average, to run the entire sequence of analysis and evaluation on my local M1 Macbook using the integrated GPU, it took 30-45 minutes.
- The LLM I used is relatively small. To substantially improve the performance (especially for sentiment), it may be necessary to use multiple models and/or use larger models to analyze the data.
- Currently, we're only evaluating English datasets and to handle non-English text, we translate the original text to English first, then perform analysis. This is undoubtably a huge limitation as capturing the small nuances and language-specific semantics is challenging when translating. In the future, it will be best to test the model's performance on multiple other languages as-is (no finetuning) to determine its multilingual capabilities.