ActiveFence — Take-Home Assignment: Feature Engineering for Text Classification

Report by Ohad Cohen

2 July 2025

1 Introduction

This report presents the submission for the Feature Engineering for Text Classification task. The objective of this assignment was to build a toxicity classifier for comments using only classical machine learning and manual feature engineering.

The goal was to achieve strong performance while keeping the model transparent and interpretable without relying on prebuilt classifiers or deep learning libraries. The model was trained on the provided dataset using a NVIDIA A100 GPU for training.

2 Workflow

2.1 Data Loading

I started by loading the provided dataset (*toxicity_toy_dataset.csv*), which included 497 labeled comments indicating whether each comment was toxic (1) or neutral/respectful (0).

2.2 Raw Text Pre-processing

I implemented basic text cleaning steps, including removing punctuation to standardize the text and converting all words to lowercase to merge variants. Additionally, I removed stop words to reduce noise and applied stemming to normalize inflections (e.g., stealing → steal). However, I observed that these two additional approaches did not lead to any improvement in accuracy.

2.3 Feature Engineering

I combined custom numeric features and text features. The numeric features are:

- Comment length (characters and words).

- Number of exclamation and question marks.
- Proportion of uppercase letters.
- Emoji count (using a Unicode regex).

Swear word count did not show any improvement. The numeric features were standardized with *StandardScaler*. For text features, I used TF-IDF representations to extract unigrams (single words) and bigrams (pairs of consecutive words). Rare words that occurred only once were ignored to reduce noise (since rare words don't help generalization) and to limit the size of the feature space. In addition, I used character-level features to capture patterns and detect bad words. This approach helped identify misspellings, partial words, and obfuscated swear words.

2.4 Model Selection

I selected Logistic Regression as the classifier because it is easy to interpret, allowing direct examination of the learned coefficients. It also works efficiently with sparse feature representations like TF-IDF and character n-grams, and it provides strong performance when the relationship between features and labels is approximately linear. The maximum number of iterations taken the model to converge was set to 1000.

2.5 Model Training and Evaluation

I trained the model on an 80%-20% stratified train-test split and evaluated using accuracy as the metric score and the classification report. I also performed 5-fold cross-validation to validate robustness.

3 Results

The Logistic Regression classifier achieved strong overall performance. On the test set, the model reached an accuracy of 95%. To further assess robustness, I performed 5-fold cross-validation across the entire dataset, which produced a mean cross-validation accuracy score of 95.96%.

In addition, Fig. 1 presents the confusion matrix for the model. The confusion matrix compares the actual target and predicted labels, showing the percentage of correct and incorrect predictions for each class. Beyond measuring accuracy, it also reveals the distribution of errors across the two classes, helping to identify specific misclassifications.

Although the overall accuracy was high, there were a few misclassified examples that illustrate some of the limitations of the approach. For instance, the comment **"Hard disagree, respectfully not really"** was a true toxic example predicted as neutral. In contrast, several clearly neutral comments were incorrectly classified as toxic such as **"Your restraint is admirable"**. More misclassified examples are presented in the notebook.

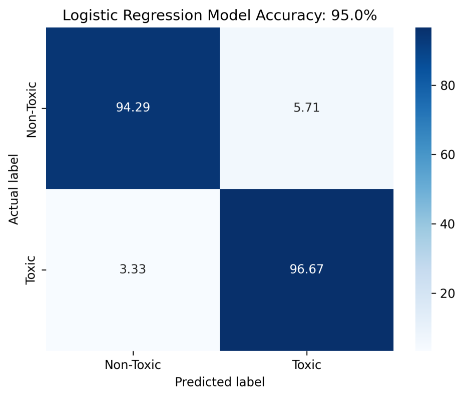

Figure 1: Confusion matrix of the results of Logistic Regression classifier

Overall, these results demonstrate that combining TF-IDF features, character n-grams, and handcrafted numeric features provided a robust foundation for accurate toxicity classification using classical machine learning.

3.1 Feature Importance Analysis

I examined the model's feature importance by analyzing the coefficients learned by the Logistic Regression classifier. A positive coefficient increases the likelihood that a comment will be classified as toxic, while a negative coefficient reduces that likelihood. The magnitude of the coefficient reflects the strength of the feature's influence on the prediction. Fig. 2 shows the top ten fea-

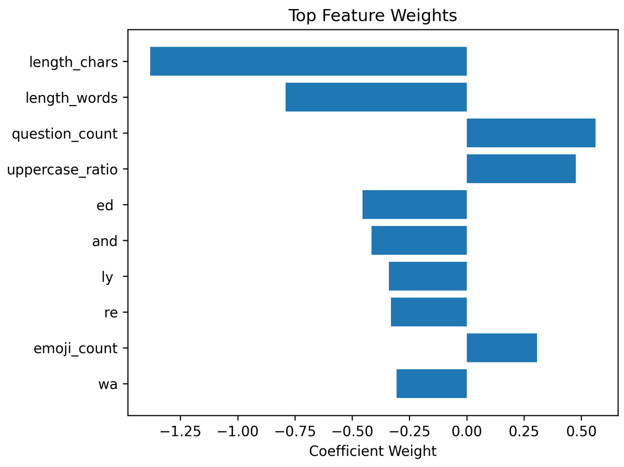

Figure 2: Top ten feature importance of the Logistic Regression classifier

tures ranked by absolute coefficient weight. The most influential feature was *length_chars*, which had a strong negative coefficient, indicating that longer comments were more likely to be classified as neutral. *length_words* followed the same pattern. On the other hand, *question_count* and *uppercase_ratio* had positive coefficients, suggesting that comments with more question marks or higher use of uppercase letters were more likely to be toxic.

The feature *emoji_count* also had a positive coefficient. This implies that comments containing more emojis tended to be associated with toxic labels. This may reflect sarcastic or mocking use of emojis, which are commonly used in toxic or dismissive remarks.

The remaining features (*ed*, *and*, *ly*, *re*, and *wa*) had relatively small weights, indicating limited but measurable influence on the prediction. These are likely n-gram fragments that appeared frequently enough to contribute to the model but were not individually strong indicators of class.

4 Summery

4.1 Methods Considered

In addition to the approach I implemented, I thought of three alternative models. Note that these were only ways of thinking and not actual implementations. Naive Bayes is fast and works well for text, but can be too simplistic to capture more nuanced patterns. Random Forest handles non-linear relationships but is less efficient with high-dimensional sparse data. Support Vector Machines perform well but are slow and resource-intensive on large feature sets. Finally, I chose Logistic Regression for its speed, interpretability and high performance. For the input features, Bag of Words was another approach, but I decided against it because it does not capture word order or context.

4.2 Challenges Faced

I faced several challenges. One was handling emojis. To deal with them, I had to create custom regex rules to clean and count them. Another challenge was choosing the best way to represent the text for the model. I tried different options, such as using only unigrams, adding bigrams, and including character n-grams to capture partial or hidden words. Additionally, I found that relying solely on numeric features was not sufficient for achieving good performance.

4.3 Strengths of the Approach

The model is highly interpretable, with all features being human-readable, and demonstrated robust accuracy performance of 95% on the test set and a mean cross-validation accuracy score of 95.96%. The pipeline is lightweight, requiring no dependency on deep learning frameworks, and remains flexible, making it easy to adapt to new data or different toxicity criteria.

4.4 Weaknesses of the Approach

The approach has limited handling of context, as simple n-grams do not fully capture nuances such as sarcasm or implied toxicity. It is also sensitive to vocabulary, meaning rare or creative word forms may be ignored or misclassified. Additionally, it does not use semantic embeddings, so unlike modern methods such as BERT, it cannot understand synonyms or semantics sentences.

4.5 What I Would Do with More Time

If I had more time, I would have used an existing tokenizer for pre-processing the text more effectively and explored deep learning approaches. For example, I could have fine-tuned a pre-trained DistilBERT model, which is designed to capture rich semantic relationships between words and sentences. This strategy might have led to significantly higher performance, particularly in understanding context, detecting subtle forms of toxicity, and handling synonyms more accurately.

4.6 How I Used AI Tools

I used AI tool like ChatGPT to review errors and refine sample code snippets, including generating regular expressions for handling emoji Unicode and text cleaning processes. Examples of prompts that I used: ”how to pre-processes emoji?”, ”Emoji detection regex in python”, ”I thought to use Bag of Words. Do you think this is a good approach?”