Part A : IMDb Sentiment Analysis

Final Report

1. Objective

The goal of this part of the project was to build a machine learning model that can look at a movie review and decide if it's **positive** or **negative**. It's a classic **binary text classification problem**, and it gave me a good chance to apply everything I've learned about natural language processing, data cleaning, and model building.

2. Understanding the Dataset

I started by loading the file named ***`Imdb-data_imdb.csv`***. It had **50,000 reviews** with two columns:

- `Review`: The actual user review text
- `Sentiment`: Whether the review is *positive* or *negative*

I printed the first few rows, checked the shape of the dataset, and ran some quick checks:

- **Class balance**: 25,000 positive and 25,000 negative, which is great because models learn better when classes are balanced.
- **Missing values**: None, which meant I didn't have to deal with cleaning null entries.

This helped me get a sense of what I was working with and guided the preprocessing decisions I made next.

3. Preprocessing the Reviews

Raw text data is messy. So the next big step was cleaning and preparing the reviews for modeling. Here's what I did, and why:

a. Remove HTML

Some reviews had embedded HTML tags like ***`<br>`***. These aren't helpful, so I used **BeautifulSoup** to strip them out.

b. Convert to Lowercase

I changed everything to lowercase to avoid treating "Great" and "great" as different words.

c. Remove Special Characters and Punctuation

I used a regular expression to remove anything that isn't a letter. This helped keep only the meaningful parts of the review.

d. Tokenization

I split each review into individual words (tokens), so they could be processed one by one.

e. Remove Stopwords

Words like *"is", "the", "of"* are so common that they don't carry much meaning. I removed them using NLTK's stopword list to reduce noise.

f. Lemmatization

Instead of treating *"running"*, *"runs"*, and *"ran"* as different words, I used **WordNetLemmatizer** to reduce them to their root form: *"run"*. This makes the model's job easier by reducing vocabulary size without losing meaning.

The cleaned-up version of each review was saved in a new column called ***`clean_review`.***

4. Feature Extraction

The cleaned reviews are still plain text, and machine learning models need numbers. So I used **TF-IDF Vectorization** to turn text into numbers.

TF-IDF helps identify which words are more important. Words that appear often in one review but not in others get higher scores. This is better than just counting word frequency.

Vectorization Settings:

- I limited it to the **top 5000 words** to avoid overfitting or making the model too slow.

- The final shape of the features was **`(50000, 5000)`**, meaning each review is now a row of 5000 numbers.

5. Preparing the Labels

The ***`sentiment`*** column had strings like *"positive"* and *"negative"*. I converted them to:

- 1 for positive
- 0 for negative

This is important for the model to understand the output it needs to learn.

6. Training the Model

I used **Logistic Regression** because:

- It's simple and fast
- It often works surprisingly well for text classification
- It's a good baseline to compare more complex models later

Train-Test Split

I split the data like this:

- **80% training**
- **20% testing**

This way, the model learns from most of the data but is evaluated on unseen data.

Training & Evaluation

I set ***`max_iter=200`*** to make sure the model had enough time to converge.

Then I predicted on the test set and checked how well the model performed.

7. Results

Accuracy: 88.5%

That means the model correctly classified 8.85 out of 10 reviews

Classification Report:

Sentiment	Precision	Recall	F1-Score
Negative	0.89	0.87	0.88
Positive	0.88	0.90	0.89

The scores are very close for both classes.

8. Conclusion

The sentiment analysis model successfully learned to classify movie reviews with good accuracy. The performance is balanced, and the results are reliable for a basic Logistic Regression approach.