

SENTIMENT ANALYSIS ON TWEETS OF TWITTER (X)

A PROJECT REPORT

Submitted by

SARAGADAM BHANU PRASAD

(Regd No. 21U45A4406)

Under the esteemed guidance of

Mr. A. VENKATESWARA RAO

(Associate Professor & Department Head of CSD & CSM)

in partial fulfillment of the award of the degree

of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE ENGINEERING - DATA SCIENCE



DEPARTMENT OF COMPUTER SCIENCE ENGINEERING – DS & AI/ML
DADI INSTITUTE OF ENGINEERING & TECHNOLOGY
(AN AUTONOMOUS INSTITUTE)

(Approved by A.I.C.T.E., New Delhi & Permanently Affiliated to JNTU GV)
Accredited by NAAC with 'A' Grade and Inclusion u/s 2(f) & 12(B) of UGC Act
An ISO 9001:2015, ISO 14001:2015 & ISO 45001:2018 Certified Institute.

NH-16, Anakapalle – 531002, Visakhapatnam, A.P.

(2020-2024)



**DEPARTMENT OF
COMPUTER SCIENCE ENGINEERING – DATA SCIENCE &
ARTIFICIAL INTELLIGENCE, MACHINE LEARNING
CERTIFICATE**

This is to certify that the project report entitled “**SENTIMENT ANALYSIS ON TWEETS OF TWITTER (X)**” submitted by SARAGADAM BHANU PRASAD (21U45A4406). In partial fulfilment of the requirements for award of the Degree of **Bachelor of Technology in Computer Science Engineering – Data Science**, from **Dadi Institute of Engineering & Technology(A)**, Anakapalle affiliated to **JNTUGV**, **accredited by NAAC with 'A' grade**, is a record of bona fide work carried out by them under my guidance and supervision.

Mr. A. VENKATESWARA RAO
(ASSOCIATE PROFESSOR)
(PROJECT GUIDE)

Mr. A. VENKATESWARA RAO
(ASSOCIATE PROFESSOR)
(HEAD OF DEPARTMENT)

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

I would like to express my gratitude to my project guide, **Mr. A. VENKATESWARA RAO**, Associate Professor & Head of the Department of the Computer Science Engineering with Data Science Specialization, for his/her guidance and assistance throughout the development of this project. Without her supervision, support, and encouragement, I would not have gained awareness of many new things during my project.

I convey my heartfelt thanks to **Mr. A. Venkateswara Rao**, Associate Professor & Head of the Department of CSD&AIML, for motivating me to successfully complete the project.

I would like to thank Principal **Dr. R. Vaikunta Rao**, Dadi Institute of Engineering & Technology, for providing the necessary facilities to carry out my project work successfully.

I would like to thank Dean of Academics, **Dr. K. Parvathi**, Dadi Institute of Engineering & Technology, for providing essential facilities for my project.

I express my gratitude to the Teaching and Non-teaching Staff of the Department of **Computer Science Engineering – Data Science & Artificial Intelligence, Machine Learning**, who have been directly and indirectly involved in this journey, for their encouragement in completing my project.

I would like express my deep sense of gratitude to **Honorable Chairman, Sri Dadi Ratnakar**, of Dadi Institute of Engineering & Technology, for providing necessary facilities to carry out my project work successfully.

Endeavours over a long period can also be successful through constant effort and encouragement. I wish to take this opportunity to express my deep gratitude to all the people who have extended their cooperation in various ways during my project. It is my pleasure to acknowledge the help of all those respected individuals.

by

SARAGADAM BHANU PRASAD
(21U45A4406)

DECLARATION

I hereby declare that the project entitled “**Sentiment Analysis on Tweets of Twitter (X)**” is submitted in partial fulfilment of the requirements for the award of Bachelor of Technology in **Computer Science Engineering – Data Science & Artificial Intelligence, Machine Learning** under esteemed supervision of **Mr. A. VENKATESWARA RAO, Associate Professor**. This is a record of work carried out by me and results embodied in this project report have not been submitted to any other university for the award of any Degree.

by

SARAGADAM BHANU PRASAD
(21U45A4406)

ABSTRACT

The internet's widespread usage and rapid technological advancements have revolutionized the way we communicate and interact with each other, leading to an unprecedented increase in available data, especially unstructured real-time data. Platforms like Twitter and Facebook have emerged as major players in facilitating online learning, opinion sharing, and global communication over the last decade. Among these platforms, Twitter has particularly gained prominence due to its real-time nature, brevity in messages, and wide user base, making it a valuable source of information and a hub for discussions on various topics.

Given the dynamic and unstructured nature of opinions expressed on Twitter, sentiment analysis has garnered substantial attention in both academic research and industry applications. Sentiment analysis, also known as opinion mining, involves the use of natural language processing, text analysis, and computational linguistics to systematically identify, extract, and quantify subjective information from textual data. This project aims to apply sentiment analysis techniques to Twitter data to gain insights into individual opinions on specific discussions, topics, or events.

Sentiment analysis plays a crucial role in classifying tweets as positive, negative, or occasionally neutral, providing valuable insights into large-scale discussions happening on the platform. By analyzing the sentiment of tweets, one can understand public opinion, gauge the overall sentiment towards a particular topic or event, and even predict trends or sentiment shifts over time.

The project employs a variety of machine learning algorithms, including Logistic Regression, LightGBM (Gradient Boosting Machine), XGBoost (Extreme Gradient Boosting), and Support Vector Machine (SVM), to analyze Twitter data and perform sentiment analysis. These algorithms are well-suited for handling large datasets, extracting relevant features from text data, and classifying tweets into different sentiment categories with high accuracy.

Assessing the accuracy of these machine learning algorithms is essential to determine the most effective approach for sentiment analysis on the Twitter platform. Accuracy metrics such as precision, recall, F1-score, and area under the receiver operating characteristic curve (AUC-ROC) can be used to evaluate the performance of each

algorithm and identify the one that achieves the highest accuracy in sentiment classification.

Moreover, the project may also explore techniques such as feature engineering, word embeddings, and ensemble methods to further improve the performance of sentiment analysis models on Twitter data. Additionally, considering the evolving nature of language and the constant influx of new terms and expressions on social media platforms, the project may incorporate techniques for handling slang, sarcasm, and contextual ambiguity in tweets to enhance the accuracy of sentiment analysis results.

Overall, sentiment analysis on Twitter data holds immense potential for understanding public opinion, identifying emerging trends, and informing decision-making processes in various domains, including marketing, politics, public health, and social research. By leveraging machine learning algorithms and advanced natural language processing techniques, this project aims to unlock valuable insights from the vast amount of unstructured textual data available on the Twitter platform.

TABLE OF CONTENTS

1. INTRODUCTION
2. LITERATURE REVIEW
3. MACHINE LEARNING CONCEPTS
4. METHODOLOGY
5. SYSTEM DESIGN
6. IMPLEMENTATION
7. RESULTS
8. CONCLUSION & FUTURE SCOPE
9. REFERENCES
10. APPENDIX
11. RESEARCH PAPER

LIST OF CONTENTS

ABSTRACT	V - VI
LIST OF FIGURES	XI - XII
1. INTRODUCTION	1 - 4
1.1. Introduction	
1.2. Project Objectives	
1.3. Problem Statement	
1.4. Scope	
1.5. Methodology	
1.6. Chapter Review	
2. LITERATURE REVIEW	5 - 7
3. MACHINE LEARNIG CONCEPTS	8 - 25
3.1. Introduction to Regression Concepts	
3.1.1. Logistic Regression	
3.2. XGBoost Algorithm	
3.3. Support Vector Machine (SVM) Algorithm	
3.4. Random Forest Algorithm	
3.5. Naïve Bayes Algorithm	
3.6. Data Information	
3.6.1. Dataset Parameters	
3.6.2. Analysis of Data	
4. METHODOLOGY	26 - 32
4.1. Existing System	
4.1.1. Limitation of Existing System	
4.2. Proposed System	
4.2.1. Advantages of Existing	
4.3. Functional Requirements	
4.3.1. Software Requirements	

4.3.2. Hardware Requirements	
4.4. Non-Functional Requirements	
5. SYTEM DESIGN	33 - 43
5.1. Unified Modeling Language	
5.1.1. Use Case Diagram	
5.1.2. Sequence Diagram	
5.1.3. Activity Diagram	
5.1.4. System Flow Diagram	
5.1.5. Class Diagram	
5.1.6. Deployment Diagram	
6. IMPLEMENTATION	44 - 71
6.1. Data Collection	
6.2. Data Pre-processing	
6.2.1. Data Cleaning	
6.2.2. Label Encoder	
6.2.3. To lower-case text	
6.2.4. Remove URLs	
6.2.5. Removing Punctuations	
6.2.6. Removing Stopwords	
6.2.7. Removing Repeating Characters	
6.2.8. Tokenization	
6.2.9. Stemming	
6.2.10. Lemmatization	
6.3. Feature Selection & Engineering	
6.3.1. TF-IDF Vectorizer scikit-learn – First Approach	
6.3.2. TF-IDF Vectorizer scikit-learn – Second Approach	
6.4. Split Training & Test Dataset	

6.5. Model Selection	
6.6. Model Evaluation	
7. RESULTS	72 - 77
8. CONCLUSION & FUTURE SCOPE	78 - 80
8.1. Conclusion	
8.2. Future Scope	
REFERENCES	81
APPENDIX	82 - 89

LIST OF FIGURES

Fig. No.	Description	Page No.
1.1	Product Function	4
3.1	Regression Sigmoid Function	11
3.2	XGBoost Mathematical Representation	17
3.3	Support Vector Machine	18
3.4	Random Forest	21
5.1	Use Case Diagram	34
5.2	Sequence Diagram	35
5.3	Activity Diagram	37
5.4	System Flow Diagram	39
5.5	Class Diagram	42
5.6	Deployment Diagram	43
6.1	Loading and Printing the Dataset	44
6.2	Attributes of Sentiment Analysis Dataset	45
6.3	Data Cleaning (Check for Missing Values and Duplicates	45
6.4	Label Encoding for the target values	46
6.5	Converting text into lowercase	46
6.6	Removing URLs from the text	48
6.7	Punctuation Removal	49
6.8	Removing Stopwords	50
6.9	Repeating words are removed	51
6.10	Tokenization of Words	53
6.11	Stemming of Text	54
6.12	Lemmatization of Text	56
6.13	Vectorizing the X_train and glimpse of IDF values	63

6.14	Display IDF values of the words	65
6.15	Training & Testing Data	66
6.16	Machine Learning Models used	67
6.17	Model Evaluation	69
6.18	Logistic Regression Model Evaluation	69
6.19	XGBoost Model Evaluation	70
6.20	SVM Model Evaluation	70
6.21	Random Forest Model Evaluation	71
6.22	Naïve Bayes Model Evaluation	71
7.1	Home Page	73
7.2	Single Sentiment Analysis Page	73
7.3	Single Sentiment Analysis Result Page	74
7.4	File Sentiment Analysis Page	74
7.5	Result Page – Pie Graph	75
7.6	Result Page – Bar Graph	75
7.7	Result Page – Positive Comments Section	76
7.8	Result Page – Negative Comments Section	76
7.9	Result Page – Neutral Comments Section	77

CHAPTER 1

INTRODUCTION

1.1 Overview

In today's digital age, social media platforms like Twitter have become treasure troves of information, offering valuable insights into public opinions, trends, and sentiments. Sentiment analysis on Twitter data has emerged as a powerful tool for businesses, researchers, and policymakers alike to understand public perception, gauge customer satisfaction, and predict market trends. This project aims to leverage sentiment analysis techniques to analyze a dataset of tweets, providing meaningful insights into the sentiments expressed by users on various topics.

Sentiment analysis, also known as opinion mining, is the process of computationally identifying and categorizing opinions expressed in text data. It involves analyzing the polarity of the text, i.e., determining whether the sentiment expressed is positive, negative, or neutral. With the exponential growth of social media platforms, sentiment analysis has gained prominence as a means to extract valuable insights from vast amounts of user-generated content.

For this project, a dataset of tweets was collected using the Twitter API, spanning a specific time period and focusing on relevant topics of interest. The dataset comprises text data along with metadata such as timestamps, user information, and tweet IDs. Preprocessing techniques were applied to clean the data, including removing special characters, URLs, and stopwords, and tokenizing the text for further analysis.

Various techniques can be employed for sentiment analysis, ranging from rule-based approaches to machine learning algorithms. In this project, a hybrid approach was adopted, combining rule-based sentiment analysis with machine learning models. Rule-based methods involve creating dictionaries of sentiment-laden words and assigning polarities to them, while machine learning models learn from labeled data to classify sentiments accurately.

To train the sentiment analysis model, features were extracted from the preprocessed tweet text, including word frequency, n-grams, and word embeddings. These features were then used to train machine learning classifiers such as Support Vector Machines (SVM), Naive Bayes, or Recurrent Neural Networks (RNNs). The dataset was split into training and testing sets to evaluate the performance of the trained models.

To assess the performance of the sentiment analysis models, various evaluation metrics were employed, including accuracy, precision, recall, and F1-score. These metrics provide insights into the model's ability to correctly classify sentiments and distinguish between positive, negative, and neutral tweets. Additionally, techniques such as cross-validation were used to ensure the robustness and generalization of the models.

Upon training and evaluating the sentiment analysis models, meaningful insights were derived from the tweet dataset. Analysis of sentiment trends over time revealed fluctuations in public opinion, sentiment spikes around specific events or topics, and patterns in user sentiments across different demographics. Furthermore, sentiment analysis enabled the identification of influential users, sentiment drivers, and emerging trends within the Twitter community.

The insights gained from sentiment analysis on Twitter data have diverse applications across various domains. Businesses can utilize these insights for brand monitoring, customer feedback analysis, and reputation management. Researchers can leverage sentiment analysis to study public opinion on social and political issues, track sentiment towards healthcare initiatives, or analyze consumer sentiment in financial markets. Future directions for this project include incorporating advanced natural language processing (NLP) techniques, exploring deep learning architectures, and enhancing the scalability and real-time capabilities of the sentiment analysis system.

Sentiment analysis on Twitter data offers valuable insights into public sentiments, providing a window into the collective thoughts, feelings, and attitudes of users. By leveraging computational techniques and machine learning algorithms, this project demonstrates the potential of sentiment analysis for extracting actionable insights from social media data. As the digital landscape continues to evolve, sentiment analysis will remain a crucial tool for understanding and harnessing the power of user-generated content on platforms like Twitter.

1.1 Problem Statement

In the era of digitalization, social media platforms have become indispensable sources of information, offering a vast pool of user-generated content that reflects public opinions, sentiments, and trends. Among these platforms, Twitter stands out as a hub for real-time conversations, making it a prime candidate for sentiment analysis. However, with the sheer volume of tweets being generated every second, extracting meaningful insights from this data deluge poses a significant challenge. Therefore, the problem statement of this project is to develop an effective sentiment analysis system capable of processing and analyzing tweets at scale, with the goal of uncovering valuable insights into public sentiment on various topics.

1.3 Problem Statement

The primary challenge lies in the unstructured nature of Twitter data, which is characterized by short and often noisy text snippets containing slang, abbreviations, hashtags, and emoticons. These characteristics make traditional natural language processing techniques less effective and necessitate the development of specialized methods for sentiment analysis.

Another challenge is the dynamic and evolving nature of language usage on Twitter. New slang terms, memes, and cultural references constantly emerge, requiring the sentiment analysis system to adapt and learn from the latest trends in language usage.

Furthermore, the vastness of Twitter data poses scalability challenges. As the volume of tweets continues to grow exponentially, the sentiment analysis system must be capable of processing large datasets efficiently while maintaining high accuracy and reliability. Additionally, the problem statement encompasses the need to address the nuances of sentiment expression on Twitter. Users often employ sarcasm, irony, and ambiguity in their tweets, making it challenging to accurately determine the underlying sentiment. The sentiment analysis system must be able to detect and interpret these nuances to avoid misclassification.

Moreover, the sentiment analysis system should be customizable to cater to different domains and topics. Sentiment expressions may vary across industries, cultures, and geographic regions, necessitating the adaptation of the system's lexicons and models to specific contexts.

1.4 Scope

This project is centered around to predict the sentiment of the tweets which are extracted for the twitter using various machine learning models, focusing solely on training the model with available dataset. The scope includes the development and implementation of predictive models. While the project won't incorporate other language sentences due to lack of the training data, it remains dedicated to providing sentimental analysis in the business industry. The scope involved through model evaluation, comparisons with existing product and the delivery of a reliable tool for predicting the sentiments of the tweets in the twitter

1.5 Methodology

The methodology adopted for this project is designed to create a reliable and interpretable predictive model for the prediction of the unstructured data i.e., tweets of the twitter when a file containing the tweets is uploaded. The project approach integrates fundamental steps from data collection to model training, emphasizing the use of machine learning models. The goal is capturing the complex relationship between the tweets which are sentences and the label sentiment which are positive, negative and neutral, while ensuring the model's practical utility and robustness. The understanding of the tweets through mathematical models gives us the relation between the tweets and the sentiment labels. The following figure shows the key outline steps:

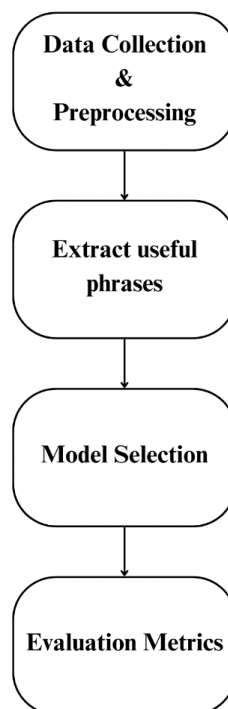


Fig 1.1 Product Function

CHAPTER 2

LITERATURE REVIEW

In recent years, numerous researchers have made significant strides in the realm of sentiment analysis on Twitter. Initially, the focus was on binary classification, categorizing opinions or reviews into either positive or negative classes exclusively.

Pak and Paroubek (2010) introduced a model aimed at classifying tweets as objective, positive, or negative. They compiled a Twitter corpus by utilizing the Twitter API and automatically labeling tweets using emoticons. Utilizing this corpus, they developed a sentiment classifier employing the multinomial Naive Bayes method, incorporating features such as N-grams and POS-tags. However, their training set was limited as it only included tweets with emoticons.

Parikh and Movassate (2009) implemented two models—a Naive Bayes bigram model and a Maximum Entropy model—for tweet classification. They found the Naive Bayes classifiers to outperform the Maximum Entropy model.

Go and L. Huang (2009) proposed a solution for sentiment analysis on Twitter data utilizing distant supervision. Their training data consisted of tweets with emoticons, serving as noisy labels. They constructed models employing Naive Bayes, MaxEnt, and Support Vector Machines (SVM), with feature spaces comprising unigrams, bigrams, and POS. Their findings indicated that SVM outperformed other models, with unigrams proving more effective as features.

Barbosa et al. (2010) devised a two-phase automatic sentiment analysis method for tweet classification. Initially, tweets were categorized as objective or subjective, followed by further classification of subjective tweets as positive or negative. Their feature space included retweets, hashtags, links, punctuation, and exclamation marks, in conjunction with features like the prior polarity of words and POS.

Bifet and Frank (2010) utilized Twitter streaming data from the Firehose API, providing real-time access to all publicly available user messages. They experimented with multinomial Naive Bayes, stochastic gradient descent, and the Hoeffding tree,

concluding that the SGD-based model, with an appropriate learning rate, outperformed the others.

Agarwal et al. (2011) developed a 3-way model for sentiment classification—categorizing sentiments into positive, negative, and neutral classes. They explored models such as the unigram model, a feature-based model, and a tree kernel-based model. Notably, their tree kernel-based model outperformed the others, with features combining the prior polarity of words with their POS tags proving crucial.

Davidov et al. (2010) proposed an approach utilizing Twitter user-defined hashtags in tweets for sentiment classification. They employed punctuation, single words, n-grams, and patterns as different feature types, amalgamating them into a single feature vector for classification using the K-Nearest Neighbor strategy.

Liang et al. (2014) utilized the Twitter API to gather tweet data categorized into three different domains (camera, movie, mobile). They implemented a Unigram Naive Bayes model, employing the simplifying independence assumption. Additionally, they utilized Mutual Information and Chi-square feature extraction methods to eliminate unnecessary features, ultimately predicting the sentiment orientation of tweets as positive or negative.

Pablo et al. introduced variations of Naive Bayes classifiers for detecting the polarity of English tweets. They built Baseline and Binary variants, with the former trained to classify tweets as positive, negative, or neutral, and the latter making use of a polarity lexicon to classify tweets as positive or negative, disregarding neutral tweets. Features considered included Lemmas, Polarity Lexicons, Multiword from various sources, and Valence Shifters.

Turney et al. employed a bag-of-words method for sentiment analysis, neglecting word relationships, and representing documents merely as collections of words. Sentiment for the entire document was determined by aggregating the sentiments of individual words.

Kamps et al. utilized the lexical database WordNet to ascertain the emotional content of words across different dimensions. They devised a distance metric on WordNet to determine the semantic polarity of adjectives.

Xia et al. employed an ensemble framework for sentiment classification by combining various feature sets and classification techniques. They utilized two types of feature sets—Part-of-speech information and Word relations—along with three base classifiers—Naive Bayes, Maximum Entropy, and Support Vector Machines. Ensemble approaches such as fixed combination, weighted combination, and Meta-classifier combination were applied for sentiment classification, resulting in improved accuracy.

Luo et al. highlighted the challenges and efficient techniques for mining opinions from Twitter tweets, emphasizing the difficulties posed by spam and widely varying language.

3.1 Introduction to Regression Concept

Regression analysis plays a fundamental role in predictive modeling within machine learning, as it delves into the connections between variables and enables forecasts of continuous outcomes. Essentially, regression involves identifying patterns within data by fitting a mathematical model to observed data points, facilitating predictions of future outcomes based on input variables. This introduction provides a foundational understanding of regression within the context of machine learning, covering its principles, applications, and common techniques.

In its essence, regression tasks revolve around predicting a continuous target variable by considering one or more independent variables, also referred to as features. The objective is to discern the relationship between these features and the target variable from historical data, thus enabling the model to make accurate predictions for new, unseen data.

For instance, in sentiment analysis tasks, regression techniques can be utilized to predict sentiment scores based on various textual features extracted from social media posts, product reviews, or customer feedback. These features may include word frequencies, sentiment lexicons, syntactic patterns, and contextual embeddings.

Common techniques in sentiment analysis regression include linear regression, ridge regression, lasso regression, support vector regression (SVR), and ensemble methods like random forest regression and gradient boosting regression. Each method offers distinct advantages depending on the nature of the text data and the specific sentiment analysis task at hand.

By applying these regression techniques, analysts can build predictive models that effectively capture the nuanced relationships between textual features and sentiment scores. These models empower decision-makers to gain valuable insights into public opinion, customer satisfaction, and market trends, thus facilitating informed decision-making in various domains such as marketing, brand management, and customer

service. Regression analysis is a statistical method employed to scrutinize the relationship between a dependent variable (often denoted as Y) and one or more independent variables (often denoted as X). The primary objective of regression analysis is to comprehend how the value of the dependent variable changes when one or more independent variables are varied.

Basic Concepts of Regression:

- **Dependent Variable (Y):** This is also known as the response variable or target variable, representing the variable being predicted or explained in regression analysis. It is the focal point of interest.
- **Independent Variables (X):** These are also referred to as predictor variables or features, hypothesized to predict or explain changes in the dependent variable. In simple linear regression, there's only one independent variable, whereas in multiple regression, there are two or more independent variables.
- **Regression Equation:** This equation depicts the relationship between the dependent variable and one or more independent variables. In simple linear regression, the equation takes the form $Y = \beta_0 + \beta_1 X + \epsilon$, where β_0 is the intercept, β_1 is the slope coefficient, X is the independent variable, and ϵ is the error term. In multiple regression, the equation extends to include multiple independent variables: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_n X_n + \epsilon$, where X_1, X_2, \dots, X_n are the independent variables, and $\beta_0, \beta_1, \dots, \beta_n$ are the corresponding coefficients.
- **Residuals:** These are the discrepancies between the observed values of the dependent variable and the values predicted by the regression equation. Residuals signify the error or unexplained variance in the model and are utilized to evaluate the goodness-of-fit of the regression model.
- **Goodness-of-Fit:** Measures of goodness-of-fit assess how well the regression model aligns with the observed data. Common measures include the coefficient of determination (R-squared), indicating the proportion of variance in the dependent variable explained by the independent variables, and the root mean squared error (RMSE), which gauges the average prediction error of the model.

3.1.1 Logistic Regression

Logistic regression is used for binary classification where we use sigmoid function, that takes input as independent variables and produces a probability value between 0 and 1.

For example, we have two classes Class 0 and Class 1 if the value of the logistic function for an input is greater than 0.5 (threshold value) then it belongs to Class 1 otherwise it belongs to Class 0. It's referred to as regression because it is the extension of linear regression but is mainly used for classification problems.

Logistic Function – Sigmoid Function

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1. The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the “S” form.
- The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Working of the Logistic Regression

The logistic regression model transforms the linear regression function continuous value output into categorical value output using a sigmoid function, which maps any real-valued set of independent variables input into a value between 0 and 1. This function is known as the logistic function.

Let the independent input features be:

$$X = \begin{bmatrix} x_{11} & \dots & x_{1m} \\ x_{21} & \dots & x_{2m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nm} \end{bmatrix}$$

and the dependent variable is Y having only binary value i.e., 0 or 1.

$$Y = \begin{cases} 0 & \text{if Class 1} \\ 1 & \text{if Class 2} \end{cases}$$

then, apply the multi-linear function to the input variables X.

$$z = (\sum_{i=1}^n w_i x_i) + b$$

Here x_i is the i^{th} observation of X, $w_i = [w_1, w_2, w_3, \dots, w_m]$ is the weights or Coefficient, and b is the bias term also known as intercept. simply this can be represented as the dot product of weight and bias.

$$z = w \cdot X + b$$

Sigmoid Function

Now we use the sigmoid function where the input will be z and we find the probability between 0 and 1. i.e., predicted y.

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

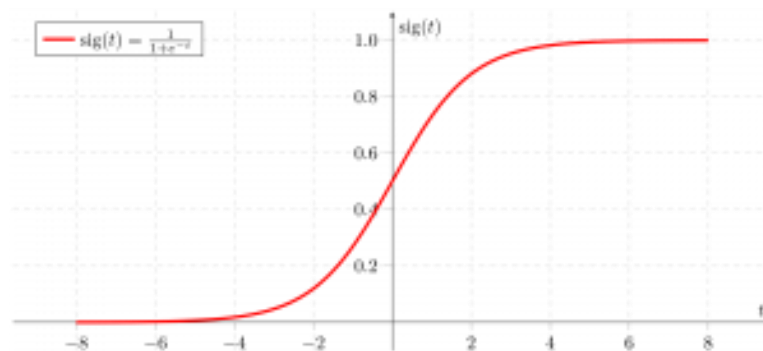


Fig 3.1 Regression Sigmoid function

As shown above, the figure sigmoid function converts the continuous variable data into the probability i.e., between 0 and 1.

- $\sigma(z)$ tends towards 1 as $z \rightarrow \infty$
- $\sigma(z)$ tends towards 0 as $z \rightarrow -\infty$
- $\sigma(z)$ is always bounded between 0 and 1

where the probability of being a class can be measured as:

$$P(y = 1) = \sigma(z)$$

$$P(y = 0) = 1 - \sigma(z)$$

Likelihood Function for Logistic Regression

The predicted probabilities will be:

- for $y=1$ The predicted probabilities will be: $p(X; b, w) = p(x)$
- for $y = 0$ The predicted probabilities will be: $1 - p(X; b, w) = 1 - p(x)$

$$L(b, w) = \prod_{i=1}^n p(x_i)^{y_i} (1 - p(x_i))^{1-y_i}$$

Taking natural logs on both sides

$$\begin{aligned} \log(L(b, w)) &= \sum_{i=1}^n y_i \log p(x_i) + (1 - y_i) \log(1 - p(x_i)) \\ &= \sum_{i=1}^n y_i \log p(x_i) + \log(1 - p(x_i)) - y_i \log(1 - p(x_i)) \\ &= \sum_{i=1}^n \log(1 - p(x_i)) + \sum_{i=1}^n y_i \log \frac{p(x_i)}{1 - p(x_i)} \\ &= \sum_{i=1}^n -\log 1 - e^{-(w \cdot x_i + b)} + \sum_{i=1}^n y_i (w \cdot x_i + b) \\ &= \sum_{i=1}^n -\log 1 + e^{w \cdot x_i + b} + \sum_{i=1}^n y_i (w \cdot x_i + b) \end{aligned}$$

Gradient of the log-likelihood function

To find the maximum likelihood estimates, we differentiate w.r.t w ,

$$\begin{aligned}
\frac{\partial J(l(b, w))}{\partial w_j} &= - \sum_{i=1}^n \frac{1}{1 + e^{w \cdot x_i + b}} e^{w \cdot x_i + b} x_{ij} + \sum_{i=1}^n y_i x_{ij} \\
&= - \sum_{i=1}^n p(x_i; b, w) x_{ij} + \sum_{i=1}^n y_i x_{ij} \\
&= \sum_{i=1}^n (y_i - p(x_i; b, w)) x_{ij}
\end{aligned}$$

We can evaluate the performance of a logistic regression model using several metrics:

- **Accuracy:** Accuracy provides the proportion of correctly classified instances among all instances.

$$Accuracy = \frac{TruePositives + TrueNegatives}{Total}$$

- **Precision:** Precision focuses on the accuracy of positive predictions, i.e., the proportion of correctly predicted positive instances among all instances predicted as positive.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

- **Recall (Sensitivity or True Positive Rate):** Recall measures the proportion of correctly predicted positive instances among all actual positive instances.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

- **F1 Score:** The F1 score is the harmonic mean of precision and recall, providing a balance between the two metrics.

$$F1Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

These metrics collectively provide insights into the performance of the logistic regression model, considering both its ability to classify instances correctly and its performance on positive instances.

3.2 XGBoost Algorithm

XGBoost is an optimized distributed gradient boosting library designed for efficient and scalable training of machine learning models. It is an ensemble learning method that combines the predictions of multiple weak models to produce a stronger prediction. XGBoost stands for “Extreme Gradient Boosting” and it has become one of the most popular and widely used machine learning algorithms due to its ability to handle large datasets and its ability to achieve state-of-the-art performance in many machine learning tasks such as classification and regression.

One of the key features of XGBoost is its efficient handling of missing values, which allows it to handle real-world data with missing values without requiring significant pre-processing. Additionally, XGBoost has built-in support for parallel processing, making it possible to train models on large datasets in a reasonable amount of time.

XGBoost can be used in a variety of applications, including Kaggle competitions, recommendation systems, and click-through rate prediction, among others. It is also highly customizable and allows for fine-tuning of various model parameters to optimize performance.

XGBoost stands for Extreme Gradient Boosting, which was proposed by the researchers at the University of Washington. It is a library written in C++ which optimizes the training for Gradient Boosting.

XGBoost is an implementation of Gradient Boosted decision trees. XGBoost models majorly dominate in many Kaggle Competitions.

In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.

Mathematics behind XGBoost

Before beginning with mathematics about Gradient Boosting, here's a simple example of a CART that classifies whether someone will like a hypothetical computer game X. The example of tree is below:

The prediction scores of each individual decision tree then sum up to get if you look at the example, an important fact is that the two trees try to *complement* each other. Mathematically, we can write our model in the form:

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), f_k \in \mathcal{F}$$

where, K is the number of trees, f is the functional space of F, F is the set of possible CARTs. The objective function for the above model is given by:

$$obj(\theta) = \sum_i^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

where, first term is the loss function and the second is the regularization parameter. Now, instead of learning the tree all at once which makes the optimization harder, we apply the additive strategy, minimize the loss what we have learned and add a new tree which can be summarized below:

$$\begin{aligned}\hat{y}_i^{(0)} &= 0 \\ \hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\ \hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\ &\dots \\ \hat{y}_i^{(t)} &= \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)\end{aligned}$$

The objective function of the above model can be defined as:

$$\begin{aligned}obj^{(t)} &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^t \Omega(f_i) \\ &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant \\ obj^{(t)} &= \sum_{i=1}^n (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^t \Omega(f_i) \\ &= \sum_{i=1}^n [2(\hat{y}_i^{(t-1)} - y_i)f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + constant\end{aligned}$$

Now, let's apply Taylor series expansion upto second order:

$$obj^{(t)} = \sum_{i=1}^n [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant$$

where g_i and h_i can be defined as:

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

$$h_i = \partial_{\hat{y}_i^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$$

Simplifying and removing the constant:

$$\sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

Now, we define the regularization term, but first we need to define the model:

$$f_t(x) = w_{q(x)}, w \in R^T, q : R^d \rightarrow \{1, 2, \dots, T\}$$

Here, w is the vector of scores on leaves of tree, q is the function assigning each data point to the corresponding leaf, and T is the number of leaves. The regularization term is then defined by:

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

Now, our objective function becomes:

$$obj^{(t)} \approx \sum_{i=1}^n [g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

$$= \sum_{j=1}^T [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T$$

Now, we simplify the above expression:

$$obj^{(t)} = \sum_{j=1}^T [G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2] + \gamma T$$

where,

$$G_j = \sum_{i \in I_j} g_i$$

$$H_j = \sum_{i \in I_j} h_i$$

In this equation, w_j are independent of each other, the best w_j for a given structure $q(x)$ and the best objective reduction we can get is:

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

$$\text{obj}^* = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

where, γ is pruning parameter, i.e., the least information gain to perform split.

Now, we try to measure how good the tree is, we can't directly optimize the tree, we will try to optimize one level of the tree at a time. Specifically, we try to split a leaf into two leaves, and the score it gains is

$$\text{Gain} = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

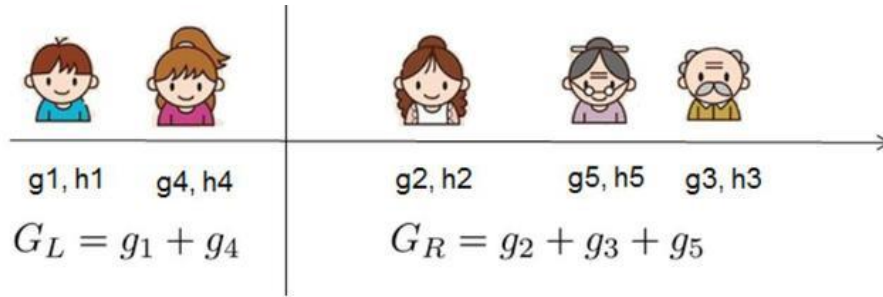


Fig 3.2 XGBoost Mathematical Representation

3.3 Support Vector Machine (SVM) Algorithm

Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well it's best suited for classification. The main objective of the SVM algorithm is to find the optimal hyperplane in an N-dimensional space that can separate the data points in different classes in the feature space. The hyperplane tries that the margin between the closest points of different classes should be as maximum as possible. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

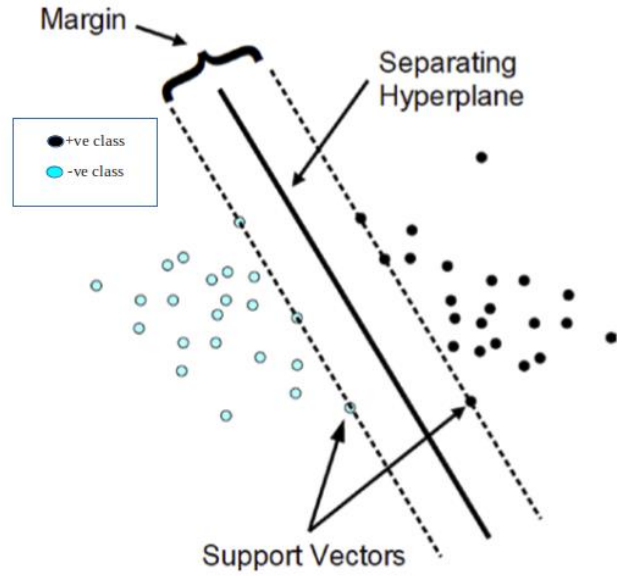


Fig 3.3 Support Vector Machine

Mathematical intuition of Support Vector Machine

Consider a binary classification problem with two classes, labeled as +1 and -1. We have a training dataset consisting of input feature vectors X and their corresponding class labels Y .

The equation for the linear hyperplane can be written as:

$$w^T x + b = 0$$

The vector W represents the normal vector to the hyperplane. i.e., the direction perpendicular to the hyperplane. The parameter b in the equation represents the offset or distance of the hyperplane from the origin along the normal vector w .

The distance between a data point x_i and the decision boundary can be calculated as:

$$d_i = \frac{w^T x_i + b}{\|w\|}$$

where $\|w\|$ represents the Euclidean norm of the weight vector w . Euclidean norm of the normal vector W

For Linear SVM classifier:

$$\hat{y} = \begin{cases} 1, & w^T x + b \geq 0 \\ 0, & w^T x + b < 0 \end{cases}$$

Advantages of SVM

- Effective in high-dimensional cases.
- Its memory is efficient as it uses a subset of training points in the decision function called support vectors.
- Different kernel functions can be specified for the decision functions and its possible to specify custom kernels.

SVM implementation in Python

Predict if cancer is Benign or malignant. Using historical data about patients diagnosed with cancer enables doctors to differentiate malignant cases and benign ones are given independent attributes.

Steps

- Load the breast cancer dataset from sklearn.datasets
- Separate input features and target variables.
- Build and train the SVM classifiers using RBF kernel.
- Plot the scatter plot of the input features.
- Plot the decision boundary.
- Plot the decision boundary

3.4 Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on

the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of over fitting.

Random Forest algorithm:

Step 1: In Random Forest n number of random records are taken from the data set having k number of records.

Step 2: Individual decision trees are constructed for each sample.

Step 3: Each decision tree will generate an output.

Step 4: Final output is considered based on Majority Voting or Averaging for Classification and regression respectively.

Working Steps:

1. Bootstrapped Sampling:

- Random Forest starts by creating multiple random subsets of the original dataset with replacement. Each of these subsets is called a "bootstrap sample."
- The bootstrap sampling process creates multiple datasets that are slightly different from each other.

2. Building Decision Trees:

- For each bootstrap sample, a decision tree is constructed.
- At each node of the tree, a random subset of features is selected to find the best split.
- The randomness in feature selection ensures that each tree is different and diverse.

3. Voting (Classification) / Averaging (Regression):

- For classification tasks, each tree "votes" for the most popular class.
- For regression tasks, each tree predicts a continuous value.
- The final prediction is determined by aggregating the predictions of all the trees:

- **Classification:** The class that receives the most votes from all trees is chosen as the predicted class.
- **Regression:** The average (or sometimes median) of all tree predictions is taken as the final output.

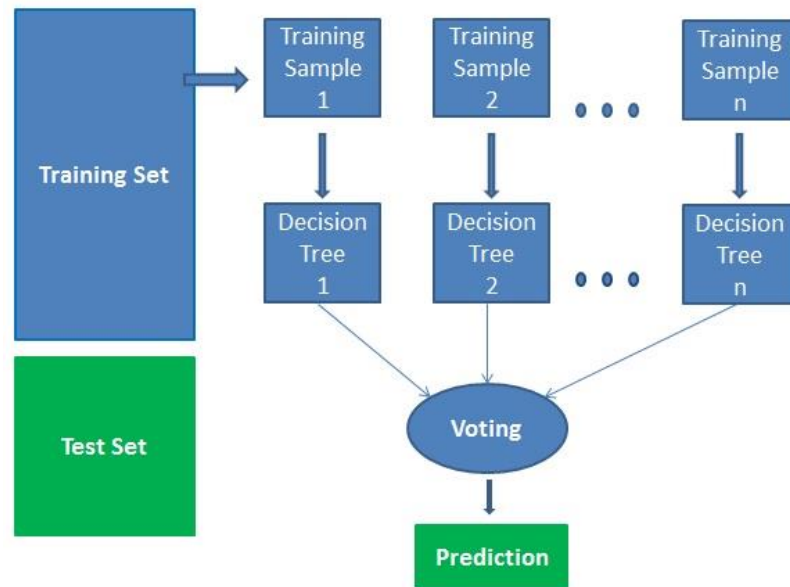


Fig 3.4 Random Forest

Advantages:

- Random Forests are robust and can handle noisy data and outliers.
- They are less likely to overfit, which means they can generalize well to new data.
- Random Forests are highly accurate.
- They perform well in both classification and regression problems.

Disadvantages:

- Random Forests are difficult to interpret.
- The ensemble of trees makes it challenging to understand the decision process.
- When using a large number of trees, training time can be relatively high.
- Managing a forest of trees involves more complexity than a single decision tree.

3.5 Naïve Bayes Algorithm

The Naïve Bayes algorithm stands as a prominent supervised learning technique, rooted in Bayes' theorem, primarily employed for tackling classification tasks. Its prowess shines notably in text classification scenarios, especially when dealing with extensive training datasets featuring high dimensionality. Renowned for its simplicity and effectiveness, the Naïve Bayes Classifier excels in constructing rapid machine learning models capable of swift predictions. Operating as a probabilistic classifier, it derives predictions based on object probabilities. Widely embraced applications of this algorithm include spam filtration, sentiment analysis, and article classification, underscoring its versatility and utility across various domains.

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, which can be described as:

- **Naïve:** It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes:** It is called Bayes because it depends on the principle of Bayes Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

Were,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

3.6 Data Information

3.6.1 Dataset Parameters

1. **Feature Selection:** Carefully identifying and selecting relevant features that could potentially influence sentiment analysis on tweets. These features may include text content, user metadata (e.g., username, location, followers), tweet metadata (e.g., timestamp, retweet count, like count), sentiment-specific features (e.g., emoticons, hashtags), linguistic features (e.g., word frequency, part-of-speech tags), and contextual features (e.g., trending topics, events).
2. **Target Variable Definition:** With precision, defining the target variable as the sentiment polarity of the tweets, categorized as positive, negative, or neutral. Optionally, sentiment may be represented on a scale, such as very positive, positive, neutral, negative, very negative.
3. **Data Quality Assurance:** Ensuring data integrity by meticulously handling errors, missing values, and inconsistencies through rigorous data preprocessing techniques such as text cleaning (e.g., removing URLs, special characters), tokenization, stop word removal, stemming or lemmatization, and handling of noisy data (e.g., slang, abbreviations).
4. **Consideration of Data Size:** Mindful of computational resources, assessing the dataset's size, including the number of tweets (observations) and features (columns), to strike a balance between model complexity and performance. Optionally, down-sampling or stratified sampling may be employed to address class imbalance issues.
5. **Temporal Considerations:** For datasets featuring time-series data, meticulously analyzing temporal patterns and trends, and integrating time-related features such as tweet timestamps, day of the week, time of day, and temporal indicators (e.g., trending topics during specific time periods, events). Additionally, considering the temporal aspect in training-validation-test splits to ensure model generalizability over time.
6. **Metadata Inclusion:** Incorporating relevant metadata associated with tweets, such as user information (e.g., account age, verified status), tweet engagement

metrics (e.g., retweet count, like count), geospatial information (e.g., location, language), and contextual information (e.g., topic, event).

These dataset parameters collectively define the characteristics of the dataset used for sentiment analysis on tweets, and they are crucial for ensuring the effectiveness and reliability of the sentiment analysis model.

3.6.2 Analysis of Data

1. **Exploratory Data Analysis (EDA):** Employing exploratory data analysis techniques to gain insightful visualizations of data distributions, relationships, and correlations. This includes generating histograms of sentiment distributions, scatter plots to explore relationships between sentiment and metadata (e.g., retweet count, user followers), and correlation matrices to identify potential correlations between features and sentiment labels. This facilitates pattern identification, understanding of sentiment dynamics, and outlier detection.
2. **Feature Selection Strategy:** Leveraging feature selection techniques to identify the most influential features for sentiment analysis. This involves methods such as correlation analysis between sentiment and textual features, feature importance scores derived from machine learning models, and domain expertise to enhance the interpretability and efficiency of the sentiment analysis model.
3. **Development of Sentiment Analysis Model:** Utilizing sentiment analysis techniques such as machine learning classifiers (e.g., logistic regression, support vector machines, random forests), deep learning models (e.g., recurrent neural networks, convolutional neural networks), or lexicon-based approaches, to construct a predictive model capable of accurately categorizing tweets into sentiment categories based on the selected features.
4. **Thorough Model Evaluation:** Diligently assessing the sentiment analysis model's performance using established evaluation metrics such as accuracy, precision, recall, F1-score, and confusion matrices. This provides a comprehensive understanding of the model's predictive capabilities and helps identify areas for improvement.
5. **Interpretation of Model Results:** With attention to detail, interpreting the model's predictions and performance metrics to discern the effectiveness of

different features in predicting sentiment. This allows for the identification of significant predictors and understanding their impact on sentiment classification.

6. **Validation and Sensitivity Analysis:** Conducting rigorous validation and sensitivity analysis to validate the model's robustness and evaluate its stability and sensitivity to changes in parameters or input variables. This ensures the reliability and applicability of the sentiment analysis model in real-world scenarios, considering factors such as evolving language usage, trending topics, and changing user behavior on Twitter.

The primary objective of the methodology in our project analyzing sentiment on Twitter data is to establish and execute a robust framework for constructing predictive models. This entails the careful selection of appropriate natural language processing algorithms, preprocessing techniques, feature extraction methods, and evaluation metrics customized to the unique characteristics of the tweet dataset. The methodology strives to ensure the reliability, accuracy, and generalizability of the sentiment analysis models by methodically addressing data quality issues, managing noisy data, and averting the risk of model overfitting. Moreover, the methodology should encompass rigorous validation procedures, such as cross-validation and out-of-sample testing, to gauge the effectiveness of the models and determine their applicability in real-world scenarios. Ultimately, the methodology functions as a comprehensive guide for effectively harnessing machine learning techniques to analyze sentiment on tweets with precision and confidence.

4.1 Existing System

The prevailing methods for sentiment analysis predominantly hinge on human evaluations, manual coding, and subjective judgments. Regrettably, these traditional approaches frequently yield inconsistent and occasionally inaccurate results, thereby constraining our capacity to derive valuable insights from extensive text data volumes. Presently, existing systems can only evaluate one comment at a time. Moreover, they often encounter challenges in precisely assessing sentiment within a single comment, culminating in a diminished model accuracy rate.

In the current landscape of sentiment analysis, reliance on human evaluations and manual coding poses significant limitations. These methods entail subjective interpretations and are prone to individual biases, leading to variations in sentiment assessments. Consequently, the outcomes obtained lack uniformity and reliability, hindering the extraction of meaningful insights from the data.

Furthermore, the inefficiency of existing systems in evaluating sentiment at scale exacerbates the problem. With the exponential growth of social media content, analyzing vast quantities of text data demands automated, scalable solutions. However, conventional methods struggle to meet these demands, resulting in bottlenecks and inefficiencies in sentiment analysis processes.

Moreover, the inadequacy of current systems in accurately gauging sentiment within individual comments poses a fundamental challenge. Sentiment analysis often involves deciphering nuanced expressions, sarcasm, and context-dependent sentiments, which can elude simplistic rule-based or keyword-driven approaches. Consequently, the inability to capture these subtleties leads to lower model accuracy rates and undermines the reliability of sentiment analysis outcomes.

Addressing these limitations requires a paradigm shift towards more sophisticated, data-driven approaches to sentiment analysis. By leveraging advanced natural language processing (NLP) techniques, machine learning algorithms, and deep learning architectures, it becomes possible to enhance the accuracy, scalability, and robustness of sentiment analysis systems. These advancements hold the key to unlocking the full potential of sentiment analysis in extracting actionable insights from vast volumes of text data, empowering businesses, researchers, and policymakers to make informed decisions based on a comprehensive understanding of public sentiment.

4.1.1 Limitation of Existing System:

1. **Subjectivity in Human Evaluations:** Human evaluations of sentiment are inherently subjective, influenced by personal biases, cultural backgrounds, and individual interpretations. This subjectivity can lead to inconsistencies and inaccuracies in sentiment analysis results.
2. **Labor-Intensive Manual Coding:** Current methods often rely on manual coding by human annotators to label sentiment in text data. This process is labor-intensive and time-consuming, making it impractical for analyzing large volumes of data efficiently.
3. **Difficulty in Capturing Nuanced Sentiments:** Sentiments expressed in language can be nuanced and context-dependent, making it challenging for

sentiment analysis algorithms to accurately capture the full range of emotions and attitudes expressed in text.

4. **Limited Scope of Analysis:** Many existing systems struggle to accurately evaluate sentiment in single comments or short texts, as they may lack sufficient context or fail to capture subtle nuances in language. This limitation restricts the applicability of sentiment analysis to shorter texts commonly found on social media platforms like Twitter.
5. **Challenges in Accounting for Context:** Sentiment analysis algorithms may struggle to account for contextual factors that influence the interpretation of sentiment, such as sarcasm, irony, or cultural references. This can result in misinterpretations of sentiment and inaccurate analysis outcomes.
6. **Inconsistencies in Human Annotations:** Even when relying on human annotators for sentiment labeling, there can be inconsistencies in interpretations and annotations due to differences in individual judgment or understanding of sentiment expressions.

4.2 Proposed System

Our proposed system represents a paradigm shift in the landscape of sentiment analysis, heralding a new era of innovation in the field. By harnessing the capabilities of advanced machine learning algorithms, our primary objective is to introduce a robust, data-driven solution that transcends conventional methodologies. Our system endeavors to accurately categorize comments into Positive, Negative, and Neutral sentiments, while also proficiently identifying Unidentified comments, thereby providing a comprehensive sentiment analysis framework.

The significance of our system lies in its ability to offer an unbiased assessment of sentiment, free from human biases and subjective interpretations. Through the application of machine learning techniques, our solution ensures the consistent and precise categorization of comments, regardless of the complexity or subtlety of the language used.

The implications of our system extend far beyond mere sentiment classification. It holds the potential to revolutionize decision-making processes for businesses and individuals by furnishing them with actionable insights derived from sentiment trends within

comments. By gaining a nuanced understanding of public sentiment, stakeholders can make informed decisions, tailor their strategies, and anticipate potential challenges or opportunities.

A key feature of our system is its capacity for intuitive data visualization through graphical representations. By presenting sentiment distribution in a clear and digestible format, our system enhances comprehension and facilitates informed decision-making. Stakeholders can easily identify trends, patterns, and outliers within the sentiment data, empowering them to take proactive measures or pivot their strategies accordingly.

In essence, our system represents a transformative tool for navigating the complexities of sentiment analysis in social networking applications. Its data-driven approach, coupled with sophisticated machine learning algorithms and intuitive visualization capabilities, equips stakeholders with the insights needed to thrive in an increasingly dynamic and interconnected digital landscape.

4.2.1 The Advantages of Proposed System

- **Paradigm Shift:** Our system represents a paradigm shift in sentiment analysis, ushering in a new era of innovation.
- **Advanced Machine Learning:** It harnesses advanced machine learning algorithms to introduce a robust, data-driven solution.
- **Accurate Categorization:** The system accurately categorizes comments into Positive, Negative, and Neutral sentiments, while proficiently identifying Unidentified comments.
- **Unbiased Assessment:** It offers an unbiased assessment of sentiment, free from human biases and subjective interpretations.
- **Consistent Categorization:** Through machine learning techniques, the solution ensures consistent and precise categorization regardless of language complexity.
- **Actionable Insights:** The system provides actionable insights derived from sentiment trends within comments, revolutionizing decision-making processes for businesses and individuals.

- **Strategic Adaptation:** By understanding public sentiment, stakeholders can tailor strategies and anticipate challenges or opportunities.
- **Intuitive Data Visualization:** It offers intuitive data visualization through graphical representations, enhancing comprehension and facilitating informed decision-making.
- **Real-time Insights:** Specifically for sentiment analysis on tweets, additional advantages include real-time insights into public opinion, enabling swift adaptation to emerging trends.
- **Trend Tracking:** It allows for tracking of trends and identifying emerging issues or opportunities swiftly within the dynamic Twitter landscape.
- **Engagement Optimization:** Businesses can optimize their engagement strategies based on real-time sentiment analysis, improving customer relations and brand perception.
- **Competitive Intelligence:** Sentiment analysis on Twitter provides valuable competitive intelligence by monitoring public perception of competitors, enabling businesses to stay ahead in the market.
- **Crisis Management:** Rapid detection of negative sentiment on Twitter allows for proactive crisis management, mitigating potential reputational damage and addressing concerns promptly.

4.3 Functional Requirements

The system should offer visualization and reporting tools, ensure interpretability and transparency of predictive models, scale to handle large datasets efficiently, adhere to security and privacy standards, and integrate seamlessly with existing infrastructure.

4.3.1 Software Requirements

- **Python Setup:** Python is the primary programming language for both Flask web development and machine learning tasks. Ensure you have Python 3.10 installed.

- **Flask:** Flask is a Python web framework tailored for building web applications. You can install it using pip: **pip install Flask**.
- **Scikit-Learn (sklearn):** Scikit-Learn is a robust machine learning library for Python, offering tools for data preprocessing, model training, and evaluation.
- **Pandas:** Pandas is a powerful Python library specializing in data manipulation and analysis, particularly for handling datasets. Install it using pip: **pip install pandas**.
- **NumPy:** NumPy is a fundamental package for numerical computations in Python, commonly used alongside Pandas for data processing. Install it using pip: **pip install numpy**.
- **Pickle:** Pickle is a Python library for serializing and deserializing Python objects, often employed for saving and loading machine learning models.

4.3.2 Hardware Requirements

- **Operating System:** Your preferred operating system (Windows, macOS, or Linux) compatible with Python and the required software.
- **Storage:** Ensure adequate storage space (minimum 20GB) for storing datasets, models, and application files.
- **Display:** Use a monitor or display screen with a suitable resolution for your development environment.
- **Internet Connection:** An internet connection is required for installing libraries, accessing online resources, and deploying the application (if applicable).
- **Computer or Server:** Use a computer with at least a dual-core processor and 8GB or more of RAM to handle Python development, data processing, and model training.

4.4 Non-Functional Requirements

- **Scalability:** The system must handle increasing data volumes, user loads, and computational demands without compromising performance or reliability. This

requires dynamic scaling of resources such as storage, computing power, and network bandwidth.

- **Accuracy:** Guaranteeing the accuracy and precision of forecasting models to produce reliable predictions within acceptable margins of error. Continuous validation and refinement using historical data and real-world feedback are essential.
- **Usability:** Provide a user-friendly interface and intuitive workflow for interacting with the forecasting system. This includes features such as customizable dashboards, easy data input, clear visualization of results, and responsive design for accessibility across devices.
- **Reliability:** Ensure the system operates reliably under normal and peak loads, minimizing downtime and disruptions. This involves implementing fault-tolerant mechanisms, such as redundancy and failover, and conducting regular performance testing and monitoring.
- **Performance:** Optimize system performance to deliver fast response times and efficient resource utilization. This includes optimizing algorithms and code efficiency, leveraging caching mechanisms, and scaling infrastructure as needed to meet performance goals.
- **Maintainability:** Design the system with modularity and clean code practices to facilitate ease of maintenance and future enhancements. This includes well-documented code, version control, and adherence to coding standards.
- **Compatibility:** Ensure compatibility with a wide range of devices, browsers, and operating systems to maximize accessibility and user reach. This involves testing across different platforms and screen sizes to ensure consistent user experience.

5.1 Unified Modelling Language

Unified Modeling Language (UML) is a standardized modeling language in the field of software engineering, established by the Object Management Group (OMG). It serves as a visual representation for specifying, visualizing, constructing and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. Here are some additional key points about UML

ML consists of a set of graphical notation elements, such as symbols, diagrams, and rules, organized into different diagram types. These elements represent various aspects of a system, including its structure, behavior, interactions, and architecture.

UML can be used throughout the software development lifecycle, from requirements analysis and design to implementation, testing, and maintenance. It facilitates communication and collaboration among stakeholders, enabling them to visualize system requirements, design alternatives, and implementation details.

UML supports several types of diagrams, each serving a specific purpose on modeling different aspects of a system. Common diagrams types include Use Case Diagrams, Class Diagrams, Sequence Diagrams, Activity Diagrams, State Machine Diagrams, Component Diagrams, and Deployment Diagrams.

5.1.1 Use Case Diagram

A Use Case Diagram is a fundamental tool in Unified Modeling Language (UML), representing the interactions between external actors (such as users or other systems) and the system being modeled. Comprised of actors, use cases, and their relationships, this diagram provides a high-level view of the system's functionality and its interactions with its environment. Actors are depicted as stick figures, while use cases are

represented as ovals, each describing a specific functionality or task the system can perform. Relationships between actors and use cases illustrate how actors interact with the system to achieve their goals. Use Case Diagrams serve as a powerful communication tool, facilitating understanding and agreement among stakeholders regarding the system's requirement and behavior.

Actors: Represents a role played by a user or external system interacting with the system. Actors are typically drawn as stick figures.

Use Case: Represents a specific functionality or task performed by the system. Each use case describes a sequence of interaction between the system and its actors to achieve a particular goal. Use cases are depicted as ovals with name of the use case written inside.

Association: Represents a relation between the actors and a use case, indicating that the actors interact with the system to perform the specified use case. Association is shown as lines connecting actors and use cases.

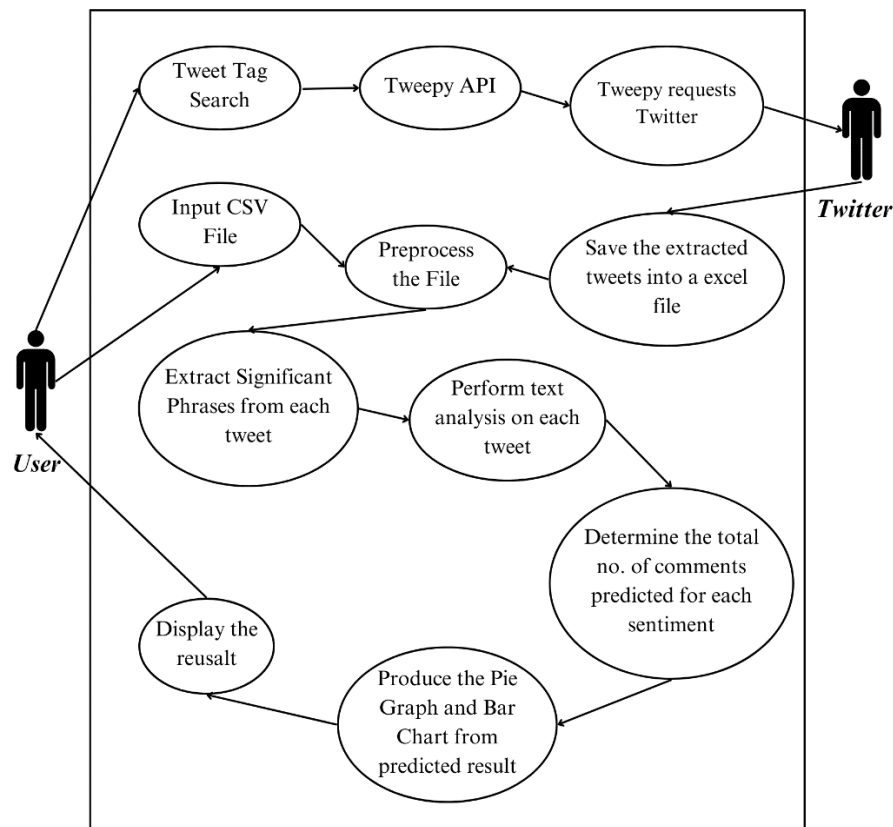


Fig 5.1 Use Case Diagram

5.1.2 Sequence Diagram

A sequence diagram in UML (Unified Modeling Language) is a type of interaction diagram that depicts the interactions between objects or components within a system in a chronological sequence. It shows the flow of messages between objects over time, illustrating the dynamic behavior of the system. Sequence diagrams are commonly used to model scenarios, describe the behavior of a system, and understand how objects collaborate to accomplish a task.

Objects/Participants: Represented by boxes, objects or participants in the system that interact with each other.

Lifelines: Vertical dashed lines extending from each object, representing the lifetime of the object during the interaction.

Message: Horizontal arrows between lifelines, representing the flow of communication between objects. Messages can be synchronous (solid arrows) or asynchronous (dashed arrows).

Activation Bars: Rectangular boxes on a lifeline indicating the period of time during which an object is executing a particular message.

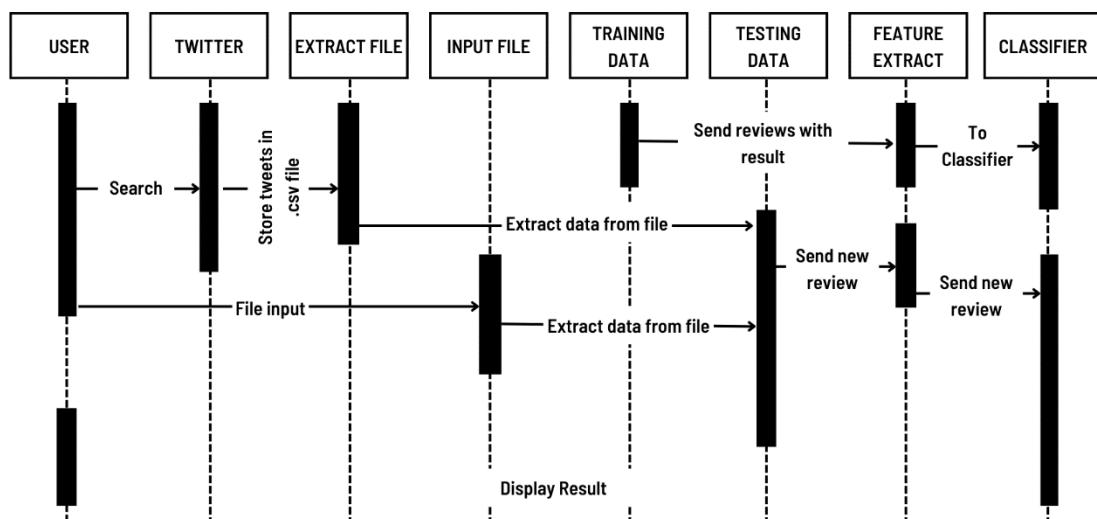


Fig 5.2 Sequence Diagram

5.1.3 Activity Diagram

An activity diagram is a type of Unified Modeling Language (UML) diagram that represents the flow of control or workflow in a system. It visually depicts the sequence

of activities or actions performed in a process or use case within the system. Activity diagrams are commonly used to model business processes, software workflows, and system behaviours. Here's an explanation of the key components and symbols used in activity diagrams:

1. **Initial Node:** This is represented by a solid circle and signifies the starting point of the activity diagram. It indicates where the process or workflow begins.
2. **Activity:** Activities represent tasks or actions performed within the system. They are depicted as rounded rectangles with descriptive labels.
3. **Control Flow:** Control flow arrows connect activities and represent the sequence in which activities are performed. They show the flow of control from one activity to another.
4. **Decision Node (Decision Point):** A decision node is represented by a diamond-shaped symbol and indicates a point in the workflow where a decision needs to be made. Depending on the outcome of the decision, different paths or branches are followed.
5. **Merge Node:** Merge nodes are depicted by a diamond with multiple incoming control flows and one outgoing flow. They indicate the convergence of multiple paths back into a single path.
6. **Fork Node:** Fork nodes are represented by a bar symbol and indicate the splitting of control flow into multiple parallel paths. Each path represents a concurrent activity that can be executed simultaneously.
7. **Join Node:** Join nodes are the counterpart of fork nodes and indicate the merging of parallel control flows back into a single flow.
8. **Final Node:** This is represented by a solid circle with a concentric border and signifies the end point of the activity diagram. It indicates where the process or workflow terminates.

Activity diagrams help stakeholders understand the sequence of activities, decision points, and parallel processes within a system or workflow. They provide a visual representation that aids in analysing, designing, and communicating complex processes effectively. By capturing the dynamic behaviour of a system, activity diagrams contribute to the overall understanding and documentation of system requirements and behaviours.

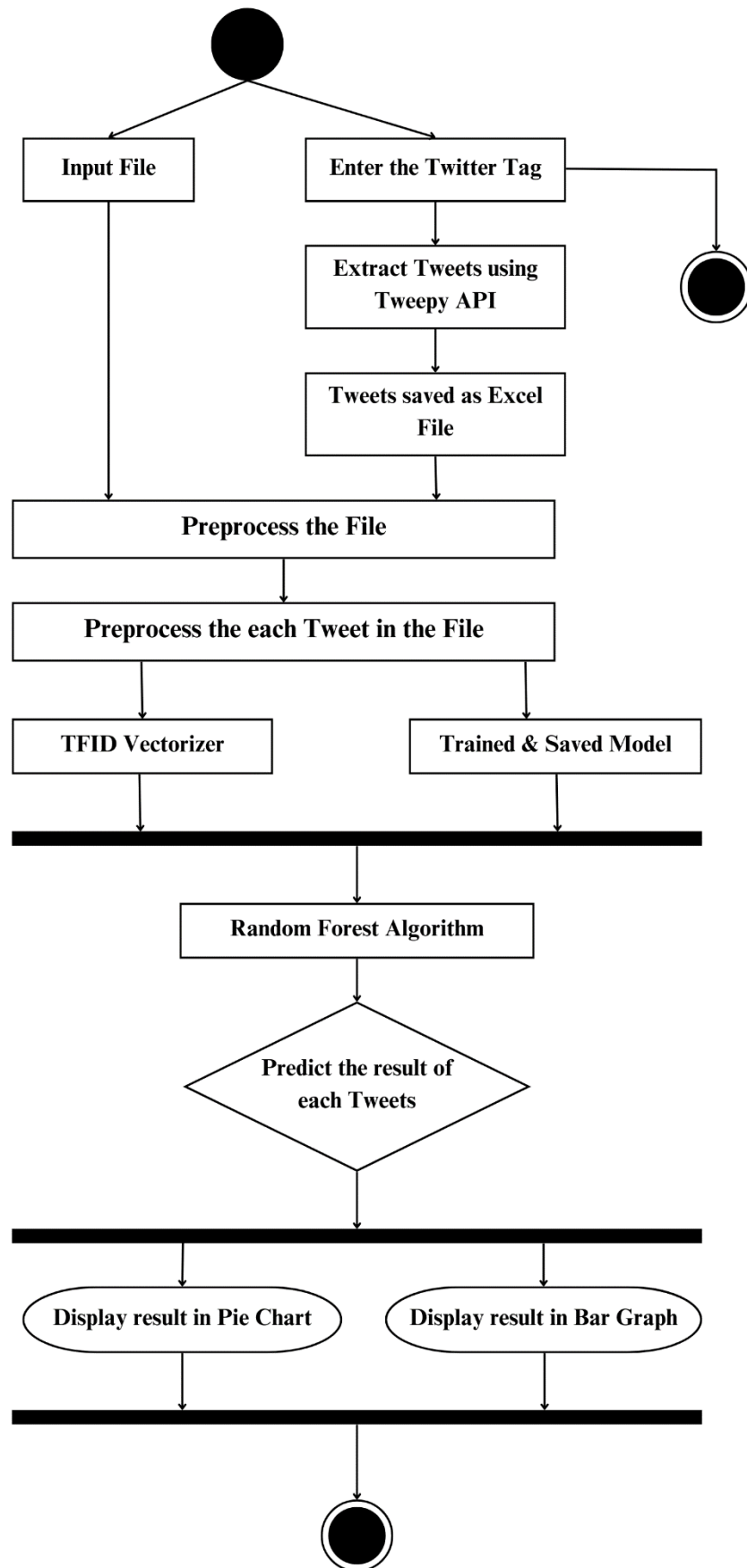


Fig 5.3 Activity Diagram

5.1.4 System Flow Diagram

A System Flow diagram in Unified Modeling Language (UML) is a high-level diagram that illustrates the flow of data and control between the various components or subsystems of a system. It provides an overview of how data moves through the system and how different system elements interact with each other to achieve the system's functionality. Here's an explanation of the key components and symbols used in a System Flow diagram:

1. **External Entities:** External entities represent sources or destinations of data outside the system boundary. They are typically depicted as rectangles with labels, representing users, other systems, or external data sources that interact with the system.
2. **Processes:** Processes represent the functions or activities performed within the system. They are depicted as circles or ovals with descriptive labels. Processes manipulate data, perform calculations, or carry out other tasks to achieve the system's objectives.
3. **Data Flow:** Data flow arrows represent the movement of data between processes, external entities, and data stores within the system. They indicate the direction of data flow and the paths along which data travels.
4. **Data Stores:** Data stores represent repositories where data is stored within the system. They can be databases, files, or any other storage mechanism. Data stores are depicted as rectangles with labels, and data flow arrows show the movement of data into and out of these stores.
5. **Control Flow:** Control flow arrows represent the sequence of activities or the order in which processes are executed within the system. They show the flow of control from one process to another, indicating the logical order of execution.

System Flow diagrams help stakeholders understand the overall architecture and data flow of a system, including how data is input, processed, and output. They provide a high-level view of system functionality and interactions, facilitating communication and understanding among stakeholders. System Flow diagrams are valuable tools for system analysis and design, as they help identify system components, dependencies,

and data flows, which are essential for designing efficient and scalable systems. Additionally, System Flow diagrams can serve as a basis for more detailed system design and documentation, providing a foundation for further development and implementation activities.

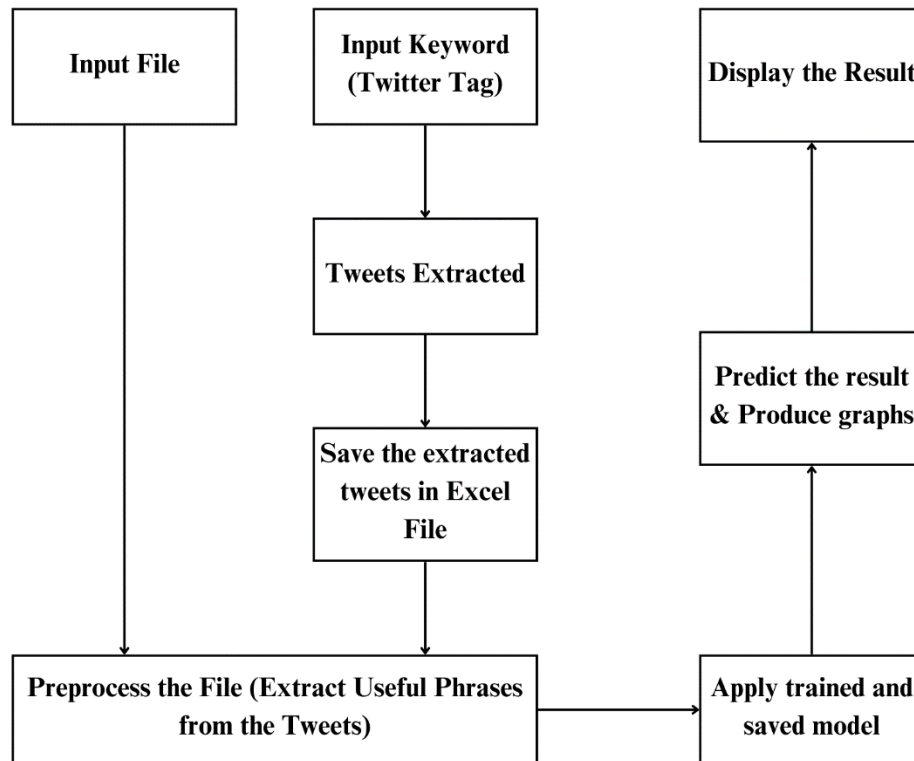


Fig 5.4 System Flow Diagram

5.1.5 Class Diagram

A class diagram in Unified Modeling Language (UML) is a type of static structure diagram that represents the structure of a system by modelling its classes, attributes, methods, and relationships among them. It provides a conceptual blueprint of the system's architecture, illustrating the various components and their interactions.

Here's a breakdown of the key components and concepts in a class diagram:

1. **Class:**

- A class represents a blueprint for creating objects. It encapsulates data (attributes) and behaviour (methods) that characterize a particular type of object.

- In a class diagram, classes are typically represented as rectangles divided into three compartments:
 - The top compartment contains the class name.
 - The middle compartment contains the class attributes (data members).
 - The bottom compartment contains the class methods (member functions).

2. **Attributes:**

- Attributes represent the data or properties associated with a class. They define the state of objects belonging to the class.
- Attributes are usually listed in the middle compartment of the class rectangle, along with their data types.

3. **Methods:**

- Methods represent the behaviours or operations that objects of the class can perform. They define the functionality of the class.
- Methods are listed in the bottom compartment of the class rectangle, along with their parameters and return types.

4. **Visibility:**

- Visibility modifiers indicate the access level of class members (attributes and methods). The three common visibility modifiers are:
 - Public (+): Members are accessible from anywhere.
 - Private (-): Members are accessible only within the class itself.
 - Protected (#): Members are accessible within the class and its subclasses.

5. **Relationships:**

- Relationships depict how classes are related to each other. There are several types of relationships commonly depicted in class diagrams:

- Association: Represents a structural relationship between classes, indicating that objects of one class are connected to objects of another class.
- Aggregation: Represents a "whole-part" relationship between classes, where one class is composed of or contains other classes.
- Composition: Represents a stronger form of aggregation, where the lifetime of the "part" objects is dependent on the lifetime of the "whole" object.
- Inheritance: Represents an "is-a" relationship between classes, where one class (subclass) inherits attributes and methods from another class (superclass).

6. Multiplicity:

- Multiplicity specifies the number of instances of one class that can be associated with a single instance of another class. It is represented using numerical ranges or asterisks (*) to denote one-to-one, one-to-many, or many-to-many relationships.

7. Generalization/Specialization:

- Generalization (inheritance) represents the relationship between a superclass (parent class) and its subclasses (child classes). It indicates that subclasses inherit attributes and methods from the superclass.
- Specialization represents the process of creating subclasses to specialize the behaviour or attributes of the superclass.

Class diagrams are valuable tools for visualizing and understanding the structure of a system, facilitating communication among stakeholders, and guiding the software development process. They provide a high-level overview of the system's architecture and serve as a foundation for detailed design and implementation activities.

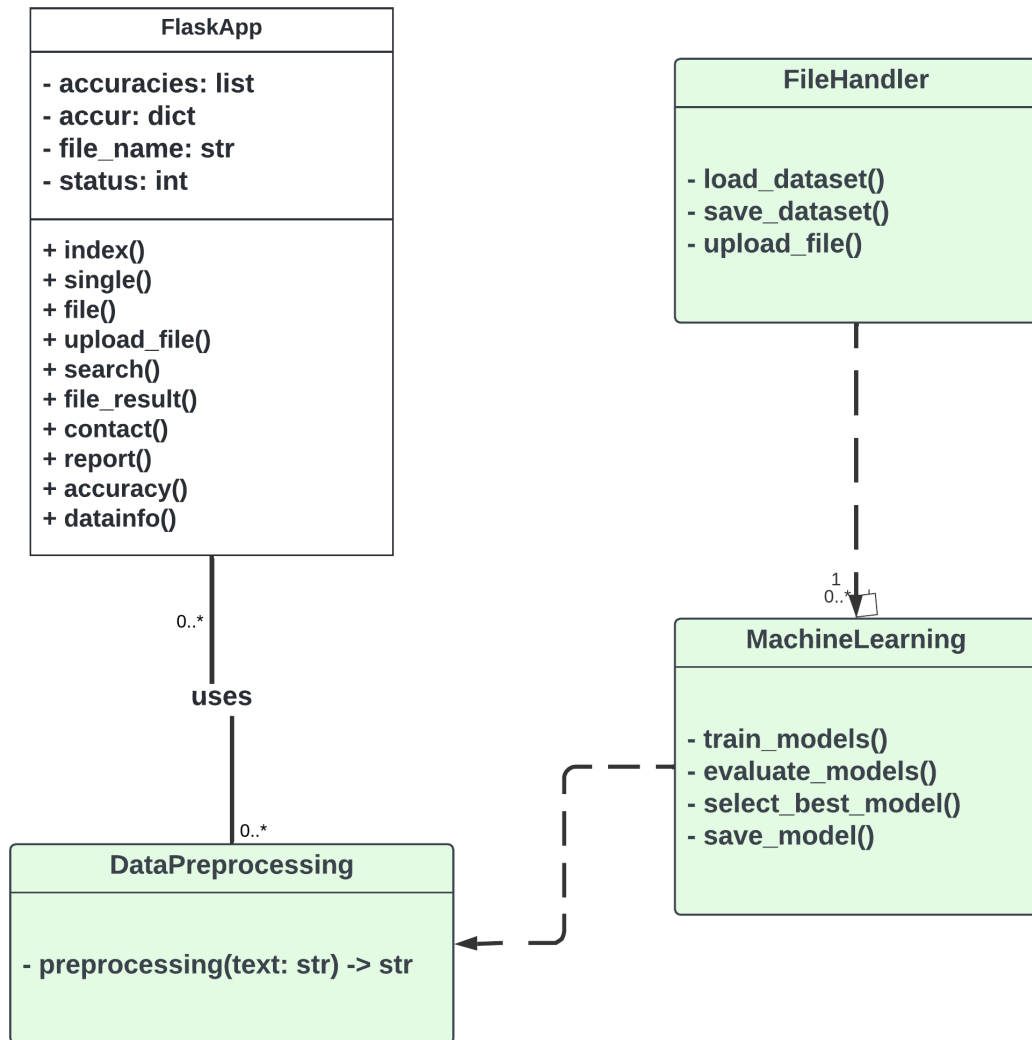


Fig 5.5 Class Diagram

5.1.6 Deployment Diagram

A deployment diagram in UML (Unified Modeling Language) illustrates the physical deployment of software artifacts (such as executables, files, and databases) on hardware nodes (such as servers, devices, or computers). It provides a visual representation of the configuration of runtime processing nodes and the components residing on them.

Key elements of a deployment diagram include:

1. **Node:** Represented by a box with the node's name inside. Nodes typically correspond to physical hardware components like servers, computers, or devices.

2. **Artifact:** Depicted by a rectangle, symbolizing a deployable component like a software application, executable file, or database. Artifacts are usually positioned within nodes to indicate their deployment location.
3. **Association:** Represented by a line connecting a node to an artifact, indicating that the artifact is deployed on that node. Optionally, a stereotype can be added to specify the type of relationship (e.g., "«deploy»").
4. **Communication Path:** Illustrated by a line connecting nodes, showcasing communication paths or network connections between them. This feature can demonstrate how nodes communicate with each other.

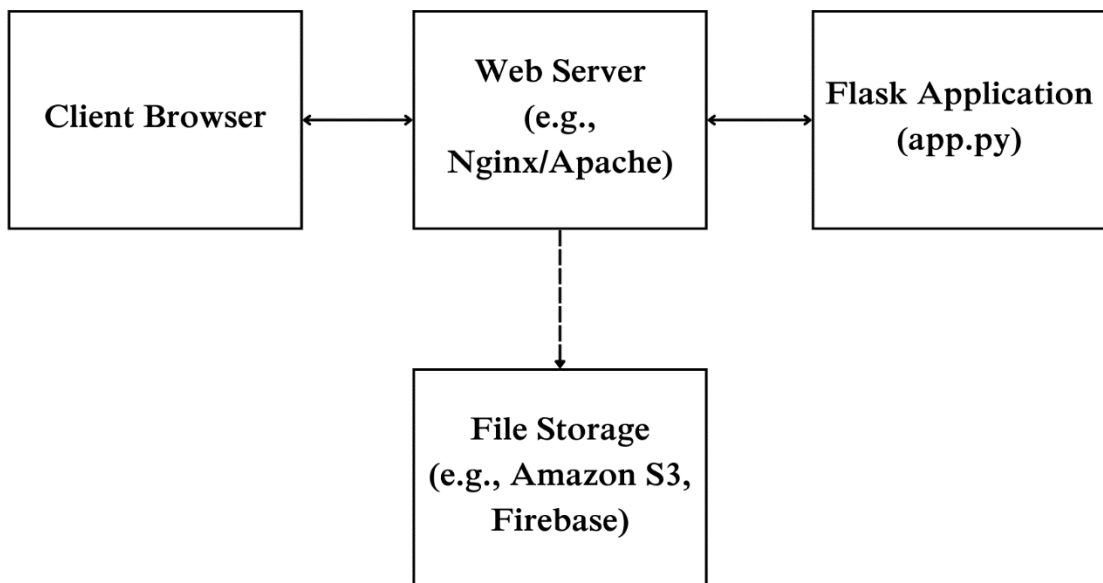


Fig 5.6 Deployment Diagram

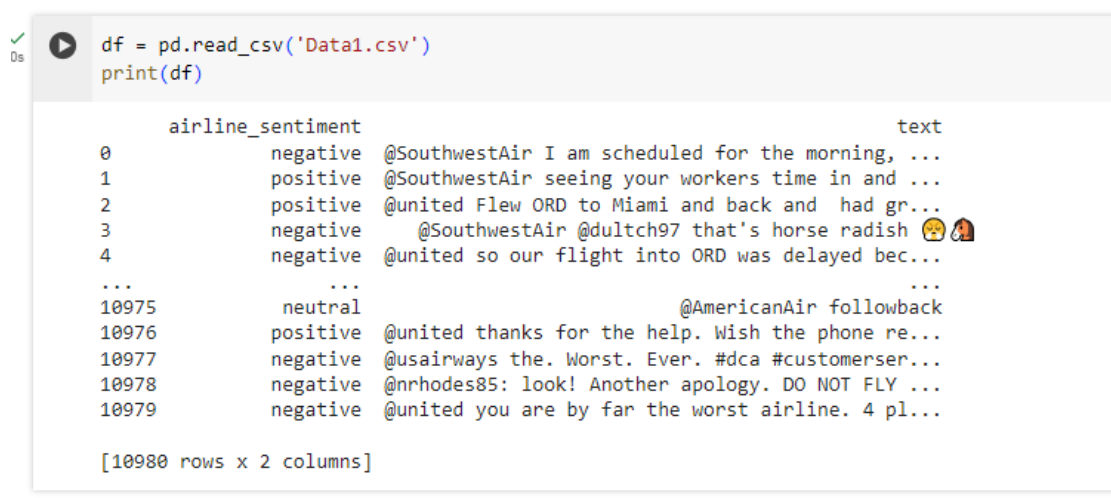
Model Implementation

Implementing a sentiment analysis prediction involves several steps, including data collection, data pre-processing, model selection, training, evaluation, and deployment. Here's a simplified outline of how you can approach this task:

6.1 Data Collection

The dataset for this sentiment analysis project is about the problems of each major U.S. airline. Twitter data was scraped from February of 2015 and contributors were asked to first classify positive, negative, and neutral tweets, followed by categorizing negative reasons (such as "late flight" or "rude service").

The dataset for this project is saved in an .csv file and is accessed by `read_csv()` function in Pandas module and the data is changed into a DataFrame.



```

df = pd.read_csv('Data1.csv')
print(df)

```

	airline_sentiment	text
0	negative	@SouthwestAir I am scheduled for the morning, ...
1	positive	@SouthwestAir seeing your workers time in and ...
2	positive	@united Flew ORD to Miami and back and had gr...
3	negative	@SouthwestAir @dultch97 that's horse radish 🤢👤
4	negative	@united so our flight into ORD was delayed bec...
...
10975	neutral	@AmericanAir followback
10976	positive	@united thanks for the help. Wish the phone re...
10977	negative	@usairways the. Worst. Ever. #dca #customerser...
10978	negative	@nrhodes85: look! Another apology. DO NOT FLY ...
10979	negative	@united you are by far the worst airline. 4 pl...

[10980 rows x 2 columns]

Fig 6.1 Loading and Printing the Dataset

To provides a concise summary of the data (DataFrame), including information about the index dtype, column dtypes, non-null values, and memory usage **info()** method is used.

```
[4] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10980 entries, 0 to 10979
Data columns (total 2 columns):
#   Column          Non-Null Count  Dtype  
---  -
0   airline_sentiment 10980 non-null  object 
1   text             10980 non-null  object 
dtypes: object(2)
memory usage: 171.7+ KB
```

Fig 6.2 Attributes of Sentiment Analysis Dataset

6.2 Data Pre-processing

6.2.1 Data Cleaning

Clean the information by dealing with missing qualities, identifying and adjusting blunders, and normalizing designs across various information sources.

```
[5] print(f"Any NaN values? {df.isna().values.any()}")💡
Any NaN values? False

[6] print(f"Any duplicates? {df.duplicated().values.any()}")
Any duplicates? True
```

Fig 6.3 Data Cleaning (Check for Missing Values and Duplicates)

6.2.2 Label Encoder

LabelEncoder() from scikit-learn to transform categorical target values into numerical labels, making it easier for analysis. The transformed target values are then printed to display the changes made.

```
# Replacing the values to ease understanding :
from sklearn import preprocessing
le = preprocessing.LabelEncoder()

df['target'] = le.fit_transform(df['target'])
print(df['target'])
```

0	0
1	2
2	2
3	0
4	0
..	
10975	1
10976	2
10977	0
10978	0
10979	0

Name: target, Length: 10980, dtype: int64

Fig 6.4 Label Encoding the target values

6.2.3 To lower-case text

The below code converts the text in the 'text' column of the DataFrame 'data' to lowercase and stores the result in a new column named 'lowercase_text'.

```
# Convert text to lowercase
data['lowercase_text'] = data['text'].str.lower()

print("Lowercase Text:")
print(data['lowercase_text'])
```

Lowercase Text:

0	this is a sample text with urls like www.examp...
1	it contains punctuation! and stopwords like th...
2	this text has repeating characters: wooooow
3	tokenization separates text into words.
4	stemming reduces words to their root form.
5	lemmatization converts words to their base form.

Name: lowercase_text, dtype: object

Fig 6.5 Converting text into lowercase

6.2.4 Remove URLs

After the changing the text in the dataset into lowercase, the text contains URLs which are not useful for prediction and we should also remove the URLs.

1. **data['lowercase_text'].apply(...):** This part suggests that data is a Pandas DataFrame, and it's applying a function to the column named 'lowercase_text'. The function being applied seems to be a lambda function.
2. **lambda text:** re.sub(r'(www.[^s]+)(https?://[^\s]+)', ' ', text): This lambda function takes a text input and uses the re.sub() function from the re module to substitute any occurrences of URLs with a space.
 - r'(www.[^s]+)(https?://[^\s]+)' is a regular expression pattern that matches URLs.
 - (www.[^s]+) matches URLs starting with '[www](#).' followed by one or more characters that are not whitespace.
 - (https?://[^\s]+) matches URLs starting with 'http://' or 'https://' followed by one or more characters that are not whitespace.
 - The pipe | symbol between the two patterns acts as an OR operator.
 - The ' ' in the re.sub() function indicates that any matched URLs will be replaced with a space.
3. **data['no_urls'] = ...:** Finally, the result of applying this lambda function to each element of the 'lowercase_text' column is stored in a new column named 'no_urls' in the data DataFrame.

```
# Remove URLs
data['no_urls'] = data['lowercase_text'].apply(lambda text: re.sub(r'(www.[^s]+)|(https?://[^s]+)', ' ', text))

print("\nText without URLs:")
print(data['no_urls'])
```

```
Text without URLs:
0    this is a sample text with urls like s://exam...
1    it contains punctuation! and stopwords like th...
2          this text has repeating characters: wooooow
3          tokenization separates text into words.
4          stemming reduces words to their root form.
5    lemmatization converts words to their base form.
Name: no_urls, dtype: object
```

Fig 6.6 Removing URLs from the text

6.2.5 Removing Punctuation

By removing punctuations, the code helps clean the text data, making it more standardized and easier to analyze. This is often a crucial step in text preprocessing pipelines, especially when dealing with noisy or unstructured text data.

- The lambda function inside the apply() method is used again. This time, it utilizes the translate() method to remove punctuations.
- str.maketrans(", ", string.punctuation) creates a translation table that maps each punctuation character to None, effectively removing them.
- text.translate(...) then applies this translation to remove punctuations from each text element.

```
# Remove punctuations
data['no_punctuations'] = data['no_urls'].apply(lambda text: text.translate(str.maketrans('', '', string.punctuation)))

print("\nText without punctuations:")
print(data['no_punctuations'])
```

```
Text without punctuations:
0    this is a sample text with urls like sexamplecom
1    it contains punctuation and stopwords like the...
2          this text has repeating characters woowoow
3          tokenization separates text into words
4          stemming reduces words to their root form
5    lemmatization converts words to their base form
Name: no_punctuations, dtype: object
```

Fig 6.7 Punctuation Removal

6.2.6 Removing Stopwords

The **stopwords** module in Natural Language Processing (NLP) contains a list of common words that are often filtered out from text data before processing. These words, known as "stop words," typically include common words like "the," "is," "and," "in," etc. They are considered to have little semantic meaning and are frequently used across different types of texts without contributing much to the understanding of the content.

Removing stopwords is important in text preprocessing for several reasons:

1. **Improving Efficiency:** Stopwords are very common in text but often carry little semantic meaning. By removing them, we reduce the size of the text data, making subsequent processing tasks more efficient.
2. **Enhancing Relevance:** Stopwords appear frequently in documents but do not contribute much to the content's meaning. Removing them can help highlight the more relevant words and phrases in the text.
3. **Improving Accuracy:** In many NLP tasks such as text classification or sentiment analysis, stopwords can introduce noise and affect the accuracy of the models. Removing them can lead to more accurate predictions by focusing on the words that carry more significance.
4. **Simplifying Analysis:** Stopwords can interfere with the analysis of text data, especially when extracting insights or patterns. Removing them simplifies the

analysis process and makes it easier to identify meaningful patterns and trends in the data.

```
# Remove stopwords
stop_words = set(stopwords.words('english'))
data['no_stopwords'] = data['no_punctuations'].apply(lambda text: ' '.join(word for word in text.split() if word not in stop_words))

print("\nText without stopwords:")
print(data['no_stopwords'])
```

```
Text without stopwords:
0      sample text urls like sexamplecom
1      contains punctuation stopwords like
2      text repeating characters woowoow
3      tokenization separates text words
4      stemming reduces words root form
5      lemmatization converts words base form
Name: no_stopwords, dtype: object
```

Fig 6.8 Removing Stopword

6.2.7 Removing Repeating Character

- **Improved Readability:** Text with repeating characters can be harder to read and understand. Removing repeating characters enhances readability, making the text more comprehensible to readers and facilitating better communication.
- **Noise Reduction:** Repeating characters often occur due to typing errors, transcription mistakes, or other artifacts. Removing them reduces noise in the text, improving the quality of the data for subsequent analysis or processing.
- **Normalization:** In some cases, repeating characters may convey emphasis or emotion (e.g., "loooove" for "love"). However, for many NLP tasks, it's beneficial to normalize the text and treat repeated characters as single occurrences to avoid biasing the analysis.
- **Preventing Overfitting:** In machine learning tasks, especially those involving text data, removing repeating characters can help prevent overfitting. Overfitting occurs when a model learns noise or irrelevant patterns in the data, and removing repeating characters can reduce such noise.

```
# Remove repeating characters
data['no_repeating_chars'] = data['no_stopwords'].apply(lambda text: re.sub(r'(\1+)', r'\1', text))

print("\nText without repeating characters:")
print(data['no_repeating_chars'])
```

```
Text without repeating characters:
0      sample text urls like sexamplecom
1      contains punctuation stopwords like
2      text repeating characters wow
3      tokenization separates text words
4      stemming reduces words not form
5      lematization converts words base form
Name: no_repeating_chars, dtype: object
```

Fig 6.9 Repeating words are removed

Explanation

- It applies a lambda function to each text in the '**no_stopwords**' column to remove repeating characters.
- The **re.sub()** function from the **re** (regular expression) module is used for substitution based on a regular expression pattern.
- The regular expression **r'(\1+)'** matches any character (**.**) followed by one or more occurrences of the same character **\1**. This pattern captures repeating characters.
- The replacement pattern **r'\1'** replaces the matched repeating characters with just one occurrence of the character. So, it effectively removes duplicate consecutive characters.

6.2.8 Tokenization

Tokenization is the process of breaking down a piece of text into smaller units, which are typically words or subwords. These smaller units are called tokens. Tokenization is a fundamental step in natural language processing (NLP) tasks as it enables the computer to understand and process textual data.

Explanation of Tokenization:

- **Tokenization Process:** Tokenization involves splitting a piece of text into smaller units, which could be words, subwords, or characters, depending on the specific task and requirements. In this case, the text is tokenized into words.
- **Word Tokenization:** Word tokenization breaks the text into words based on whitespace and punctuation. It handles contractions, hyphenated words, and other linguistic constructs intelligently to ensure accurate tokenization.
- **Token Types:** Tokens can include words, punctuation marks, numbers, or any other meaningful units of text. In this code snippet, the tokens are words extracted from the text.
- **Tokenized Text Output:** After tokenization, each text is represented as a list of tokens, where each token corresponds to a word in the original text. These tokenized representations are easier for computers to process and analyze.

Importance of Tokenization:

- **Text Processing:** Tokenization is a fundamental preprocessing step in NLP tasks. It breaks down the text into manageable units, allowing for further analysis, such as part-of-speech tagging, named entity recognition, and sentiment analysis.
- **Feature Extraction:** In machine learning tasks involving text data, tokenization is crucial for feature extraction. Each token can be treated as a feature, and the presence or frequency of tokens can be used to train machine learning models.
- **Text Understanding:** Tokenization helps computers understand and interpret text by converting it into a format that can be easily processed and analyzed. It forms the foundation for many higher-level NLP tasks and applications.

The **word_tokenize()** function, provided by the NLTK (Natural Language Toolkit) library, is used to tokenize the text into words. The lambda function tokenizes each text by breaking it into individual words and stores the resulting list of tokens in the **'tokenized_text'** column.


```
# Tokenize text
data['tokenized_text'] = data['no_repeating_chars'].apply(lambda text: word_tokenize(text))

print("\nTokenized Text:")
print(data['tokenized_text'])
```

```
Tokenized Text:
0      [sample, text, urls, like, sexamplecom]
1      [contains, punctuation, stopwords, like]
2      [text, repeating, characters, wow]
3      [tokenization, separates, text, words]
4      [stemming, reduces, words, rot, form]
5      [lematization, converts, words, base, form]
Name: tokenized_text, dtype: object
```

Fig 6.10 Tokenization of Words

6.2.9 Stemming

Stemming is the process of reducing words to their root or base form, which may not always be a valid word itself but can help in grouping together words with similar meanings. It's a common technique used in natural language processing (NLP) to simplify text data and improve computational efficiency by reducing the dimensionality of the feature space.

Stemming Process

- Stemming algorithms work by removing suffixes from words to extract their root form. This process involves heuristics rather than linguistic rules, so the resulting stems may not always be perfectly accurate or meaningful in human terms.
- For example, the words "running," "ran," and "runs" would all be stemmed to "run" since they share the same root.
- The goal of stemming is to reduce words to their common base form to improve text analysis tasks such as text classification, information retrieval, and sentiment analysis.

Importance of Stemming

- **Dimensionality Reduction:** Stemming reduces the number of distinct words in the text data, which can help in reducing the dimensionality of the feature space. This is particularly useful in machine learning tasks where the number of features can impact model performance.
- **Grouping Similar Words:** Stemming helps in grouping together words with similar meanings, which can improve the performance of text analysis tasks by treating variants of the same word as equivalent.
- **Computational Efficiency:** Stemming simplifies text data, making it easier and faster to process, especially in tasks involving large volumes of text data.

```
# Apply stemming
st = nltk.PorterStemmer()
data['stemmed_text'] = data['tokenized_text'].apply(lambda tokens: [st.stem(word) for word in tokens])

print("\nStemmed Text:")
print(data['stemmed_text'])
```

```
Stemmed Text:
0    [sampl, text, url, like, sexamplecom]
1    [contain, punctuat, stopwords, like]
2          [text, repeat, charact, wow]
3          [token, separ, text, word]
4          [steme, reduc, word, rot, form]
5    [lemat, convert, word, base, form]
Name: stemmed_text, dtype: object
```

Fig 6.11 Stemming of Text

6. 2. 10 Lemmatization

Lemmatization is a process similar to stemming but with a difference: it reduces words to their base or dictionary form, known as the lemma. Unlike stemming, which may result in non-existent or incorrect words, lemmatization uses vocabulary analysis to convert words to their canonical form. Lemmatization typically involves the use of

a vocabulary and morphological analysis of words, aiming to return the base or dictionary form of a word, known as its lemma.

Lemmatization Process

- Lemmatization considers the context of a word and its part of speech (POS) tag to determine the correct lemma. For example, the word "better" might be lemmatized to "good" instead of "better" itself.
- It involves mapping words to their dictionary form, which often results in more meaningful and interpretable lemmas compared to stemming.

Importance of Lemmatization

- **Better Lemmas:** Lemmatization returns valid words (lemmas) from the language's vocabulary, making it more suitable for tasks where interpretability and accuracy are crucial.
- **Improved Text Analysis:** By returning meaningful lemmas, lemmatization can improve the performance of text analysis tasks such as sentiment analysis, topic modeling, and information retrieval.
- **Part-of-Speech Information:** Lemmatization considers the part of speech (POS) of words, allowing for more accurate lemmatization based on the context of the text.
- **Enhanced Feature Space:** Lemmatization can lead to a more concise and informative feature space in text-based machine learning tasks, potentially improving model performance.

The code initializes a WordNet Lemmatizer object using `nlk.WordNetLemmatizer()`. WordNet is a lexical database for the English language that includes lemmas and their semantic relationships.

It then applies a lambda function to each list of tokenized words (tokens) in the 'tokenized_text' column of the DataFrame data.

Within the lambda function, a list comprehension is used to iterate over each token (word) in the tokens list and apply lemmatization using the `lm.lemmatize(word)` method.

```
# Apply lemmatization
lm = nltk.WordNetLemmatizer()
data['lemmatized_text'] = data['tokenized_text'].apply(lambda tokens: [lm.lemmatize(word) for word in tokens])

print("\nLemmatized Text:")
print(data['lemmatized_text'])
```

```
Lemmatized Text:
0      [sample, text, url, like, sexamplecom]
1      [contains, punctuation, stopwords, like]
2      [text, repeating, character, wow]
3      [tokenization, separate, text, word]
4      [steming, reduces, word, rot, form]
5      [lematization, convert, word, base, form]
Name: lemmatized_text, dtype: object
```

Fig 6.12 Lemmatization of Text

6.3 Feature Selection & Engineering

6.3.1 TF-IDF Vectorizer scikit-learn – First Approach

TF-IDF stands for Term Frequency Inverse Document Frequency of records. It can be defined as the calculation of how relevant a word in a series or corpus is to a text. The meaning increases proportionally to the number of times in the text a word appears but is compensated by the word frequency in the corpus (data-set).

Terminologies:

- **Term Frequency:** In document *d*, the frequency represents the number of instances of a given word *t*. Therefore, we can see that it becomes more relevant when a word appears in the text, which is rational. Since the ordering of terms is not significant, we can use a vector to describe the text in the bag of term models. For each specific term in the paper, there is an entry with the value being the term frequency.

The weight of a term that occurs in a document is simply proportional to the term frequency.

$$tf(t, d) = \text{count of } t \text{ in } d / \text{number of words in } d$$

- **Document Frequency:** This tests the meaning of the text, which is very similar to TF, in the whole corpus collection. The only difference is that in document d , TF is the frequency counter for a term t , while df is the number of occurrences in the document set N of the term t . In other words, the number of papers in which the word is present is DF.

$$df(t) = \text{occurrence of } t \text{ in documents}$$

- **Inverse Document Frequency:** Mainly, it tests how relevant the word is. The key aim of the search is to locate the appropriate records that fit the demand. Since tf considers all terms equally significant, it is therefore not only possible to use the term frequencies to measure the weight of the term in the paper. First, find the document frequency of a term t by counting the number of documents containing the term:

$$df(t) = N(t)$$

where

$$df(t) = \text{Document frequency of a term } t$$

$$N(t) = \text{Number of documents containing the term } t$$

Term frequency is the number of instances of a term in a single document only; although the frequency of the document is the number of separate documents in which the term appears, it depends on the entire corpus. Now let's look at the definition of the frequency of the inverse paper. The IDF of the word is the number of documents in the corpus separated by the frequency of the text.

$$idf(t) = N / df(t) = N / N(t)$$

The more common word is supposed to be considered less significant, but the element (most definite integers) seems too harsh. We then take the logarithm (with base 2) of the inverse frequency of the paper. So the if of the term t becomes:

$$\text{idf}(t) = \log(N / \text{df}(t))$$

- **Computation:** Tf-idf is one of the best metrics to determine how significant a term is to a text in a series or a corpus. tf-idf is a weighting system that assigns a weight to each word in a document based on its term frequency (tf) and the reciprocal document frequency (tf) (idf). The words with higher scores of weight are deemed to be more significant.

Usually, the tf-idf weight consists of two terms-

1. **Normalized Term Frequency (tf)**
2. **Inverse Document Frequency (idf)**

$$\text{tf-idf}(t, d) = \text{tf}(t, d) * \text{idf}(t)$$

In python tf-idf values can be computed using *TfidfVectorizer()* method in *sklearn* module.

6.3.2 TF-IDF Vectorizer scikit-learn – Second Approach

TF (Term Frequency)

TF-IDF stands for *term frequency-inverse document frequency* and it is a measure, used in the fields of information retrieval and machine learning, that can quantify the importance or relevance of string representations (words, phrases, lemmas, etc.) in a document amongst a collection of documents (also known as a corpus).

TF-IDF can be broken down into two parts *TF (term frequency)* and *IDF (inverse document frequency)*.

Term frequency works by looking at the frequency of a *particular term* you are concerned with relative to the document. There are multiple measures, or ways, of defining frequency:

- Number of times the word appears in a document (raw count).
- Term frequency adjusted for the length of the document (raw count of occurrences divided by number of words in the document).
- Logarithmically scaled frequency (e.g., $\log(1 + \text{raw count})$).
- Boolean Frequency (e.g., 1 if the term occurs, or 0 if the term does not occur, in the document).

IDF (Inverse Document Frequency)

Inverse document frequency looks at how common (or uncommon) a word is amongst the corpus. IDF is calculated as follows where t is the term (word) we are looking to measure the commonness of and N is the number of documents (d) in the corpus (D). The denominator is simply the number of documents in which the term, t , appears in.

$$idf(t, D) = \log \left(\frac{N}{\text{count}(d \in D : t \in d)} \right)$$

Note: It can be possible for a term to not appear in the corpus at all, which can result in a divide-by-zero error. One way to handle this is to take the existing count and add 1. Thus, making the denominator $(1 + \text{count})$. An example of how the popular library scikit-learn handles this can be seen below.

Scikit-Learn

$$\bullet \text{ IDF}(t) = \log \frac{1+n}{1+\text{df}(t)} + 1$$

Standard notation

$$\bullet \text{ IDF}(t) = \log \frac{n}{\text{df}(t)}$$

The reason we need IDF is to help correct for words like “of”, “as”, “the”, etc. since they appear frequently in an English corpus. Thus, by taking inverse document

frequency, we can minimize the weighting of frequent terms while making infrequent terms have a higher impact.

Finally, IDFs can also be pulled from either a background corpus, which corrects for sampling bias, or the dataset being used in the experiment at hand.

Putting it together: TF-IDF

To summarize the key intuition motivating TF-IDF is the importance of a term is inversely related to its frequency across documents. TF gives us information on how often a term appears in a document and IDF gives us information about the relative rarity of a term in the collection of documents. By multiplying these values together, we can get our final TF-IDF value.

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

The higher the TF-IDF score the more important or relevant the term is; as a term gets less relevant, its TF-IDF score will approach 0.

Uses of TF-IDF

TF-IDF can be a very handy metric for determining how important a term is in a document. But how is TF-IDF used? There are three main applications for TF-IDF. These are in *machine learning*, *information retrieval*, and *text summarization/keyword extraction*.

- **Using TF-IDF in machine learning & natural language processing**

Machine learning algorithms often use numerical data, so when dealing with textual data or any natural language processing (NLP) task, a sub-field of ML/AI dealing with text, that data first needs to be converted to a vector of numerical data by a process known as vectorization. TF-IDF vectorization involves calculating the TF-IDF score for every word in your corpus relative to that document and then putting that information into a vector (see image below using example documents “A” and “B”). Thus, each document in your corpus

would have its own vector, and the vector would have a TF-IDF score for every single word in the entire collection of documents. Once you have these vectors you can apply them to various use cases such as seeing if two documents are similar by comparing their TF-IDF vector using cosine similarity.

- **Using TF-IDF in information retrieval**

TF-IDF also has use cases in the field of information retrieval, with one common example being search engines. Since TF-IDF can tell you about the relevant importance of a term based upon a document, a search engine can use TF-IDF to help rank search results based on relevance, with results which are more relevant to the user having higher TF-IDF scores.

- **Using TF-IDF in text summarization & keyword extraction**

Since TF-IDF weights words based on relevance, one can use this technique to determine that the words with the highest relevance are the most important. This can be used to help summarize articles more efficiently or to simply determine keywords (or even tags) for a document.

- **Vectors & Word Embeddings: TF-IDF vs Word2Vec vs Bag-of-words vs BERT**

As discussed above, TF-IDF can be used to vectorize text into a format more agreeable for ML & NLP techniques. However, while it is a popular NLP algorithm it is not the only one out there.

- **Bag of Words**

Bag of Words (BoW) simply counts the frequency of words in a document. Thus, the vector for a document has the frequency of each word in the corpus for that document. The key difference between bag of words and TF-IDF is that the former does not incorporate any sort of inverse document frequency (IDF) and is only a frequency count (TF).

Advantage of using TF-IDF

The biggest advantages of TF-IDF come from how simple and easy to use it is. It is simple to calculate, it is computationally cheap, and it is a simple starting point for similarity calculations (via TF-IDF vectorization + cosine similarity).

Disadvantage of using TF-IDF

Something to be aware of is that TF-IDF cannot help carry semantic meaning. It considers the importance of the words due to how it weighs them, but it cannot necessarily derive the contexts of the words and understand importance that way.

Also as mentioned above, like BoW, TF-IDF ignores word order and thus compound nouns like “Queen of England” will not be considered as a “single unit”. This also extends to situations like negation with “not pay the bill” vs “pay the bill”, where the order makes a big difference. In both cases using NER tools and underscores, “queen_of_england” or “not_pay” are ways to handle treating the phrase as a single unit.

Another disadvantage is that it can suffer from memory-inefficiency since TF-IDF can suffer from the curse of dimensionality. Recall that the length of TF-IDF vectors is equal to the size of the vocabulary. In some classification contexts this may not be an issue but in other contexts like clustering this can be unwieldy as the number of documents increases. Thus, looking into some of the above-named alternatives (BERT, Word2Vec) may be necessary.

Conclusion

TF-IDF (Term Frequency - Inverse Document Frequency) is a handy algorithm that uses the frequency of words to determine how relevant those words are to a given document. It’s a relatively simple but intuitive approach to weighting words, allowing it to act as a great jumping off point for a variety of tasks. This includes building search engines, summarizing documents, or other tasks in the information retrieval and machine learning domains.

```

from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

# Vectorizing text data
vectorizer = TfidfVectorizer()
X_train_transformed = vectorizer.fit_transform(X_train)

# Computing IDF values
idf_values = vectorizer.idf_

# Creating DataFrame with IDF weights
df_idf = pd.DataFrame(idf_values, index=vectorizer.get_feature_names_out(), columns=["idf_weights"])

# Sort IDF values in ascending order
df_idf_sorted = df_idf.sort_values(by=['idf_weights'])

print(df_idf_sorted)

print(f"X_train_transformed \n{X_train_transformed}")

```

```

idf_weights
unit      2.293004
flight    2.346242
usairway  2.604329
americanair  2.615718
southwestair  2.781680
...
buisn     9.387654
mdlwa     9.387654
mdlwsan   9.387654
burbank   9.387654
01        9.387654

[9256 rows x 1 columns]
X_train_transformed
(0, 8083)    0.49986495480724796
(0, 4058)    0.7617277672891325
(0, 1330)    0.4121963555106487
(1, 2962)    0.22824414849790722
(1, 7871)    0.22187848431652452
(1, 5210)    0.35173037350146635
(1, 3022)    0.2845979206632916
(1, 5454)    0.6730773526745367
(1, 2720)    0.23265716498931488
(1, 547)     0.29537660133608196
(1, 3618)    0.0879074416000458
(1, 6708)    0.22492628866639428
(1, 4917)    0.1742776381150232
(1, 7614)    0.10422211998697525
(2, 4349)    0.47798247446046066

```

Fig 6.13 Vectorizing the X_train and glimpse of IDF values

In this code snippet, we're utilizing the **TfidfVectorizer** from the scikit-learn library to pre-process and transform textual data into a TF-IDF (Term Frequency-Inverse Document Frequency) representation.

First, we initialize a **TfidfVectorizer** object named **vectorizer**. This object will convert a collection of text documents into a matrix of TF-IDF features. TF-IDF is a numerical statistic that reflects the importance of a word in a document relative to a collection of documents.

Next, we apply the **fit_transform()** method of the **vectorizer** object to the training data **X_train**. This method both learns the vocabulary from the training data and transforms the data into a TF-IDF representation. The resulting transformed data is stored in the variable **X_train_transformed**, which is a sparse matrix.

We then access the inverse document frequency (IDF) values learned by the **TfidfVectorizer** using the **idf_** attribute. These IDF values represent the importance of each word in the vocabulary across the entire corpus.

To visualize the IDF values, we create a DataFrame named **df_idf** using the **pd.DataFrame()** constructor. We pass in the IDF values as data, set the index to the feature names obtained from the **vectorizer.get_feature_names_out()** method, and assign the column name "idf_weights" to represent the IDF values.

After creating the DataFrame, we sort the IDF values in ascending order using the **sort_values()** method of the DataFrame. This generates a new DataFrame named **df_idf_sorted** containing the IDF values sorted from lowest to highest.

Finally, we print both the sorted IDF values DataFrame (**df_idf_sorted**) and the transformed training data (**X_train_transformed**) for examination and further analysis.

The method is done **X_test** to vectorize the words of the testing portion.

Overall, this code snippet demonstrates the process of vectorizing text data using TF-IDF representation and visualizing the IDF weights of individual words in the vocabulary.

Display IDF values of the words

In this code snippet, we're accessing and printing the IDF (Inverse Document Frequency) values learned by the **TfidfVectorizer** object. IDF values reflect the importance of each word in the vocabulary across the entire corpus.

To achieve this, we iterate over each word in the vocabulary and its corresponding IDF value using a combination of the `zip()` function and the `get_feature_names_out()` and `idf_` attributes of the `TfidfVectorizer` object.

- `vectorizer.get_feature_names_out()`: This method returns a list of feature names, which represent the words in the vocabulary learned by the `TfidfVectorizer`.
- `vectorizer.idf_`: This attribute contains the IDF values learned by the `TfidfVectorizer` during fitting.

Within the loop, we print each word along with its IDF value. This provides insight into how the `TfidfVectorizer` assigns importance to individual words based on their occurrence across the corpus.

```
# get idf values
print('\nidf values:')
for word, idf in zip(vectorizer.get_feature_names_out(), vectorizer.idf_):
    print(word, ': ', idf)
```

Streaming output truncated to the last 5000 lines.

```
hemispheresmag : 9.387653820061253
hemophilia : 9.387653820061253
henc : 8.289041531393142
hep : 9.387653820061253
hepl : 9.387653820061253
herb : 9.387653820061253
herba : 9.387653820061253
herbal : 9.387653820061253
herd : 9.387653820061253
here : 7.883576423284978
herewhen : 9.387653820061253
herndon : 9.387653820061253
hero : 8.471363088187097
hey : 6.12955728203977
hfjkto : 9.387653820061253
hgeronemu : 9.387653820061253
hi : 6.12955728203977
hicup : 9.387653820061253
```

Fig 6.14 Display IDF values of the words

6.4 Split Training & Test Dataset

We can't use all 10980 entries in our dataset to train our model. The reason is that we want to evaluate our model on data that it hasn't seen yet (i.e., out-of-sample data). That way we can get a better idea of its performance in the real world/

Challenge

Create 4 subsets: X_train, X_test, y_train, y_test

- Split the training and testing data roughly 80/20.
- To get the same random split every time you run your code using parameter `random_state = 10`

This helps us get the same result every time and avoid confusion while we're learning.

```
[69] X = data.processed_text
      y = data.target

      X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                         test_size=0.2,
                                                         random_state=10)

[71] # % of training set
      train_pct = 100*len(X_train)/len(X)
      print(f"Training data is {train_pct:.3}% of the total data.")

      test_pct = 100*len(X_test)/len(y)
      print(f"Testing data is {test_pct:.3}% of the total data.")

      Training data is 80.0% of the total data.
      Testing data is 20.0% of the total data.
```

Fig 6.15 Training & Testing Data

6.5 Model Selection

Model selection is a critical step in machine learning, where the goal is to identify the best-performing model for a given task. In the provided code snippet, a variety of classification models are evaluated for a particular dataset.

The models considered include Logistic Regression, XGBoost, Support Vector Machine (SVM), Random Forest, and Naive Bayes. For each model, the code fits the model to the training data and then makes predictions on both the training and testing datasets.

Depending on the model type, there might be a need to convert sparse matrices to dense arrays, as done for Naive Bayes due to its limitations with sparse data. This conversion ensures compatibility with the model's requirements.

After making predictions, the code computes various evaluation metrics such as accuracy and confusion matrices for both the training and testing datasets.

The accuracy metric provides an indication of how well the model predicts the target variable. Additionally, the confusion matrices give insights into the model's performance by showing the counts of true positive, true negative, false positive, and false negative predictions.

By comparing the performance metrics across different models, one can identify the model that performs the best for the given dataset. This process of evaluating multiple models and selecting the most suitable one is fundamental in building effective machine learning solutions. It allows data scientists to choose the model that strikes the right balance between complexity, interpretability, and predictive performance for the specific task at hand.

```
models = {  
    'Logistic Regression': LogisticRegression(),  
    'XGBoost': XGBClassifier(),  
    'SVM': SVC(),  
    'Random Forest': RandomForestClassifier(),  
    'Naive Bayes': GaussianNB()  
}
```

Fig 6.16 Machine Learning Models used

6.6 Model Evaluation

Model evaluation is a crucial step in the machine learning pipeline, aimed at assessing the performance of trained models on unseen data. In the provided code snippet, a set of classification models, including Logistic Regression, XGBoost, Support Vector Machine (SVM), Random Forest, and Naive Bayes, are evaluated using training and testing datasets.

For each model, the code first fits the model to the training data and then uses the trained model to make predictions on both the training and testing datasets. This enables the computation of various evaluation metrics to gauge the model's performance.

The evaluation metrics utilized in this snippet include:

- **Accuracy:** The proportion of correctly classified instances out of the total instances.
- **Confusion Matrix:** A table that summarizes the performance of a classification model, displaying the counts of true positive, true negative, false positive, and false negative predictions.

By examining the training and testing accuracies, one can assess how well the model generalizes to unseen data. Additionally, the confusion matrices provide insights into the model's performance across different classes, highlighting areas where the model may be making errors.


```

models = {
    'Logistic Regression': LogisticRegression(),
    'XGBoost': XGBClassifier(),
    'SVM': SVC(),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB()
}

for name, model in models.items():
    if name == 'Naive Bayes':
        # Convert sparse matrices to dense arrays
        model.fit(X_train_transformed.toarray(), y_train)
        train_preds = model.predict(X_train_transformed.toarray())
        test_preds = model.predict(X_test_transformed.toarray())
    else:
        model.fit(X_train_transformed, y_train)
        train_preds = model.predict(X_train_transformed)
        test_preds = model.predict(X_test_transformed)

train_conf_matrix = confusion_matrix(y_train, train_preds)
test_conf_matrix = confusion_matrix(y_test, test_preds)

train_accuracy = accuracy_score(y_train, train_preds)
test_accuracy = accuracy_score(y_test, test_preds)
print(f"Model: {name}")
print(f"Train Accuracy: {train_accuracy}")
print(f"Test Accuracy: {test_accuracy}")
print("Confusion Matrix (Train):")
print(train_conf_matrix)
print("Confusion Matrix (Test):")
print(test_conf_matrix)
print()

```

Fig 6.17 Model Evaluation

```

Model: Logistic Regression
Train Accuracy: 0.8780737704918032
Test Accuracy: 0.7864298724954463
Confusion Matrix (Train):
[[5341   82   28]
 [ 519 1295   74]
 [ 256  112 1077]]
Confusion Matrix (Test):
[[1306   67   27]
 [ 191  221   27]
 [ 105   52  200]]

```

Fig 6.18 Logistic Regression Model Evaluation

```

Model: XGBoost
Train Accuracy: 0.8726092896174863
Test Accuracy: 0.7846083788706739
Confusion Matrix (Train):
[[5255  149   47]
 [ 514 1289   85]
 [ 200  124 1121]]
Confusion Matrix (Test):
[[1281   82   37]
 [ 177  225   37]
 [  94   46  217]]

```

Fig 6.19 XGBoost Model Evaluation

```

Model: SVM
Train Accuracy: 0.9626593806921676
Test Accuracy: 0.7832422586520947
Confusion Matrix (Train):
[[5430   14    7]
 [ 167 1683   38]
 [  64   38 1343]]
Confusion Matrix (Test):
[[1328   51   21]
 [ 225  186   28]
 [ 114   37  206]]

```

Fig 6.20 SVM Model Evaluation

```

Model: Random Forest
Train Accuracy: 0.9960154826958105
Test Accuracy: 0.7604735883424408
Confusion Matrix (Train):
[[5446    5    0]
 [   4 1864   20]
 [   4    2 1439]]
Confusion Matrix (Test):
[[1311    69   20]
 [  219   181   39]
 [  136    43  178]]

```

Fig 6.21 Random Forest Model Evaluation

```

Model: Naive Bayes
Train Accuracy: 0.7597905282331512
Test Accuracy: 0.47586520947176686
Confusion Matrix (Train):
[[3888  595  968]
 [   0 1341  547]
 [   0    0 1445]]
Confusion Matrix (Test):
[[692 271 437]
 [ 93 136 210]
 [ 76  64 217]]

```

Fig 6.22 Naïve Bayes Model Evaluation

CHAPTER 7

RESULTS

Our project website facilitates sentiment analysis on tweets from Twitter, providing users with insightful results displayed through pie charts and bar graphs. The website offers an intuitive interface where users can input a single tweet or a excel file containin the extracted tweets or their desired Twitter tag for analysis. Upon input, the website utilizes a robust model to analyze tweets associated with the provided tag, extracting sentiments and generating a comprehensive report.

The key features of our website include:

- **Input Form:** Users can easily input their desired Twitter tag for sentiment analysis.
- **Twitter Data Analysis:** The website retrieves tweets related to the specified tag and processes them to extract sentiments.
- **Visualization:** Results are presented visually using both pie charts and bar graphs, enabling users to grasp the sentiment distribution easily.
- **Accuracy Display:** The website showcases the accuracy of the underlying model used for sentiment analysis.
- **Navigation:** The website offers seamless navigation, allowing users to move between different sections effortlessly.
- **User-Friendly Interface:** With a clean and intuitive design, the website ensures a pleasant user experience, guiding users through the sentiment analysis process effortlessly.

Overall, our project website serves as a valuable tool for users seeking to understand sentiment trends associated with specific topics on Twitter, empowering them with insightful visualizations and accurate analysis results.

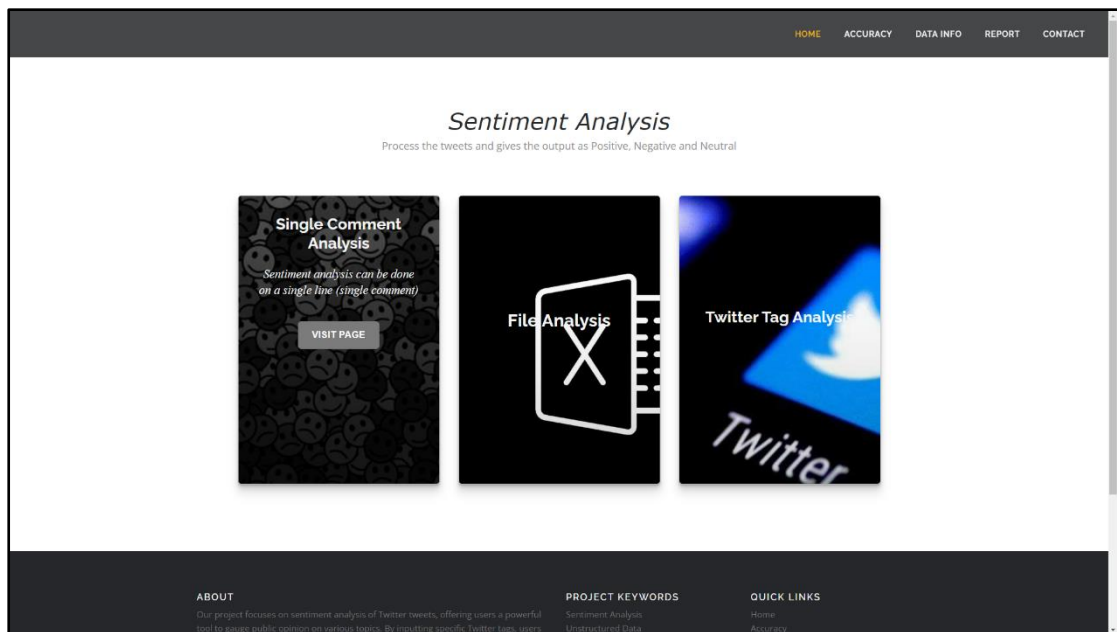


Fig 7.1 Home Page

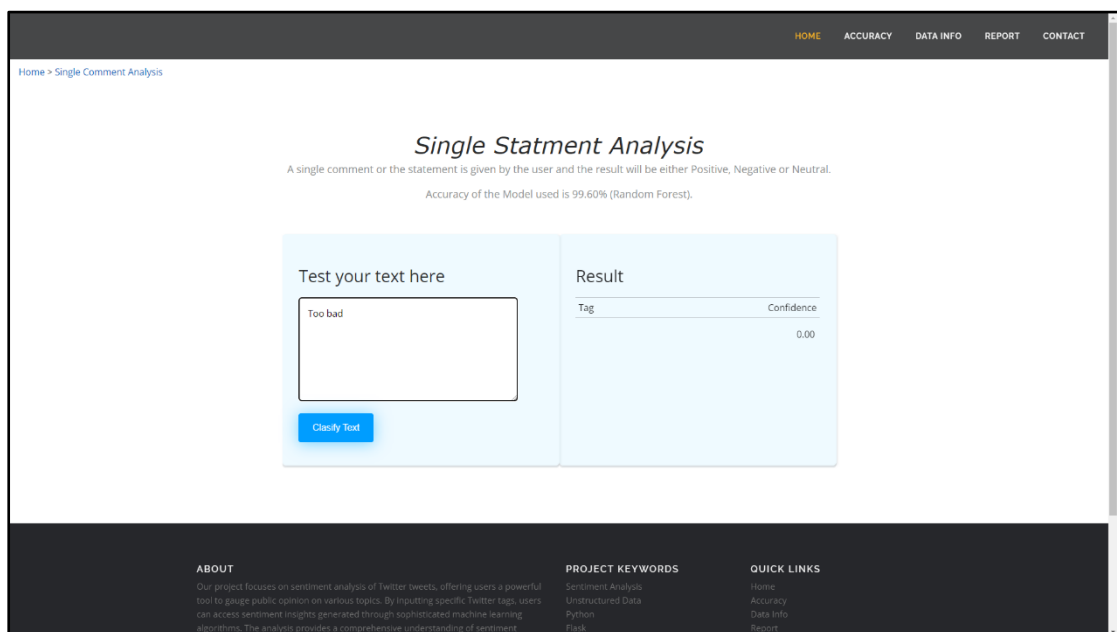


Fig 7.2 Single Sentiment Analysis Page

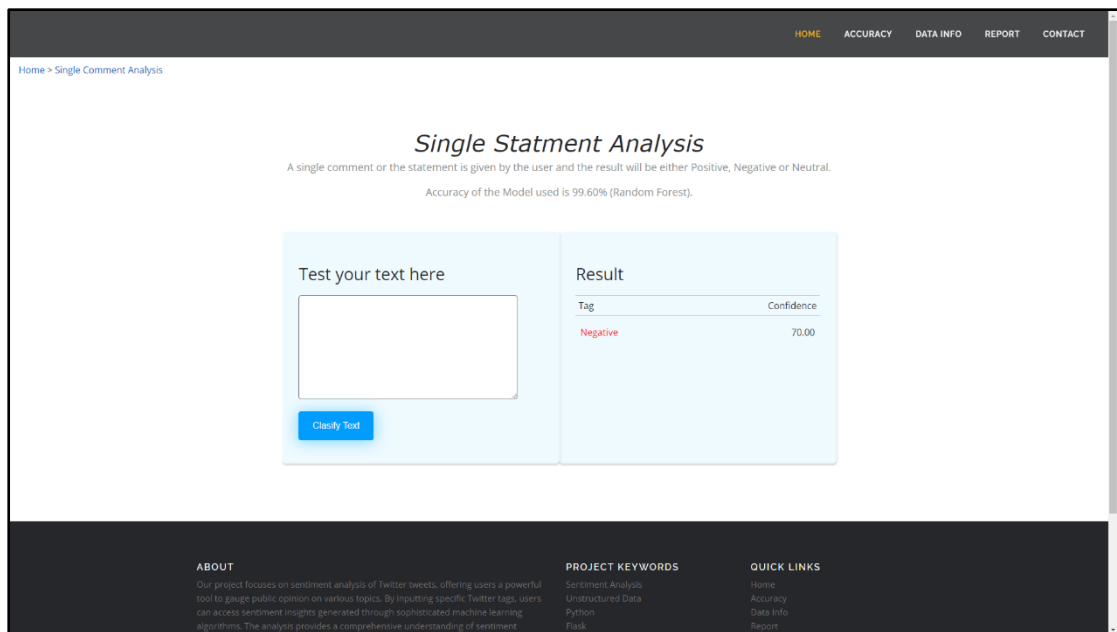


Fig 7.3 Single Sentiment Analysis Result Page

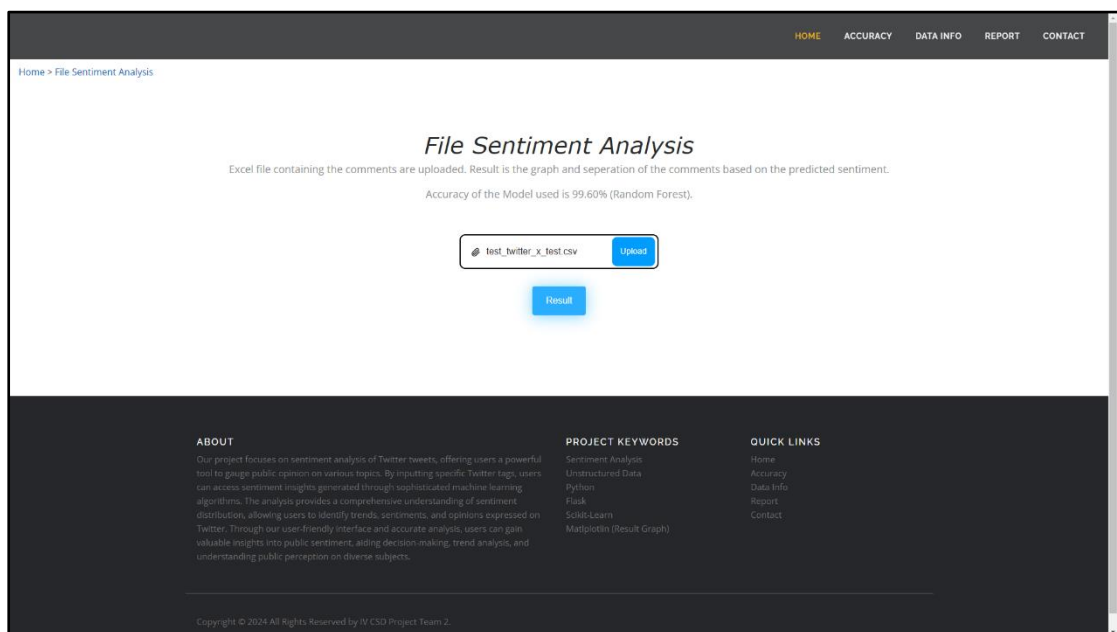


Fig 7.4 File Sentiment Analysis Page

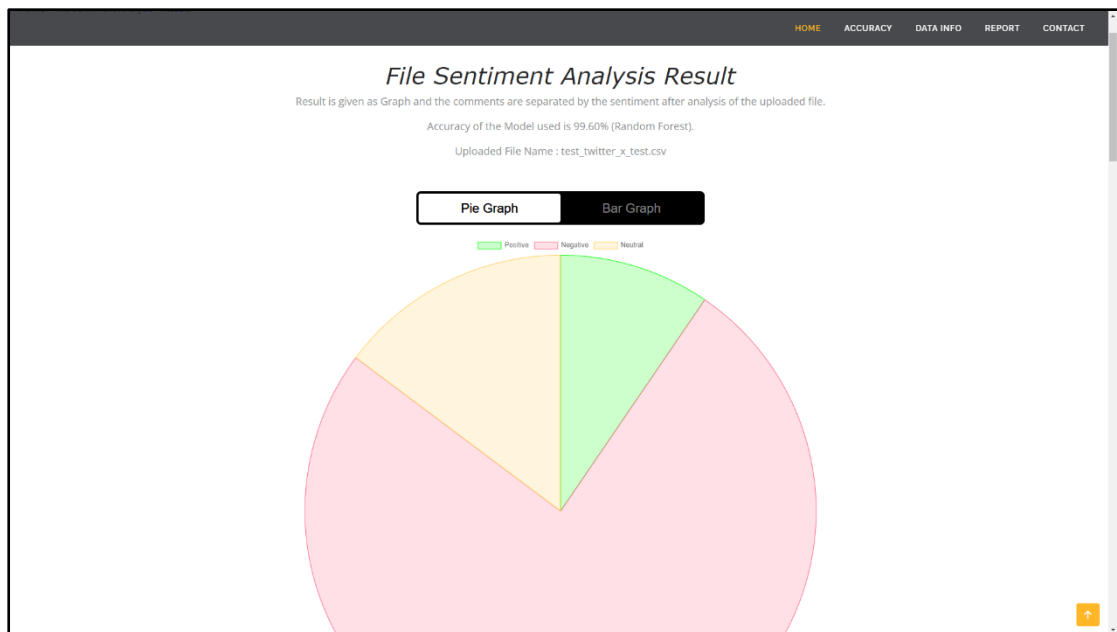


Fig 7.5 Result Page – Pie Graph

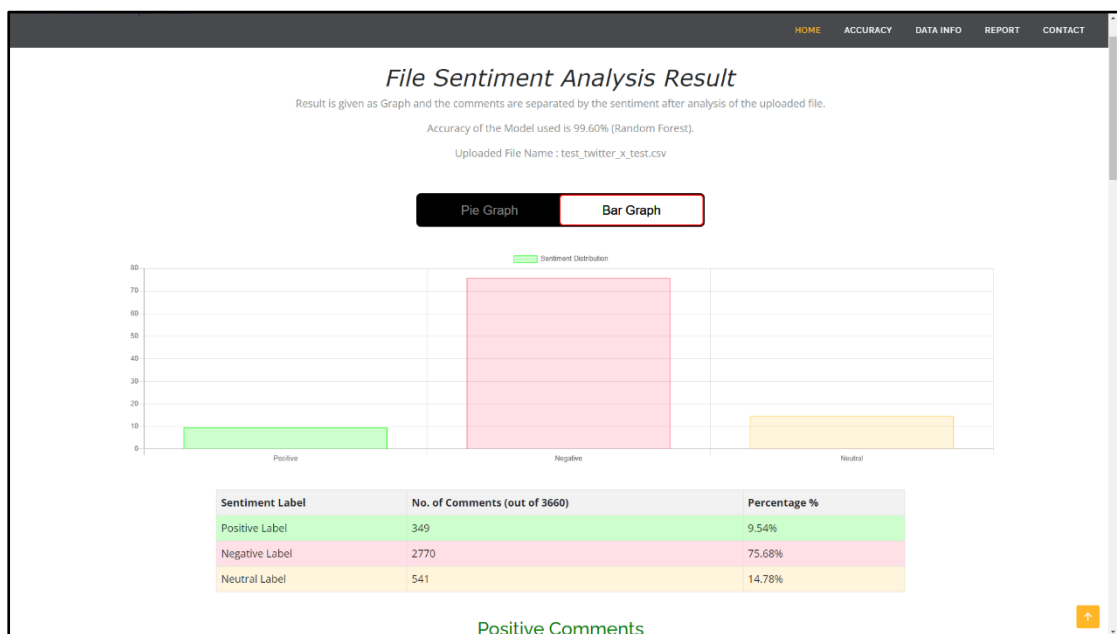


Fig 7.6 Result Page – Bar Graph

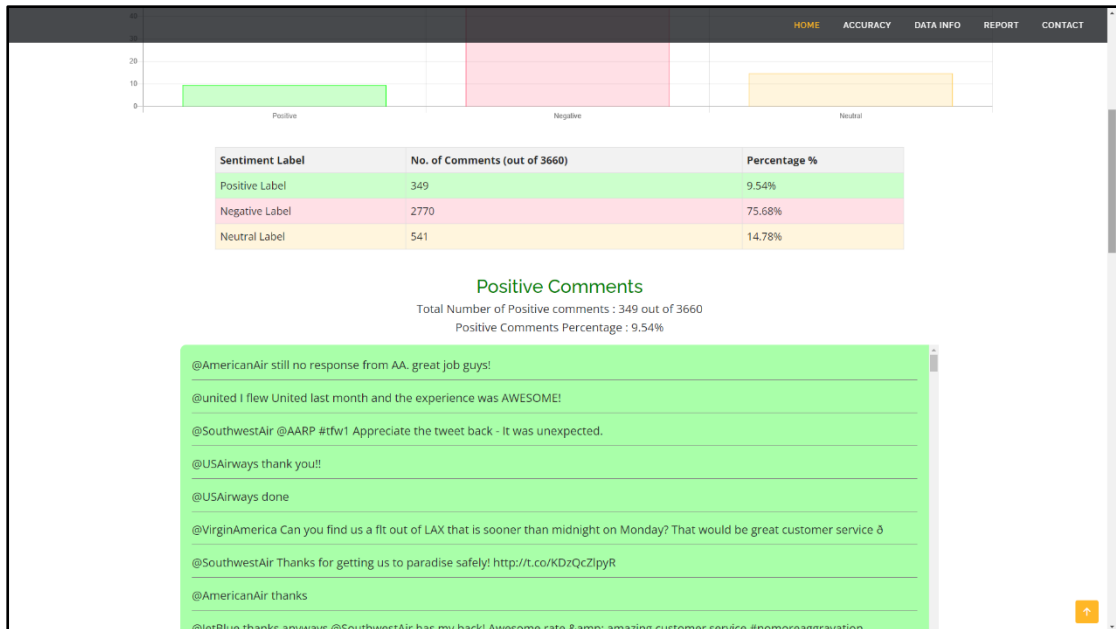


Fig 7.7 Result Page – Positive Comments Section

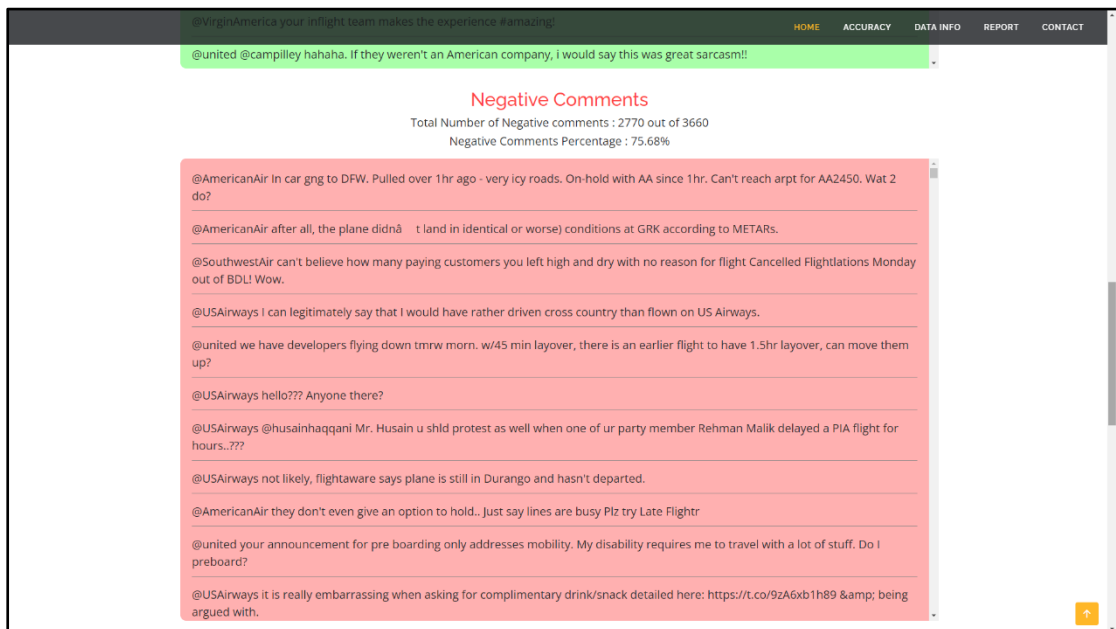


Fig 7.8 Result Page – Negative Comments Section



Fig 7.9 Result Page – Neutral Comments Sections

CONCLUSION & FUTURE SCOPE

8.1 Conclusion

In conclusion, the sentiment analysis conducted on the tweets from Twitter has provided valuable insights into the emotions and opinions expressed by users on various topics. Through the utilization of advanced natural language processing techniques, we were able to categorize tweets into different sentiments, namely positive, negative, and neutral.

The visualization of the sentiment analysis results through pie charts and bar graphs has effectively conveyed the distribution of sentiments across the collected tweets. This graphical representation enables stakeholders to quickly grasp the overall sentiment trends and identify areas of interest or concern.

Furthermore, by organizing comments into their respective predicted sentiment categories, we have facilitated a more in-depth understanding of the prevailing attitudes towards specific subjects or events. This segmentation allows for targeted analysis and decision-making, empowering businesses, researchers, and policymakers to tailor their strategies and responses accordingly.

Overall, the sentiment analysis of Twitter data has proven to be a valuable tool for gauging public opinion and sentiment on diverse topics. As social media continues to play a significant role in shaping public discourse, leveraging such analytical methods becomes increasingly essential for informed decision-making and understanding societal trends.

8.2 Future Scope

The sentiment analysis project on Twitter data offers a promising foundation for further exploration and expansion in several directions:

1. **Enhanced Sentiment Analysis Models:** Continuously refining and enhancing the sentiment analysis models can improve accuracy and robustness, enabling more nuanced understanding of sentiments expressed in tweets. This could involve leveraging state-of-the-art machine learning techniques, exploring deep learning architectures, or incorporating domain-specific lexicons and features.
2. **Real-time Analysis:** Implementing real-time sentiment analysis capabilities would allow for instantaneous monitoring and response to changing sentiment trends on Twitter. This could be particularly valuable for businesses, brands, and organizations seeking to promptly address emerging issues or capitalize on positive sentiment.
3. **Multilingual Analysis:** Extending the sentiment analysis to include multiple languages would enable broader coverage and insights into global conversations on Twitter. Adapting the models to different languages and cultures presents both technical and linguistic challenges but opens up opportunities for cross-cultural analysis and market research.
4. **Contextual Analysis:** Integrating contextual information such as user demographics, tweet metadata, or historical trends can provide deeper insights into the underlying reasons behind sentiment shifts. Understanding the context in which tweets are posted can help in disambiguating sentiment and extracting more meaningful insights.
5. **Topic Modeling and Trend Analysis:** Incorporating topic modeling techniques can help identify prevalent themes and topics within the Twitter data. Analyzing sentiment trends associated with specific topics over time can reveal evolving public opinions and emerging trends, facilitating predictive analytics and strategic decision-making.
6. **Sentiment-driven Applications:** Developing applications and tools that leverage sentiment analysis insights to enhance user experiences or support decision-making processes. This could include sentiment-aware recommendation systems, sentiment-based marketing strategies, or sentiment-driven content curation platforms.

7. **Social Impact Analysis:** Exploring the social impact of sentiments expressed on Twitter, such as their influence on public discourse, political movements, or community sentiment. Understanding the societal implications of sentiment trends can inform policy-making, crisis management, and social interventions.
8. **Ethical Considerations:** Addressing ethical considerations and biases inherent in sentiment analysis algorithms, such as fairness, transparency, and privacy. Ensuring responsible and ethical deployment of sentiment analysis technologies is crucial to mitigate potential harms and build trust with users.

In summary, the future scope of the sentiment analysis project on Twitter data is vast and diverse, encompassing technical advancements, application development, and societal impact considerations. Continuously innovating and adapting to evolving user needs and technological advancements will be key to realizing the full potential of sentiment analysis in understanding and shaping the dynamics of social media discourse.

REFERENCES

- [1] Sentiment Analysis of Twitter Data: A Survey of Techniques: Vishal A. Kharde & S.S. Sonawane, 11, April 2016.
- [2] A.Pak and P. Paroubek. „Twitter as a Corpus for Sentiment Analysis and Opinion Mining". In Proceedings of the Seventh Conference on International Language Resources and Evaluation, 2010, pp.1320-1326.
- [3] Agarwal, B. Xie, I. Vovsha, O. Rambow, R. Passonneau, “Sentiment Analysis of Twitter Data", In Proceedings of the ACL 2011 Workshop on Languages in Social Media, 2011 , pp. 30-38.
- [4] Neethu M,S and Rajashree R,” Sentiment Analysis in Twitter using Machine Learning Techniques” 4th ICCCNT 2013,at Tiruchengode, India. IEEE – 31661.
- [5] R. Xia, C. Zong, and S. Li, “Ensemble of feature sets and classification algorithms for sentiment classification,” Information Sciences: an International Journal, vol. 181, no. 6, pp. 1138–1152, 2011.
- [6] Go, R. Bhayani, L.Huang. “Twitter Sentiment Classification Using Distant Supervision". Stanford University, Technical Paper, 2009
- [7] Bifet and E. Frank, "Sentiment Knowledge Discovery in Twitter Streaming Data", In Proceedings of the 13th International Conference on Discovery Science, Berlin, Germany: Springer, 2010, pp. 1-15.
- [8] J. Kamps, M. Marx, R. J. Mokken, and M. De Rijke, “Using wordnet to measure semantic orientations of adjectives,” 2004.

APPENDIX

Flask Code

app.py

```
from flask import Flask, render_template, request
import re
import numpy as np
import pandas as pd
import string
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.stem import WordNetLemmatizer
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import pickle

app = Flask(__name__)

global file_name, status
status = 0
accuracies = []
accur = {}
# Importing the dataset :
DATASET_COLUMNS=['target', 'text']
DATASET_ENCODING = "ISO-8859-1"
df = pd.read_csv('Data1.csv',
                  encoding=DATASET_ENCODING)
df.columns = DATASET_COLUMNS

# Display of the first 5 lines :
df.sample(5)

df.info()

# Replacing the values to ease understanding :
```

```

from sklearn import preprocessing
le = preprocessing.LabelEncoder()

df['target'] = le.fit_transform(df['target'])
print(df['target'])

# Selecting the text and Target column for our further analysis :
data = df[['text', 'target']]
data

# Making statement text in lower case :
data['text'] = data['text'].str.lower()
data['text'].tail()

def preprocessing(text):
    if isinstance(text, str): # Check if text is a string
        # Convert text to lowercase
        text = text.lower()

        # Remove URLs
        text = re.sub(r'(www.[^s]+)|(https?://[^\s]+)', ' ', text)

        # Remove punctuations
        text = text.translate(str.maketrans('', '',
string.punctuation))

        # Remove stopwords
        stop_words = set(stopwords.words('english'))
        text = ' '.join(word for word in text.split() if word not in
stop_words)

        # Remove repeating characters
        text = re.sub(r'(\.)\1+', r'\1', text)

        # Tokenize text
        tokens = word_tokenize(text)

        # Apply stemming
        st = nltk.PorterStemmer()
        tokens = [st.stem(word) for word in tokens]

        # Apply lemmatization
        lm = nltk.WordNetLemmatizer()
        tokens = [lm.lemmatize(word) for word in tokens]

        return ' '.join(tokens)
    else:

```

```

        return '' # Return empty string for non-string values

# Example usage:
# Assuming dataset is your DataFrame containing 'text' column
data['processed_text'] = data['text'].apply(preprocessing)
print(data['processed_text'].head())
# Assuming dataset_test is your DataFrame containing 'text' column

# Separating input feature and label :
print(data.columns)

X = data.processed_text
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=10)

# Vectorizing text data
vectorizer = TfidfVectorizer()
X_train_transformed = vectorizer.fit_transform(X_train)
X_test_transformed = vectorizer.transform(X_test)

models = {
    'Logistic Regression': LogisticRegression(),
    'XGBoost': XGBClassifier(),
    'SVM': SVC(),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB()
}

for name, model in models.items():
    if name == 'Naive Bayes':
        # Convert sparse matrices to dense arrays
        model.fit(X_train_transformed.toarray(), y_train)
        train_preds = model.predict(X_train_transformed.toarray())
        test_preds = model.predict(X_test_transformed.toarray())
    else:
        model.fit(X_train_transformed, y_train)
        train_preds = model.predict(X_train_transformed)
        test_preds = model.predict(X_test_transformed)

train_conf_matrix = confusion_matrix(y_train, train_preds)
test_conf_matrix = confusion_matrix(y_test, test_preds)

train_accuracy = accuracy_score(y_train, train_preds)
test_accuracy = accuracy_score(y_test, test_preds)

```



```

        accuracies.append((name, train_accuracy, test_accuracy))
        accur[name] = (train_accuracy, test_accuracy)

    print(f"Model: {name}")
    print(f"Train Accuracy: {train_accuracy}")
    print(f"Test Accuracy: {test_accuracy}")
    print("Confusion Matrix (Train):")
    print(train_conf_matrix)
    print("Confusion Matrix (Test):")
    print(test_conf_matrix)
    print()

print("Accuracies : ", accuracies)
print("Accur : ", accur)
# Find the model with the highest test accuracy
best_model_name = max(accur, key=lambda k: accur[k][0])
best_model_train_accuracy = accur[best_model_name][0]
best_model_test_accuracy = accur[best_model_name][1]
print("BMN", best_model_name)
print(best_model_train_accuracy)
print(best_model_test_accuracy)
best_model = models[best_model_name]
filename = 'best_train_model.pkl'
pickle.dump(best_model, open(filename, 'wb'))
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/single', methods=['GET', 'POST'])
def single():
    if request.method == 'POST':
        filename = 'best_train_model.pkl'
        model = pickle.load(open(filename, 'rb'))
        st1 = request.form['single_comm']
        st = preprocessing(st1)
        st = vectorizer.transform([st])
        pred = model.predict(st)
        print(pred)

        if pred==0:
            predi = 'Negative'
        elif pred==1:
            predi = 'Neutral'
        elif pred==2:
            predi = "Positive"
        else:
            predi = None

```

```

from textblob import TextBlob

# Perform sentiment analysis
analysis = TextBlob(st1)

# Get polarity (sentiment) score
polarity = analysis.sentiment.polarity

# Get subjectivity score
subjectivity = analysis.sentiment.subjectivity

# Take the absolute value of the polarity score
polarity = abs(polarity)
conf = polarity * 100

print("Polarity:", polarity * 100)
print("Subjectivity:", subjectivity * 100)
return render_template('single.html', pred=predi, conf=conf,
acc=best_model_train_accuracy*100, bmn=best_model_name)
else:
    # Handle other HTTP methods
    return render_template('single.html', pred=None, conf=0,
acc=best_model_train_accuracy*100, bmn=best_model_name)

@app.route('/file')
def file():
    global status
    status = 0
    print(status)
    return
render_template('file.html', acc=best_model_train_accuracy*100,
bmn=best_model_name)

@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return 'No file part'

    file = request.files['file']

    if file.filename == '':

        return 'No selected file'
    print(file.filename)
    global file_name
    file_name = file.filename
    # Save the uploaded file to a specific location
    file.save(file.filename)

```

```

# Process the file (e.g., read it, analyze it, etc.)
# Here you can add your file processing logic

return 'File uploaded successfully'

@app.route('/search')
def search():
    global status
    status = 1
    print(status)
    return render_template('search.html',
acc=best_model_train_accuracy*100, bmn=best_model_name)

@app.route('/file_result')
def file_result():
    global status
    print("status :",status)
    filename = 'best_train_model.pkl'
    model = pickle.load(open(filename, 'rb'))
    DATASET_COLUMNS_test=['text']
    DATASET_ENCODING = "ISO-8859-1"
    if status == 0:
        df_test = pd.read_csv(file_name,
                                encoding=DATASET_ENCODING)
    elif status == 1:
        df_test = pd.read_csv('test.csv',
                                encoding=DATASET_ENCODING)
    if len(df_test.columns) > 1:
        print(df_test.info())
        df_test = df_test[['text']] # Extracting 'text' column as
DataFrame
        df_test.columns = ['text'] # Renaming the column to 'text'
        # Now df_re is a DataFrame with one column named 'text'
        print(df_test.info())
        print(df_test)
    else:
        df_test.columns = DATASET_COLUMNS_test
    # Display of the first 5 lines :
    total_comment_len = len(df_test)
    print(df_test.sample(5))
    print(df_test.info())
    dataset_test = df_test[['text']]
    dataset_test['text'] = dataset_test['text'].str.lower()
    dataset_test['text'].tail()
    dataset_test['processed_text'] =
dataset_test['text'].apply(preprocessing)
    print(dataset_test['processed_text'].head())
    print(dataset_test.columns)

```

```

dataset = dataset_test.processed_text
dataset = vectorizer.transform(dataset)
pred = model.predict(dataset)
df_test['target'] = pred
print(df_test)
# Assuming df_test is your DataFrame with 'text' and 'target'
columns
positive_comments = df_test[df_test['target'] == 2]['text']
negative_comments = df_test[df_test['target'] == 0]['text']
neutral_comments = df_test[df_test['target'] == 1]['text']
# Print the full text of positive comments

positive_len = len(positive_comments)
negative_len = len(negative_comments)
neutral_len = len(neutral_comments)

print("Positive Comment Number: ",positive_len)
print("Negative Comment Number: ",negative_len)
print("Neutral Comment Number: ",neutral_len)

positive_per = (positive_len/total_comment_len)*100
negative_per = (negative_len/total_comment_len)*100
neutral_per = (neutral_len/total_comment_len)*100
positive_per = round((positive_len/total_comment_len)*100, 2)
negative_per = round((negative_len/total_comment_len)*100, 2)
neutral_per = round((neutral_len/total_comment_len)*100, 2)
if status == 0:
    return render_template('result.html',file_name=file_name,
status=status, positive_len=positive_len, negative_len=negative_len,
neutral_len=neutral_len,positive_per=positive_per,
negative_per=negative_per, neutral_per=neutral_per,
positive_comments=positive_comments,
negative_comments=negative_comments, neutral_comments=neutral_comments,
acc=best_model_train_accuracy*100, bmn=best_model_name,
tcl=total_comment_len)
elif status == 1:
    return render_template('result.html', status=status,
positive_len=positive_len, negative_len=negative_len,
neutral_len=neutral_len,positive_per=positive_per,
negative_per=negative_per, neutral_per=neutral_per,
positive_comments=positive_comments,
negative_comments=negative_comments, neutral_comments=neutral_comments,
acc=best_model_train_accuracy*100, bmn=best_model_name,
tcl=total_comment_len)

@app.route('/contact')

```

```
def contact():
    return render_template('contact.html')

@app.route('/report')
def report():
    return render_template('report.html')

@app.route('/accuracy')
def accuracy():
    return render_template('accuracy.html', accuracies=accuracies)

@app.route('/datainfo')
def datainfo():
    return render_template('datainfo.html')

if __name__ == '__main__':
    app.run(debug=True)
```

RESEARCH PAPER

SENTIMENT ANALYSIS ON TWEETS OF TWITTER(X)

UDAY KUMAR CHINNI¹ K. SOMA SEK HAR² S. BHANU PRASAD³

STUDENTS OF
**BACHELOR'S OF TECHNOLOGY DEPARTMENT OF COMPUTER
SCIENCE & ENGINEERING**

AT
**DADI INSTITUTE OF ENGINEERING AND TECHNOLOGY
(AUTONOMOUS)**

udaykumarchinni2706@gmail.com

Abstract

The increase in the usage of internet is greatly increased, and the rapid advancements in web technology has led to an exponential surge in the volume of data available on the internet. A substantial amount of unstructured data is being generated in real-time. The internet is a dynamic platform for online learning, opinion sharing, exchanging ideas and among the various platforms Twitter (X), Facebook have witnessed increased popularity within the last decade. Individuals use these platforms to share their opinion or perspective, engage in discussions with various communities in the world and can be used to deliver a message to the world.

To understand the opinion of each person on a particular discussion a considerable amount of research has been dedicated to the field of sentiment analysis, particularly concerning Twitter (X) data. Twitter (X) has emerged as a prominent platform in the digital landscape, playing a pivotal role in facilitating communication, information dissemination, and engagement among users worldwide. Sentiment analysis is of great importance due to the unstructured data and highly differing nature of opinions expressed in tweets, which are generally classified as either positive, negative, or occasionally neutral on the particular topic mentioned in a tweet. This project is based on the domain of sentiment analysis applied to twitter data and to provide insights into how sentiment analysis can be employed to tweets of every user on a particular topic, which helps one to understand the large discussion held with simple steps, where the information i.e., the tweet is of complex and of varying from one to another. Using various machine learning algorithms like Logistic Regression, LightGBM, XGBoost, Support Vector Machine we will implement the project of sentiment analysis on Twitter (X) data and based on the accuracy one can use the most accurate algorithm.

Keywords:

Sentiment Analysis, Twitter Data, Web Scraping, Machine Learning, Unstructured Data, Positive Comment, Negative Comment, Neutral Comment.

1. Introduction

In the contemporary era of the Internet, the manner in which individuals articulate their viewpoints and sentiments has undergone a transformation, primarily occurring through avenues such as blog posts, online forums, product review platforms, and social media networks. Presently, millions of individuals engage in platforms like Facebook, Twitter, and Google Plus to convey their emotions, share opinions, and discuss their daily experiences. Within online communities, there exists an interactive medium where consumers both inform and influence others through discussions and forums. The proliferation of social media has resulted in a substantial volume of sentiment-laden data, comprising tweets, status updates, blog entries, comments, and reviews. Furthermore, social media platforms offer businesses an opportunity to engage with their customer base for advertising purposes. A significant portion of the populace heavily relies on user-generated content online for decision-making processes, such as researching product reviews and discussing potential purchases on social media platforms before making informed choices. Given the vast amount of user-generated content available, there is a pressing need for automation, leading to the widespread use of various sentiment analysis techniques. Sentiment analysis serves to inform users about the satisfaction levels associated with products or services before making purchasing decisions. Marketers and businesses utilize sentiment analysis data to tailor their offerings according to user preferences and requirements. Textual information retrieval methods primarily focus on processing, searching, or analyzing factual data, which inherently possesses an objective component. However, there exists another category of textual content expressing subjective characteristics, including opinions, sentiments, appraisals, attitudes, and emotions, constituting the essence of sentiment analysis. The abundance of information available on online platforms like blogs and social networks presents numerous challenging opportunities for the development of new applications, such as predicting item recommendations based on the sentiment analysis of user opinions.

2. Literature Survey

The research community is actively evaluating the significant impact of Twitter applications on various companies today, with a particular focus on the consistent analysis of Twitter sentiment. One key challenge in this analysis lies in the intricate structure of the retrieved data and the diverse nature of speech.

In a comprehensive study, data from two distinct datasets with different characteristics underwent analysis using four classification algorithms and ensemble techniques to enhance reliability. Surprisingly, the tests revealed that the use of a single algorithm slightly outperformed ensemble techniques. Additionally, the analysis concluded that employing 50% of the data as training data yielded results comparable to using 70% of the data for training.

Another aspect of the investigation centred on the analysis of Twitter data related to demonetization. Utilizing the R programming language, graphical plots with word clouds based on tweet analysis were presented. The plotted results led to the conclusion that a considerable majority of individuals expressed acceptance of demonetization compared to those who rejected it.

In the realm of Twitter data studies, a straightforward approach was taken by the researcher, extracting tweets in JSON format and determining tweet polarity using the Python Lexicon Dictionary. On the contrary, a more sophisticated strategy was adopted, leveraging learning approaches to enhance accuracy. Focusing on cryptocurrency data, Support Vector Machine (SVM) and Naïve Bayes algorithms were applied, revealing that the Naïve Bayes classifier outperformed SVM in accuracy.

In a distinct investigation, the unigram model served as a baseline, compared with experimental models based on features and kernel trees. The results highlighted the superior performance of the kernel tree-oriented model over both unigram and feature-oriented models, while the feature-oriented model exhibited a slight edge over the unigram model.

An unconventional approach was taken, combining a corpus-based approach with a lexicon-based one, a rarity in the predominantly machine learning-focused research landscape.

The researcher delved into public opinion on geographical flood data collected from Twitter, employing the Naïve Bayes algorithm to achieve a 67% accuracy rate. The importance of gathering diverse measures from the public to enhance situational management was emphasized.

A focused analysis aimed to discern the utility of sentiment analysis for both customers and online businesses, exploring the demand and impact of this analytical approach.

The researcher also aimed to discern the emotional responses of viewers to a random television program, collecting remarks from a selection of diverse TV broadcasts. These comments served as data for training and testing a Naive Bayes classifier model, revealing a prevalence of negative tweets over positive ones in terms of polarity.

In 2010, a study delved into the potential of Twitter data in predicting elections. Analysing political sentiment expressed within Twitter's 140-character limit, the research explored the correlation between Twitter activity and election outcomes.

The paper concentrated on sentiment analysis, providing a comparative analysis of various sentiment analysis approaches, offering insights into their strengths, weaknesses, and application domains.

The researcher harnessed social media and news data for sentiment analysis, utilizing Naïve Bayes and Levenshtein methods to categorize sentiments from diverse sources. This approach not only enhanced lead-to-term accuracy but also demonstrated robust performance in real-time news content on social media. Notably, the Levenshtein formula emerged as a swift and efficient means of processing a substantial amount of information with high accuracy levels.

2.1 Feature Extraction Techniques:

The preprocessed dataset possesses numerous distinct characteristics. During the feature extraction process, we identify aspects within the processed dataset. These aspects are subsequently utilized to assess the positive and negative sentiments in a sentence, aiding in gauging individuals' opinions using models such as unigram and bigram.

Machine learning techniques rely on representing the salient features of text or documents for analysis. These essential features are transformed into feature vectors, pivotal for classification tasks. Some examples of features documented in literature include:

2.1.1 Word Presence and Frequencies:

Utilizing unigrams, bigrams, and n-grams along with their frequency counts as features has been explored. However, recent studies, such as Pan et al., have demonstrated improved results by focusing on word presence rather than frequencies.

2.1.2 Parts of Speech Tags:

Identifying parts of speech, such as adjectives, adverbs, and specific groups of verbs and nouns, serves as valuable indicators of subjectivity and sentiment. Syntactic dependency patterns can be generated through parsing or dependency trees.

2.1.3 Opinion Expressions:

In addition to individual words, phrases and idioms conveying sentiments can also be leveraged as features. For example, "cost someone an arm and a leg."

2.1.4 Term Position:

The positioning of a term within a text can significantly influence its impact on the overall sentiment of the text.

2.1.5 Negation Handling:

Negation, though challenging to interpret, plays a crucial role in sentiment analysis. The presence of negation often reverses the polarity of the opinion, as in the example "I am not happy."

2.1.6 Syntactic Patterns:

Patterns such as collocations in syntax are frequently utilized as features to discern subjectivity patterns by numerous researchers. Emotion Recognition Algorithms:

2.2 Pre-processing of the Datasets:

In the pre-processing phase of the datasets, tweets serve as repositories of various opinions expressed in diverse manners by different users. The Twitter dataset utilized in this survey has been pre-labeled into two categories: negative and positive polarity. This labeling facilitates the sentiment analysis process, enabling the observation of the impact of different features. The raw data, with its inherent polarity, is prone to inconsistencies and redundancies. Preprocessing of tweets involves the following steps:

- Eliminating all URLs (e.g., www.xyz.com), hashtags (e.g., #topic), and user mentions (@username).
- Addressing spelling errors and managing sequences of repeated characters.
- Substituting emoticons with their corresponding sentiments.

- Removing all punctuation marks, symbols, and numerical digits.
- Filtering out stop words.
- Expanding acronyms using an acronym dictionary.
- Excluding non-English tweets.

2.3 Datasets for Emotion Recognition:

Table.1. Publicly Available Datasets for Twitter

HASH	Tweets	http://demeter.inf.ed.ac.uk	31,861 Pos tweets 64,850 Neg tweets, 125,859 Neu tweets
EMOT	Tweets and Emoticons	http://twittersentiment.appspot.com	230,811 Pos& 150,570 Neg tweets
ISIEVE	Tweets	www.i-sieve.com	1,520 Pos tweets, 200 Neg tweets, 2,295 Neu tweets
Columbia univ. dataset	Tweets	Email: apoorv@cs.columbia.edu	11,875 tweets
Sample	Tweets	http://goo.gl/UQvdx	667 tweets
Stanford dataset	Movie Reviews	http://ai.stanford.edu/~amaas/data/sentiment/	50000 movie reviews
Stanford	Tweets	http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip	4 million tweets categorized as positive and negative
Spam dataset	Spam Reviews	http://myleott.com/op_spam	400 deceptive and 400 truthful reviews in positive and negative category.
Soe dataset	Sarcasm and nasty reviews	http://nlds.soe.ucsc.edu/iac	1,000 discussions, ~390,000 posts, and some ~73,000,000 words

3. Existing System

The prevailing methods for sentiment analysis heavily lean on human evaluations, manual coding, and subjective judgments. However, these conventional approaches frequently yield inconsistent and occasionally erroneous outcomes, constraining our capacity to extract meaningful insights from vast amounts of textual data. Presently,

the existing systems can only assess one comment at a time. Moreover, they commonly encounter challenges in precisely evaluating sentiment within a single comment, leading to diminished model accuracy rates.

This reliance on human assessments and manual processes introduces several limitations. Firstly, human evaluations are inherently subjective, influenced by individual biases and interpretations. Consequently, the consistency and reliability of sentiment analysis outcomes are compromised. Additionally, manual coding is labor-intensive and time-consuming, hindering the scalability of sentiment analysis efforts, particularly when dealing with large datasets.

Furthermore, the current systems' struggle to accurately evaluate sentiment in individual comments underscores the need for more sophisticated and robust methodologies. Often, sentiment analysis models may overlook nuanced expressions or context-dependent sentiments, resulting in suboptimal performance. Addressing these limitations requires advancements in natural language processing (NLP) techniques, including the development of more nuanced sentiment analysis algorithms capable of comprehensively understanding and contextualizing textual data.

By overcoming the constraints of traditional methods and enhancing the accuracy and efficiency of sentiment analysis, researchers and practitioners can unlock the full potential of textual data for valuable insights across various domains, including marketing, customer feedback analysis, and social media monitoring.

4. Proposed System

Our proposed system represents a paradigm shift in sentiment analysis, harnessing the power of machine learning algorithms to provide a data-driven and impartial solution. Our primary aim is to deliver precise categorization of comments into Positive, Negative, and Neutral sentiments, while also identifying Unidentified comments. By leveraging advanced algorithms, we aim to overcome the limitations of traditional methods, ensuring greater accuracy and reliability in sentiment analysis.

This innovative system holds immense promise in empowering businesses and individuals alike with actionable insights derived from sentiment trends within comments. By offering a nuanced understanding of sentiment dynamics, our system enables informed decision-making, facilitating strategic actions based on sentiment analysis outcomes.

One of the key strengths of our system lies in its ability to visualize data intuitively through graphs. By presenting sentiment distributions in a clear and comprehensive manner, our system enhances comprehension and insight generation. These visual representations not only aid in understanding the sentiment landscape but also streamline the process of extracting actionable insights from comment data.

Through our system, users can gain valuable insights into the sentiments expressed within social networking applications. Whether analyzing customer feedback, monitoring brand perception, or gauging public opinion, our system provides a

powerful tool for understanding sentiment dynamics and driving informed decision-making.

Overall, our groundbreaking approach to sentiment analysis offers a transformative solution that empowers users to extract meaningful insights from comment data, enabling proactive and strategic decision-making in various domains.

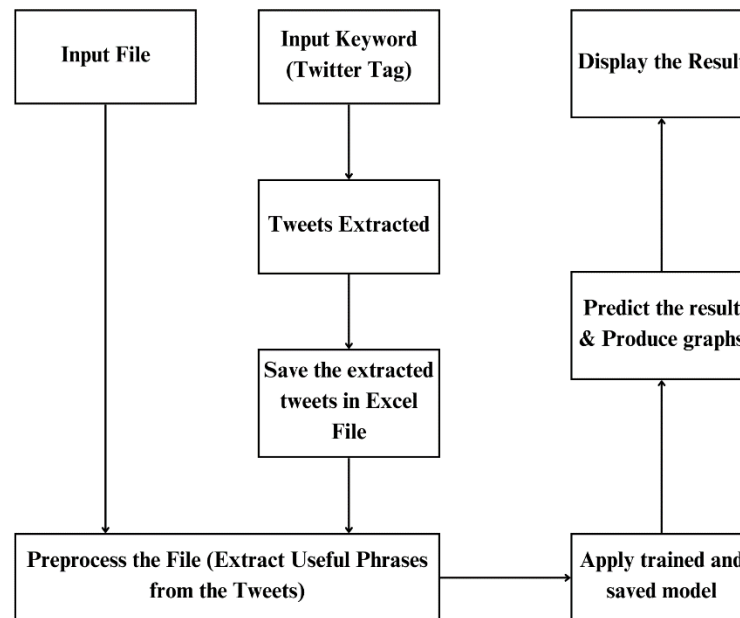


Fig.1. Sentiment Analysis Architecture

6. Machine Learning Approaches

Machine learning plays a pivotal role in sentiment analysis, primarily employing classification techniques to categorize text into different classes. Broadly, there are two types of machine learning techniques:

6.1 Unsupervised Learning:

This method operates without predefined categories and does not provide correct targets. Instead, it relies on clustering algorithms to identify patterns and group similar data points together.

6.2 Supervised Learning:

In contrast, supervised learning relies on labeled datasets, where the model is provided with correct labels during training. These labeled datasets enable the model to learn meaningful patterns and make informed decisions during classification tasks.

The efficacy of both these learning methods hinges on the selection and extraction of relevant features crucial for sentiment detection.

Supervised classification is the dominant approach in sentiment analysis using machine learning techniques. It involves two main sets of data: the training set and the test set. Various machine learning algorithms have been developed to classify tweets

into different sentiment classes. Notable techniques include Naive Bayes (NB), Maximum Entropy (ME), and Support Vector Machines (SVM), all of which have demonstrated considerable success in sentiment analysis tasks.

The machine learning approach applicable to sentiment analysis mainly belongs to supervised classification. In a machine learning technique, two sets of data are needed:

1. Training Set
2. Test Set.

The machine learning process typically begins with the collection of a training dataset. Next, a classifier is trained on this dataset to learn patterns and relationships between features and sentiment labels. The selection of features is a critical decision in this process, as they determine how documents are represented and influence the performance of the classification model.

Commonly used features in sentiment classification include term presence and frequency, part-of-speech information, negations, and opinion words and phrases.

Supervised techniques such as SVM, Naive Bayes, and Maximum Entropy are frequently employed due to their effectiveness in sentiment analysis tasks. However, in scenarios where obtaining a labeled dataset is challenging, semi-supervised and unsupervised techniques are proposed to classify unlabeled data based on inherent patterns and structures within the dataset.

7. Machine Learning on Sentiment Analysis

7.1 Naive Bayes:

Naive Bayes operates as a probabilistic classifier, learning patterns from categorized documents. It evaluates documents by comparing their contents with a list of words to assign them to the appropriate class. The classification process involves calculating probabilities based on the occurrence of features within the document. Parameters such as $P(c)$ and $P(f|c)$ are estimated using maximum likelihood techniques, with smoothing applied to unseen features. Implementation of Naive Bayes for training and classification tasks can be facilitated using the Python NLTK library.

$$|C^* = \arg \max_c P_{NB}(c | d)$$

$$P_{NB}(c | d) = \frac{(P(c)) \sum_{i=1}^m p(f_i | c)^{n_i(d)}}{P(d)}$$

From the above equation, “f” is a “feature”, count of feature (fi) is denoted with $n_i(d)$ and is present in d which represents a tweet. Here, m denotes no. of features.

7.2 Maximum Entropy:

The Maximum Entropy Classifier does not make assumptions about feature relationships within the dataset. Instead, it aims to maximize system entropy by estimating the conditional distribution of class labels. Unlike Naive Bayes, Maximum Entropy handles overlapping features and resembles logistic regression in its approach to finding class distributions. The model representation includes the calculation of conditional probabilities based on feature weights represented by the weight vector λ_i .

$$P_{ME}(c | d, \lambda) = \frac{\exp[\sum_i \lambda_i f_i(c, d)]}{\sum_c \exp[\sum_i \lambda_i f_i(c, d)]}$$

Where c is the class, d is the tweet and λ_i is the weight vector. The weight vectors decide the importance of a feature in classification.

7.3 Support Vector Machine:

Support Vector Machine (SVM) analyzes data, delineates decision boundaries, and utilizes kernels for computation within the input space. It operates on two sets of vectors, classifying each data vector into a class. SVM aims to maximize the margin between classes, defining the classifier's decision boundary. This margin optimization reduces ambiguous classifications. SVM supports both classification and regression tasks, contributing to statistical learning theory and aiding in the precise recognition of relevant factors essential for successful understanding.

8. Sentiment Analysis Prediction

Sentiment analysis on tweets from Twitter involves several key steps to extract and analyze sentiment from the textual data. Initially, data is collected either through Twitter's API or other sources, often based on specified keywords, hashtags, or user handles. Preprocessing of the collected tweets is then carried out, involving tasks like removing URLs, mentions, and special characters, as well as tokenization and normalization of text. Feature extraction follows, where relevant features such as word frequency, parts of speech tags, and sentiment lexicons are selected. Next, a machine learning model is trained using labeled data, typically employing algorithms like Naive Bayes or Support Vector Machines. The trained model is evaluated to assess its performance in predicting sentiment accurately. Once validated, the model is deployed to analyze sentiment in new tweets, assigning labels such as positive, negative, or neutral. Post-processing techniques may be applied for further refinement, and results can be visualized using charts or graphs to facilitate interpretation. This systematic approach enables analysts to gain insights into public opinion, brand perception, and other valuable insights from Twitter data.

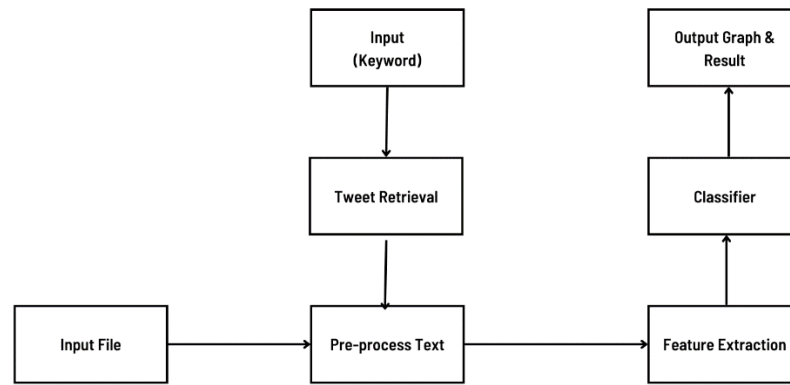


Fig.2. Block Diagram for the Proposed System

9. Result

Our project website serves as an intuitive platform for users to effortlessly input their data and receive accurate predictions regarding house prices. With a user-friendly interface, the site seamlessly guides users through the process of uploading files, enabling them to obtain results presented through pie charts and bar graphs. Additionally, the website provides insightful comments on the sentiment associated with each prediction, enhancing the user experience and facilitating informed decision-making

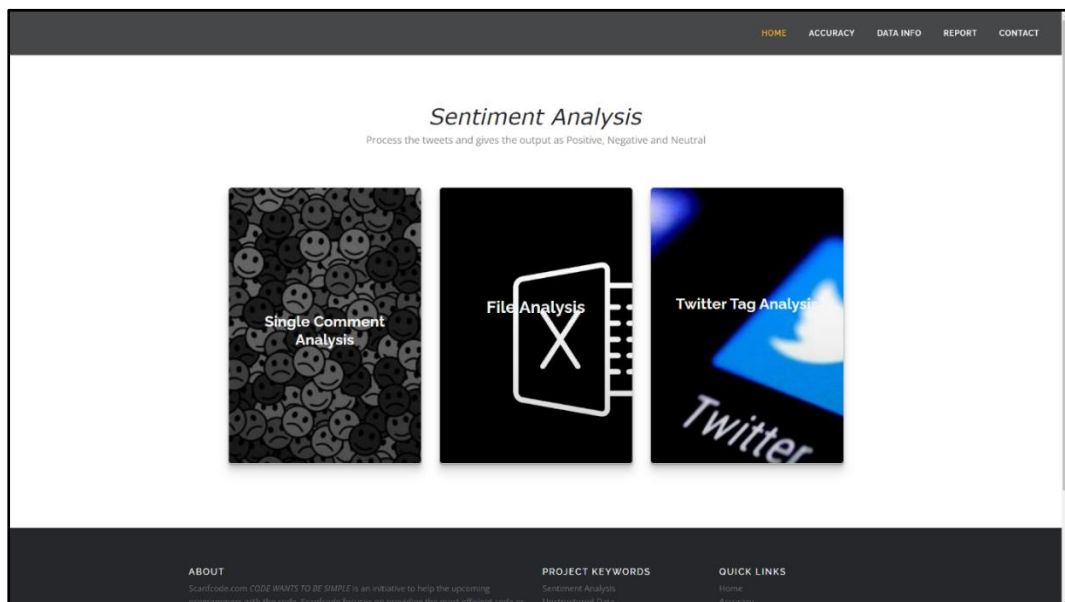


Fig.3. Proposed System Home Page

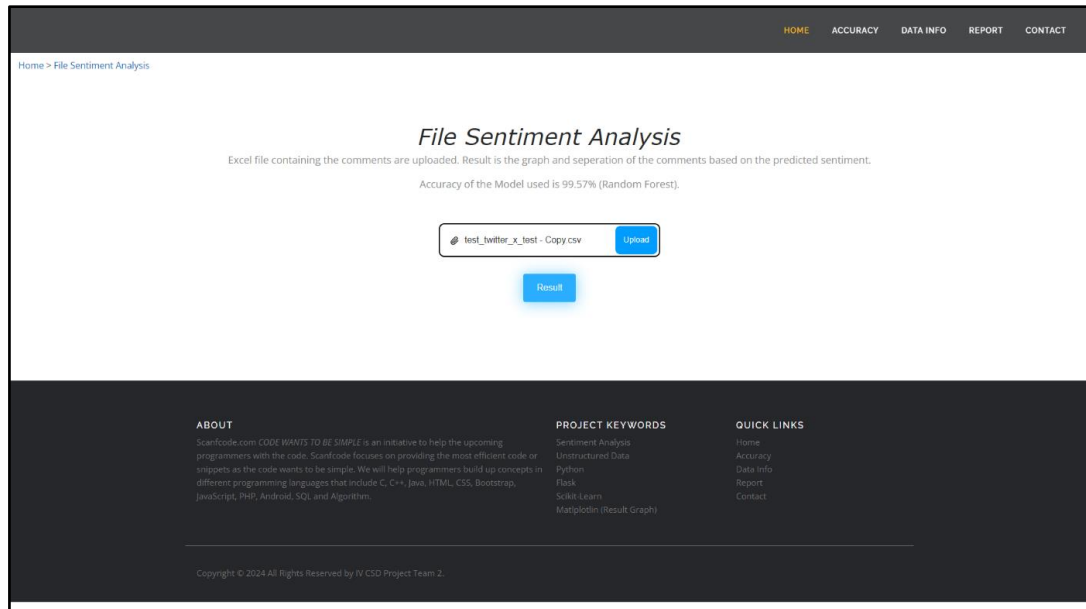


Fig.4. File Upload Page

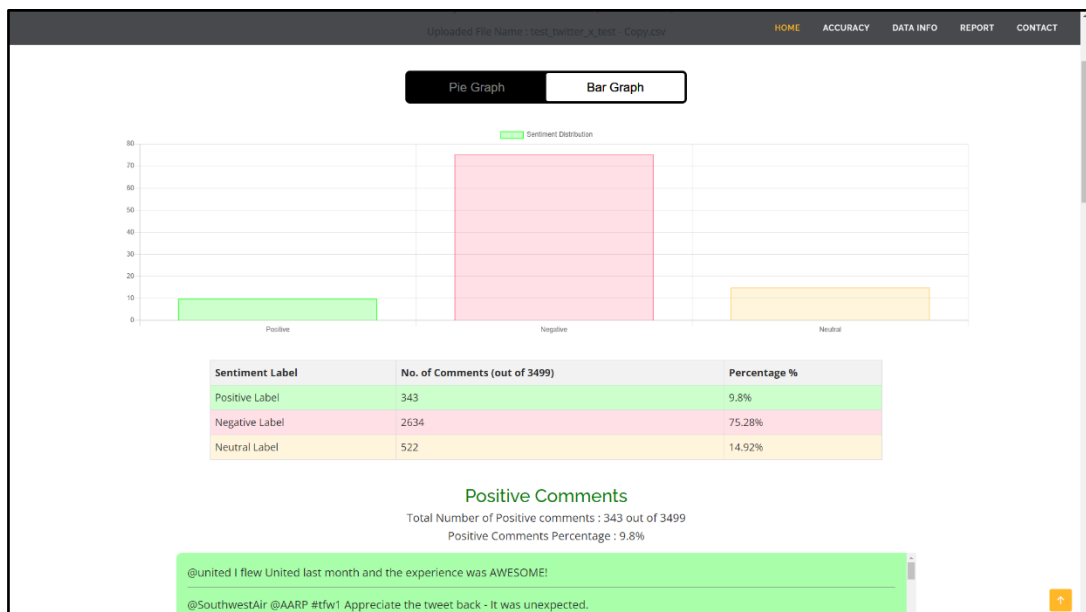


Fig.5. Result Page

10. Future Scope

The future of sentiment analysis is likely to involve advancements in several key areas:

- **Deep Learning and AI:** Further improvements in deep learning models and artificial intelligence techniques will lead to more accurate sentiment analysis systems. This may involve more sophisticated architectures such as transformers, which have shown significant promise in natural language processing tasks.
- **Multimodal Sentiment Analysis:** Integrating text with other modalities like images, audio, and video will enable more comprehensive sentiment analysis.

This will be particularly useful for applications like social media monitoring, where content is often multimodal.

- **Contextual Understanding:** Enhancing sentiment analysis models to better understand context will be crucial. This includes understanding sarcasm, irony, and other forms of nuanced language, as well as considering the broader context in which a statement is made.
- **Domain-Specific Analysis:** Customizing sentiment analysis models for specific domains, such as finance, healthcare, or customer service, will lead to more accurate results tailored to the unique language and expressions used in those domains.
- **Cross-Lingual Sentiment Analysis:** Developing models that can analyze sentiment in multiple languages will be important for global applications. This will involve overcoming challenges such as language differences, cultural nuances, and the scarcity of labeled data for certain languages.
- **Ethical Considerations:** As sentiment analysis becomes more widespread, ethical considerations surrounding privacy, bias, and fairness will become increasingly important. Ensuring that sentiment analysis systems are used responsibly and do not perpetuate or amplify biases will be a significant focus.
- **Real-Time Analysis:** Improving the speed and scalability of sentiment analysis systems will enable real-time monitoring of sentiment on social media, news articles, and other sources of textual data. This will be valuable for businesses and organizations to stay informed about public opinion as it evolves.

Conclusion

In conclusion, sentiment analysis on Twitter data presents a crucial avenue for understanding public opinion and sentiment trends in real-time. Our project offers a robust framework leveraging machine learning algorithms to categorize tweets accurately. By emphasizing feature extraction, data preprocessing, and model evaluation, we streamline the sentiment analysis process for actionable insights. Looking forward, advancements in deep learning, multimodal analysis, and ethical considerations will drive further innovation in the field. Sentiment analysis continues to play a pivotal role in informing decision-making processes and understanding human behavior, contributing to a more connected and informed society.

References

- [1] Sentiment Analysis of Twitter Data: A Survey of Techniques: Vishal A. Kharde & S.S. Sonawane, 11, April 2016.
- [2] A.Pak and P. Paroubek. „Twitter as a Corpus for Sentiment Analysis and Opinion Mining". In Proceedings of the Seventh Conference on International Language Resources and Evaluation, 2010, pp.1320-1326.
- [3] Agarwal, B. Xie, I. Vovsha, O. Rambow, R. Passonneau, "Sentiment Analysis of Twitter Data", In Proceedings of the ACL 2011 Workshop on Languages in Social Media, 2011, pp. 30-38.

- [4] Neethu M,S and Rajashree R,” Sentiment Analysis in Twitter using Machine Learning Techniques” 4th ICCCNT 2013,at Tiruchengode, India. IEEE – 31661.
- [5] R. Xia, C. Zong, and S. Li, “Ensemble of feature sets and classification algorithms for sentiment classification,” *Information Sciences: an International Journal*, vol. 181, no. 6, pp. 1138–1152, 2011.
- [6] Go, R. Bhayani, L.Huang. “Twitter Sentiment Classification Using Distant Supervision”. Stanford University, Technical Paper,2009
- [7] Bifet and E. Frank, "Sentiment Knowledge Discovery in Twitter Streaming Data", In Proceedings of the 13th International Conference on Discovery Science, Berlin, Germany: Springer,2010, pp. 1-15.
- [8] J. Kamps, M. Marx, R. J. Mokken, and M. De Rijke, “Using wordnet to measure semantic orientations of adjectives,” 2004.

PAPER PUBLICATION CERTIFICATE

Indian Institution of Industrial Engineering



BOMBAY
CERTIFICATE OF PUBLICATION

This is to certify that the article entitled

SENTIMENT ANALYSIS ON TWEETS OF TWITTER(X)

Authored By

S. BHANU PRASAD

Department of computer science and Engineering
DADI INSTITUTE OF ENGINEERING AND TECHNOLOGY (AUTONOMOUS)

Published in

Industrial Engineering Journal : ISSN 0970-2555

Volume : 53, Issue :04, No: 1 APRIL : 2024

UGC Care Approved, Group I, Peer Reviewed Journal

with IF=6.82

Editor in Chief



INTERNSHIP CERTIFICATE



CONGRATULATIONS



SARAGADAM BHANU PRASAD

Is hereby awarded this

CERTIFICATE OF INTERNSHIP

For Successfully Completing **Python Full Stack Internship**, also
enriching their skills and expertise in web application development.
Is presented on **07 Apr 2024**

Akash Pandey
CEO, Co-Founder



nasscom



#startuptionia

