Sentiment Analysis Using Machine Learning

Python | NLP | scikit-learn

Project Report

February 19, 2026

Contents

1 Introduction

Sentiment analysis is a Natural Language Processing (NLP) technique used to determine the emotional tone behind a body of text. It is widely used in social media monitoring, customer feedback analysis, brand reputation management, and market research.

This project implements a complete sentiment analysis pipeline that classifies text (tweets) into **Positive** or **Negative** sentiment categories. The system includes data preprocessing, feature extraction, model training, evaluation, a command-line interface (CLI) for predictions, and an interactive web application built with Streamlit.

1.1 Objectives

- Load and clean raw text data (tweets)
- Preprocess text using NLP techniques (stopword removal, lemmatization)
- Extract features using TF-IDF Vectorization (unigrams + bigrams)
- Train and evaluate a Machine Learning classifier
- Build a CLI tool and a web app for real-time sentiment prediction
- Save and reuse the trained model using serialization

2 Dataset

2.1 Source

The dataset used is the **Sentiment140** dataset, a widely used benchmark dataset for sentiment analysis research. It was originally collected by Stanford University researchers.

- **Source:** https://www.kaggle.com/datasets/kazanova/sentiment140
- **Total Records:** 1,600,000 tweets
- **Training Sample Used:** 100,000 tweets (for computational efficiency)
- **Format:** CSV (Comma Separated Values)
- **Encoding:** Latin-1 (ISO 8859-1)

2.2 Dataset Structure

The raw CSV file contains 6 columns with no header row:

Table 1: Dataset Column Description

Index	Column	Type	Description
0	target	Integer	Sentiment label (0 = Negative, 4 = Positive)
1	id	Integer	Unique tweet ID
2	date	String	Timestamp of the tweet
3	flag	String	Query flag (NO_QUERY if not applicable)
4	user	String	Username of the tweet author
5	text	String	The raw tweet text

2.3 Label Mapping

The original dataset uses the following labels:

- **0** → Negative
- **4** → Positive

For binary classification, label `4` is remapped to `1`:

- **0** → Negative
- **1** → Positive

The dataset is approximately balanced with an equal number of positive and negative tweets.

3 Tech Stack and Libraries

Table 2: Libraries and Their Purpose

Library	Version	Purpose
Python	3.x	Core programming language
pandas	latest	Data loading and manipulation
scikit-learn	latest	ML pipeline (TF-IDF, Logistic Regression, metrics)
NLTK	latest	NLP preprocessing (stopwords, lemmatization)
Matplotlib	latest	Confusion matrix visualization
Joblib	latest	Model serialization (save/load)
Streamlit	latest	Interactive web application

4 Project Structure

```
Sentiment-Analysis-ML/
|
|-- data/
|   |-- sentiment.csv            # Dataset file (Sentiment140)
```

```
|
|-- src/
|   |-- preprocess.py            # Text cleaning, lemmatization,
   data loading
|   |-- train.py                 # Model training, evaluation,
   confusion matrix
|   |-- predict.py               # CLI prediction script
|
|-- models/
|   |-- sentiment_model.pkl      # Saved trained model
|   |-- confusion_matrix.png     # Confusion matrix visualization
|
|-- app.py                       # Streamlit web application
|-- main.py                      # Simple project runner
|-- requirements.txt             # Python dependencies
|-- report.tex                   # This report (LaTeX source)
|-- README.md                    # GitHub documentation
|-- .gitignore                   # Git ignore rules
```

5 Methodology

The project follows a standard Machine Learning pipeline:

`Raw Data` → `Preprocessing` → `Feature Extraction` → `Model Training` → `Evaluation` → `Prediction`

5.1 Data Loading

The dataset is loaded using `pandas.read_csv()` with `latin-1` encoding. Only the `target` (label) and `text` (tweet) columns are retained. The positive label is remapped from 4 to 1. A random sample of 100,000 rows is drawn for training efficiency.

5.2 Text Preprocessing

Each tweet undergoes the following cleaning steps (implemented in `src/preprocess.py`):

1. **Lowercasing:** Convert all text to lowercase for uniformity.
2. **URL Removal:** Remove all HTTP/HTTPS links using regex.
3. **Mention & Hashtag Removal:** Strip Twitter-specific `@user` mentions and `#` symbols.
4. **Punctuation Removal:** Remove all non-alphanumeric characters except spaces.
5. **Number Removal:** Remove all numeric digits.
6. **Stopword Removal:** Remove common English words (“the”, “is”, “at”, etc.) using NLTK’s stopword corpus.

7. **Lemmatization:** Reduce words to their base/root form using NLTK's WordNet Lemmatizer (e.g., "running" → "run", "better" → "better").
8. **Short Word Removal:** Remove single-character tokens.

Example:

```
Input:  "@user I am SO happy!!! Check http://example.com #love"
Output: "happy check love"
```

5.3 Feature Extraction: TF-IDF Vectorization

Text data is converted into numerical feature vectors using **Term Frequency–Inverse Document Frequency (TF-IDF)**.

The TF-IDF score for a term t in document d from corpus D is calculated as:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

Where:

$$\text{TF}(t, d) = \frac{\text{Number of times } t \text{ appears in } d}{\text{Total number of terms in } d}$$

$$\text{IDF}(t, D) = \log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right)$$

Configuration:

- `max_features = 10,000` — Top 10,000 most important features
- `ngram_range = (1, 2)` — Both unigrams (single words) and bigrams (word pairs)

Bigrams are important because they capture phrases like "not good" or "very bad" which carry different meaning than individual words.

5.4 Model: Logistic Regression

Logistic Regression is a linear classification algorithm that models the probability of a binary outcome. It is well-suited for text classification due to its:

- Fast training and prediction speed
- Good performance on high-dimensional sparse data (like TF-IDF vectors)
- Interpretability
- Ability to output probability scores

The model computes the probability using the sigmoid function:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}}$$

Configuration:

- `max_iter = 1000` — Maximum iterations for convergence
- `random_state = 42` — For reproducibility

The entire pipeline (TF-IDF + Logistic Regression) is encapsulated in a `sklearn.pipeline.Pipelin` for clean, reproducible execution.

5.5 Train/Test Split

The data is split into training and testing sets:

- **Training Set:** 80% (80,000 tweets)
- **Testing Set:** 20% (20,000 tweets)
- **Random State:** 42 (for reproducibility)

6 Results and Evaluation

6.1 Accuracy

The trained model achieved an overall accuracy of:

$$\boxed{\text{Accuracy} = 76.95\%}$$

6.2 Classification Report

Table 3: Classification Report

Class	Precision	Recall	F1-Score	Support
Negative	0.77	0.75	0.76	9,822
Positive	0.77	0.78	0.78	10,085
Accuracy				**0.77**
Macro Avg	0.77	0.77	0.77	19,907
Weighted Avg	0.77	0.77	0.77	19,907

6.3 Metrics Explanation

- **Precision:** Of all tweets predicted as a class, what fraction was correct?

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall:** Of all actual tweets of a class, what fraction was correctly identified?

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-Score:** Harmonic mean of Precision and Recall.

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

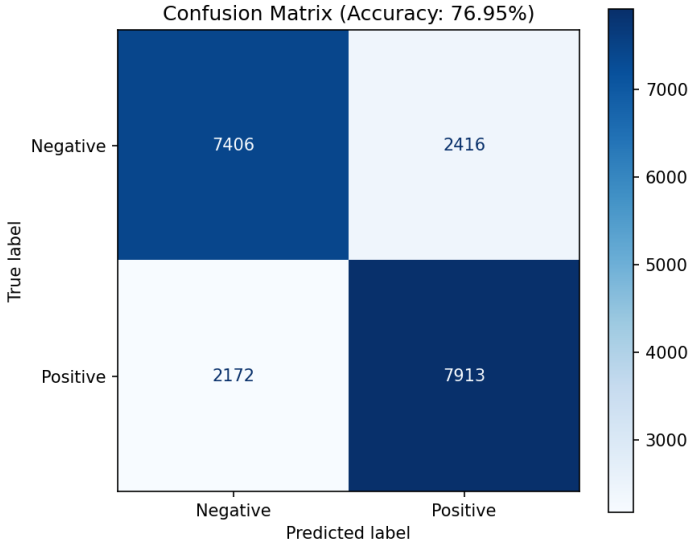

Figure 1: Confusion Matrix of the Logistic Regression Model

6.4 Confusion Matrix

The confusion matrix visualizes correct and incorrect predictions:

The diagonal elements represent correct predictions (True Positives and True Negatives), while the off-diagonal elements represent misclassifications.

7 Model Serialization

The trained pipeline (TF-IDF vectorizer + Logistic Regression model) is saved to disk using `joblib`:

```
import joblib
joblib.dump(pipeline, 'models/sentiment_model.pkl')
```

This allows the model to be loaded later without retraining:

```
model = joblib.load('models/sentiment_model.pkl')
prediction = model.predict(["I love this!"])
```

8 Application Interfaces

8.1 Command-Line Interface (CLI)

The file `src/predict.py` provides an interactive CLI where users can type text and receive instant sentiment predictions:

```
$ python src/predict.py

--- Sentiment Analysis CLI ---
Model loaded successfully!
Enter text to analyze (type 'exit' to quit):
```

```
Input: I love this movie so much
Sentiment: Positive

Input: This is the worst experience ever
Sentiment: Negative
```

8.2 Streamlit Web Application

The file `app.py` provides a browser-based web interface with:

- **Single Text Analysis:** Enter one sentence and get sentiment + confidence score
- **Batch Analysis:** Enter multiple sentences (one per line) and analyze all at once
- **Sidebar:** Displays model information and the confusion matrix

To run the web app:

```
$ python -m streamlit run app.py
```

The app opens in the browser at `http://localhost:8501`.

9 How to Run the Project

1. **Clone the repository:**

```
git clone https://github.com/<username>/Sentiment-Analysis-ML.
   git
cd Sentiment-Analysis-ML
```

2. **Install dependencies:**

```
pip install -r requirements.txt
```

3. **Train the model:**

```
python src/train.py
```

4. **Run CLI predictions:**

```
python src/predict.py
```

5. **Run the web app:**

```
python -m streamlit run app.py
```

10 Improvements Made

During development, the following improvements were applied to boost performance:

Table 4: Improvements and Impact

#	Improvement	Before	After
1	Naive Bayes → Logistic Regression	73.87%	76.95%
2	Added Lemmatization	–	Cleaner features
3	Added Bigrams to TF-IDF	unigrams only	unigrams + bigrams
4	Increased training data (50K → 100K)	50,000	100,000
5	Increased TF-IDF features (5K → 10K)	5,000	10,000
6	Added Confusion Matrix visualization	–	
7	Added Streamlit Web App	CLI only	CLI + Web App

11 Future Scope

- Train on the full 1.6 million tweet dataset for higher accuracy
- Experiment with SVM or Random Forest classifiers
- Use word embeddings (Word2Vec, GloVe) instead of TF-IDF
- Implement deep learning models (LSTM, BERT) for state-of-the-art performance
- Add neutral sentiment class for three-way classification
- Deploy the Streamlit app to Streamlit Cloud or Heroku
- Add real-time Twitter API integration for live sentiment tracking

12 Conclusion

This project successfully demonstrates a complete sentiment analysis pipeline from raw data to a deployable application. By using NLP preprocessing techniques (lemmatization, stopword removal), TF-IDF feature extraction with bigrams, and Logistic Regression classification, the model achieves **76.95% accuracy** on the Sentiment140 Twitter dataset.

The project includes both a command-line interface and an interactive Streamlit web application, making it practical for real-world use and portfolio presentation.

References

1. Go, A., Bhayani, R., & Huang, L. (2009). *Twitter Sentiment Classification using Distant Supervision.* Stanford University.
2. scikit-learn Documentation: https://scikit-learn.org/stable/
3. NLTK Documentation: https://www.nltk.org/
4. Streamlit Documentation: https://docs.streamlit.io/
5. Sentiment140 Dataset: https://www.kaggle.com/datasets/kazanova/sentiment140