# Final Report: Sentiment Analysis on Social Media Reviews

**Krish Chaudhary**

**COE 379L – Machine Learning Applications**

**May 5, 2025**

## Introduction and Problem Statement

Social media has become one of the most powerful tools for sharing public opinion, emotion, and commentary in real time. Every second, thousands of tweets and comments flood platforms like Twitter and Reddit, expressing joy, anger, sarcasm, humor, and frustration. With so much raw, unstructured, and often messy text being generated, one compelling challenge in the field of machine learning is figuring out how to make sense of it all. That's where sentiment analysis comes in.

For this project, my goal was to build a sentiment classification pipeline that could take tweets as input and predict whether the sentiment was positive, negative, or neutral. While this might sound straightforward at first glance, the reality is more nuanced. Social media language is informal, short, and often filled with emojis, slang, sarcasm, and abbreviations—elements that traditional natural language processing (NLP) systems often struggle with. So rather than just throwing a model at the data, I wanted to go deeper: explore how well classic machine learning models perform, analyze where they fall short, and investigate what techniques could be used to improve their performance. I also wanted to visualize the results in meaningful ways and consider the broader implications of applying such a system to real-world platforms.

# Data Sources and Technologies Used

The core dataset I worked with is called the **Sentiment140** dataset, which contains 1.6 million tweets that were labeled automatically based on emoticons. Tweets with :) or :-D were labeled as positive, those with :( or >:( were labeled as negative, and those without emoticons were considered neutral. While this auto-labeling approach isn't perfect, it provided a solid foundation to explore large-scale sentiment classification.

Given time and resource constraints, I randomly sampled **10,000 tweets** from the full dataset to keep training and testing fast. Each tweet came with metadata like user ID, timestamp, and query, but I only focused on the tweet text and its sentiment label.

For tools and libraries, I used Python along with the following stack:

- **Pandas and NumPy** for data manipulation
- **NLTK** for text preprocessing (tokenization, stopword removal, lemmatization)
- **Scikit-learn** for model training and evaluation
- **Matplotlib and Seaborn** for visualizations
- **WordCloud** for visually analyzing the most frequent words in each sentiment category
- **Jupyter Notebook** as my coding and experimentation environment
- **GitHub** for version control and submission

# Methods Employed

### Step 1: Preprocessing the Raw Tweets

Raw tweet text is messy. It often contains user mentions (e.g., @user), URLs, emojis, special characters, hashtags, and inconsistent casing. To prepare the tweets for model training, I created a preprocessing function that:

- Lowercased all text (to avoid "Love" and "love" being treated as different words)
- Removed URLs, mentions, and hashtags
- Removed punctuation
- Tokenized the text using nltk.word_tokenize
- Filtered out stopwords (like "is", "the", "on") to focus on the meaningful words
- Lemmatized the tokens to reduce them to their root form (e.g., "running" becomes "run")

This cleaned text was stored in a new column called clean_text. One benefit I noticed immediately was that the resulting vocabulary was more consistent, with far fewer duplicates caused by variation in word form or punctuation.

## Step 2: Exploring the Dataset (EDA)

Before jumping into modeling, I wanted to get a feel for the data. I plotted the distribution of sentiment classes, which revealed a slightly imbalanced dataset—neutral tweets were the most common, followed by positive, and then negative. I also generated **word clouds** for each sentiment category. These visualizations were surprisingly insightful. For instance, the positive class was dominated by words like "love", "thank", and "awesome", while the negative class had words like "miss", "tired", and "hate". The neutral class had a mix of generic terms like "today", "work", and "going". These visuals helped validate that the sentiment labels broadly aligned with the underlying language.

## Step 3: Feature Extraction with TF-IDF

Next, I needed a way to convert the cleaned text into numerical features that machine learning models could understand. I chose **TF-IDF (Term Frequency-Inverse Document Frequency)**, a commonly used technique that gives more weight to words that are unique and informative. For example, common words like "the" appear in nearly every tweet and get low weight, while words like "thrilled" or "horrible" are much more telling and receive higher importance.

The resulting matrix was sparse but effective. Each tweet was now a vector in a high-dimensional space, and ready to be fed into a classifier.

## Step 4: Training the Models

I started with two models:

- **Logistic Regression**
- **Naive Bayes (MultinomialNB)**

Both are standard, interpretable baseline models for text classification tasks. I trained each on 80% of the data and tested on the remaining 20%. Logistic Regression slightly outperformed Naive Bayes with an overall F1-score of **0.70**, compared to **0.71** for Naive Bayes. However, Naive Bayes had better recall on the negative class, making it a good alternative depending on the application.

## Step 5: Evaluating and Interpreting Results

I used classification_report from scikit-learn to get precision, recall, and F1-scores per class. I also visualized a **confusion matrix** for Logistic Regression. This showed that most misclassifications were between neutral and either positive or negative, which makes sense: neutral tweets often lack strong emotional signals, making them harder to distinguish.

## Step 6: Hyperparameter Tuning with GridSearchCV

To push performance a little further, I used **GridSearchCV** to search for the best value of the regularization parameter C in Logistic Regression. After trying a few values, the best performing model used C = 1, achieving a cross-validated accuracy of **71.6%**. This showed that even with a simple model like Logistic Regression, tuning can lead to noticeable improvements.

### Step 7: Visualizing with Word Clouds

I wanted to include some visuals that go beyond standard metrics. So I generated **word clouds for each sentiment class**, based on the cleaned text. This served two purposes: it made the report more engaging and gave us an intuitive sense of what kinds of words were driving each sentiment prediction. These visuals could be especially helpful in a real-world dashboard, helping companies monitor public perception of their brand or product.

## Results and Discussion

Overall, the models performed as expected. Logistic Regression proved to be a reliable baseline, achieving a macro-average F1-score of **0.70**. Naive Bayes performed similarly, and in some cases better for minority classes. The confusion matrix revealed that most errors occurred with tweets labeled as neutral, indicating that even humans might disagree on some of these classifications.

One interesting insight was how the models struggled with sarcasm or subtle emotional context. For example, a tweet like “Just great. My car broke down.” might be classified as positive due to the word "great", even though the actual sentiment is clearly negative. This limitation points to a potential next step: using transformer-based models like BERT, which are better at capturing context.

That said, even with these limitations, this project demonstrated that classical ML techniques can still yield solid results on noisy, real-world text. In practice, a lightweight model like Logistic Regression could be deployed in resource-constrained environments where deep learning isn't feasible.

# Future Directions and Broader Impact

This project could easily be extended in several exciting directions. One would be to include a **transformer model like DistilBERT**, using pretrained embeddings and fine-tuning for classification. This would allow the system to understand more nuanced patterns, like sarcasm and slang.

Another path would be to implement **real-time inference** through a REST API. For example, a web app could take a tweet as input and instantly return the predicted sentiment. This could be useful in marketing, political polling, or customer service, where quick feedback is crucial.

Lastly, from a societal perspective, sentiment analysis has massive potential. It can be used to track mental health trends, detect online harassment, analyze election discourse, or even assist in disaster response by identifying urgent needs from social media posts. The pipeline we built lays the groundwork for more advanced and impactful applications.

# References


1. Sentiment140 Dataset: https://www.kaggle.com/datasets/kazanova/sentiment140
2. Bird, Steven, Edward Loper and Ewan Klein (2009). *Natural Language Processing with Python*. O'Reilly Media.
3. Scikit-learn Documentation: https://scikit-learn.org/
4. NLTK Documentation: https://www.nltk.org/
5. WordCloud Library: https://github.com/amueller/word_cloud