ITNPAI3: AI for NLP

Sentiment Analysis of Twitter Data

A Comparative Evaluation of Naive Bayes and Logistic Regression across Bag-of-Words and TF-IDF Representations

Student ID: 3539054

February 26, 2026

1 Data Preprocessing

Raw tweets contain significant noise and stylistic variations that can confuse machine learning algorithms. The first preprocessing step was converting all text to lowercase. While this removes the emotional intensity often conveyed through capital letters (e.g., "HATE" versus "hate"), it was a necessary trade-off for statistical models. Without lowercasing, models using Bag-of-Words or TF-IDF treat "Bad", "bad", and "BAD" as three entirely separate mathematical features. Lowercasing consolidates these variations, preventing feature fragmentation and preserving the vocabulary limit.

Next, regular expressions (Regex) were used to remove URLs, user mentions, and hashtags, as these rarely contribute to the underlying sentiment. However, because social media users frequently rely on punctuation to express strong feelings, multiple exclamation marks or asterisks (often used to mask profanity) were explicitly translated into readable text tokens (e.g., `exclamationmark`) before the remaining punctuation was stripped away.

```
# Convert to lowercase
text = text.lower()

# Remove platform-specific noise (URLs, user mentions, hashtag symbols)
text = re.sub(r"http\S+|www\S+|https\S+", '', text, flags=re.MULTILINE)
text = re.sub(r'\@\w+', '', text)
text = re.sub(r'\#', '', text)

# Preserve sentiment-heavy characters by converting them to text tokens
text = re.sub(r'\*{2,}', ' censoredswear ', text)
text = re.sub(r'!+', ' exclamationmark ', text)
text = re.sub(r'\?+', ' questionmark ', text)
```

Code Snippet 1: Regex logic for noise removal and punctuation preservation

Finally, the `spaCy` natural language processing library was utilised to reduce words to their base dictionary form (lemmatisation) and remove common "stop words" to reduce data sparsity. Crucially, sentiment-altering words such as "not", "no", and "never" were retained. A custom negation handler was implemented to link these negations directly to the subsequent word. For instance, the phrase "not good" was transformed into `not_good`. This ensures the model treats the negated phrase as a distinct, negative feature rather than misinterpreting "good" as a purely positive signal

```
# Flag the subsequent token for negation prefixing
if lemma in ["not", "no", "never", "cannot"]:
    negate_next = True
    # Append immediately to preserve if it is the final token
    tokens.append(lemma)
    continue

# Apply negation prefix by combining with the previous negation term
if negate_next:
    if tokens:  # Safety check to ensure the token list is not empty
        prev_negation = tokens.pop()
        lemma = f"{prev_negation}_{lemma}"
    negate_next = False
```

Code Snippet 2: Custom negation handler linking modifiers to subsequent words

Raw Tweet	**Preprocessed Tweet**
Sooo SAD I will miss you here in San Diego!!!	sooo sad miss san diego exclamationmark
Sons of ****, why couldn‘t they put them on the releases we already bought	son censoredswear release buy
you can ride one, you can catch one, but its not summer til you pop open one	ride catch but not_summer til pop open

Table 1: Examples of raw text transformed by the custom preprocessing and negation pipeline.

2 Feature Representation

Because machine learning algorithms cannot process raw text, the preprocessed tweets had to be converted into numerical formats. This project evaluated two different mathematical representations: Bag-of-Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF).

The Bag-of-Words approach converts text into a simple matrix where each column represents a unique word, and the values represent how many times that word appears in a specific tweet. While this method is straightforward, it treats all words as equally important. In human language, this is rarely true; highly frequent but emotionally neutral words can mathematically overshadow rare but highly emotional words.

To address this limitation, the TF-IDF method was also evaluated. TF-IDF does not just count how often a word appears in a single tweet; it scales that count based on how often the word appears across the entire dataset. If a word is extremely common everywhere, its mathematical weight is penalised. If a word is rare overall but appears in a specific tweet, its weight is heavily boosted. Theoretically, TF-IDF is better suited for capturing sentiment because human emotion is often conveyed through distinct, specialised words rather than common vocabulary.

For both methods, the feature matrix was strictly limited to the top 10,000 most frequent words. This limit prevents the dataset from becoming overwhelmed by rare misspellings, reducing noise and preventing the models from overfitting. How these two representations ultimately interact with the specific learning mechanics of Naive Bayes and Logistic Regression is detailed in the following section.

3 Model Training & Evaluation

With the text converted into numerical features, two statistical classifiers were trained and evaluated: Multinomial Naive Bayes and Logistic Regression.

Naive Bayes is a probabilistic model that relies on the frequency of events. As a result, it performed significantly better when paired with the Bag-of-Words representation. Bag-of-Words provides explicit, raw integer counts of how many times a word appears, which perfectly aligns with the mathematical assumptions of Naive Bayes. When paired with TF-IDF, the fractional, scaled weights distorted these probability calculations, leading to a noticeable drop in accuracy.

Contrarily, Logistic Regression is a discriminative model that learns to draw a boundary between classes by assigning continuous weights to each feature. To ensure the model performed optimally on a sparse matrix of 10,000 features, its inverse regularisation strength was tuned to `C=10.0`, giving it enough mathematical flexibility to learn complex patterns without underfitting. Because Logistic Regression excels at balancing feature weights, it thrived on the TF-IDF representation. The TF-IDF scaling provided a normalised, rich feature space where highly emotional, rare words were mathematically emphasised.

As demonstrated in Table 2, the Logistic Regression model paired with TF-IDF vectorisation emerged as the champion model. It successfully leveraged the penalised feature weights to achieve the highest scores across all core classification metrics, peaking at an overall accuracy of 0.692.

Model & Representation	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Naive Bayes (BoW)	0.686	0.691	0.686	0.687
Naive Bayes (TF-IDF)	0.658	0.687	0.658	0.657
Logistic Regression (BoW, C=10.0)	0.687	0.691	0.687	0.688
Logistic Regression (TF-IDF, C=10.0)	**0.692**	**0.697**	**0.692**	**0.694**

Table 2: Performance metrics for all model and feature representation combinations.

4 Error Analysis

While the Logistic Regression (TF-IDF) model achieved the highest overall accuracy, analysing its misclassifications provides critical insight into the limitations of statistical text processing. Figure 1 displays the confusion matrix, revealing that the model is highly effective at identifying "Neutral" text (995 correct predictions). However, its primary failure mode is being overly conservative; it frequently misclassifies "Positive" and "Negative" tweets as "Neutral" (275 and 288 errors, respectively) because it struggles to confidently detect sentiment without explicit emotional keywords.

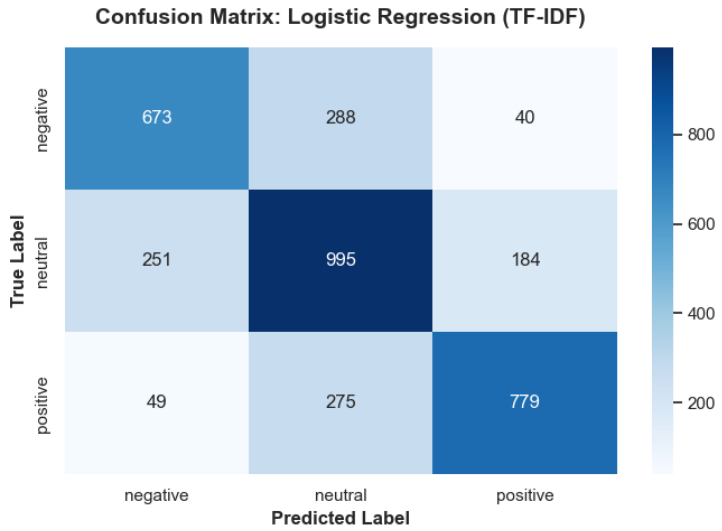

Figure 1: Confusion Matrix for Logistic Regression (TF-IDF, C=10.0)

To understand these boundaries, a qualitative error analysis was conducted on specific misclassified tweets (Table 3):

Raw Tweet (Misclassified Examples)	**True**	**Predicted**
The underwire in my bra is sticking out and poking me in the armpit	Negative	Neutral
I want to see David cook!!	Positive	Neutral
.. and you‘re on twitter! Did the tavern bore you that much?	Neutral	Negative
was so excited to eat the wartermelon i bought the other day and it was terrible and not sweet	Neutral	Negative

Table 3: Selected examples demonstrating classic NLP failure modes.

1. Implicit Sentiment & Context (False Neutrals): Statistical models rely heavily on explicit emotional vocabulary. In the first example, a "poking underwire" is frustrating, but lacking mathematically strong negative words like "hate" or "bad", the model defaults to "Neutral". Similarly, in the second example, the model fails to recognise the positive excitement conveyed by the slang "cook", interpreting the text literally rather than capturing the user's enthusiasm.

2. Sarcasm: The third example uses the word "bore", which carries a strong negative weight in the training data. However, the conversational, slightly sarcastic context is entirely "Neutral". The model processes words in isolation and easily falls victim to these contextual traps.

3. Flawed Dataset Labels: The final example highlights a common issue when datasets are labelled automatically. The user clearly states that the watermelon was "terrible" and "not sweet", yet the dataset's True Label is "Neutral". The model correctly predicted "Negative", which shows that some of the model's "errors" are actually just mistakes in the dataset itself.