

Day 19 Training Report

17 July 2025

Final Project: Sentiment Analysis on Tweets/Product Reviews

On **Day 19**, we undertook a **comprehensive, end-to-end NLP project** to analyze sentiment in real-world text data, such as **tweets or product reviews**. This day was the culmination of everything learned in previous NLP sessions, focusing on **practical implementation, problem-solving, and result interpretation**.

1. Project Objectives

- Build a complete **sentiment analysis pipeline** using NLP techniques.
 - Apply all preprocessing and feature extraction methods learned earlier.
 - Train, evaluate, and optimize a machine learning model for sentiment classification.
 - Extract **actionable insights** from textual data.
-

2. Dataset Selection

Students worked with datasets such as:

- **Tweets** (positive, negative, neutral sentiment labels)
- **Product reviews** (Amazon, Flipkart, or similar e-commerce reviews)

Dataset Characteristics:

- Large number of short text entries (tweets) or longer reviews.
 - Noisy data including typos, emojis, hashtags, URLs, and special characters.
 - Class imbalance (e.g., more positive reviews than negative).
-

3. Data Preprocessing

1. **Text Cleaning:**
 - Removed URLs, mentions (@username), hashtags, punctuation, and special characters.
 - Converted all text to lowercase for consistency.
2. **Tokenization:** Split text into words.
3. **Stopwords Removal:** Removed common words that add no value (e.g., “the”, “is”).
4. **Stemming/Lemmatization:** Reduced words to root forms for uniformity.

Hands-on Example:

```
import re
```

```
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess(text):
    text = text.lower()
    text = re.sub(r"http\S+|@\S+|#\S+|[\^A-Za-z\s]", "", text)
    tokens = text.split()
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    return " ".join(tokens)
```

4. Feature Extraction

- Converted text into numerical vectors using **TF-IDF vectorization**.
- Captured **word importance** across all documents, giving more weight to words that are frequent in a document but rare across others.

Example:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(preprocessed_texts)
```

5. Model Training & Evaluation

- Split dataset into **training and testing sets**.
- Trained classifiers such as **Logistic Regression, Naive Bayes, or Random Forest**.
- Evaluated model performance using:
 - Accuracy
 - Confusion Matrix
 - Precision, Recall, F1-score

Example:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
```

Insights Students Learned:

- Handling **imbalanced classes** is important; metrics like precision and recall are more meaningful than accuracy alone.
 - Some preprocessing steps (like emoji removal or lemmatization) significantly improve model performance.
 - Visualization of word importance (top positive and negative words) helps interpret results.
-

6. Visualization & Reporting

- Plotted **word clouds** for positive vs negative words.
- Created **bar charts** for sentiment distribution.
- Presented key trends from tweets or reviews.

Hands-on Visualization Example:

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

```
positive_words = " ".join([text for text, label in zip(preprocessed_texts, labels) if label==1])
wordcloud = WordCloud(width=800, height=400).generate(positive_words)
```

```
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```