# Lab 2: Building an NLP Pipeline for Sentiment Analysis

This lab demonstrates the creation of a Natural Language Processing (NLP) pipeline for preprocessing tweets for sentiment analysis, utilizing Python's NLTK and other libraries. Below is a detailed summary and explanation.



| Student Name | ID | Section |
|---|---|---|
| Lujain Bukassim Almarri | S20106753 | 1 |



**Learning Objectives**

1. Load and explore a dataset: Work with a balanced Twitter dataset from NLTK (5,000 positive and 5,000 negative tweets).

2. Text preprocessing steps: Transform raw text into a clean, structured format suitable for analysis.

3. Text representation: Use techniques like Bag of Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF) to represent text as numerical data.

---

**Steps in the Lab**

**1. Loading and Exploring the Dataset**

- Dataset: The Twitter dataset from NLTK contains labeled tweets, separated into positive and negative categories.

- Objective: Understand the dataset structure, such as data types and the balance between classes.

**Key Code and Outputs:**

```
from nltk.corpus import twitter_samples
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')
```

The dataset is stored as a list of strings where each string represents a tweet.

Visualization: A pie chart is used to show the balance between positive and negative tweets.

---

**2. Text Preprocessing**

Preprocessing involves cleaning and transforming text data for machine learning.

Steps in Preprocessing:

Lowercasing and cleaning:

Remove retweet marks (RT), hashtags, hyperlinks, and special characters using regex.

Example: My beautiful sunflowers #happy → my beautiful sunflowers happy

**Code:**

```
tweet_clean = re.sub(r'^RT[\s]+', '', tweet)
tweet_clean = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet_clean)
tweet_clean = re.sub(r'#', '', tweet_clean)
```

**Tokenization:**

Break text into individual words using NLTK's TweetTokenizer.

Example: my beautiful sunflowers happy → ['my', 'beautiful', 'sunflowers', 'happy']

**Code:**

```
tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True, reduce_len=True)
tokens = tokenizer.tokenize(tweet_clean)
```

**Stopword and punctuation removal:**

**Remove common stopwords like "the", "is", and punctuation using nltk.corpus.stopwords and Python's string.punctuation.**

**Example: ['my', 'beautiful', 'sunflowers', 'happy'] → ['beautiful', 'sunflowers', 'happy']**

**Code:**

```
stopwords_english = stopwords.words('english')
tokens_cleaned = [word for word in tokens if word not in stopwords_english and word not
```

**Stemming:**

**Reduce words to their base or root forms using PorterStemmer.**

**Example: ['beautiful', 'sunflowers', 'happy'] → ['beauti', 'sunflow', 'happi']**

**Code:**

```
stemmer = PorterStemmer()
stems = [stemmer.stem(word) for word in tokens_cleaned]
```

---

**3. Text Representation (Vectorization)**

**Purpose: Convert cleaned text data into numerical formats for machine learning models.**

**Techniques:**

**Bag of Words (BoW):**

**Represents text as word counts.**

**Example: ['happy', 'sunflowers', 'happy'] → {happy: 2, sunflowers: 1}**

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
bow = vectorizer.fit_transform([tweet_clean])
print(vectorizer.get_feature_names_out())  # ['beautiful', 'sunflowers', 'happy']
```

**TF-IDF (Term Frequency-Inverse Document Frequency):**

**Weights words based on their frequency in the document and across all documents.**

**Useful for distinguishing significant words from common ones.**

**Example output: [('happy', 0.45), ('sunflowers', 0.35)]**

**Code:**

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
tfidf = tfidf_vectorizer.fit_transform([tweet_clean])
print(tfidf_vectorizer.get_feature_names_out())  # ['beautiful', 'sunflowers', 'h
```

---

**Key Takeaways**

**Preprocessing Pipeline:**

Crucial steps: Lowercasing, tokenization, stopword removal, punctuation removal, and stemming.

Customizable based on the dataset and application.

Data Representation:

Bag of Words captures word frequency but lacks context.

TF-IDF adds context by penalizing common words.

**Applications:**

Cleaned and vectorized data can now be used for sentiment classification using machine learning models like logistic regression or neural networks.