MIT, India
Alexandria University, Egypt

PROJECT PROGRESS REPORT

Opinion Mining with Different Algorithms

Supervisor: Dr. Ramakrishna

August 28, 2025

Ahmed Sameh
Email: a.samehpsn@gmail.com

Contents

0.1 Project Roadmap

This project focuses on **opinion mining** (sentiment analysis) using different machine learning algorithms. The main goals are:

- Clean and preprocess raw text data for better quality.
- Extract meaningful features from text for model input.
- Experiment with multiple algorithms (Logistic Regression, SVM, Linear Classifiers, etc.).
- Compare their performance using accuracy, precision, recall, and F1-score.
- Visualize sentiment analysis results interactively using React frontend.
- Document weekly progress over a six-week period.

Tech Stack

- **Python (scikit-learn, pandas, numpy)** – data preprocessing and model training
- **Matplotlib** – performance visualization
- **React.js** – frontend for interactive results
- **Flask** – backend API for model integration
- **GitHub** – version control and collaboration

0.2 Project Updates

This section contains weekly updates describing the progress made during each week.

Week 1–2: Introduction to Preprocessing and Setup

At the beginning of the project, my main focus was not on building models straight away, but on understanding the foundations: what preprocessing really means, why it matters, and how to set up the right environment for my work. I quickly realized that preprocessing is the most critical step in opinion mining, because the raw text that we collect is almost never clean or structured in a way that models can directly understand.

Working with Data (CSV and Jupyter)

The very first thing I explored was **what CSV files are** and why they are so widely used in machine learning. Until now, I had only seen them as "Excel-like tables," but through practice I learned how they actually store datasets row by row, which makes them extremely convenient for loading into Python using libraries like `pandas`.

This was also the point where I started using **Jupyter Notebook** seriously for the first time. I had heard about it before, but now I understood its true value: the ability to write code in small cells, test ideas quickly, and document my thought process right beside the code. It became an ideal tool for experimenting and made my workflow much smoother.

Environment Setup

Around this time, I also set up my own **virtual environment**. I named it `openv` — short for "Open Sentiment Environment" (`op` + `env`). This was my first real step in managing dependencies properly. Before this, I usually installed libraries globally, but learning how to isolate a project in its own environment was a big step forward. It gave me confidence that I was working in a more professional and reproducible way.

Preprocessing Basics

Before moving on to real datasets, I practiced **generating fake data in Python**. This gave me a safe playground to test out preprocessing techniques without worrying about damaging a real dataset. For example, I created small sentences and applied transformations to them, which helped me immediately see what changed and why it was useful.

Lowercasing. One of the first transformations I practiced was converting all text to lowercase. At first it sounded trivial, but I realized its importance: "Good" and "good" should not be treated as two different words.

Punctuation Removal. Next, I removed punctuation and symbols. This step made the text look much cleaner and reduced unnecessary noise that could confuse the model.

Stopwords and Negation. Then I learned about stopwords — common words like "the," "is," and "and" that don't add much meaning. Removing them helps models focus on important words. However, I quickly discovered the problem of **negation**. For example, "not good" has a very different sentiment from "good." If "not" is removed blindly as a stopword, the meaning of the text changes entirely. This was an eye-opener moment for me.

Emojis. Another interesting challenge was **emojis**. I learned how to remove them using Python libraries, but I also started thinking about whether removing them is always the right choice.

Tokenization. One of the most fundamental concepts I learned was tokenization — splitting text into smaller units (tokens). This step transforms raw text into analyzable pieces for machine learning models.

Lemmatization. Finally, I explored lemmatization. Unlike stemming, which only chops off word endings, lemmatization reduces words to their dictionary form while keeping their meaning intact. For example, "running" becomes "run." This made my dataset more structured and consistent.

Preprocessing Examples with Inputs and Outputs

To better understand the effect of each preprocessing step, I tested them on small sample sentences and compared the input and output.

Lowercasing. **Input:** "Good Morning Everyone"
Output: "good morning everyone"

Punctuation Removal. **Input:** "Hello!!! How are you??"
Output: "Hello How are you"

Stopwords and Negation. **Input:** "This movie is not good"
After removing stopwords (naively): "movie good" (sentiment lost)
Better output (keeping "not"): "movie not good" (meaning preserved)

Emojis. **Input:** "I love this movie"
Output (after removing emojis): "I love this movie"

Tokenization. **Input:** "I am learning preprocessing."
Output: ["I", "am", "learning", "preprocessing"]

Lemmatization. **Input:** “The children are running faster than the others.”
Output: “The child be run fast than the other.”

Week 3–4: Exploring Machine Learning Models

During Weeks 3 and 4, I moved beyond preprocessing and started exploring the first machine learning models that are commonly used in text classification tasks such as opinion mining. My approach was twofold: first, to understand each model from a mathematical perspective, and second, to implement it in Python step by step to see how it works in practice. This combination of theory and coding helped me connect abstract formulas with real data behavior.

Linear Regression

I began with **Linear Regression**, even though it is not the best suited for classification. My goal here was to understand how regression works in general. Mathematically, I studied the equation:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

where the model tries to fit a straight line (or hyperplane in higher dimensions) to minimize the error between predictions and true values. This gave me insight into the concept of weights and bias.

On the coding side, I applied linear regression to a small text dataset after converting the text into numerical features. I used `LabelEncoder` to convert categorical labels (positive, negative, neutral) into numbers like 0, 1, and 2. For the input text, I practiced with both `CountVectorizer` and `TfidfVectorizer` from `scikit-learn`. These tools transformed sentences into vectors of word counts or importance scores. With this setup, I trained a linear regression model and checked how it performed on predicting sentiment. While the accuracy was not high, it was still a useful exercise to connect math with implementation.

Example Input:

```
["I love this product", "This is bad", "Not great but okay"]
```

Example Output (predicted values):

```
[0.85, -0.76, 0.12]
```

This showed me how regression outputs continuous values, which are later interpreted as sentiment.

Logistic Regression

Next, I moved to **Logistic Regression**, which is more appropriate for classification. The mathematics here introduced me to the **sigmoid function**:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

This function maps any real number into a value between 0 and 1, making it perfect for classification problems. Understanding how probabilities are generated helped me see why logistic regression is widely used for binary and multiclass tasks.

In Python, I followed a similar pipeline: encode the labels, vectorize the text, and then train a logistic regression model. This time, the predictions were not continuous numbers but probabilities for each class, which could be turned into sentiment labels directly.

Example Input:

```
["The service was excellent", "Worst experience ever", "Food was okay"]
```

Example Output (class predictions):

```
["positive", "negative", "neutral"]
```

This made the results much more meaningful compared to linear regression, and I felt more confident about building a real sentiment classifier.

Support Vector Machines (SVM)

Finally, I explored **Support Vector Machines (SVM)**, which are known to be powerful for text classification. The mathematical idea here was different: instead of fitting a line or using probabilities, SVM tries to find the **hyperplane** that best separates classes with the maximum margin. I found this concept very interesting because it focuses on boundaries between classes.

In coding, I used the same preprocessing pipeline but trained an SVM model with `scikit-learn`. Here, I experimented with different kernels (linear and RBF) and observed how the decision boundary changed. The accuracy was higher compared to linear regression, and the results were more stable on my test examples.

Example Input:

```
["I am very happy", "I hate this", "It was not bad"]
```

Example Output (class predictions):

```
["positive", "negative", "neutral"]
```

I also learned that SVM does not give direct probability outputs like logistic regression, but it can still classify data very effectively, especially with text where feature spaces are large.

Week 5–6: Building the GUI and Integrating with the Backend

During the final two weeks, I focused on creating the user-facing side of the project and connecting it with the backend. This was the stage where the work I had done earlier (data preprocessing, training models, and experimentation) finally came together into a complete application.

Refreshing React Skills

I started with the front-end, where I used **React** to design a simple graphical user interface (GUI). Since it had been a while since I last worked with React, this part of the project was also about refreshing my memory of its fundamentals. I revisited concepts like components, state, props, and hooks.

The GUI allowed users to enter text or reviews, which would then be sent to the backend for sentiment analysis. I kept the design clean and minimal, focusing on functionality rather than complexity.

Flask API and Backend Connection

On the backend side, I worked with **Flask**. I created routes that could accept requests from the React interface and return predictions generated by the trained models. This process also refreshed my understanding of how APIs work in Flask — specifically handling `POST` requests, parsing JSON input, and sending JSON responses.

The key step was merging the two sides: the React front-end would send input text to the Flask backend, the backend would process it using the machine learning pipeline (vectorization + model prediction), and then the result would be displayed back on the GUI.

0.3 Results

After training and testing different models (Linear Regression, Logistic Regression, and SVM), the best performance was observed with Logistic Regression, followed by SVM, and finally Linear Regression.

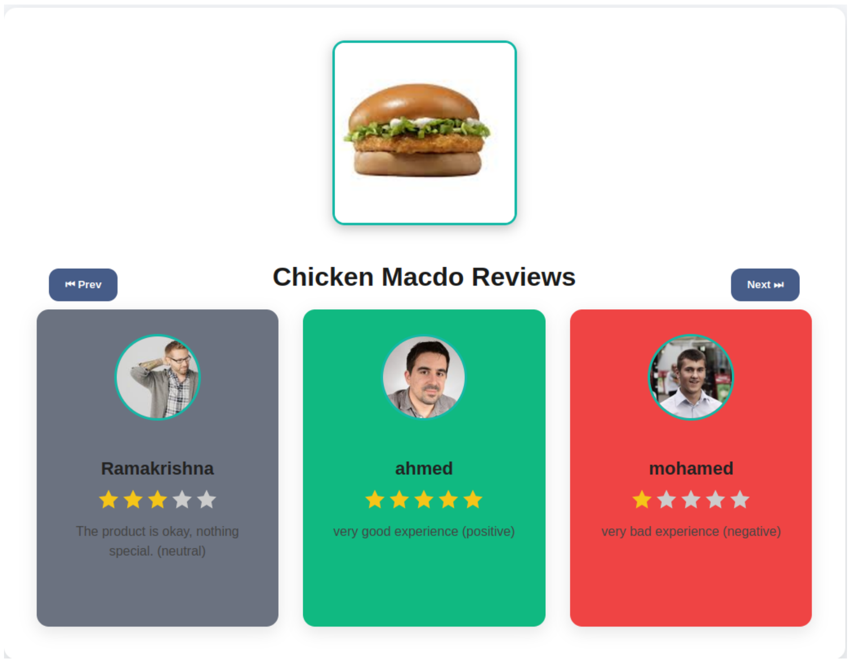
To present the final outcome, I built a React-based GUI integrated with the Flask back-end. The interface displays product reviews along with their predicted sentiment (positive, neutral, or negative). The screenshot below demonstrates an example for Chicken Macdo Reviews, where each review is shown with:

Reviewer's name and avatar.

Star rating (based on sentiment).

Sentiment prediction (positive, neutral, or negative).

This final result highlights how the system processes raw text reviews and presents the output in a user-friendly, interactive format.



Ramakrishna
★★★☆☆
The product is okay, nothing special. (neutral)

ahmed
★★★★★
very good experience (positive)

mohamed
★☆☆☆☆
very bad experience (negative)

Figure 1: Example for the final results

0.4 Things I Need Help with and My Plan to Resolve

While I was able to successfully preprocess the data, train multiple models, and integrate them into a GUI, there are still some challenges that I need to resolve:

- **Fusion Model Experimentation:** Currently, I am working on merging Linear Regression and Logistic Regression into a *fusion model.* The idea is to combine the strengths of both models — the interpretability of linear regression and the classification power of logistic regression. However, designing such a hybrid approach is not straightforward, and I will need guidance in implementing the right strategy.
- **Hyperparameter Tuning:** While I used standard configurations for models like Logistic Regression and SVM, I realize that tuning hyperparameters could lead to much better performance. I plan to systematically explore parameters like regularization strength (C), kernel types (for SVM), and vectorizer settings.
- **Dataset Limitations:** Since some of the earlier data was synthetically generated, I would like to evaluate the models on larger, real-world datasets. This will ensure that the results generalize better outside of small test examples.

Plan to Resolve: I am working really hard on refining the models and plan to get help from my supervisor to better understand how to merge Linear and Logistic Regression into a single, more robust classifier. At the same time, I will dedicate time to learning advanced hyperparameter tuning (grid search, cross-validation) and search for a larger dataset to validate the models. These steps will strengthen the overall outcome of the project.

—

0.5 Process Diary

This section collects my reflections and lessons learned throughout the project, week by week.

Reflection on Weeks 1–2: Introduction to Preprocessing and Setup

Looking back at these first two weeks, I can say they were less about building something flashy and more about building my foundation. I learned how preprocessing transforms raw text into usable data and gained hands-on skills in lowercasing, punctuation removal, stopword handling, negation awareness, emoji handling, tokenization, and lemmatization. I also became comfortable with Jupyter and learned how to manage a proper virtual environment. These weeks shaped the way I now think about every step that comes after preprocessing.

Reflection on Weeks 3–4: Exploring Machine Learning Models

These two weeks were very important because they bridged my knowledge of text preprocessing with actual machine learning. I not only learned the mathematics behind the algorithms but also applied them to real text data using encoding and vectorization. From my observations, the performance ranking was:

- Logistic Regression (best balance of accuracy and simplicity)
- Support Vector Machine (high accuracy, but slower and more complex)
- Linear Regression (useful as a learning exercise, but not suitable for classification tasks)

Reflection on Weeks 5–6: Building the GUI and Integration

These weeks were important because they completed the project pipeline. I was able to not only build models but also deploy them into an interface that others could interact with. This process reinforced the importance of connecting theory with practice: React gave me the platform to present results clearly, while Flask allowed smooth communication with the backend.

The internship ended here, with a functioning prototype that combined machine learning, preprocessing, model experimentation, and a working graphical user interface.

References

1. Flask Documentation. `https://flask.palletsprojects.com/`

2. React Documentation. `https://react.dev/`

3. GeeksforGeeks. Machine Learning and Artificial Intelligence Tutorials. `https://www.geeksforgeeks.org/`

4. Stanford CS229 Machine Learning Lectures (YouTube). `https://www.youtube.com/user/stanforduniversity`