A Text Preprocessing Pipeline for Airline Review Sentiment Analysis

Juan Carlos Miguel Timoteo

Computer Studies & Engineering

José Rizal University

Mandaluyong City, Philippines

Juancarlosmiguel.timoteo@my.jru.edu

***Abstract*—With the rapid growth of user-generated content on the web, sentiment analysis has become a key tool for organizations to understand public opinion. The quality of this analysis, however, is highly dependent on the quality of the input data. Raw text from online reviews is inherently noisy and requires systematic cleaning before it can be used for any meaningful Natural Language Processing (NLP) task. This paper presents a practical and detailed implementation of a text preprocessing pipeline applied to a dataset of 3,700 airline customer reviews. The pipeline addresses multiple forms of noise, including standard textual inconsistencies and domain-specific patterns such as flight durations and industry-specific codes. Key steps include lowercasing, custom pattern removal using regular expressions, punctuation removal, stopword removal using the standard NLTK library, and tokenization. The effectiveness of the pipeline is demonstrated through a quantitative analysis, which shows a significant reduction in data volume by 48.%, and a qualitative comparison of the text before and after cleaning. This work underscores the critical role of tailored preprocessing in preparing unstructured text data for reliable NLP applications.**

Keywords—text preprocessing, natural language processing, sentiment analysis, data cleaning, text mining, stopword removal.

I. INTRODUCTION

In the contemporary digital era, the proliferation of user-generated content across online platforms has created vast repositories of textual data holding immense value for organizations [1], [2]. A key technique for unlocking insights within this data is sentiment analysis, a subfield of Natural Language Processing (NLP) concerned with the automated extraction and classification of opinions expressed in text. However, the raw, unstructured nature of user-generated text presents significant challenges for direct computational analysis [3].

Raw text data is inherently noisy, characterized by grammatical inconsistencies, colloquialisms, irregular capitalization, and the presence of non-semantic elements such as punctuation, hyperlinks, and symbols [4]. This noise can severely degrade the performance of machine learning models and other analytical tools, making robust data preparation essential. Thus, a systematic text preprocessing pipeline is not merely a preliminary step but a foundational requirement for any successful text mining or NLP application [5].

This paper presents a practical application and detailed documentation of a text preprocessing pipeline applied to a real-world dataset comprising 3,700 customer reviews of a major airline. The primary objective of this work is to implement a sequence of standard preprocessing techniques—including lowercasing, custom noise removal, stopword removal, and tokenization—to systematically clean the dataset. The effectiveness of this pipeline is evaluated by quantifying data volume reduction and through a qualitative inspection of raw versus cleaned text.

The remainder of this paper is structured as follows: Section II reviews related work. Section III outlines the methodology. Section IV presents results. Section V offers a discussion, and Section VI concludes with future research directions.

II. RELATED WORK

A. Normalization Techniques

Normalization, including lowercasing and punctuation removal, is among the most established preprocessing techniques. Lowercasing prevents the model from treating "Flight" and "flight" as distinct tokens, a necessary standardization particularly in informal text [6]. Removing punctuation and special characters helps reduce the feature space and eliminate noise [7].

B. Stopword Removal

Stopwords are common words that provide grammatical structure but limited semantic meaning. Removing them reduces dimensionality and enhances model focus on content-rich terms. While many pipelines utilize standard lists such as those from NLTK, domain-specific stopword lists can further improve performance [8], [9].

C. Tokenization

Stopwords are common words that provide grammatical structure but limited semantic meaning. Removing them reduces dimensionality and enhances model focus on content-rich terms. While many pipelines utilize standard lists such as those from NLTK, domain-specific stopword lists can further improve performance [8], [9].

III. METHODOLOGY

A. Dataset Description

The dataset comprises 3,700 reviews of British Airways, sourced from Kaggle. The structured part includes metadata like traveler type and route; however, our focus is the ReviewBody column containing free-form text.

B. Noise Identification

Preliminary analysis revealed multiple noise types

- Standard Noise: Capitalization, punctuation, and stopwords.
- Domain-Specific Noise:
  - **Time expressions**: e.g., "4h", "2 hours".
  - **Currency and fees**: e.g., "$500", "£100".
  - **Codes**: Airport and aircraft codes like "LHR", "A380".
  - **Encoding issues**: e.g., "â€™" for apostrophes.

C. Preprocessing Pipeline

The preprocessing pipeline implemented in this study was developed in Python using the pandas library for data manipulation and the NLTK (Natural Language Toolkit) library for fundamental NLP tasks. The design of the pipeline follows widely accepted preprocessing strategies in the field [1]–[3], but it also incorporates domain-specific customizations to address the unique challenges of airline review text.

The pipeline consists of five sequential stages:

1. Lowercasing and Encoding Correction

The first step involves converting all text to lowercase. This is a common normalization technique that reduces the dimensionality of the feature space by ensuring that tokens like "Flight", "flight", and "FLIGHT" are treated identically [4]. This is particularly useful when processing informal, user-generated text, where capitalization is often inconsistent and non-semantic.

In addition, this stage includes the correction of character encoding artifacts—non-standard characters that result from text scraping, incorrect encoding formats, or copy-paste operations. For instance, typographic symbols such as â€™ may appear in place of standard apostrophes due to improper encoding. These anomalies were corrected using targeted replacement functions and the ftfy (Fixes Text For You) Python library, which is specifically designed to resolve common Unicode errors in text data [5].

2. Custom Noise Removal

```
[ ]  # Define a single, powerful function to remove multiple categories of custom noise
     def remove_custom_noise(text):
         # Remove URLs (which are a common form of noise in user-generated content)
         text = re.sub(r'https?://\S+|www\.\S+', '', text)

         # Remove dates (e.g., 24th october 2023, 15 july 2023)
         text = re.sub(r'\d{1,2}(st|nd|rd|th)?\s+\w+\s+\d{4}', '', text)

         # Remove times and durations (e.g., 4h, 2 hours, 45 mins)
         text = re.sub(r'\b\d+\s?(h|hr|hrs|hour|hours|minute|minutes|min|mins)\b', '', text)

         # Remove prices and currencies (e.g., £100, $500, 75 pounds)
         text = re.sub(r'[£$€]\d+(\.\d{1,2})?', '', text)
         text = re.sub(r'\b\d+\s?(pounds|gbp|aud|euro|eur)\b', '', text)

         # Remove common airport and aircraft codes
         text = re.sub(r'\b(lhr|jfk|cpt|man|sjc|mxp|gatwick)\b', '', text) # Airports
         text = re.sub(r'\b([a-z]\d{2,3}|[b]?\d{3})\b', '', text)       # Aircraft (a380, b777, 787)
```

Figure. 3.

- Temporal references: e.g., "4h", "45 mins", "2 hours".
- Monetary amounts: e.g., "£200", "$500"
- Airline-specific identifiers: airport codes (e.g., "LHR", "JFK") and aircraft models (e.g., "A380", "B777").
- URLs and email addresses: irrelevant for sentiment analysis but common in user-generated reviews.
- HTML/XML artifacts: e.g., escaped characters or tag remnants.

These elements are not only semantically irrelevant for sentiment classification, but they also increase vocabulary size and introduce noise into the analysis. By using comprehensive regular expressions, these patterns were filtered out, following approaches validated in recent domain-specific preprocessing studies [6], [7].

3. Special Character Removal

```
# Define a function to remove all characters that are not letters, numbers, or whitespace
def remove_special_characters(text):
  # The regex [^a-zA-Z0-9\s] matches any character that is NOT a letter, number, or space
  return re.sub(r'[^a-zA-Z0-9\s]', '', text)

# Apply the function
df['Cleaned_Review'] = df['Cleaned_Review'].apply(remove_special_characters)

# Display the head of the DataFrame to see the result
print("--- DATAFRAME PREVIEW: AFTER SPECIAL CHARACTER REMOVAL ---")
df[['ReviewBody', 'Cleaned_Review']].head()
```

Figure. 1

Special characters, punctuation, and symbols (e.g., "!", "@", "#") were removed unless they contributed meaningful context to the review. While punctuation can sometimes carry sentiment (e.g., exclamation marks), its interpretability is ambiguous without a deeper sentiment model. Therefore, for simplicity and consistency, non-alphanumeric characters were excluded using pattern-based filtering. This decision aligns with prior work that emphasizes minimizing syntactic noise in sentiment corpora [8].

4. Stopword Removal

```python
import nltk
from nltk.corpus import stopwords

# Download the stopwords resource from NLTK
nltk.download('stopwords')

# Load the standard English stopwords directly from NLTK into a set for efficient
stop_words = set(stopwords.words('english'))

print(f"Using standard NLTK stopword list with {len(stop_words)} words.")

# Define the removal function
def remove_stopwords(text):
  words = text.split()
  # Filter out any word that is in our stop_words set
  filtered_words = [word for word in words if word not in stop_words]
  return " ".join(filtered_words)

# Apply the stopword removal function
df['Cleaned_Review'] = df['Cleaned_Review'].apply(remove_stopwords)

# Display the head of the DataFrame to see the result
print("\n--- DATAFRAME PREVIEW: AFTER STANDARD STOPWORD REMOVAL ---")
df[['ReviewBody', 'Cleaned_Review']].head()
```

Figure. 2

Stopwords—commonly used words that do not carry significant semantic weight—were removed using the standard English stopword list provided by the NLTK library. Examples include “the”, “is”, “and”, “but”, etc. These words are useful for grammatical structure but contribute little to the classification of sentiment. Removing them reduces dimensionality and improves model efficiency [9].

While domain-specific stopword lists can offer enhanced performance by removing frequently occurring but sentiment-neutral terms (e.g., “flight”, “airline”, “ba”), only the general-purpose NLTK stopword list was used in this initial implementation to maintain reproducibility. However, future enhancements could integrate domain-adapted stopword filtering, as advocated in [10], [11].

5. Tokenization

```python
from nltk.tokenize import word_tokenize

# Download both the standard 'punkt' model and the required 'punkt_tab' add-on
nltk.download('punkt')
nltk.download('punkt_tab')

# Apply the word_tokenize function to create the new column
# This will now work because punkt_tab is available
df['Tokenized_Review'] = df['Cleaned_Review'].apply(word_tokenize)

# Display the final tokenized output alongside the original review
print("--- FINAL DATAFRAME PREVIEW: AFTER TOKENIZATION ---")
df[['ReviewBody', 'Tokenized_Review']].head()
```

Figure. 4

The final step involves segmenting the cleaned text into individual tokens using NLTK’s word_tokenize function. Tokenization is a foundational NLP process that converts a continuous text string into discrete elements (typically words) that can be used in vectorization and modeling. Care was taken to ensure compatibility with earlier cleaning steps, especially in handling punctuation and contractions.

The selected tokenizer uses the Penn Treebank tokenization conventions, which offer a reasonable balance between linguistic correctness and computational simplicity [12]. More advanced tokenization strategies—such as Byte Pair Encoding (BPE) or subword tokenization—could be explored in future versions of the pipeline, particularly if deep learning models are to be applied.

D. Quantitative Analysis

We measured the number of words before and after cleaning. Results are summarized in TABLE I.

Metric	Value
Total Words Before	592802
Total Words After	307093
Reduction (%)	48.20%

Table I.

This reduction affirms the pipeline’s success in removing redundant or non-informative elements.

```python
print("--- WORD COUNT ANALYSIS ---")
print(f"Total words before cleaning: {words_before}")
print(f"Total words after cleaning: {words_after}")
print(f"Percentage of words removed: {reduction_percentage:.2f}%")


df.head(10)
```

```
Total words after cleaning: 307093
Percentage of words removed: 48.20%
```

Figure 5.

Qualitative Analysis

State	Text
Before	"4 Hours before takeoff... Service level far worse then Ryanair..."
After	"takeoff received mail stating cryptic message disruptions expected..."

Table II.

TABLE II contrasts a sample review before and after preprocessing.

	ReviewBody	Tokenized_Review
0	4 Hours before takeoff we received a Mail stat...	[takeoff, received, mail, stating, cryptic, me...
1	I recently had a delay on British Airways from...	[recently, delay, british, airways, bru, due, ...
2	Boarded on time, but it took ages to get to th...	[boarded, time, took, ages, get, runway, due, ...
3	5 days before the flight, we were advised by B...	[5, days, flight, advised, ba, cancelled, aske...
4	We traveled to Lisbon for our dream vacation, ...	[traveled, lisbon, dream, vacation, cruise, po...

Figure. 6

IV. Authors and Affiliations

The preprocessing pipeline achieved its goals with demonstrable efficiency. The nearly 45% word reduction reduced computational overhead and eliminated many non-informative elements. Importantly, domain-specific cleaning such as removal of codes and durations enhanced the dataset’s semantic clarity.

However, the pipeline has limitations. The use of a general stopword list allowed domain-specific terms like “flight” and “airline” to persist, potentially diminishing sentiment signal clarity. A custom domain stopword list, as discussed in [11], could further refine results. Moreover, spelling normalization and sarcasm detection were not

addressed—known open problems in preprocessing pipelines [12].

V. CONLCLUSION

This paper presented a robust text preprocessing pipeline applied to airline review sentiment analysis. It successfully reduced textual noise and prepared the data for advanced NLP applications. Quantitative and qualitative evaluations confirmed the effectiveness of lowercasing, custom regex-based noise removal, punctuation and stopword filtering, and tokenization.

Future work should focus on extending the pipeline to include domain-specific stopwords, spelling normalization, lemmatization, and context-aware techniques such as named entity recognition and sarcasm detection. A modular, adaptable pipeline will be increasingly valuable as NLP applications continue to expand across industries.

REFERENCES

[1] W. Medhat, A. Hassan, and H. Korashy, "Sentiment analysis algorithms and applications: A survey," Ain Shams Engineering Journal, vol. 5, no. 4, pp. 1093–1113, 2014. doi: 10.1016/j.asej.2014.04.011.

[2] L. Gunawardhana et al., "A Survey on Sentiment Analysis Methods, Applications, and Challenges," in 2022 Intl. Conf. on Advancements in Computing (ICAC), 2022, pp. 200–205. doi: 10.1109/ICAC57686.2022.10025000.

[3] I. Hemalatha, G. P. S. Varma, and A. Govardhan, "Preprocessing the Informal Text for Efficient Sentiment Analysis," IJETTCS, vol. 2, no. 5, pp. 119–123, 2013.

[4] S. Bird, E. Klein, and E. Loper, Natural Language Processing with Python, O'Reilly Media, 2009.

[5] U. S. Shanthamallu et al., "A brief survey of text mining," in IEEE UEMCON, 2017, pp. 455–460. doi: 10.1109/UEMCON.2017.8249092.

[6] P. Jindal and B. Liu, "Opinion spam and analysis," in WSDM, 2008.

[7] B. Pang and L. Lee, "Opinion Mining and Sentiment Analysis," Found. Trends Inf. Retr., vol. 2, no. 1–2, pp. 1–135, 2008.

[8] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," Information Processing & Management, vol. 24, no. 5, pp. 513–523, 1988.

[9] A. Al-Zaidy and A. Al-Shara, "The Impact of Using Domain-Specific Stop Words on the Accuracy of Sentiment Analysis," Journal of Engineering, vol. 25, no. 7, pp. 121–133, 2019. doi: 10.31026/j.eng.2019.07.09.

[10] S. A. K. Al-Harbi et al., "A Survey on Text Preprocessing Techniques for the Arabic Language," in IEEE CCCI, 2021. doi: 10.1109/CCCI52690.2021.9565655.

[11] M. Potthast et al., "Evaluating the Impact of Writing Style on the Detection of Fake News," in Proc. of EMNLP, 2018.

[12] L. M. Aiello et al., "The Polarization of Social Media: A Comparative Analysis of Twitter and Facebook," ACM Trans. Web, vol. 12, no. 1, pp. 1–30, 2018.

[13] R. Smith, ftfy: Fixes Text for You, Python Library, https://pypi.org/project/ftfy/

[14] M. Marcus et al., "Building a large annotated corpus of English: The Penn Treebank," Computational Linguistics, vol. 19, no. 2, pp. 313–330, 1993.