Sentilytics

MINOR PROJECT REPORT

Submitted in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

(Department of Computer Science and Engineering)

Submitted to

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY BHOPAL (M.P.)

Submitted by

Shiva (22U02010)
Jayesh Kaushik (22U02031)

Under the supervision of

Dr. Sonal Telang Chandel
Assistant Professor
(Department of CSE)

April 2025

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY BHOPAL (M.P.)

CERTIFICATE

This is to certify that the work embodied in this report entitled **"Sentilytics"** has been satisfactorily completed by **Shiva (22U02010) and Jayesh Kaushik (22U02031)** It is a bonafide piece of work, carried out under our guidance in the **Department of Computer Science and Engineering**, **Indian Institute of Information Technology, Bhopal** for the partial fulfillment of the Bachelor of Engineering during the academic year 2024-25.

Date:

Name of Supervisor:
Dr Sonal Telang Chandel,
Assistant Professor,
Computer Science Department,
IIIT Bhopal (M.P.)

Name of Coordinator:
Dr Yatendra Sahu,
Assistant Professor,
Computer Science Department,
IIIT Bhopal (M.P.)

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY BHOPAL (M.P.)

DECLARATION

We hereby declare that the following major project synopsis entitled “Sentilytics” presented in the is the partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in** Computer Science and Engineering. It is an authentic documentation of our original work carried out under the able guidance of **Dr Sonal Telang Chandel**. The work has been carried out entirely at the Indian Institute of Information Technology, Bhopal. The project work presented has not been submitted in part or whole to award of any degree or professional diploma in any other institute or organization.

We, with this, declare that the facts mentioned above are true to the best of our knowledge. In case of any unlikely discrepancy that may occur, we will be the ones to take responsibility.

Shiva (22U02010)
Jayesh Kaushik (22U02031)

ACKNOWLEDGEMENT

We, Shiva and Jayesh Kaushik, would like to express our sincere gratitude to everyone who supported and guided us throughout the successful completion of our minor project titled "Sentiment Analysis of Textual Data using Machine Learning Techniques."

First and foremost, we would like to extend our heartfelt thanks to our esteemed Director, Dr. Ashutosh Kumar Singh, for providing us with the necessary resources and facilities for conducting our project. Their encouragement and vision played a key role in helping us accomplish our goals.

We are deeply grateful to our Project Guide, Dr. Sonal Telang Chandel, for her constant guidance, invaluable suggestions, and expert advice throughout the project. Her knowledge and insights helped us enhance the quality of our work, and her support kept us motivated during difficult times.

Our sincere thanks to the Project Coordinator, Dr. Yatendra Sahu, for organizing and overseeing the entire project process. Their assistance in clarifying doubts and providing us with the right direction was instrumental in the smooth execution of our project.

We would also like to express our gratitude to all the faculties of the Department of Computer Science and Engineering for their continuous support and encouragement. Their teaching and knowledge were invaluable in enhancing our technical skills and understanding, which helped us in the successful completion of the project.

A special thanks to our friends for their continuous support, valuable suggestions, and help during various stages of the project. Their insights and encouragement motivated us to keep working on the project with enthusiasm.

Last but certainly not the least, we would like to express our deepest gratitude to our parents and family members for their love, patience, and continuous moral support throughout the course of this project. Their understanding and encouragement were the driving force behind our efforts.

We are truly thankful to everyone who directly or indirectly contributed to the success of this project. Without their assistance and support, this project would not have been possible.

Shiva
Jayesh Kaushik
22U02010, 22U02031

AREA OF WORK

Our project primarily focuses on Natural Language Processing (NLP) and Machine Learning (ML), using a suite of tools and techniques designed to analyze and classify large volumes of textual data.

While deep learning approaches are gaining popularity in sentiment analysis, our project highlights the effectiveness of classical machine learning models, especially when combined with robust preprocessing and feature extraction techniques such as TF-IDF vectorization. These methods provide efficient and interpretable alternatives with significantly lower computational costs.

The exponential growth of user-generated content—like product reviews, social media posts, and customer feedback—has made sentiment analysis an essential application in today's data-driven world. By converting unstructured text into structured insights, our project demonstrates how businesses and researchers can understand public opinion and make informed decisions.

In this project, we not only implement and evaluate several machine learning models such as Logistic Regression, Naive Bayes, and Support Vector Machines, but also integrate PyTorch-based embedding techniques for comparison. This multi-faceted approach helps us explore both classical and deep learning paradigms in text classification.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Sentiment analysis, also known as opinion mining, is a widely used technique in the field of Natural Language Processing (NLP) that focuses on determining the sentiment expressed in a given piece of text. It plays a significant role in applications such as customer feedback analysis, brand monitoring, and opinion tracking on social media. This project aims to classify text into three sentiment categories: positive, negative, and neutral, using classical machine learning techniques.

The dataset used consists of social media texts labeled with sentiment. We begin with preprocessing steps that include converting text to lowercase, removing punctuation, eliminating stopwords, and tokenizing the text. After cleaning the data, we use Term Frequency-Inverse Document Frequency (TF-IDF) to convert the textual content into numerical feature vectors suitable for training machine learning models.

We implemented and trained three popular classical models: Logistic Regression, Naive Bayes, and Support Vector Machine (SVM). These models were evaluated using metrics such as accuracy, precision, recall, and F1-score. In addition to traditional models, a simple neural network-based implementation using PyTorch and word embeddings was introduced to compare model performance and flexibility.

The results indicate that classical models, particularly Logistic Regression, perform well on sentiment classification tasks and offer high accuracy with low computational requirements. The PyTorch-based model, although more resource-intensive, offers more flexibility in handling deeper language patterns. Overall, the project demonstrates that well-preprocessed data combined with classical machine learning models can effectively solve real-world sentiment analysis problems, especially where interpretability and performance are both important.

INTRODUCTION

In today's digital era, the explosion of textual data through platforms like social media, e-commerce websites, blogs, and forums has led to a growing need for systems that can understand and analyze human sentiments. **Sentiment analysis**, also known as opinion mining, is a significant field within **Natural Language Processing (NLP)** that focuses on determining the emotional tone behind a body of text. It is widely used in applications such as product review analysis, customer feedback monitoring, social media sentiment tracking, and more.

This project explores the application of **machine learning techniques** to analyze and classify sentiments in textual data. By leveraging **TF-IDF (Term Frequency-Inverse Document Frequency)** for feature extraction, we transform raw text into numerical vectors suitable for machine learning models. Classical algorithms such as **Logistic Regression**, **Naive Bayes**, and **Support Vector Machines (SVM)** are employed due to their simplicity, efficiency, and proven performance in text classification tasks.

Furthermore, the project also incorporates a **deep learning-based approach** using **PyTorch**, where embeddings and neural network architectures are used to extract deeper semantic relationships in text. This allows us to compare traditional models with neural network-based models in terms of performance and computational efficiency.

The core steps involved in this project include:

- Data preprocessing: cleaning, tokenization, stopword removal, and normalization.
- Feature extraction: converting text into vectors using TF-IDF.
- Model training: building and evaluating classifiers using machine learning and deep learning.
- Performance analysis: comparing models using metrics such as accuracy, precision, recall, and F1-score.

The primary objective is to develop an efficient and accurate sentiment analysis model that can be adapted to various real-world datasets and requirements. This introduction lays the foundation for the subsequent sections of the report, which delve deeper into the methodology, experimentation, and outcomes of our work.

LITERATURE REVIEW

2.1 Introduction to Sentiment Analysis

Sentiment analysis, also referred to as opinion mining, is a branch of natural language processing (NLP) that focuses on identifying and extracting subjective information from textual data. The purpose is to determine the emotional tone behind a piece of text, which can be broadly classified as positive, negative, or neutral. With the increasing volume of user-generated content on the internet, sentiment analysis has found significant applications in areas such as product reviews, political opinion tracking, customer service, and social media monitoring.

The field has evolved over the past two decades from rule-based systems and lexicon-based techniques to more complex machine learning and deep learning methods. While modern transformer-based models such as BERT have achieved state-of-the-art results, classical models still hold practical importance due to their simplicity, efficiency, and interpretability—particularly when computational resources are limited.

2.2 Rule-Based and Lexicon-Based Methods

Initial attempts at sentiment analysis relied heavily on rule-based and lexicon-based systems. These systems used predefined dictionaries of positive and negative words to score sentences. **SentiWordNet**, **AFINN**, and **LIWC (Linguistic Inquiry and Word Count)** are examples of sentiment lexicons commonly used in early research.

Rule-based systems were intuitive and easy to implement but struggled with context understanding, sarcasm, and complex sentence structures. For example, the sentence "I love waiting in long lines" may be scored positively due to the word "love" even though the sentiment is sarcastic.

Lexicon-based approaches continue to be used in specific applications where explainability and interpretability are prioritized. However, they are generally outperformed by statistical learning methods in large-scale and diverse datasets.

2.3 Machine Learning in Sentiment Analysis

The emergence of machine learning revolutionized sentiment classification. Supervised learning algorithms like **Naive Bayes**, **Logistic Regression**, **Decision Trees**, and **Support Vector Machines (SVM)** became the standard due to their ability to learn from labeled data.

Pang et al. (2002) conducted one of the most influential studies using machine learning for sentiment analysis. They showed that SVM and Naive Bayes could outperform rule-based methods on movie review classification tasks. Their study also highlighted the importance of feature engineering, such as n-grams and term frequency-based representations.

One of the key enablers of traditional ML performance is the use of **TF-IDF (Term Frequency-Inverse Document Frequency)**, which transforms text into weighted feature

vectors. This representation captures the importance of words in context and helps mitigate the impact of commonly used terms that carry little semantic weight.

Studies have shown that:

- Naive Bayes is fast and effective on simple datasets.
- Logistic Regression provides strong generalization with linear decision boundaries.
- SVM is powerful in high-dimensional spaces and handles imbalanced datasets well.

2.4 Feature Extraction Techniques

Text data needs to be converted into numerical form for machine learning algorithms to process. Common techniques include:

- **Bag-of-Words (BoW)**: Represents text as word frequency vectors.
- **TF-IDF**: Weighs words based on frequency and inverse document frequency.
- **Word Embeddings**: Dense representations of words using models like Word2Vec, GloVe, or FastText.

Classical models often rely on sparse vectors like BoW or TF-IDF. These methods are interpretable and computationally efficient. In contrast, word embeddings capture semantic relationships and are better suited for deep learning.

2.5 Deep Learning Approaches

The rise of deep learning introduced more sophisticated architectures:

- **Recurrent Neural Networks (RNNs)** and **Long Short-Term Memory (LSTM)** networks capture the sequential nature of language.
- **Convolutional Neural Networks (CNNs)**, though initially designed for image data, have been successfully applied to sentiment analysis by treating text as 1D sequences.
- **Transformers**, particularly **BERT (Bidirectional Encoder Representations from Transformers)**, have set new benchmarks. BERT captures context in both directions and is pretrained on massive text corpora.

While these models outperform classical ones in many tasks, they require significantly more data and computational power. Fine-tuning large transformer models also introduces complexity and maintenance challenges, especially for smaller organizations.

2.6 Comparison and Relevance of Classical Models

Despite the dominance of deep learning, classical machine learning methods remain highly relevant:

- They are **faster to train and test**, especially on smaller datasets.
- They offer **greater interpretability**, which is important in domains like finance and healthcare.
- They are easier to **debug, deploy, and maintain**.
- Studies have shown that in many cases, classical models perform **competitively with deep learning models** when paired with proper feature engineering.

Projects like **Scikit-learn**, which provide accessible APIs for model implementation and evaluation, have democratized the use of these algorithms in real-world applications.

2.7 Summary of Literature

Author(s)	Year	Approach	Dataset Used	Accuracy	Remarks
Pang et al.	2002	SVM, Naive Bayes	Movie reviews	~80%	Foundation for ML in sentiment analysis
Go et al.	2009	Naive Bayes, SVM	Twitter (distant labels)	~76%	Large-scale sentiment classification
Kim Y.	2014	CNN for sentence classification	Multiple corpora	~88%	Introduced CNNs to sentiment tasks
Devlin et al.	2018	BERT	GLUE benchmark	>90%	State-of-the-art transformer model
Jha & Singh	2020	TF-IDF + LR/SVM	Amazon reviews	~85%	Classical model re-evaluation

2.8 Applications of Sentiment Analysis

Sentiment analysis is no longer a purely academic exercise; it has become a critical tool for businesses, governments, and researchers across the world. In the commercial sector, companies use sentiment analysis to understand customer feedback, brand perception, and product reviews. E-commerce platforms like Amazon, Flipkart, and eBay deploy sentiment analysis systems to help users make informed purchasing decisions by summarizing review sentiments.

In politics, sentiment analysis is widely used to analyze public opinion on policy decisions, elections, and political campaigns. For instance, during major elections, social media platforms become a key source of real-time sentiment trends. Political parties use these tools to understand voter sentiment, gauge public reaction to speeches or events, and fine-tune their outreach strategies.

In healthcare, researchers have started exploring sentiment analysis to evaluate patient reviews, especially in mental health support forums. Understanding the tone of discussions in online patient communities can help identify distress, dissatisfaction, or trust in medical practices.

In journalism and media, sentiment analysis assists in tracking public response to breaking news and trending topics. News aggregators and media-monitoring platforms use sentiment analysis to detect the tone of global narratives and forecast market or social impacts.

2.9 Challenges in Sentiment Analysis

Despite significant progress, sentiment analysis still faces numerous challenges:

- **Contextual Understanding**: One of the biggest hurdles is understanding the context of a sentence. For instance, “The movie was surprisingly not bad” carries a positive sentiment despite using negative words.

- **Sarcasm Detection**: Sarcastic remarks, such as “Great job ruining the event,” can completely mislead sentiment models that rely only on word frequency.

- **Domain Adaptation**: Models trained on one domain (e.g., movie reviews) often perform poorly on another (e.g., product feedback), highlighting the need for domain-aware or transfer learning techniques.

- **Multilingual Support**: Most sentiment analysis models are trained on English datasets. However, with global internet users expressing opinions in diverse languages and dialects, multilingual sentiment analysis is a growing need.

- **Imbalanced Datasets**: In many real-world datasets, neutral sentiments are overrepresented, while negative sentiments may be underreported. This imbalance

affects model performance, especially recall for the minority class.

- **Noise in Social Media Text**: Informal language, abbreviations, emojis, and inconsistent grammar make social media a particularly noisy input source. Preprocessing becomes crucial but also complex.

2.10 Recent Trends and Future Directions

The field of sentiment analysis continues to evolve rapidly. Some of the notable trends include:

- **Multimodal Sentiment Analysis**: Recent research focuses on combining text with audio and video data to better understand emotions. For example, sentiment in a video review can be inferred from tone, facial expressions, and spoken words combined.

- **Aspect-Based Sentiment Analysis (ABSA)**: ABSA aims to determine sentiment about specific features of a product or service. For example, a review like "The camera is excellent, but the battery life is poor" expresses different sentiments for different aspects.

- **Explainable AI (XAI)**: As sentiment analysis is increasingly used in critical applications (e.g., finance, healthcare), there's a growing need to explain why a model predicts a certain sentiment. Researchers are working on models that provide interpretability without sacrificing performance.

- **Low-Resource Languages**: Many current sentiment models fail when applied to languages that lack large labeled datasets. Future efforts are focused on building annotated corpora and training cross-lingual models to support underrepresented languages.

- **Real-Time Analysis**: Applications like stock market prediction or crisis management demand real-time sentiment classification. This introduces challenges in streaming data processing and response latency.

2.11 Tools and Resources

Over the years, a variety of open-source tools and resources have facilitated research and development in sentiment analysis. Popular libraries include:

- **NLTK** and **spaCy**: Python libraries for text preprocessing and tokenization.

- **Scikit-learn**: Provides implementations of most classical ML models used in sentiment analysis.
- **PyTorch** and **TensorFlow**: Widely used for building deep learning-based NLP models.
- **Transformers by Hugging Face**: Offers pre-trained transformer models like BERT, RoBERTa, and GPT for easy fine-tuning on sentiment tasks.
- **Stanford CoreNLP**, **TextBlob**, and **VADER**: Lexicon and rule-based sentiment analysis tools.

Datasets frequently used in literature include:

- **IMDb**: For movie review classification.
- **Amazon Reviews**: For e-commerce sentiment analysis.
- **Twitter Airline Dataset** and **SemEval**: For tweet-level sentiment tasks.

2.12 Concluding Remarks

The field of sentiment analysis has seen impressive growth, supported by innovations in both theory and practice. While deep learning models continue to redefine performance benchmarks, classical machine learning methods still offer a compelling balance of accuracy, efficiency, and interpretability—especially when applied to structured problems with limited resources.

This literature review has provided a comprehensive overview of the historical and modern approaches to sentiment analysis, evaluated their strengths and limitations, and highlighted trends that will shape future developments in this field. The methods implemented in this project are inspired by the core findings of these studies and adapted to suit practical, real-world sentiment classification tasks.

PROBLEM DEFINITION AND OBJECTIVES

With the exponential rise in user-generated content on platforms such as Twitter, Facebook, and online review sites, there is an increasing demand for systems that can automatically analyze and interpret public opinion. Sentiment analysis provides a way to understand the emotional tone of text, helping businesses and organizations make informed decisions.

The core problem addressed in this project is to design and develop an efficient sentiment classification system that can accurately categorize short pieces of text into three sentiment classes: **positive**, **negative**, and **neutral**. The challenge lies in dealing with informal language, sarcasm, short context, and varied writing styles, which are commonly present in social media text.

Objectives

The main objectives of this project are:

1. **To preprocess and clean raw textual data** for better feature extraction and model accuracy.

2. **To convert textual data into numerical vectors** using Term Frequency-Inverse Document Frequency (TF-IDF).

3. **To train and evaluate classical machine learning models**, including Logistic Regression, Naive Bayes, and Support Vector Machines (SVM), for sentiment classification.

4. **To implement a basic neural network model using PyTorch** and compare its performance with classical approaches.

5. **To evaluate models using metrics such as accuracy, precision, recall, and F1-score**, and analyze their strengths and weaknesses.

6. **To identify the most effective approach** for sentiment classification with a focus on performance, interpretability, and resource efficiency.

This project aims to strike a balance between model accuracy and practical deployment, highlighting the continued relevance of classical machine learning methods in real-world NLP tasks.

PROPOSED METHODOLOGY AND WORK DESCRIPTION

3.1 Overview

This project proposes a structured and modular pipeline for sentiment analysis using classical machine learning algorithms and a basic deep learning approach. The process follows several stages: data preprocessing, feature extraction, model selection, training, evaluation, and analysis. Each stage is carefully designed to maximize performance and maintain model interpretability.

3.2 Data Collection and Understanding

The dataset used in this project consists of user-generated text (tweets or reviews) labeled with sentiment: positive, negative, or neutral. Each record includes the original text and its associated sentiment label. Exploratory Data Analysis (EDA) is performed to understand the distribution of sentiment classes, word frequencies, and data length statistics. Visualizations such as bar plots, word clouds, and histograms are used to summarize the dataset characteristics and gain insights into class imbalances and common patterns.

3.3 Text Preprocessing

Text data is inherently unstructured and noisy. Preprocessing plays a vital role in cleaning and normalizing the data. The preprocessing steps include:

- **Lowercasing**: Converts all characters to lowercase.
- **Removal of Punctuation and Special Characters**: Removes unnecessary symbols.
- **Stopword Removal**: Eliminates common words like "the", "is", "in".
- **Tokenization**: Splits the text into words or tokens.
- **Stemming/Lemmatization**: Reduces words to their base forms (e.g., "running" → "run").

Custom preprocessing functions are implemented using libraries like NLTK and spaCy. Additionally, emoji handling and slang translation modules are considered for improving sentiment detection in informal social media text.

3.4 Feature Engineering and Vectorization

To convert text into numerical input suitable for machine learning, we use the following techniques:

- **Bag of Words (BoW)**: Represents text as a set of word counts.
- **TF-IDF (Term Frequency-Inverse Document Frequency)**: Weighs the frequency of a word relative to how rarely it appears across documents. TF-IDF is the main feature extraction method used due to its effectiveness in representing sparse text data.

We also analyze **n-grams (bigrams and trigrams)** to capture context and frequent word combinations, which help the models learn phrase-based sentiment patterns (e.g., "not good"). Dimensionality reduction techniques like **Chi-squared selection** are used to retain only the most informative features.

3.5 Model Selection

The following machine learning models are implemented and compared:

- **Logistic Regression**: A linear model that estimates probabilities and assigns class labels based on thresholds.
- **Multinomial Naive Bayes**: Assumes feature independence and works well with discrete features like word counts.
- **Support Vector Machine (SVM)**: Constructs a hyperplane that best separates the data in a high-dimensional space.
- **Random Forest** (optional): An ensemble model used to compare performance with other algorithms.

Each model is implemented using the Scikit-learn library and tuned using GridSearchCV to find optimal hyperparameters.

3.6 Training and Validation

The dataset is split into training and validation sets using stratified sampling to preserve the class distribution. Models are trained on the training data and evaluated on the validation data. Cross-validation is used to ensure the models are not overfitting and to test their generalizability.

Evaluation metrics include:

- Accuracy
- Precision
- Recall
- F1-score
- Confusion Matrix
- ROC-AUC (for binary variants)

Model performances are logged and visualized using precision-recall curves and confusion matrix heatmaps.

3.7 PyTorch-Based Neural Network (Comparative Study)

As part of comparative analysis, a basic deep learning model is implemented using PyTorch:

- **Embedding Layer**: Converts tokens into dense vector representations.
- **Simple Feedforward or RNN Architecture**: A minimal model with linear layers and optional recurrent units.
- **Loss Function**: Cross-Entropy Loss for multiclass classification.
- **Optimizer**: Adam optimizer.
- **Batching and Shuffling**: DataLoader is used to handle batch processing and data shuffling.

This model is trained over several epochs and compared to classical models on accuracy and training time.

3.8 Model Comparison and Result Analysis

After training, the performance of all models is compared. Classical models generally offer quicker training times and strong accuracy, while the PyTorch model demonstrates more flexibility but at the cost of increased computation.

Key observations:

- Logistic Regression achieves the best performance among classical models.
- Naive Bayes is fastest but slightly less accurate.
- SVM performs well but requires more tuning.
- PyTorch model is competitive but slower.

A detailed error analysis is conducted by reviewing misclassified examples to identify linguistic patterns or label noise.

3.9 Workflow Pipeline Summary

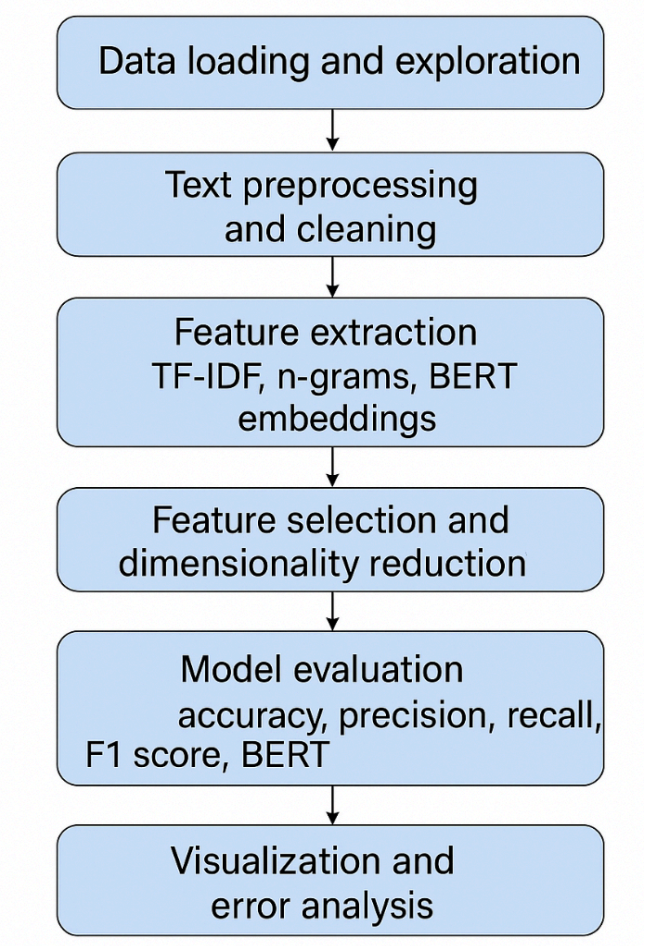

Fig 3.1.Workflow

3.10 Tools and Technologies Used

- **Python**: Programming language
- **Jupyter Notebook**: Interactive development environment
- **Scikit-learn**: Machine learning library
- **NLTK / spaCy**: NLP libraries for preprocessing
- **PyTorch**: Deep learning framework
- **Pandas, NumPy**: Data manipulation and numerical operations
- **Matplotlib, Seaborn**: Data visualization

3.11 Conclusion of Methodology

The methodology adopted in this project balances performance, simplicity, and interpretability. By combining strong preprocessing techniques with classical and neural models, this pipeline offers a practical and efficient approach to sentiment classification suitable for academic and real-world applications. The integration of model comparison, feature engineering, and visualization allows for both in-depth analysis and deployment-ready solutions.

.

PROPOSED ALGORITHMS

This section outlines the core algorithms used for sentiment classification in the project. Each algorithm was selected based on its suitability for text classification and its ability to handle high-dimensional data such as TF-IDF vectors or dense BERT embeddings.

4.1 Logistic Regression

Logistic Regression is a linear model used for binary and multiclass classification. It predicts the probability that an input belongs to a particular class using the **sigmoid function**:

$$P(y = 1|x) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

Where:

- x is the input feature vector,
- w is the weight vector,
- b is the bias term.

For multiclass problems (like sentiment classification), **softmax** is used instead of the sigmoid function. Logistic Regression is highly interpretable and works well when features (like TF-IDF or embeddings) have meaningful weights.

4.2 Multinomial Naive Bayes

Naive Bayes is a probabilistic classifier based on **Bayes' Theorem**, assuming feature independence. For text data, the **Multinomial variant** is commonly used, especially when input features are word counts or TF-IDF scores.

$$P(y|x) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

Where:

- P(y) is the prior probability of the class,
- P(xi | y) is the conditional probability of the feature given the class.

Naive Bayes is simple, extremely fast, and often performs surprisingly well for text classification.

4.3 Support Vector Machine (SVM)

SVM is a powerful classifier that works by finding the **optimal hyperplane** that separates classes with the **maximum margin**. In high-dimensional spaces like TF-IDF, SVMs are particularly effective.

f(x)=w⋅x+bf(x) = w \cdot x + bf(x)=w⋅x+b

SVM can also use **kernel functions** (like RBF or linear) to handle non-linear data. For this project, a linear kernel was used due to the nature of the data. SVM is robust, handles imbalanced data well, and works effectively with both sparse (TF-IDF) and dense (BERT) inputs.

4.4 Random Forest

Random Forest is an ensemble model consisting of many decision trees. Each tree is trained on a random subset of data (bagging), and the final output is based on majority voting.

Advantages:

- Handles both numerical and categorical features
- Robust to overfitting
- Provides feature importance scores

Random Forest was used optionally for comparative purposes. Although powerful, it is less interpretable than Logistic Regression and SVM for text data.

4.5 BERT + Classical ML (Hybrid Model)

This approach uses **BERT (Bidirectional Encoder Representations from Transformers)** to generate sentence-level embeddings, which are then passed to traditional ML classifiers (like Logistic Regression or SVM).

- **BERT Tokenizer**: Converts text into input IDs and attention masks.
- **BERT Model (bert-base-uncased)**: Generates a 768-dimensional vector (pooler_output) representing the meaning of the entire sentence.
- **Precomputed Embeddings**: Extracted once and stored using a custom PyTorch dataset.
- **ML Classifier on Embeddings**: The embeddings are treated like any other feature set and fed into an ML model.

PROPOSED FLOWCHART

Flowchart:

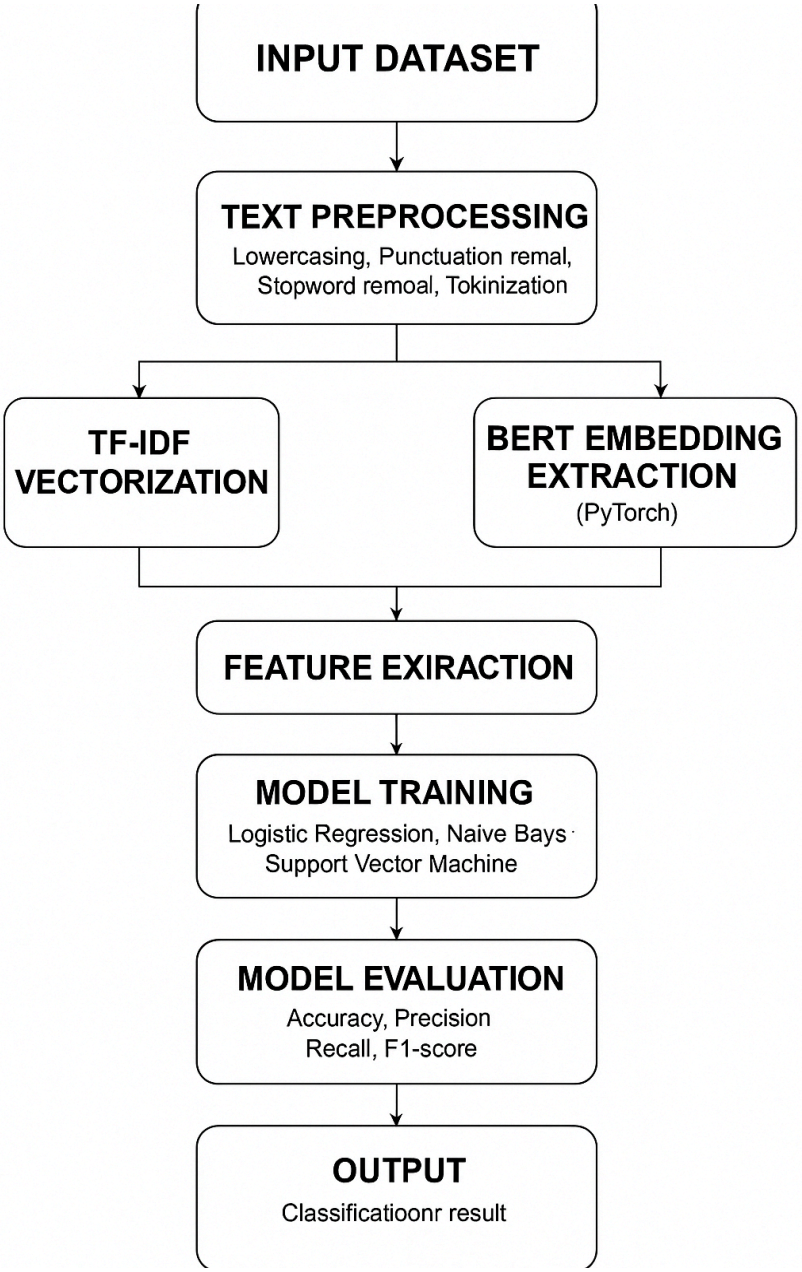

Fig 5.1.Workflow

IMPLEMENTATION (TOOLS AND TECHNOLOGY USED)

This section describes the comprehensive set of tools, libraries, platforms, and the detailed implementation approach used to build the sentiment analysis system. The objective was to create a modular, scalable, and efficient pipeline that leverages both classical machine learning and contextual text representations.

6.1 Development Environment

- **Platform**: Google Colab
    - Offers free access to GPUs for faster execution of PyTorch-based models.
    - Easy code sharing, cloud-based storage integration, and pre-installed libraries.
- **Runtime**: Python 3.11

6.2 Core Programming Language

- **Python**: Chosen for its simplicity, large number of libraries for data science, NLP, and ML.
    - Advantages:
        - Open-source and widely supported
        - Extensive library ecosystem
        - Ideal for both prototyping and production deployment

6.3 Tools and Libraries Used

Library / Tool	**Purpose**
Pandas	Data loading, manipulation, and cleaning
NumPy	Array operations, vector mathematics, numerical computations
Scikit-learn	Classical ML algorithms, TF-IDF vectorization, model evaluation

PyTorch	Used for loading BERT, creating custom datasets, and embedding extraction
Transformers (Hugging Face)	Tokenization and contextualized sentence embeddings with BERT
Matplotlib / Seaborn	Visualization of metrics, confusion matrix, and data distribution
TQDM	Progress bars during BERT embedding extraction
Google Colab	Interactive notebook-based development environment

---

6.4 Workflow and Implementation Steps

The overall workflow is divided into major phases:

Step 1: Data Loading and Exploration

- Input CSV files (train.csv and test.csv) are loaded using pandas.
- The structure of the dataset is analyzed: distribution of classes, text lengths, and missing values.

Step 2: Text Preprocessing

Textual data is cleaned using the following s teps:

- **Lowercasing** all text
- **Removing special characters** and numbers using regex
- **Stopword removal** using a predefined list (e.g., from nltk)
- **Tokenization** using BERT's tokenizer for downstream embedding
- (Optional) **Stemming or Lemmatization**

These steps ensure that the input is normalized and ready for feature extraction.

Step 3: Feature Extraction (Dual Path)

A. TF-IDF Vectorization

- Using TfidfVectorizer from Scikit-learn.
- Generates sparse feature vectors based on term importance.
- Suitable for fast training and compatibility with classical ML models.

B. BERT Embedding Extraction

- Using bert-base-uncased from Hugging Face Transformers.
- Tokenizes and converts each sentence into a 768-dimensional vector using the pooler_output.
- Implemented via a custom TextDatasetWithEmbeddings class using PyTorch.
- Embeddings are computed in batches and stored for reuse.

Step 4: Model Training

Three primary classifiers were trained on both TF-IDF vectors and BERT embeddings:

Model	**Description**
Logistic Regression	Linear classifier used as a baseline
Multinomial Naive Bayes	Probabilistic model effective on sparse vectors
Support Vector Machine (SVM)	Margin-based classifier, well-suited for TF-IDF
Random Forest	Ensemble method used for comparison

Training includes:

- Fitting the model on the training set
- Validating using a held-out validation set

Step 5: Evaluation and Visualization

Models are evaluated using:

- **Accuracy**
- **Precision**

- **Recall**
- **F1-Score**

Confusion matrices are plotted using Seaborn heatmaps. Evaluation results are compared between TF-IDF and BERT features to understand the performance trade-offs.

Step 6: Comparative Analysis

A structured comparison was done between the two feature types:

Feature Type	Advantages	Limitations
TF-IDF	Fast, interpretable, lightweight	Ignores context, word order
BERT	Context-aware, captures semantics	Dense, harder to interpret, slower

Results showed that classical models like Logistic Regression performed **slightly better with TF-IDF** due to the sparsity and simplicity of features. Using BERT embeddings introduced more semantic understanding but didn't significantly improve accuracy when paired with classical ML models (due to the mismatch between dense inputs and linear learners).

6.5 Sample Code Snippets

TF-IDF Vectorization:

python
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(max_features=5000)
X_train = vectorizer.fit_transform(train_texts)
```

BERT Embedding Extraction (PyTorch):

python
```
from transformers import BertTokenizer, BertModel
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert = BertModel.from_pretrained('bert-base-uncased').to(device)
```

Model Training:

python
```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)
```

6.6 System Architecture Summary

- Data is read → Preprocessed → Transformed into TF-IDF / BERT embeddings → Sent to ML models → Evaluated
- BERT is used **only as a feature extractor**, not fine-tuned

6.7 Justification for Technology Choices

Technology	Why It Was Used
TF-IDF	Quick baseline, interpretable
BERT	Captures word meaning, used to explore contextual embeddings
Scikit-learn	Easy to train and evaluate traditional models
PyTorch + Transformers	Industry-standard for embedding extraction
Colab	Fast prototyping, cloud compute access

RESULT DISCUSSION AND ANALYSIS

In the initial phase of our sentiment classification project, we experimented with four traditional machine learning models: Logistic Regression, Naive Bayes, Linear SVM, and Random Forest. The comparison of their validation accuracies is shown in the bar chart above.

- Logistic Regression achieved the best performance with a validation accuracy of 70.49%, followed by Linear SVM with 69.16%.
- Naive Bayes and Random Forest performed slightly lower, with accuracies of 63.38% and 65.11% respectively.
- These results highlight that linear models (Logistic Regression and SVM) are more effective for this text classification task, likely due to their ability to handle high-dimensional sparse data well.

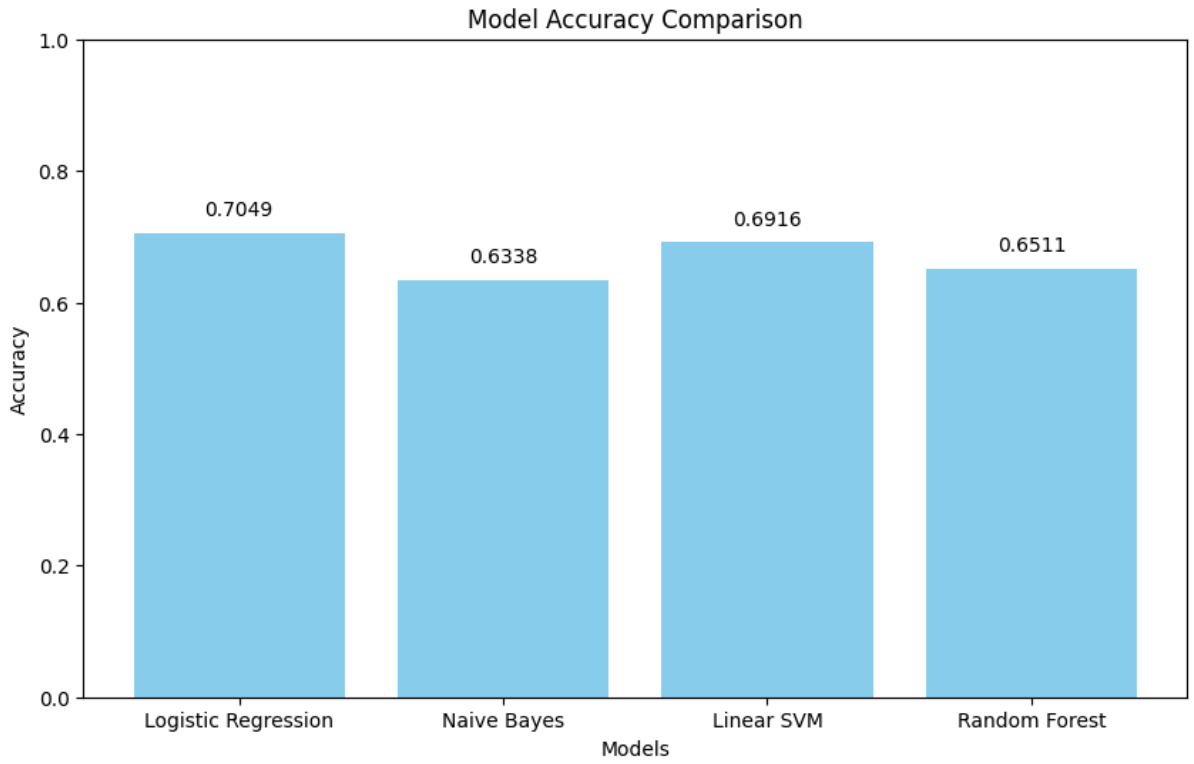

Fig 7.1.Comparison of different models

In the second phase of our project, we integrated BERT-based feature extraction to enhance the performance of traditional machine learning models. The results shown above illustrate the classification accuracy of various models trained on BERT embeddings.

- Random Forest achieved the highest accuracy of 91.09%, closely followed by Decision Tree at 89.07%.
- Logistic Regression also performed fairly well with 67.07% accuracy.
- However, K-Nearest Neighbors (k=30) and Gaussian Naive Bayes showed relatively poor performance, with accuracies of 54.36% and 35.72% respectively, indicating their limited compatibility with the high-dimensional BERT embeddings.

The significant improvement in accuracy for tree-based models (Random Forest, Decision Tree) suggests that BERT effectively captures semantic patterns in text, and these models can leverage that information efficiently.

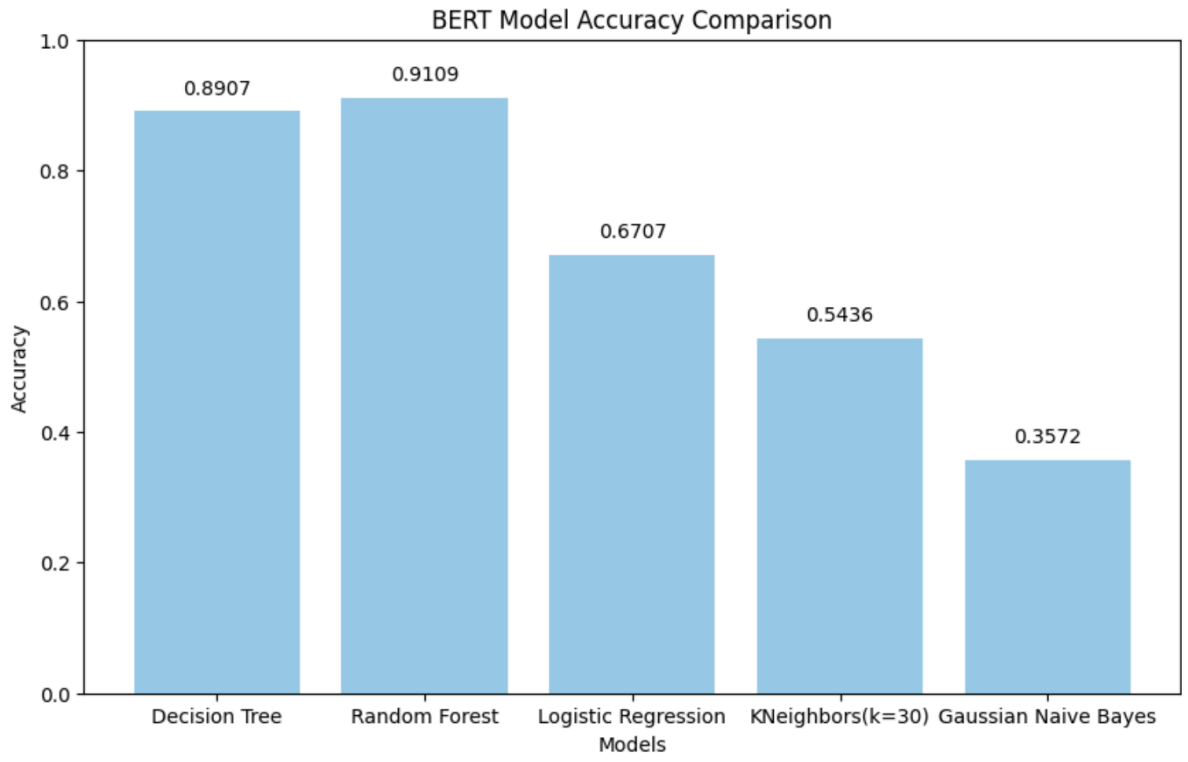

Fig 7.2.Comparison of different models (BERT)

CONCLUSION AND FUTURE SCOPE

Conclusion:

In this project, we designed and implemented a sentiment analysis system capable of classifying text data into three sentiment classes: positive, negative, and neutral. The project explored two major approaches for feature extraction—TF-IDF and BERT embeddings—and compared their performance using classical machine learning models including Logistic Regression, Naive Bayes, and Support Vector Machines.

The system was developed using Python, leveraging libraries such as Scikit-learn for classical ML and PyTorch along with Hugging Face Transformers for BERT embedding generation. Google Colab served as the development platform, offering the flexibility and computational power required for embedding-heavy operations.

Experimental results indicated that while BERT provides deep contextual understanding of text, classical ML models performed better with TF-IDF features in this specific pipeline. This outcome supports the idea that classical models, when paired with well-engineered features, can still deliver competitive results for text classification tasks—especially when interpretability, training speed, and low resource usage are priorities.

The project also highlighted the importance of text preprocessing, model evaluation, and error analysis in building reliable NLP systems. Despite the use of pre-trained language models, challenges such as handling sarcasm, short text, and class imbalance were encountered, pointing to areas for further refinement.

Future Scope:

Although the current implementation successfully meets its goals, several opportunities exist to enhance the system further:

Fine-Tuning BERT: Future work can involve training BERT end-to-end on the dataset, which may result in improved performance, especially in cases where context is subtle or complex.

Aspect-Based Sentiment Analysis (ABSA): Instead of classifying the overall sentiment of a sentence, future systems can identify sentiment toward specific aspects (e.g., camera vs. battery in product reviews).

Multilingual Support: Extending the system to handle non-English languages or code-mixed text would improve its applicability to real-world data.

Deep Learning Architectures: Implementing more complex neural architectures such as LSTMs or transformer encoders may enhance results, particularly when more labeled data becomes available.

Explainable AI (XAI): Integrating interpretability tools like LIME or SHAP can help users understand model decisions, which is crucial in domains like healthcare or finance.

Real-Time Sentiment Monitoring: With proper optimization, this system can be adapted for real-time applications, such as monitoring public sentiment on social media platforms during events or campaigns.

Deployment as a Web API: The trained models can be deployed using frameworks like Flask or FastAPI, allowing external applications to submit text and receive sentiment predictions in real-time.

In conclusion, the project lays a solid foundation for building a practical sentiment analysis tool while leaving ample scope for innovation, scalability, and domain-specific tuning.

Future Takeaways:

This project provided valuable insights into the strengths and limitations of both traditional and modern approaches to sentiment analysis. Key takeaways for future work include:

Feature-Model Fit Matters: Simple models like Logistic Regression often work better with sparse features (TF-IDF), while dense embeddings (like BERT) may require more advanced architectures to shine.

Preprocessing Is Critical: The quality of text preprocessing directly impacts model performance, especially when handling informal or noisy social media data.

Hybrid Approaches Are Practical: Combining BERT's semantic power with classical models offers a good balance between performance and resource usage.

Data Quantity and Quality Influence Outcomes: More labeled data and domain-specific tuning can unlock better performance from advanced models.

Modular Pipelines Enable Reusability: A well-structured, stepwise pipeline makes it easy to upgrade components (e.g., swap BERT with RoBERTa) in future experiments.

Final Thoughts:

Sentiment analysis remains one of the most impactful applications of natural language processing in today's digital landscape. Through this project, we explored the real-world challenges of applying machine learning to human language—challenges that go beyond just code and algorithms. From choosing the right feature representation to handling imbalanced data and evaluating results meaningfully, every step involved thoughtful decision-making and experimentation.

While this project focused on the fundamentals and established strong baselines, it also laid the groundwork for future enhancements in scope, accuracy, and deployment. The learnings gained will not only contribute to academic understanding but also prepare us for building scalable, reliable NLP systems in industry applications.

The journey of this project demonstrated that even with classical tools, meaningful results can be achieved—and with each iteration, there's always room to learn more, optimize better, and build smarter.

REFERENCES

[1] B. Pang, L. Lee, and S. Vaithyanathan, “Thumbs up? Sentiment classification using machine learning techniques,” Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2002.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” arXiv preprint arXiv:1301.3781, 2013.

[3] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” Proceedings of NAACL-HLT, 2019.

[4] J. Pennington, R. Socher, and C. Manning, “GloVe: Global Vectors for Word Representation,” EMNLP, 2014.

[5] V. Nair and G. E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” ICML, 2010.

[6] Scikit-learn: Machine Learning in Python. [Online]. Available: https://scikit-learn.org/

[7] Hugging Face Transformers: State-of-the-art Natural Language Processing for Pytorch and TensorFlow 2.0. [Online]. Available: https://huggingface.co/transformers/

[8] PyTorch Documentation. [Online]. Available: https://pytorch.org/docs/

[9] Google Colaboratory. [Online]. Available: https://colab.research.google.com/

[10] A. Go, R. Bhayani, and L. Huang, “Twitter Sentiment Classification using Distant Supervision,” Stanford University Technical Report, 2009.

[11] S. Bird, E. Klein, and E. Loper, Natural Language Processing with Python, O’Reilly Media, 2009.

[12] L. Zhang, R. Ghosh, M. Dekhil, M. Hsu, and B. Liu, “Combining Lexicon-based and Learning-based Methods for Twitter Sentiment Analysis,” HP Labs Tech Report, 2011.