X- FEED SENTIMENT ANALYSIS

A MINI PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

GOVIND KALAWATE [RA2111033010048]
GUTTUR SAIKIRAN [RA2111033010047]

Under the guidance of
Dr. Kanimozhi N

Assistant Professor, Department of Computational Intelligence

in partial fulfillment for the award of the

degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING
with specialisation in Software Engineering

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that Mini project report titled **"X-FEED SENTIMENT ANALYSIS"** is the bonafide work of **GOVIND KALAWATE (RA2111033010048) & GUTTUR SAIKIRAN (RA2111033010047)** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. Kanimozhi N

Assistant Professor

Department of Computational Intelligence.

SIGNATURE

Dr. R.Annie Uthra

Head Of the Department

Department of Computational Intelligence.

ABSTRACT

Sentiment analysis has become increasingly important in today's digital age, particularly with the widespread use of social media platforms like X. This project focuses on developing a robust sentiment analysis system tailored specifically for Twitter feeds, with the goal of accurately categorizing tweets into positive, negative, or neutral sentiments. The system architecture is designed to encompass various stages, including data collection, preprocessing, feature extraction, model building, evaluation, and data visualization.

In the data collection phase, tweets are gathered from Twitter's API or other relevant sources. These tweets undergo preprocessing steps such as tokenization, text cleaning, and normalization to ensure consistency and quality. Feature extraction techniques are then applied to convert the textual data into numerical representations that machine learning models can process. Several techniques are explored, including Bag-of-Words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF), and Word Embeddings.

The project employs machine learning algorithms like Logistic Regression, Support Vector Machines (SVM), and Neural Networks for sentiment classification. Each model's performance is evaluated using various metrics such as accuracy, precision, recall, and F1-score to determine its effectiveness in sentiment classification.

In addition to model evaluation, the project integrates interactive data visualization techniques to present sentiment analysis results in a visually intuitive manner. This enhances the interpretability of the analysis outcomes and allows for deeper insights into sentiment trends and patterns.

Upon analysis, the TF-IDF feature extraction technique emerges as the most optimized approach based on achieved accuracy scores. However, future enhancements to the system could involve integrating advanced deep learning models, further enhancing data visualization capabilities, and extending sentiment analysis to multimedia content beyond textual data sources. These enhancements would contribute to a more comprehensive and versatile sentiment analysis system capable of addressing diverse data sources and analytical needs.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

NLP - Natural Language Processing

BoW - Bag-of-Words

TF-IDF - Term Frequency-Inverse Document Frequency

SVM - Support Vector Machines

API - Application Programming Interface

LR - Logistic Regression

F1-score - F1 Measure or F1-Score

SVD - Singular Value Decomposition

PCA - Principal Component Analysis

Note: Throughout this document, "X" is used interchangeably with “Twitter"

CHAPTER 1

INTRODUCTION

X (former Twitter) has become one of the most popular social media platforms for people to express their opinions, share news, and engage in discussions on various topics. With the vast amount of data generated on X every day, sentiment analysis plays a crucial role in understanding public opinion and sentiment towards different subjects, products, events, or brands.

In this project, we aim to perform sentiment analysis on Twitter feeds using Natural Language Processing (NLP) techniques. Sentiment analysis involves determining the sentiment or emotional tone expressed in a piece of text, which can be positive, negative, or neutral. By analyzing tweets, we can gain insights into public sentiment towards specific topics, products, or events.

Objective:
The main objective of this project is to build a sentiment analysis model that can accurately classify the sentiment of X feeds into positive, negative categories. To achieve this, we will employ various NLP techniques and machine learning algorithms to preprocess the text data, extract meaningful features, and train predictive models.

Scope:
This project focuses specifically on sentiment analysis of Twitter feeds using NLP techniques and machine learning models. While Twitter provides a vast amount of data, we will limit our analysis to text-based tweets and exclude other media types such as images and videos.

Overall, the goal of this project is to demonstrate the effectiveness of NLP techniques in analysing public sentiment on social media platforms like Twitter and provide insights that can be valuable for various applications, including market research, brand management, and public opinion analysis

CHAPTER 2

LITERATURE SURVEY

1. Sentiment Analysis in Social Media:

- Researchers have extensively studied sentiment analysis in social media platforms like Twitter due to the abundance of user-generated content. Various studies have explored different approaches, including lexicon-based methods, machine learning models, and deep learning techniques, to analyze sentiment in tweets. (Reference: Pang, B., & Lee, L. (2008). Opinion Mining and Sentiment Analysis)

2. NLP Techniques for Text Processing:

- Natural Language Processing (NLP) techniques such as tokenization, stemming, and lemmatization are fundamental for preprocessing text data before sentiment analysis. These techniques help to standardize and normalize the text, making it suitable for feature extraction and modeling. (Reference: Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python)

3. Feature Extraction Methods:

- Bag-of-Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF) are popular feature extraction methods used in sentiment analysis. BoW represents text data as a matrix of word frequencies, while TF-IDF computes the importance of words in a document relative to a corpus of documents. Researchers have compared the effectiveness of these methods in sentiment classification tasks. (Reference: Jurafsky, D., & Martin, J. H. (2009). Speech and Language Processing)

4. Machine Learning Models for Sentiment Analysis:

- Logistic Regression, Support Vector Machines (SVM), Random Forest, and Neural Networks are commonly used machine learning models for sentiment analysis. These models are trained on labeled datasets to predict the sentiment of text data accurately. Researchers have evaluated the performance of these models on various sentiment analysis benchmarks and datasets. (Reference: Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval)

5. Deep Learning Approaches:

- Deep learning techniques, including Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have shown promising results in sentiment analysis tasks. These models can capture complex patterns and dependencies in text data, leading to improved sentiment classification performance. Researchers have explored the use of pre-trained word embeddings and attention mechanisms in deep learning models for sentiment analysis. (Reference: Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification)

6. Evaluation Metrics:

- Evaluation metrics such as accuracy, precision, recall, and F1-score are commonly used to assess the performance of sentiment analysis models. Researchers have discussed the importance of choosing appropriate evaluation metrics based on the specific requirements and characteristics of sentiment analysis tasks. (Reference: Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks)

7. Challenges and Future Directions:

- Despite significant advancements in sentiment analysis research, several challenges remain, including handling sarcasm, irony, and context-dependent sentiments in text data. Future research directions may focus on developing more robust sentiment analysis models that can accurately capture nuanced sentiments and contextual information from social media texts. (Reference: Liu, B. (2015). Sentiment Analysis: Mining Opinions, Sentiments, and Emotions)

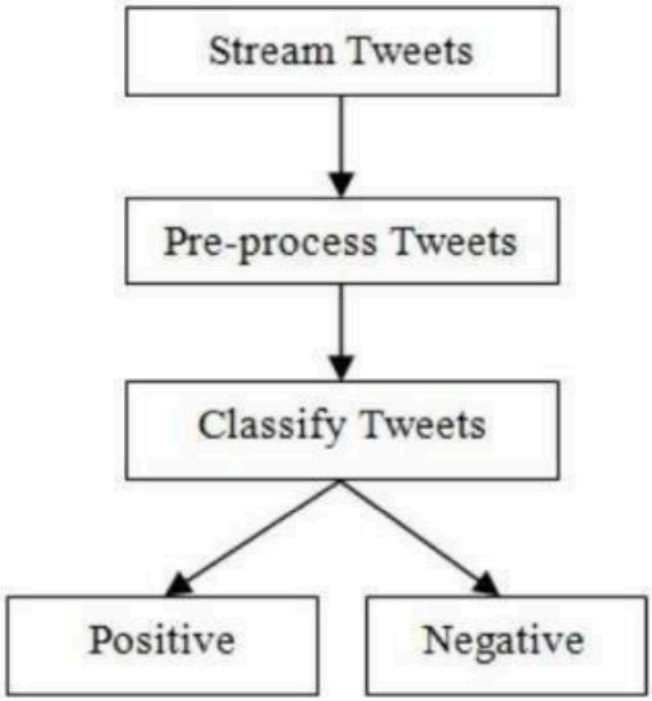
By reviewing the existing literature, we gain valuable insights into the state-of-the-art techniques and methodologies employed in sentiment analysis and NLP research. This knowledge will guide our approach and methodology for conducting sentiment analysis on X feeds in this project.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

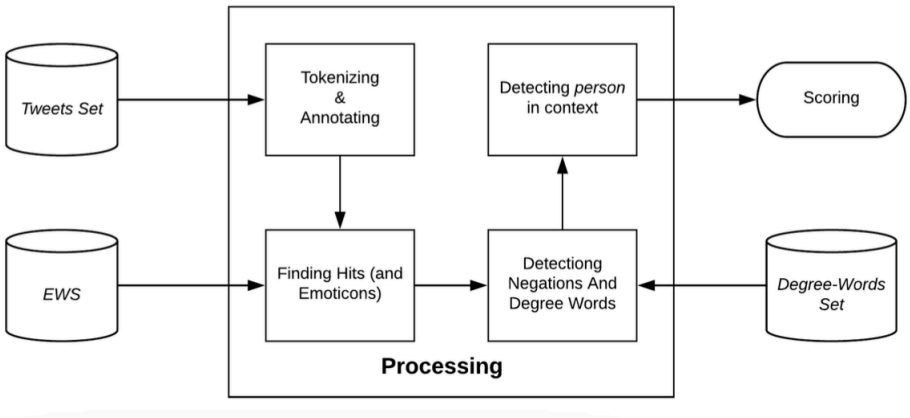
3.1 Architecture Diagram:

Our system architecture for Twitter feed sentiment analysis comprises several interconnected components designed to process, analyze, and classify tweets based on their sentiment. The architecture diagram illustrates the flow of data and operations within the system, highlighting the key modules and their interactions.

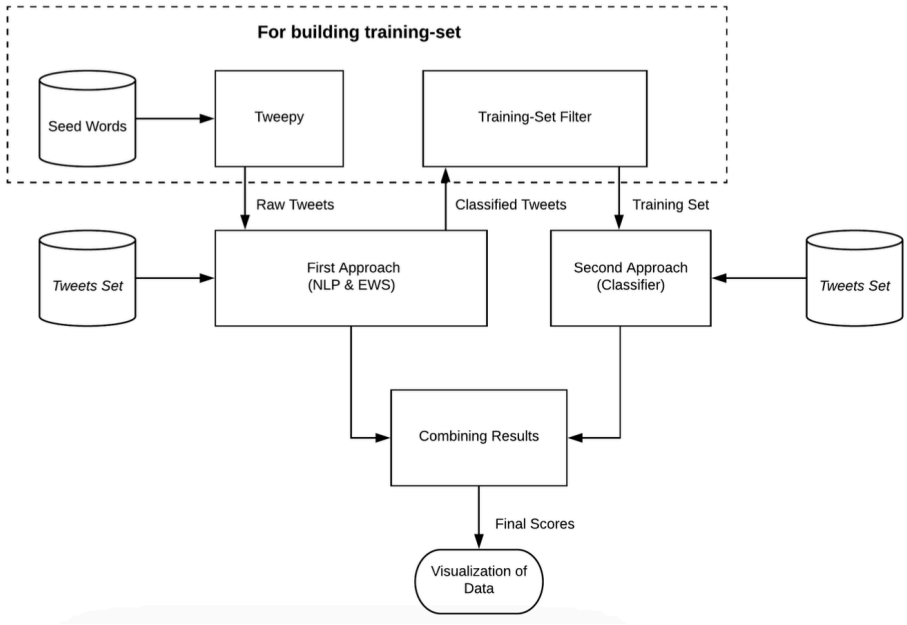


Architecture Diagram Overview 3.1.1

Step-1: Collecting the dataset.
Step-2: Then we pre-process the tweets, so that they can be fit feature extraction.
Step-3: After pre-processing we pass this data in our trained classifier, which then classify them into positive or negative class based on trained result.



Architecture Diagram - First Approach 3.1.2



Architecture Diagram - Combined 3.1.3

3.2 Modules and Components:

Our system consists of the following modules and components:

1. Data Collection Module:
- Responsible for collecting tweets from the Twitter API based on specified search queries or user timelines. This module retrieves raw tweet data, including text, user information, and metadata.

2. Preprocessing Module:
- Performs text preprocessing tasks such as tokenization, removing stopwords, stemming, and lemmatization. This module prepares the raw tweet text for feature extraction and analysis.

3. Feature Extraction Module:
- Converts the preprocessed tweet text into numerical representations (features) that machine learning models can understand. This module includes techniques such as Bag-of-Words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF), and word embeddings (e.g., Word2Vec, GloVe).

4. Sentiment Analysis Module:
- Utilizes machine learning models or deep learning architectures to analyze the sentiment of tweets. This module classifies tweets into sentiment categories such as positive, negative, or neutral based on the extracted features.

5. Model Evaluation Module:
- Evaluates the performance of sentiment analysis models using various evaluation metrics such as accuracy, precision, recall, and F1-score. This module assesses the effectiveness of the models in predicting tweet sentiment and provides insights for model refinement.

6. External APIs and Libraries:
- Utilizes external APIs and libraries for tasks such as Twitter data retrieval, natural language processing (NLP), machine learning model training, and visualization. This module leverages existing tools and resources to enhance the functionality and efficiency of the system.

By breaking down the system into modular components, we ensure flexibility, scalability, and maintainability, allowing for easy integration of new features or enhancements in the future. Each module plays a crucial role in the overall functionality and performance of the sentiment analysis system.

CHAPTER 4

METHODOLOGY

Our methodology for X feed sentiment analysis involves the following steps:

1. **Data Collection:**
   We gather tweet data from the Twitter API using specified search queries or user timelines. This step involves retrieving raw tweet text along with relevant metadata such as user information, timestamps, and retweet counts.

2. **Preprocessing:**
   The collected tweet text undergoes preprocessing to clean and prepare it for analysis. This includes tasks such as tokenization, removing stopwords, punctuation, and special characters, as well as stemming or lemmatization to normalize the text.

3. **Feature Extraction:**
   We convert the preprocessed tweet text into numerical representations (features) that machine learning models can understand. This step involves various techniques such as Bag-of-Words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF), and word embeddings (e.g., Word2Vec, GloVe).

4. **Model Building:**
   We train machine learning models or deep learning architectures to perform sentiment analysis on the tweet data. This step includes selecting appropriate algorithms such as logistic regression, support vector machines (SVM), or neural networks, as well as fine-tuning hyperparameters for optimal performance.

5. **Model Evaluation:**
   We evaluate the performance of the sentiment analysis models using metrics such as accuracy, precision, recall, and F1-score. This step involves splitting the data into training and testing sets, training the models on the training data, and assessing their performance on unseen test data.

6. **Cross-Validation:**
   We perform cross-validation to assess the generalization ability of the models and mitigate overfitting. This step involves partitioning the data into multiple

subsets, training the models on different subsets, and evaluating their performance across multiple iterations.

7. **Hyperparameter Tuning:**
   We optimize the hyperparameters of the models using techniques such as grid search or randomized search. This step involves systematically searching through a predefined hyperparameter space to identify the best combination that maximizes model performance.

8. **Deployment and Integration:**
   We deploy and integrate the sentiment analysis system into production environments or web applications. This step involves ensuring the scalability, reliability, and performance of the system in real-world settings, as well as providing a user-friendly interface for interaction.

By following these methodological steps, we can effectively analyze the sentiment of Twitter feeds and provide valuable insights for various applications such as brand monitoring, market analysis, and public opinion tracking.

CHAPTER 5

CODING AND TESTING

1. **Data Collection Script:**
   We write Python scripts utilizing the Twitter API (Tweepy library) to collect tweet data based on specified search queries or user timelines. These scripts handle authentication, data retrieval, and storage in a suitable format (e.g., CSV, JSON).

```
import tweepy
import pandas as pd

consumer_key = "HoriRM4tOpiyg2eQPmzxy43SW"
"pX7vNJjrHVxj077oKokrK4NxEmdMZ6pZRWoSkXhUOgMYfrXXSi"
access_token = "1780929605957804033-ZggmQbbO47tkdnovxM97SuTDoZfac6"
access_token_secrt= "2FRzb6VkQ0LmbH1J8SBAuGdKsbbdrMzLr6E6E1MVetgFe"
# Pass in our Twitter API authentication key
auth = tweepy.OAuth1UserHandler(
    consumer_key, consumer_secret,
    access_token, access_token_secret
)

# Instantiate the tweepy API
api = tweepy.API(auth, wait_on_rate_limit=True)

search_query = "'ref''world cup'-filter:retweets AND -filter:replies AND -filter:links"
no_of_tweets = 100

try:
    # The number of tweets we want to retrieve from the search
        tweets = api.search_tweets(q=search_query, lang="en", count=no_of_tweets,
tweet_mode='extended')

    # Pulling some attributes from the tweet
      attributes_container = [[tweet.user.name, tweet.created_at, tweet.favorite_count,
tweet.source, tweet.full_text] for tweet in tweets]

    # Creation of column list to rename the columns in the dataframe
       columns = ["User", "Date Created", "Number of Likes", "Source of Tweet",
"Tweet"]

    # Creation of Dataframe
    tweets_df = pd.DataFrame(attributes_container, columns=columns)
except BaseException as e:
```

```
print('Status Failed On,', str(e))
```

2. **Preprocessing Pipeline:**
   We develop a preprocessing pipeline to clean and prepare the raw tweet text for analysis. This pipeline typically includes tokenization, removal of stopwords, punctuation, and special characters, as well as stemming or lemmatization. We use libraries such as NLTK or spaCy for text processing tasks.

```
import pandas as pd
pd.set_option("display.max_colwidth", 200)
data = pd.read_csv('tweets.csv')
data.head()
data.info()
# check for missing values
data.isnull().sum()
# drop the id column
# data.drop(['id'], axis=1, inplace=True)
data.head()
# Check for the class balance
data[‘label'].value_counts(normalize=True)
# plot the label counts
data[‘label'].value_counts().plot(kind='bar')
data[‘tweet'][24]
import re
# Substitute 's with " is"
re.sub(r"'s\b", " is", data[‘tweet'][24])
# Removing the user mentions
data['tweet'][11]
# We'll retain on the alphabets & digits
re.sub("@[A-Za-z0-9]+", "", data[‘tweet'][11])
# remive the hashtags
data[‘tweet'][0]
re.sub("#", "", data[‘tweet'][0])
# Removing the hyperlinks
re.sub(r"http\S+", "", data[‘tweet'][0])
# Retain on the alphabets (get rid of punctuations, special char, digits)
data['tweet'][25]
re.sub(r"[^a-zA-Z]", " ", data[‘tweet'][0])
# Stop words Removal
import nltk
from nltk.corpus import stopwords
nltk_stopwords = set(stopwords.words('english'))
print(nltk_stopwords)
len(nltk_stopwords)
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
```

```
sklearn_stopwords = set(ENGLISH_STOP_WORDS)
print(sklearn_stopwords)
len(sklearn_stopwords)
# Find the common stopwords from NLTK & sklearn
print(nltk_stopwords.intersection(sklearn_stopwords))
len(nltk_stopwords.intersection(sklearn_stopwords))
# Combining the stopwords from sklearn & NLTK
combined_stopwords = nltk_stopwords.union(sklearn_stopwords)
len(combined_stopwords)
# Text Normalization: Stemming or Lemmatization (prefer)
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
data[‘tweet'][63].split()
import contractions
data[‘tweet'][24]
contractions.fix(data[‘tweet'][24])
import re
Def tweet_cleaner_without_stopwords(text):
    new_text = re.sub(r"'s\b", " is", text)
    new_text = re.sub("#", "", new_text)
    new_text = re.sub("@[A-Za-z0-9]+", "", new_text)
    new_text = re.sub(r"http\S+", "", new_text)
    new_text = contractions.fix(new_text)
    new_text = re.sub(r"[^a-zA-Z]", " ", new_text)
    new_text = new_text.lower().strip()

    cleaned_text = ''
    for token in new_text.split():
        cleaned_text = cleaned_text + lemmatizer.lemmatize(token) + ' '

    return cleaned_text
cleaned_tweets = []  # list of cleaned tweets
for twt in data['tweet']:
    cleaned_tweets.append(tweet_cleaner_without_stopwords(twt))
cleaned_tweets[24]
data[‘tweet'][1500]
data['tweet'][1500].split()
cleaned_tweets[1500]
data['cleaned_tweets_w/o_SW'] = cleaned_tweets
data.head()
```

3. Feature Extraction Module:

We implement feature extraction techniques such as Bag-of-Words (BoW), TF-IDF,N-grams, or word embeddings (Word2Vec, GloVe) to convert the preprocessed text into numerical representations. We use libraries like scikit-learn or Gensim for feature extraction.

```
data.shape

from sklearn.feature_extraction.text import CountVectorizer

CV = CountVectorizer()
CV_features = CV.fit_transform(data[‘cleaned_tweets_w/o_SW'])
CV_features.shape
CV_features[0]
type(CV_features[0])  # each row is a compressed spare row
CV_features[0].todense()  # decompressing the CSR data

import pandas as pd
df = pd.DataFrame(CV_features.todense() )
df
# Document-Term-Matrix X = [N x p ]
# N = no. of documentns
# p = no. of unique words in the vocab!!# this is your D
df.size/1e6  # these many elements/numbers are presnt in the df
import numpy as np

# Now you can use np.count_nonzero and other NumPy functions

np.count_nonzero(df)
100 * np.count_nonzero(df) / df.size  # percent of the non-zero elements in the df
print(CV.get_feature_names_out()[:50])  # these are the vocabulary words
len(CV.get_feature_names_out())
CV_features[10].todense()  # these are numbers corresponding to the 11th tweet
doc1 = 'I love cooking'
doc2 = 'Today I cooked pasta today'
from sklearn.feature_extraction.text import TfidfVectorizer
X = data['cleaned_tweets_w/o_SW']
y = data['label']

# we want to include only those words in the vocab which have min df of 5,
# means select only those words which occur ATLEAST in 5 documents!!
```

```
# AND SELECT the TOP 1000 FEATURES ONLY to build the model
TFIDF = TfidfVectorizer(stop_words=final_stopwords, min_df=5, max_features=1000)

LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)
CV_pipe = Pipeline([('TFIDF', TFIDF) , ('LR', LR1)] )
results = cross_validate(CV_pipe, X, y, cv=kfold, scoring='accuracy', return_train_score=True)

# print(results['train_score'])
print(np.round((results['train_score'].mean())*100, 2), np.round((results['train_score'].std())*100, 2))

# print(results['test_score'])
print(np.round((results['test_score'].mean())*100, 2), np.round((results['test_score'].std())*100, 2))

TFIDF.fit_transform(X)
len(TFIDF.vocabulary_)  # no. of features AFTER applying the stopwords
```

4. **Model Building and Training:**
   We build and train sentiment analysis models using machine learning algorithms such as logistic regression, support vector machines (SVM), or deep learning architectures (e.g., recurrent neural networks, convolutional neural networks). We utilize libraries like scikit-learn, TensorFlow, or PyTorch for model development.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(CV_features, data['label'], test_size=0.25, stratify=data['label'], random_state=42)
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(solver='liblinear')
LR.fit(X_train, y_train)
print(LR.score(X_train, y_train))  # train score)
print(LR.score(X_test, y_test))   # test score)
## L1-REGULARIZATION leso
LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)
LR1.fit(X_train, y_train)
```

```
print(LR1.score(X_train, y_train))  # train score)
print(LR1.score(X_test, y_test))   # test score)
##  Cross-Validate the l2-  Logistic Regression Model ?????
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import StratifiedKFold
X = CV_features
y = data['label']
## crOSS VALIDATE THE LR1 model

LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
results = cross_val_score(LR1, X, y, cv=kfold, scoring='accuracy')
print(results)
print(np.round((results.mean())*100, 2), np.round((results.std())*100, 2))

# our accuracy = 88.23 +/- 0.93 %
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
results = cross_validate(LR1, X, y, cv=kfold, scoring='accuracy', return_train_score=True)
results
print(results['train_score'])
print(np.round((results['train_score'].mean())*100, 2), np.round((results['train_score'].std())*100, 2))
print(results['test_score'])
print(np.round((results['test_score'].mean())*100, 2), np.round((results['test_score'].std())*100, 2))
## Hyper parameter tuning of the LR1 model
from sklearn.model_selection import GridSearchCV

LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1')

C_values = np.arange(0.00001, 1, 0.05) # 20 values

grid = GridSearchCV(estimator=LR1, param_grid={'C': C_values}, cv=kfold, scoring='accuracy', \
                    return_train_score=True, verbose=2, n_jobs=-1)
grid_results = grid.fit(X,y)
grid_results
grid_results.best_params_, grid_results.best_score_, grid_results.best_index_
```

```
grid_results.cv_results_.keys()
grid_results.cv_results_[‘mean_test_score'][grid_results.best_index_]*100
grid_results.cv_results_['mean_train_score'][grid_results.best_index_]*100

# means our "best-fitted" model from GridsearchCV is (could be) still OVERFITTED!!!!!!!!!
grid_results.cv_results_[‘std_test_score'][grid_results.best_index_]*100
grid_results.cv_results_[‘mean_test_score']
grid_results.cv_results_[‘mean_train_score']
.plot(grid_results.cv_results_['mean_train_score'] - grid_results.cv_results_[‘mean_test_score'])
grid_results.param_grid
grid_results.param_grid[‘C'][3]
grid_results.cv_results_['mean_train_score'] - grid_results.cv_results_[‘mean_test_score']
# Create a pipeline & cross-validate
from sklearn.pipeline import make_pipeline, Pipeline

X = data['cleaned_tweets_w/o_SW']
y = data['label']

CV = CountVectorizer()
LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)

CV_pipe = Pipeline([('CV', CV) , ('LR', LR1)] )

results = cross_val_score(CV_pipe, X, y, cv=kfold, scoring='accuracy')
print(np.round((results.mean())*100, 2), np.round((results.std())*100, 2))
CV_pipe.named_steps
CV_pipe.fit(X,y)
len(CV_pipe['CV'].vocabulary_)  # CV with stopwords applied already & word_len > 2
# Create a pipeline & cross-validate
from sklearn.pipeline import make_pipeline, Pipeline

X = data['cleaned_tweets_w/o_SW']
y = data['label']

CV = CountVectorizer(stop_words=final_stopwords)
```

```
LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)

CV_pipe = Pipeline([('CV', CV) , ('LR', LR1)] )

results = cross_val_score(CV_pipe, X, y, cv=kfold, scoring='accuracy')
print(np.round((results.mean())*100, 2), np.round((results.std())*100, 2))

CV_pipe.fit(X,y)
len(CV_pipe['CV'].vocabulary_)
```

BEST MODEL

```
# Create a pipeline & cross-validate
from sklearn.pipeline import make_pipeline, Pipeline

X = data['cleaned_tweets_w/o_SW']
y = data['label']

CV = CountVectorizer(stop_words=final_stopwords, min_df=10, max_features=None)
LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)

CV_pipe = Pipeline([('CV', CV) , ('LR', LR1)] )

results = cross_val_score(CV_pipe, X, y, cv=kfold, scoring='accuracy')
print(np.round((results.mean())*100, 2), np.round((results.std())*100, 2))

CV_pipe.fit(X,y)
len(CV_pipe['CV'].vocabulary_)
# Create a pipeline & cross-validate
from sklearn.pipeline import make_pipeline, Pipeline

X = data['cleaned_tweets_w/o_SW']
y = data['label']

CV = CountVectorizer(stop_words=final_stopwords, min_df=10, max_features=300)
LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)

CV_pipe = Pipeline([('CV', CV) , ('LR', LR1)] )

results = cross_val_score(CV_pipe, X, y, cv=kfold, scoring='accuracy')
```

```
print(np.round((results.mean())*100, 2), np.round((results.std())*100, 2))

CV_pipe.fit(X,y)
len(CV_pipe['CV'].vocabulary_)
```

TF-IDF Vector BoW Model + Logistic Regression
tf_score = how many times a word appears in a given doc(voc)/ Total no of words in that doc
IDF_SCORE = Log((1+total no of documents)/(1+no. of documents containg that "term"))// panalises words which are equally frequent in almost all the documents.

```
from sklearn.feature_extraction.text import TfidfVectorizer
X = data['cleaned_tweets_w/o_SW']
y = data['label']

# we want to include only those words in the vocab which have min df of 5,
# means select only those words which occur ATLEAST in 5 documents!!
# AND SELECT the TOP 1000 FEATURES ONLY to build the model
TFIDF = TfidfVectorizer(stop_words=final_stopwords, min_df=5, max_features=1000)

LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)
CV_pipe = Pipeline([('TFIDF', TFIDF) , ('LR', LR1)] )
results = cross_validate(CV_pipe, X, y, cv=kfold, scoring='accuracy', return_train_score=True)

# print(results['train_score'])
print(np.round((results['train_score'].mean())*100, 2), np.round((results['train_score'].std())*100, 2))

# print(results['test_score'])
print(np.round((results['test_score'].mean())*100, 2), np.round((results['test_score'].std())*100, 2))

TFIDF.fit_transform(X)
len(TFIDF.vocabulary_)  # no. of features AFTER applying the stopwords
```

5. Model Evaluation and Validation:

We evaluate the performance of the trained models using appropriate evaluation metrics such as accuracy, precision, recall, and F1-score. We

conduct testing on held-out validation datasets to assess the generalization ability of the models and ensure they perform well on unseen data.

F1 scores of all the Models implemented :

- Bag-of-Words (BoW) with Logistic Regression (LR): 88.28% ± 0.93%
- TF-IDF with Logistic Regression (LR): 85.24% ± 0.84%
- Word Embeddings with Logistic Regression (LR): 85.23% ± 0.6

6. Cross-Validation and Hyperparameter Tuning:

We perform cross-validation and hyperparameter tuning to optimize the performance of the models. We use techniques such as grid search or randomized search to systematically search through the hyperparameter space and identify the best parameter settings.

7. Data Visualization Module

The Data Visualization module focuses on generating graphical representations of sentiment analysis results for better interpretation and understanding. It includes the following components:

- **Visualization Techniques**: Utilizes various visualization techniques such as bar charts, pie charts, line plots, and heatmaps to represent sentiment distributions, trends, and correlations.
- **Interactive Dashboards**: Develops interactive dashboards using visualization libraries like Matplotlib, Seaborn, or Plotly to allow users to explore sentiment analysis results dynamically.
- **Sentiment Heatmaps**: Generates sentiment heatmaps to visualize sentiment distributions across different topics, time periods, or geographic regions.
- **Word Clouds**: Creates word clouds to visually represent the most frequently occurring words in positive, negative, or neutral tweets, providing insights into prevalent themes and sentiments.
- **Geospatial Visualization**: Integrates geospatial visualization techniques to map sentiment scores geographically, highlighting sentiment variations across different locations.

```
import nltk
import matplotlib.pyplot as plt
# a code to collect all the words from all the tweets into a single list
# Initialize an empty list to collect all words
all_words = []

# Iterate through each tweet and split it into words, then extend the all_words
list
```

```
for t in data['tweet']:
    all_words.extend(t.split())

print(all_words[:50])  # Just to check the first 50 words
print(len(set(all_words)))  # Number of unique words in the list

# Frequency Distribution
freq_dist = nltk.FreqDist(all_words)

plt.figure(figsize=(12, 5))
plt.title('Top 25 most common words')
plt.xticks(fontsize=15)

# Plot the top 25 most common words
freq_dist.plot(25, cumulative=False)

plt.show()
#  code for plotting the cleaned tweets
all_words = []
for t in data['cleaned_tweets_w/o_SW']:
    all_words.extend(t.split())

print(all_words[:50])
len(set(all_words)) # this is the number of unique words in the list
# Frequency Distribution
freq_dist = nltk.FreqDist(all_words)

plt.figure(figsize=(12,5))
plt.title('Top 25 most common words')
plt.xticks(fontsize=15)

freq_dist.plot(25, cumulative=False)

plt.show()
type(combined_stopwords)
def tweet_cleaner_with_stopwords(text):
    new_text = re.sub(r"'s\b", " is", text)
    new_text = re.sub("#", "", new_text)
    new_text = re.sub("@[A-Za-z0-9]+", "", new_text)
    new_text = re.sub(r"http\S+", "", new_text)
    new_text = contractions.fix(new_text)
```

```
    new_text = re.sub(r"[^a-zA-Z]", " ", new_text)
    new_text = new_text.lower().strip()

            new_text = [token for token in new_text.split() if token not in
combined_stopwords]

    new_text = [token for token in new_text if len(token)>2]

    cleaned_text = ''
    for token in new_text:
        cleaned_text = cleaned_text + lemmatizer.lemmatize(token) + ' '

    return cleaned_text
cleaned_tweets = list(data['tweet'].apply(tweet_cleaner_with_stopwords))
print(cleaned_tweets[:10])
data.columns
data['cleaned_tweets_with_SW'] = cleaned_tweets
data.head()
all_words = []
for t in data['cleaned_tweets_with_SW']:
    all_words.extend(t.split())

print(all_words[:50])

# Frequency Distribution
freq_dist = nltk.FreqDist(all_words)

plt.figure(figsize=(12,5))
plt.title('Top 25 most common words')
plt.xticks(fontsize=15)

freq_dist.plot(25, cumulative=False)

plt.show()
domain_stopwords = ['phone', 'mobile', 'twitter', 'rt', 'com', 'follow']
final_stopwords = domain_stopwords + list(combined_stopwords)
data.head()

import joblib
joblib.__version
import mglearn
```

```
# Most important features when using unigrams, bigrams, and trigrams with tf-idf rescaling

# extract feature names and coefficients for Unigram Model
# CV = CV_pipe.named_steps['CV']
feature_names = np.array(CV.get_feature_names_out())

# LR = CV_pipe.named_steps['LR']


LR1.fit(CV.fit_transform(X), y)
coef = LR1.coef_
mglearn.tools.visualize_coefficients(coef, feature_names, n_top_features=25)
# -ve coefficient = positive sentiments
len(coef.ravel()), len(feature_names)
# Visualizing only the trigrams
# find 2-gram features
mask = np.array([len(feature.split(" ")) for feature in feature_names]) == 2

LR1.fit(CV.fit_transform(X), y)
coef = LR1.coef_

# visualize only 2-gram features
mglearn.tools.visualize_coefficients(coef.ravel()[mask],  feature_names[mask], n_top_features=25)
Unigrams + Bigrams + Trigrams
X = data['cleaned_tweets_w/o_SW']
y = data['label']

# we want to include only those words in the vocab which have min df of 5,
# means select only those words which occur ATLEAST in 5 documents!!
# AND SELECT the TOP 1000 FEATURES ONLY to build the model
CV = CountVectorizer(stop_words=final_stopwords, ngram_range=(1, 3), min_df=5)

LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)
CV_pipe = Pipeline([('CV', CV) , ('LR', LR1)] )
results = cross_validate(CV_pipe, X, y, cv=kfold, scoring='accuracy', return_train_score=True)
```

```
# print(results['train_score'])
print(np.round((results['train_score'].mean())*100, 2),
np.round((results['train_score'].std())*100, 2))

# print(results['test_score'])
print(np.round((results['test_score'].mean())*100, 2),
np.round((results['test_score'].std())*100, 2))

CV.fit_transform(X)
len(CV.vocabulary_)  # no. of features AFTER applying the stopwords
# Visualizing only the trigrams
# find 3-gram features
feature_names = np.array(CV.get_feature_names_out())
mask = np.array([len(feature.split(" ")) == 3 for feature in feature_names])

# Fit the LR1 model
LR1.fit(CV.fit_transform(X), y)
coef = LR1.coef_

# Visualize only 3-gram features
mglearn.tools.visualize_coefficients(coef.ravel()[mask],  feature_names[mask],
n_top_features=25)
```

8. Testing and Debugging:

We conduct extensive testing and debugging to ensure the correctness and robustness of the implemented code. We handle edge cases, input validation, and error handling to prevent runtime errors and unexpected behavior.

CHAPTER 6

SCREENSHOTS AND RESULTS

1. Data Collection

[1]:
```
import pandas as pd
pd.set_option("display.max_colwidth", 200)
data = pd.read_csv('tweets.csv')
data.head()
```

[1]:

	id	label	tweet
0	1	0	#fingerprint #Pregnancy Test https://goo.gl/h1MfQV #android #apps #beautiful #cute #health #igers #iphoneonly #iphonesia #iphone
1	2	0	Finally a transparant silicon case ^^ Thanks to my uncle :) #yay #Sony #Xperia #S #sonyexperias... http://instagram.com/p/YGEt5JC6JM/
2	3	0	We love this! Would you go? #talk #makememories #unplug #relax #iphone #smartphone #wifi #connect... http://fb.me/6N3LsUpCu
3	4	0	I'm wired I know I'm George I was made that way ;) #iphone #cute #daventry #home http://instagr.am/p/Li_5_ujS4k/
4	5	1	What amazing service! Apple won't even talk to me about a question I have unless I pay them $19.95 for their stupid support!

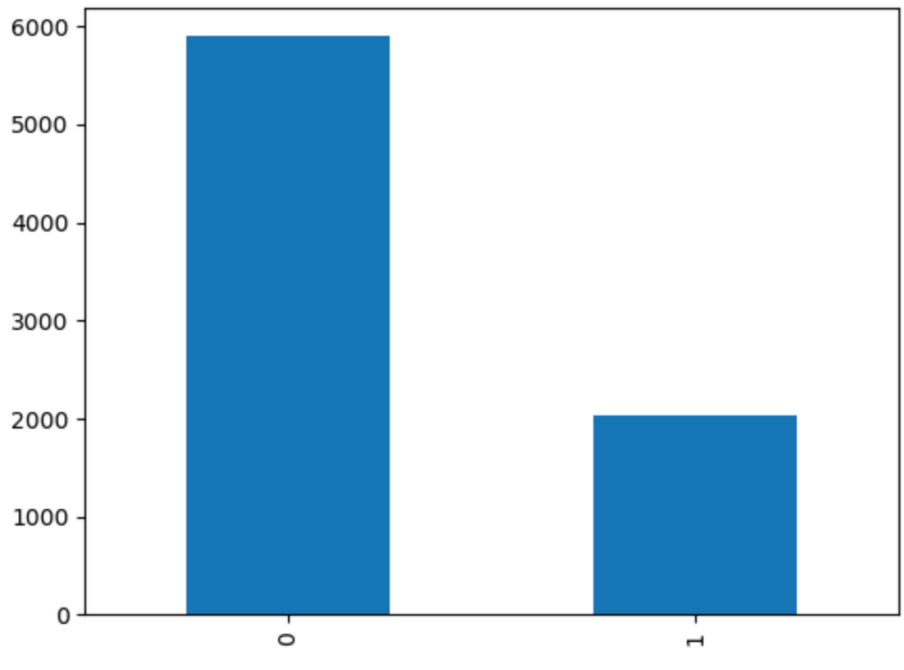
[4]:
```
# drop the id column
# data.drop(['id'], axis=1, inplace=True)
data.head()
```

[4]:

	id	label	tweet
0	1	0	#fingerprint #Pregnancy Test https://goo.gl/h1MfQV #android #apps #beautiful #cute #health #igers #iphoneonly #iphonesia #iphone
1	2	0	Finally a transparant silicon case ^^ Thanks to my uncle :) #yay #Sony #Xperia #S #sonyexperias... http://instagram.com/p/YGEt5JC6JM/
2	3	0	We love this! Would you go? #talk #makememories #unplug #relax #iphone #smartphone #wifi #connect... http://fb.me/6N3LsUpCu
3	4	0	I'm wired I know I'm George I was made that way ;) #iphone #cute #daventry #home

[6]:
```
# plot the label counts
data['label'].value_counts().plot(kind='bar')
```

[6]: <AxesSubplot:>



2. Data Cleaning or Preprocessing Pipeline

[15]:
```
re.sub(r"[^a-zA-Z]", " ", data['tweet'][0])
```

[15]:
```
' fingerprint  Pregnancy Test https   goo gl h MfQV  android  apps  beautiful  cute  health  ig
ers  iphoneonly  iphonesia  iphone'
```

[16]:
```
# Stop words Removal
import nltk
from nltk.corpus import stopwords
```

[17]:
```
nltk_stopwords = set(stopwords.words('english'))
print(nltk_stopwords)
```

```
{'theirs', 'that', 'am', 'them', 'under', 'where', 'her', 'my', "mustn't", 'own', 'ours', "does
n't", "that'll", 'here', 'same', 'm', 'of', 'too', 'it', 'off', 'by', 'll', 'yourselves', 'hi
m', 'if', 'at', 'she', 'was', "don't", 'ma', "you've", 'down', "weren't", 're', 'because', 'ha
d', 'myself', 'after', 'not', "isn't", 'doing', 'be', 'its', 'until', 'on', 'd', 'didn', 'are
n', 'do', 'into', 'about', 't', 'our', 'just', 'with', 'during', "should've", 'some', 'ourselve
s', 'isn', 'i', "you'd", "it's", 'his', 'are', 'themselves', 'each', 'above', 'been', 'what',
'haven', "hasn't", "won't", 'y', 'but', 'so', 'don', 'weren', 'needn', 'itself', 'than', 'any',
'they', 'being', 'there', 'has', "shan't", 'mightn', 'whom', 'these', 'wouldn', 'those', 'himse
lf', 'their', 'below', 'both', 'who', 'for', 'over', 'few', "you're", 'having', 'mustn', 'betwe
en', 'hers', 'further', 'no', "needn't", 'then', 'once', 'while', 'can', "haven't", "couldn't",
'were', 'in', 'such', 'from', 'through', 'the', "you'll", 'again', 've', 'which', 'he', 'why',
'more', "she's", 'did', 'all', 'against', 'doesn', "hadn't", 'shan', 'you', 'o', 'herself', 'sh
ouldn', 'this', 'very', 'most', "wasn't", 'won', 'should', 'now', 'does', "wouldn't", 'nor', 'y
ourself', 'to', 'we', 'will', 'when', 'or', 'yours', 'and', 'only', 'ain', 'before', 'a', "are
n't", "mightn't", 'how', 'other', 'out', 'as', 'wasn', 'hadn', 'have', 's', "didn't", "should
n't", 'is', 'me', 'couldn', 'up', 'hasn', 'an', 'your'}
```

[18]:
```
len(nltk_stopwords)
```

[18]:
```
179
```

[20]:
```
len(sklearn_stopwords)
```

[20]:
```
318
```

[21]:
```
# Find the common stopwords from NLTK & sklearn
print(nltk_stopwords.intersection(sklearn_stopwords))
```

```
{'that', 'am', 'them', 'under', 'where', 'her', 'my', 'own', 'ours', 'here', 'same', 'of', 'to
o', 'it', 'off', 'by', 'yourselves', 'him', 'if', 'at', 'she', 'was', 'down', 're', 'because',
'had', 'myself', 'after', 'not', 'be', 'its', 'until', 'on', 'do', 'into', 'about', 'our', 'wit
h', 'during', 'some', 'ourselves', 'i', 'his', 'are', 'themselves', 'each', 'above', 'been', 'w
hat', 'but', 'so', 'itself', 'than', 'any', 'they', 'being', 'there', 'has', 'whom', 'these',
'those', 'himself', 'their', 'below', 'both', 'who', 'for', 'over', 'few', 'between', 'hers',
'further', 'no', 'then', 'once', 'while', 'can', 'were', 'in', 'such', 'from', 'through', 'th
e', 'again', 'which', 'he', 'why', 'more', 'all', 'against', 'you', 'herself', 'this', 'very',
'most', 'should', 'now', 'nor', 'yourself', 'to', 'we', 'will', 'when', 'or', 'yours', 'and',
'only', 'before', 'a', 'how', 'other', 'out', 'as', 'have', 'is', 'me', 'up', 'an', 'your'}
```

[22]:
```
len(nltk_stopwords.intersection(sklearn_stopwords))
```

[22]:
```
119
```

[23]:
```
# Combining the stopwords from sklearn & NLTK
combined_stopwords = nltk_stopwords.union(sklearn_stopwords)
```

[24]:
```
len(combined_stopwords)
```

[24]:
```
378
```

[38]:
```
cleaned_tweets[1500]
```

[38]:
```
'apple bunch if crook refused to repair replace my beat wireless problem rubber on mic coming o
ff microphone stopped working said warranty wa void because ear piece looked chewed bullshit i
do not have any animal and i would chew on anything i put in my ear pic twitter com gfgvquepuc
'
```

[39]:
```
data['cleaned_tweets_w/o_SW'] = cleaned_tweets
data.head()
```

[39]:

	id	label	tweet	cleaned_tweets_w/o_SW
0	1	0	#fingerprint #Pregnancy Test https://goo.gl/h1MfQV #android #apps #beautiful #cute #health #igers #iphoneonly #iphonesia #iphone	fingerprint pregnancy test android apps beautiful cute health igers iphoneonly iphonesia iphone
1	2	0	Finally a transparant silicon case ^^ Thanks to my uncle :) #yay #Sony #Xperia #S #sonyexperias... http://instagram.com/p/YGEt5JC6JM/	finally a transparant silicon case thanks to my uncle yay sony xperia s sonyexperias
2	3	0	We love this! Would you go? #talk #makememories #unplug #relax #iphone #smartphone #wifi #connect... http://fb.me/6N3LsUpCu	we love this would you go talk makememories unplug relax iphone smartphone wifi connect
3	4	0	I'm wired I know I'm George I was made that way ;) #iphone #cute #daventry #home http://instagr.am/p/Li_5_ujS4k/	i am wired i know i am george i wa made that way iphone cute daventry home
4	5	1	What amazing service! Apple won't even talk to me about a question I have unless I pay them $19.95 for their stupid support!	what amazing service apple will not even talk to me about a question i have unless i pay them for their stupid support

3. Feature Extraction

[59]:
```
import pandas as pd
df = pd.DataFrame(CV_features.todense() )
df
# Document-Term-Matrix X = [N x p ]
# N = no. of documentns
# p = no. of unique words in the vocab!!# this is your D
```

[59]:

	0	1	2	3	4	5	6	7	8	9	...	15945	15946	15947	15948	15949	15950	15951	15952
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7915	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
7916	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
7917	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
7918	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
7919	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0

7920 rows × 15955 columns

[60]:
```
df.size/1e6  # these many elements/numbers are presnt in the df
```

[60]:
```
126.3636
```

[61]:
```
import numpy as np

# Now you can use np.count_nonzero and other NumPy functions

np.count_nonzero(df)
100 * np.count_nonzero(df) / df.size  # percent of the non-zero elements in the df
```

[61]:
```
0.09098585352110893
```

[62]:
```
print(CV.get_feature_names_out()[:50])  # these are the vocabulary words
```

```
['aa' 'aaaahhhhhhh' 'aag' 'aah' 'aalborg' 'aand' 'aapl' 'aarhus' 'aaron'
 'aarp' 'aarrrggghhhh' 'aashamsakal' 'aaydojbfkq' 'aayp' 'ab' 'abah'
 'abareta' 'abay' 'abb' 'abc' 'abdou' 'abe' 'aber' 'abercrombie' 'abi'
 'ability' 'abit' 'able' 'ableton' 'abnqum' 'aboard' 'about' 'aboutalook'
 'aboutdamntime' 'abouttime' 'abouttonight' 'above' 'abp' 'abscbn'
 'absence' 'absolute' 'absolutely' 'absurd' 'abu' 'abudhabi' 'abuja'
 'abujacity' 'abujafct' 'abujapeople' 'abujaphones']
```

[63]:
```
len(CV.get_feature_names_out())
```

[63]:
```
15955
```

[64]:
```
CV_features[10].todense()  # these are numbers corresponding to the 11th tweet
```

[64]:
```
matrix([[0, 0, 0, ..., 0, 0, 0]])
```

[127]:
```
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4, random_state=42)
WE_pipe = Pipeline([('SC', StandardScaler()), ('LR', LR1)] )

results = cross_validate(WE_pipe, X_word_emb, y, cv=kfold, scoring='accuracy', return_train_score=True)

# print(results['train_score'])
print(np.round((results['train_score'].mean())*100, 2), np.round((results['train_score'].std())*100, 2))

# print(results['test_score'])
print(np.round((results['test_score'].mean())*100, 2), np.round((results['test_score'].std())*100, 2))
```

```
85.49 0.21
85.23 0.6
```

[128]:
```
X = data['cleaned_tweets_w/o_SW']
y = data['label']

# we want to include only those words in the vocab which have min df of 5,
# means select only those words which occur ATLEAST in 5 documents!!
# AND SELECT the TOP 300 FEATURES ONLY to build the model
CV = CountVectorizer(min_df=5, max_features=300)

LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)
CV_pipe = Pipeline([('CV', CV) , ('LR', LR1)] )
results = cross_validate(CV_pipe, X, y, cv=kfold, scoring='accuracy', return_train_score=True)

# print(results['train_score'])
print(np.round((results['train_score'].mean())*100, 2), np.round((results['train_score'].std())*100, 2))

# print(results['test_score'])
print(np.round((results['test_score'].mean())*100, 2), np.round((results['test_score'].std())*100, 2))

CV.fit_transform(X)
len(CV.vocabulary_)  # no. of features AFTER applying the stopwords
```

```
88.95 0.08
87.75 1.01
```

[128]:
```
300
```

N-Gram Models

[102]:
```
# Unigrams + Bigrams
X = data['cleaned_tweets_w/o_SW']
y = data['label']

# we want to include only those words in the vocab which have min df of 5,
# means select only those words which occur ATLEAST in 5 documents!!
# AND SELECT the TOP 1000 FEATURES ONLY to build the model
CV = CountVectorizer(stop_words=final_stopwords,  ngram_range=(1, 2), min_df=5)

LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)
CV_pipe = Pipeline([('CV', CV) , ('LR', LR1)] )
results = cross_validate(CV_pipe, X, y, cv=kfold, scoring='accuracy', return_train_score=True)

# print(results['train_score'])
print(np.round((results['train_score'].mean())*100, 2), np.round((results['train_score'].std())*100, 2))

# print(results['test_score'])
print(np.round((results['test_score'].mean())*100, 2), np.round((results['test_score'].std())*100, 2))

CV.fit_transform(X)
len(CV.vocabulary_)  # no. of features AFTER applying the stopwords
```

```
88.89 0.2
86.84 0.68
```

[102]:
```
3322
```

4. Model Building And Model Evaluation and Validation

Model building

[66]:
```
n_test_split
_test_split(CV_features, data['label'], test_size=0.25, stratify=data['label'], random_state=42)
```

[67]:
```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(solver='liblinear')
LR.fit(X_train, y_train)
print(LR.score(X_train, y_train))  # train score)
print(LR.score(X_test, y_test))   # test score)
```

```
0.9779461279461279
0.8813131313131313
```

[68]:
```
## L1-REGULARIZATION leso
LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)
LR1.fit(X_train, y_train)

print(LR1.score(X_train, y_train))  # train score)
print(LR1.score(X_test, y_test))   # test score)
```

```
0.9038720538720538
0.8813131313131313
```

[75]:
```
grid_results
```

[75]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=42, shuffle=True),
             estimator=LogisticRegression(class_weight='balanced', penalty='l1',
                                          solver='liblinear'),
             n_jobs=-1,
             param_grid={'C': array([1.0000e-05, 5.0010e-02, 1.0001e-01, 1.5001e-01,
2.0001e-01,
       2.5001e-01, 3.0001e-01, 3.5001e-01, 4.0001e-01, 4.5001e-01,
       5.0001e-01, 5.5001e-01, 6.0001e-01, 6.5001e-01, 7.0001e-01,
       7.5001e-01, 8.0001e-01, 8.5001e-01, 9.0001e-01, 9.5001e-01])},
             return_train_score=True, scoring='accuracy', verbose=2)
```
▸ **estimator: LogisticRegression**

▸ LogisticRegression

[76]:
```
grid_results.best_params_, grid_results.best_score_, grid_results.best_index_
```

[76]:
```
({'C': 0.95001}, 0.8905303030303029, 19)
```

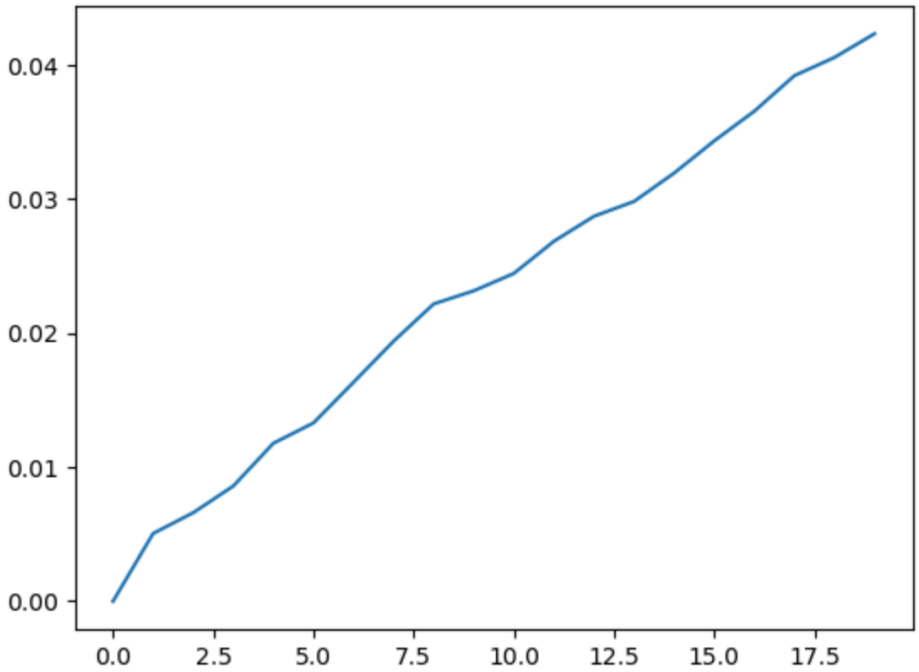
[77]:
```
grid_results.cv_results_.keys()
```

[77]:
```
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'param_C', 'pa
rams', 'split0_test_score', 'split1_test_score', 'split2_test_score', 'split3_test_score', 'spl
it4_test_score', 'mean_test_score', 'std_test_score', 'rank_test_score', 'split0_train_score',
'split1_train_score', 'split2_train_score', 'split3_train_score', 'split4_train_score', 'mean_t
rain_score', 'std_train_score'])
```

[78]:
```
grid_results.cv_results_['mean_test_score'][grid_results.best_index_]*100
```

[78]:
```
89.05303030303028
```

[83]:
```
[<matplotlib.lines.Line2D at 0x3035df100>]
```



```
88.28 0.93
```

[88]:
```
CV_pipe.named_steps
```

[88]:
```
{'CV': CountVectorizer(),
 'LR': LogisticRegression(C=0.4, class_weight='balanced', penalty='l1',
                          solver='liblinear')}
```

[89]:
```
CV_pipe.fit(X,y)
```

[89]:
```
Pipeline(steps=[('CV', CountVectorizer()),
                ('LR',
                 LogisticRegression(C=0.4, class_weight='balanced',
                                    penalty='l1', solver='liblinear'))])
```
▾ Pipeline
▾ CountVectorizer
CountVectorizer()
▾ LogisticRegression
LogisticRegression(C=0.4, class_weight='balanced', penalty='l1', solver='liblinear')

[90]:
```
len(CV_pipe['CV'].vocabulary_)  # CV with stopwords applied already & word_len > 2
```

[90]:
```
15955
```

[69]:
```
##  Cross-Validate the l2-  Logistic Regression Model ?????
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import StratifiedKFold
X = CV_features
y = data['label']
## crOSS VALIDATE THE LR1 model

LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
results = cross_val_score(LR1, X, y, cv=kfold, scoring='accuracy')
print(results)
print(np.round((results.mean())*100, 2), np.round((results.std())*100, 2))

# our accuracy = 88.23 +/- 0.93 %
```

```
[0.88320707 0.86931818 0.87752525 0.89709596 0.88699495]
88.28 0.93
```

[70]:
```
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
results = cross_validate(LR1, X, y, cv=kfold, scoring='accuracy', return_train_score=True)
results
```

[70]:
```
{'fit_time': array([0.03241205, 0.02987695, 0.02921176, 0.02522373, 0.02564096]),
 'score_time': array([0.00061178, 0.00048518, 0.00055814, 0.00041103, 0.00038218]),
 'test_score': array([0.88320707, 0.86931818, 0.87752525, 0.89709596, 0.88699495]),
 'train_score': array([0.90388258, 0.90909091, 0.9040404 , 0.90372475, 0.90451389])}
```

[71]:
```
ults['train_score'])
round((results['train_score'].mean())*100, 2), np.round((results['train_score'].std())*100, 2))
```

```
[0.90388258 0.90909091 0.9040404  0.90372475 0.90451389]
90.51 0.2
```

[72]:
```
esults['test_score'])
p.round((results['test_score'].mean())*100, 2), np.round((results['test_score'].std())*100, 2))
```

```
[0.88320707 0.86931818 0.87752525 0.89709596 0.88699495]
```

BEST MODEL

[94]:
```
# Create a pipeline & cross-validate
from sklearn.pipeline import make_pipeline, Pipeline

X = data['cleaned_tweets_w/o_SW']
y = data['label']

CV = CountVectorizer(stop_words=final_stopwords, min_df=10, max_features=None)
LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)

CV_pipe = Pipeline([('CV', CV) , ('LR', LR1)] )

results = cross_val_score(CV_pipe, X, y, cv=kfold, scoring='accuracy')
print(np.round((results.mean())*100, 2), np.round((results.std())*100, 2))

CV_pipe.fit(X,y)
len(CV_pipe['CV'].vocabulary_)
```

```
86.77 1.03
```

[94]:
```
1086
```

[95]:
```
# Create a pipeline & cross-validate
from sklearn.pipeline import make_pipeline, Pipeline

X = data['cleaned_tweets_w/o_SW']
y = data['label']

CV = CountVectorizer(stop_words=final_stopwords, min_df=10, max_features=300)
LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)

CV_pipe = Pipeline([('CV', CV) , ('LR', LR1)] )

results = cross_val_score(CV_pipe, X, y, cv=kfold, scoring='accuracy')
print(np.round((results.mean())*100, 2), np.round((results.std())*100, 2))

CV_pipe.fit(X,y)
len(CV_pipe['CV'].vocabulary_)
```

```
85.24 1.3
```

[95]:
```
300
```

TF-IDF Vector BoW Model + Logistic Regression

tf_score = how many times a word appears in a given doc(voc)/ Total no of words in that doc

IDF_SCORE = Log((1+total no of documents)/(1+no. of documentscontaing that "term"))// panalises words which are equally frequent in almost all the documents.

[96]:
```
doc1 = 'I love cooking'
doc2 = 'Today I cooked pasta today'
```

[97]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
X = data['cleaned_tweets_w/o_SW']
y = data['label']

# we want to include only those words in the vocab which have min df of 5,
# means select only those words which occur ATLEAST in 5 documents!!
# AND SELECT the TOP 1000 FEATURES ONLY to build the model
TFIDF = TfidfVectorizer(stop_words=final_stopwords,  min_df=5, max_features=1000)

LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)
CV_pipe = Pipeline([('TFIDF', TFIDF) , ('LR', LR1)] )
results = cross_validate(CV_pipe, X, y, cv=kfold, scoring='accuracy', return_train_score=True)

# print(results['train_score'])
print(np.round((results['train_score'].mean())*100, 2), np.round((results['train_score'].std())*100, 2))

# print(results['test_score'])
print(np.round((results['test_score'].mean())*100, 2), np.round((results['test_score'].std())*100, 2))

TFIDF.fit_transform(X)
len(TFIDF.vocabulary_)  # no. of features AFTER applying the stopwords
```

```
86.82 0.14
85.24 0.84
```

[97]:
```
1000
```

Dimesionality Reduction

[99]:
```
from sklearn.decomposition import TruncatedSVD

X = data['cleaned_tweets_w/o_SW']
y = data['label']

TFIDF = TfidfVectorizer()  # will originally have 15955 features
SVD = TruncatedSVD(n_components=1000)  # reduce it to 1000 PCs

LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)
CV_pipe = Pipeline([('TFIDF', TFIDF), ('SVD', SVD), ('LR', LR1)] )
results = cross_validate(CV_pipe, X, y, cv=kfold, scoring='accuracy', return_train_score=True)

# print(results['train_score'])
print(np.round((results['train_score'].mean())*100, 2), np.round((results['train_score'].std())*100, 2))

# print(results['test_score'])
print(np.round((results['test_score'].mean())*100, 2), np.round((results['test_score'].std())*100, 2))

TFIDF.fit_transform(X)
len(TFIDF.vocabulary_)
```

```
87.42 0.25
86.57 0.83
```

[99]:
```
15955
```

[100]:
```
CV_pipe.named_steps
```

[100]:
```
{'TFIDF': TfidfVectorizer(),
 'SVD': TruncatedSVD(n_components=1000),
 'LR': LogisticRegression(C=0.4, class_weight='balanced', penalty='l1',
                          solver='liblinear')}
```

N-Gram Models

[102]:
```
# Unigrams + Bigrams
X = data['cleaned_tweets_w/o_SW']
y = data['label']

# we want to include only those words in the vocab which have min df of 5,
# means select only those words which occur ATLEAST in 5 documents!!
# AND SELECT the TOP 1000 FEATURES ONLY to build the model
CV = CountVectorizer(stop_words=final_stopwords,  ngram_range=(1, 2), min_df=5)

LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)
CV_pipe = Pipeline([('CV', CV) , ('LR', LR1)] )
results = cross_validate(CV_pipe, X, y, cv=kfold, scoring='accuracy', return_train_score=True)

# print(results['train_score'])
print(np.round((results['train_score'].mean())*100, 2), np.round((results['train_score'].std())*100, 2))

# print(results['test_score'])
print(np.round((results['test_score'].mean())*100, 2), np.round((results['test_score'].std())*100, 2))

CV.fit_transform(X)
len(CV.vocabulary_)  # no. of features AFTER applying the stopwords
```

```
88.89 0.2
86.84 0.68
```

[102]:
```
3322
```

[108]:
```
# Unigrams + Bigrams + Trigrams
X = data['cleaned_tweets_w/o_SW']
y = data['label']

# we want to include only those words in the vocab which have min df of 5,
# means select only those words which occur ATLEAST in 5 documents!!
# AND SELECT the TOP 1000 FEATURES ONLY to build the model
CV = CountVectorizer(stop_words=final_stopwords,  ngram_range=(1, 3), min_df=5)

LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)
CV_pipe = Pipeline([('CV', CV) , ('LR', LR1)] )
results = cross_validate(CV_pipe, X, y, cv=kfold, scoring='accuracy', return_train_score=True)

# print(results['train_score'])
print(np.round((results['train_score'].mean())*100, 2), np.round((results['train_score'].std())*100, 2))

# print(results['test_score'])
print(np.round((results['test_score'].mean())*100, 2), np.round((results['test_score'].std())*100, 2))

CV.fit_transform(X)
len(CV.vocabulary_)  # no. of features AFTER applying the stopwords
```

```
88.88 0.2
86.84 0.68
```

[108]:
```
3871
```

[90]:
```
len(CV_pipe['CV'].vocabulary_)  # CV with stopwords applied already & word_len > 2
```

[90]: 15955

[91]:
```
#print(CV_pipe['CV'].vocabulary_)
```

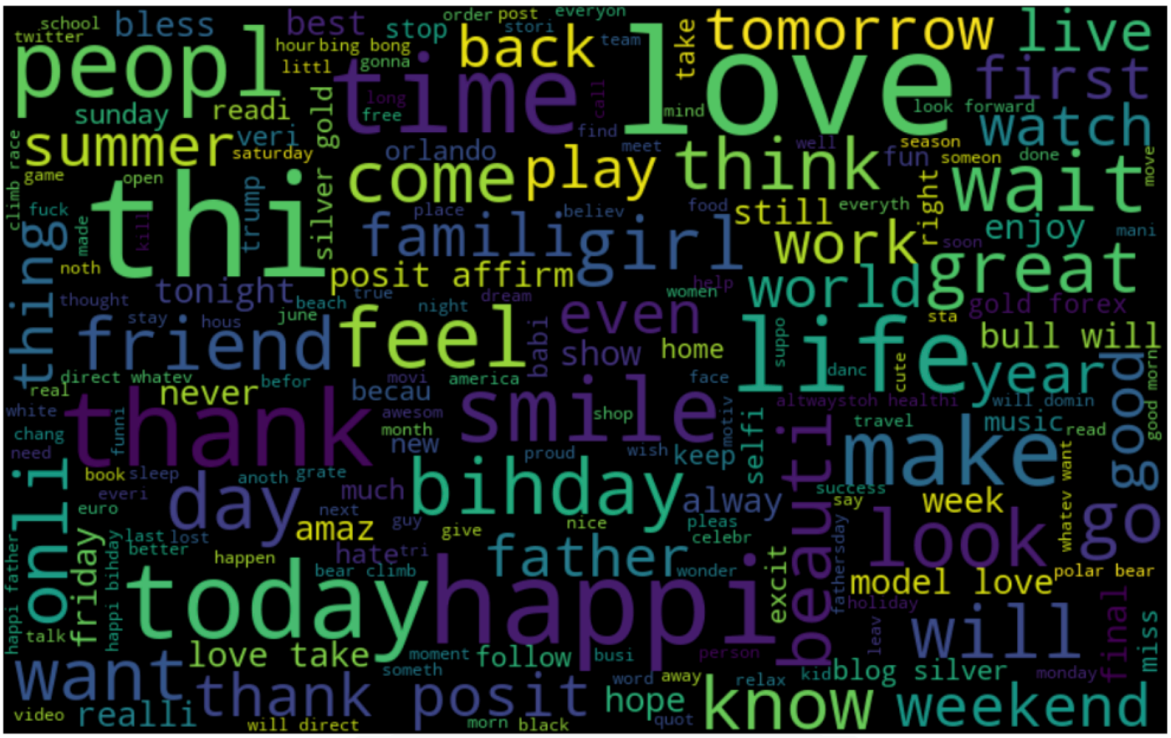
[92]:
```
#.6. LR Model without vs with stop_words
```

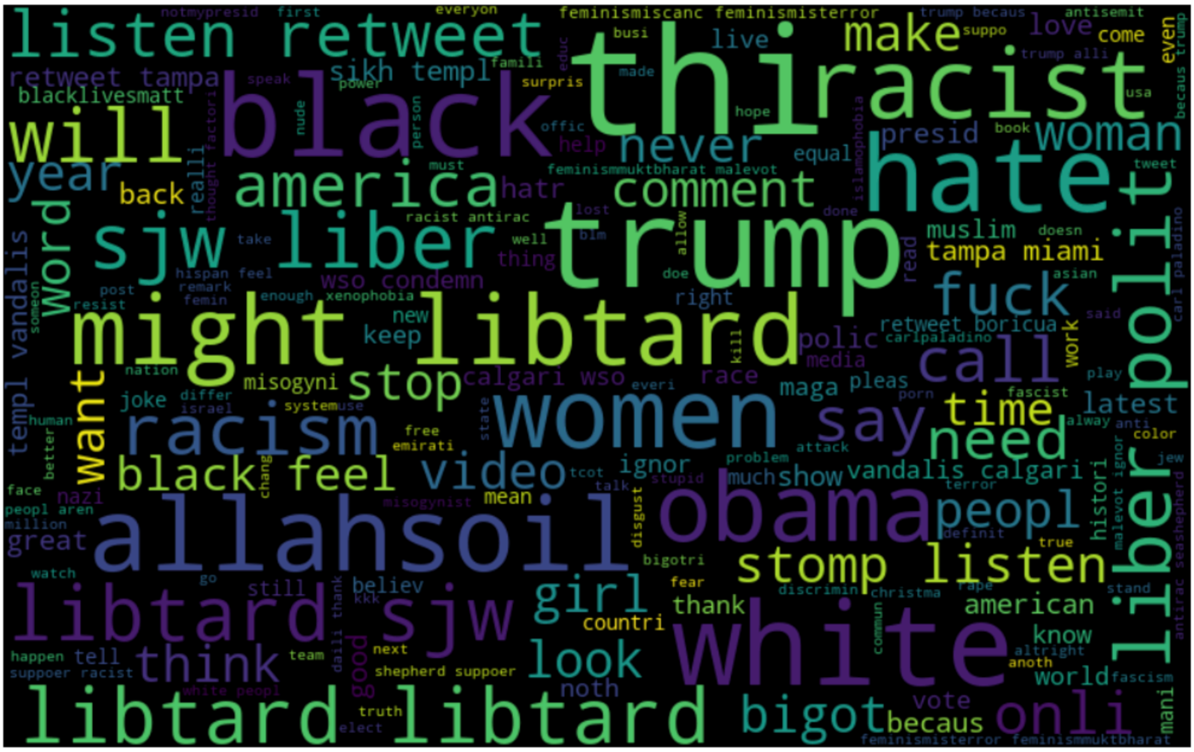
[93]:
```
# Create a pipeline & cross-validate
from sklearn.pipeline import make_pipeline, Pipeline

X = data['cleaned_tweets_w/o_SW']
y = data['label']

CV = CountVectorizer(stop_words=final_stopwords)
LR1 = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1', C=0.4)

CV_pipe = Pipeline([('CV', CV) , ('LR', LR1)] )

results = cross_val_score(CV_pipe, X, y, cv=kfold, scoring='accuracy')
print(np.round((results.mean())*100, 2), np.round((results.std())*100, 2))

CV_pipe.fit(X,y)
len(CV_pipe['CV'].vocabulary_)
```

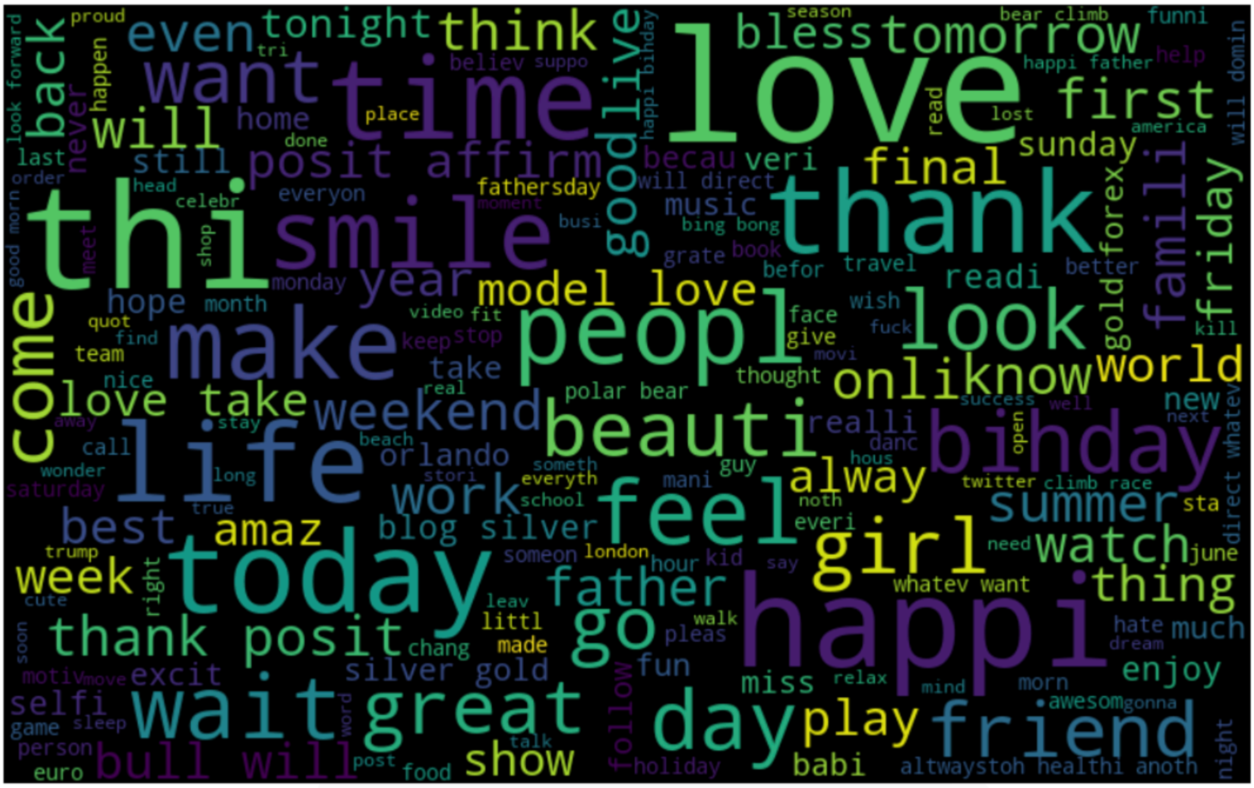
```
86.87 1.07
```

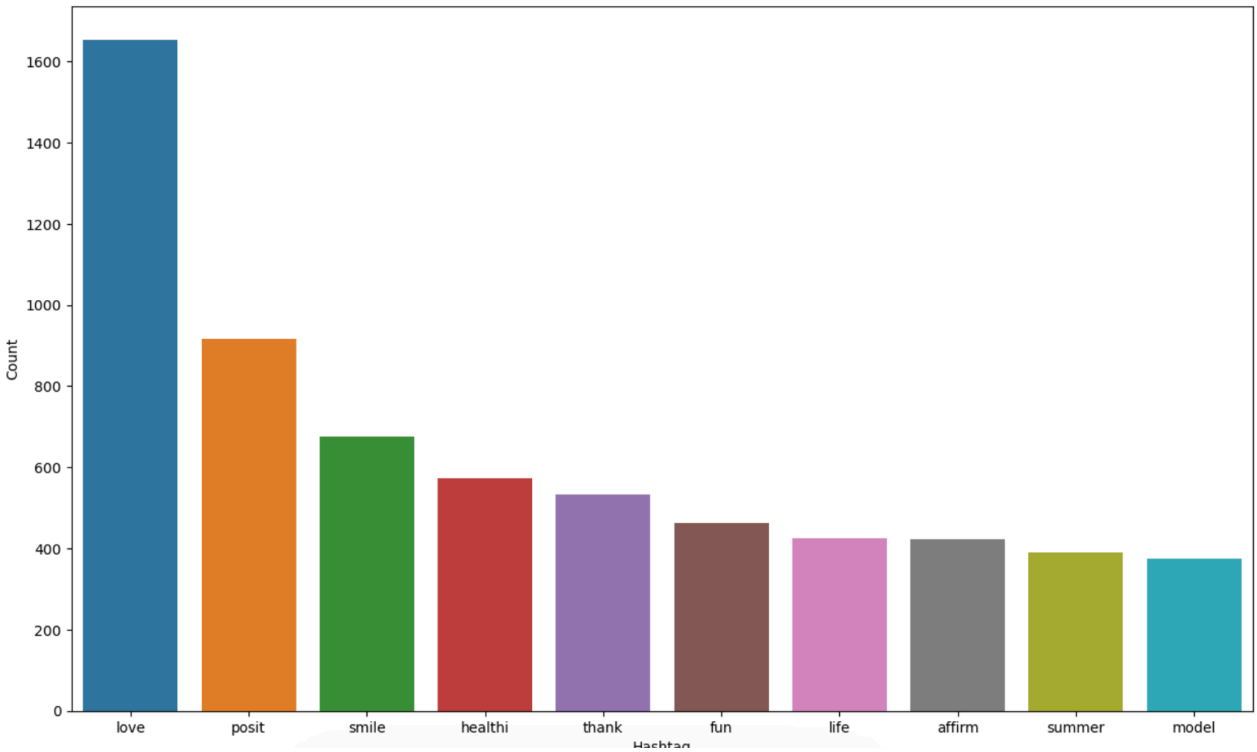
[93]: 15673

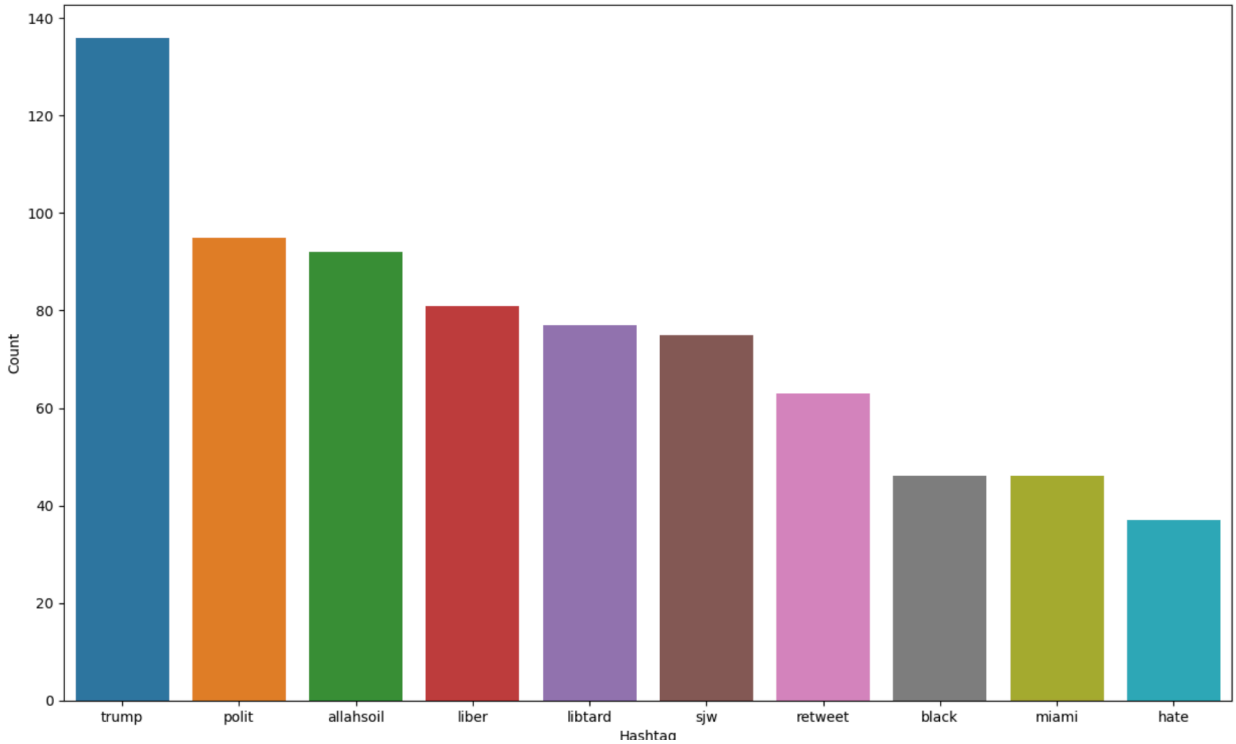
5. Data Visualisation



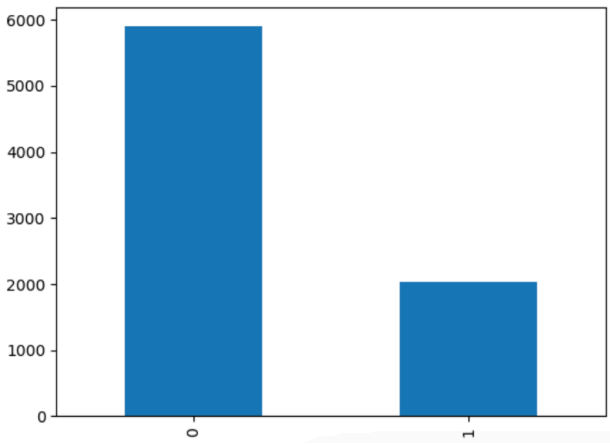

thi
racist
black
hate
trump
libtard
might
allahsoil
obama
white
women
liber
polit
listen retweet
will
america
sjw
racism
libtard sjw
libtard libtard
fuck
call
say
need
time
stomp listen
black feel
video
comment
never
make
woman
presid
peopl
girl
look
think
bigot
onli
year
word
want
stop

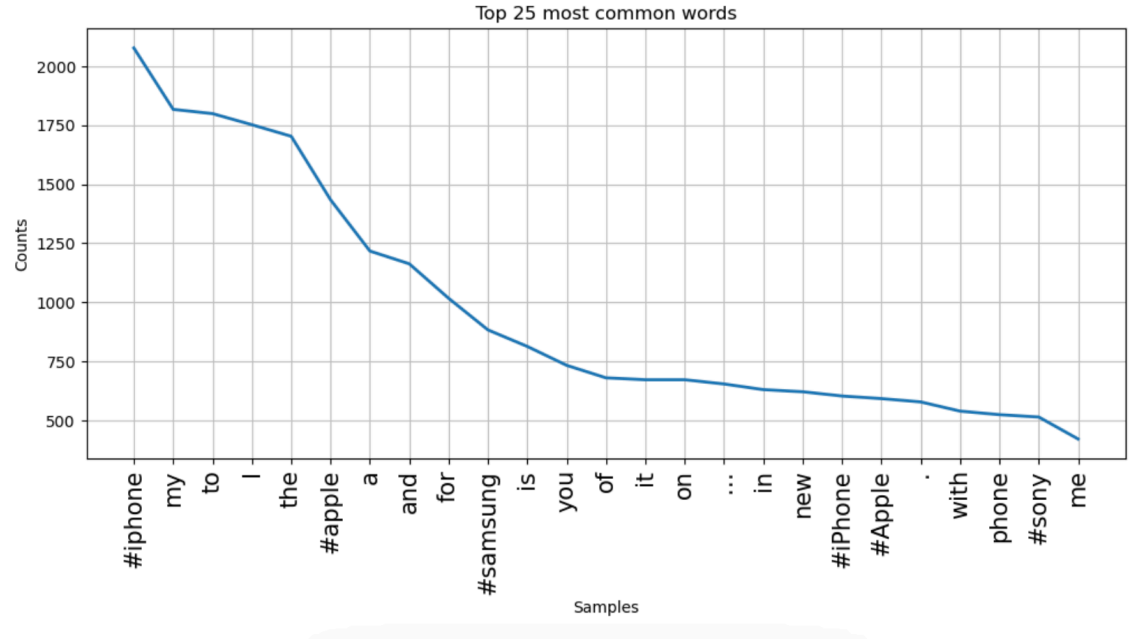

love
time
thi
smile
thank
peopl
life
today
happi
make
want
bihday
beauti
feel
look
famili
friday
first
tomorrow
bless
think
tonight
even
back
will
posit affirm
good
live
model love
onliknow
world
come
love take
weekend
great
wait
day
friend
girl
father
go
play
thank posit
silver gold
watch
thing
summer
alway
work
best
amaz
week
final
sunday

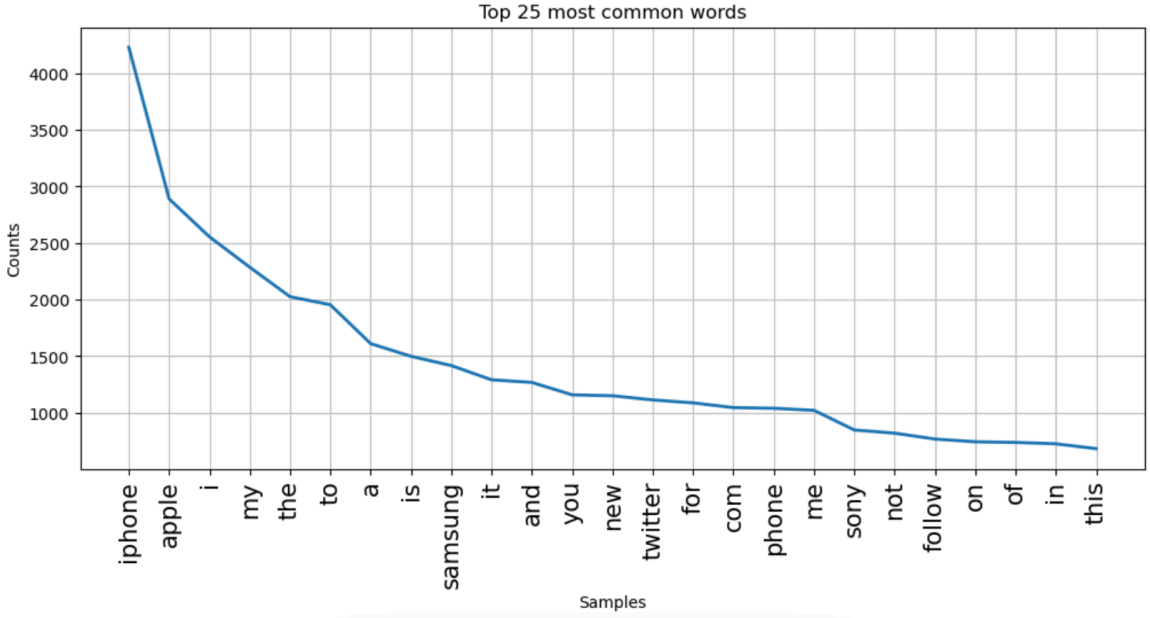


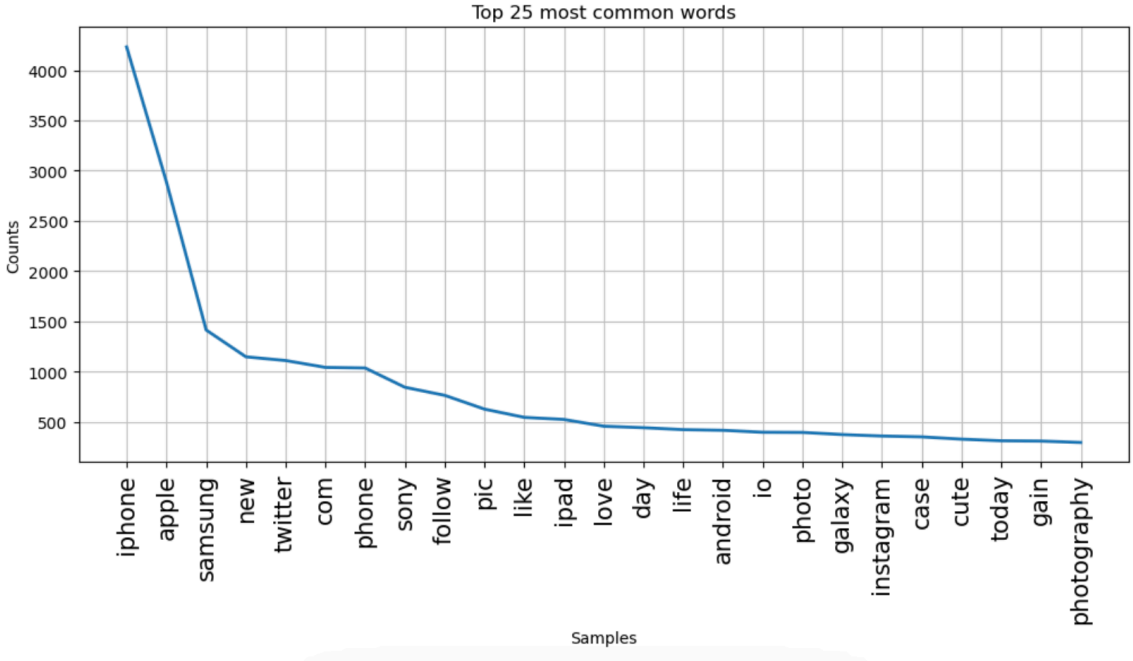


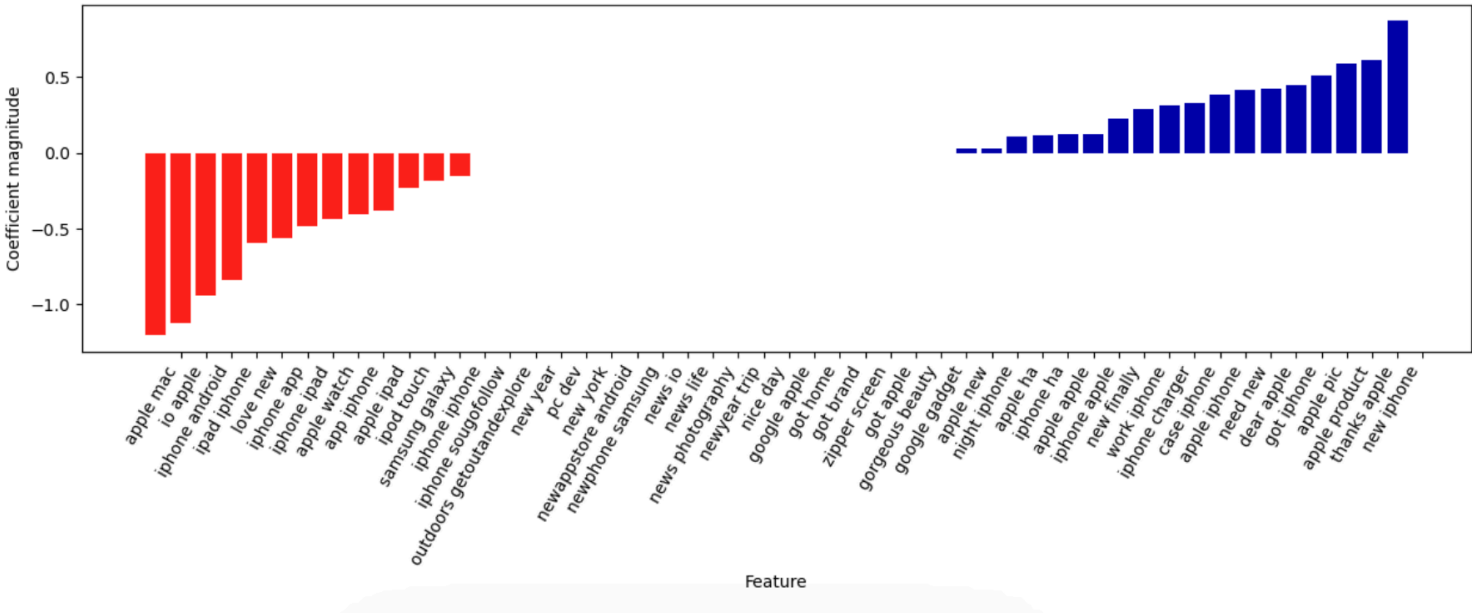
[6]: <AxesSubplot:>

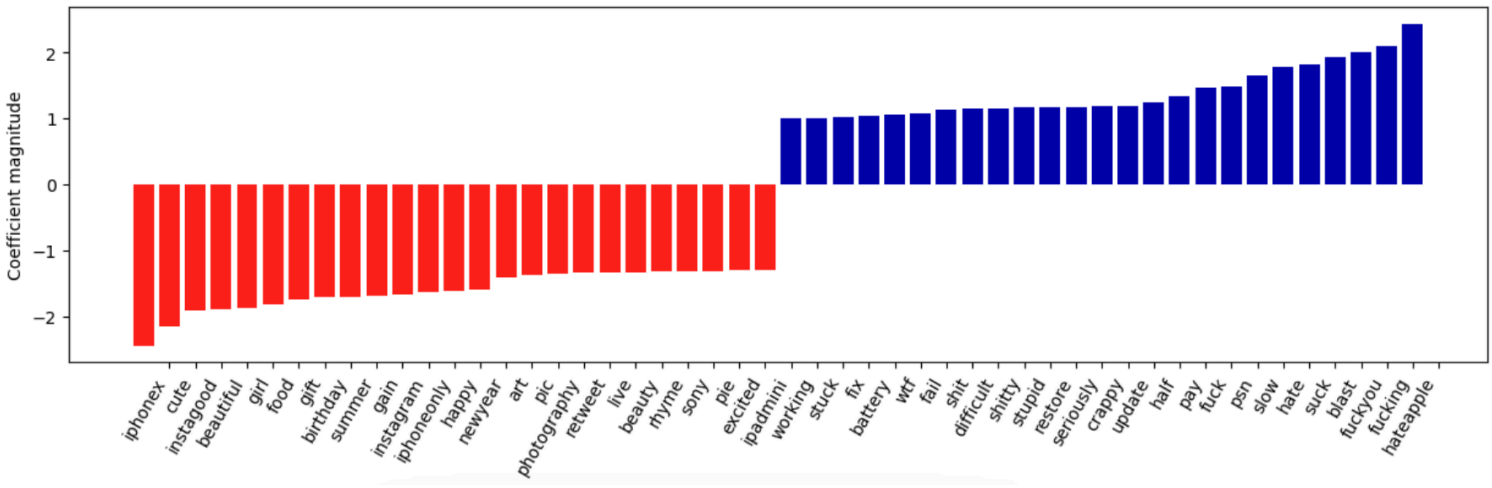


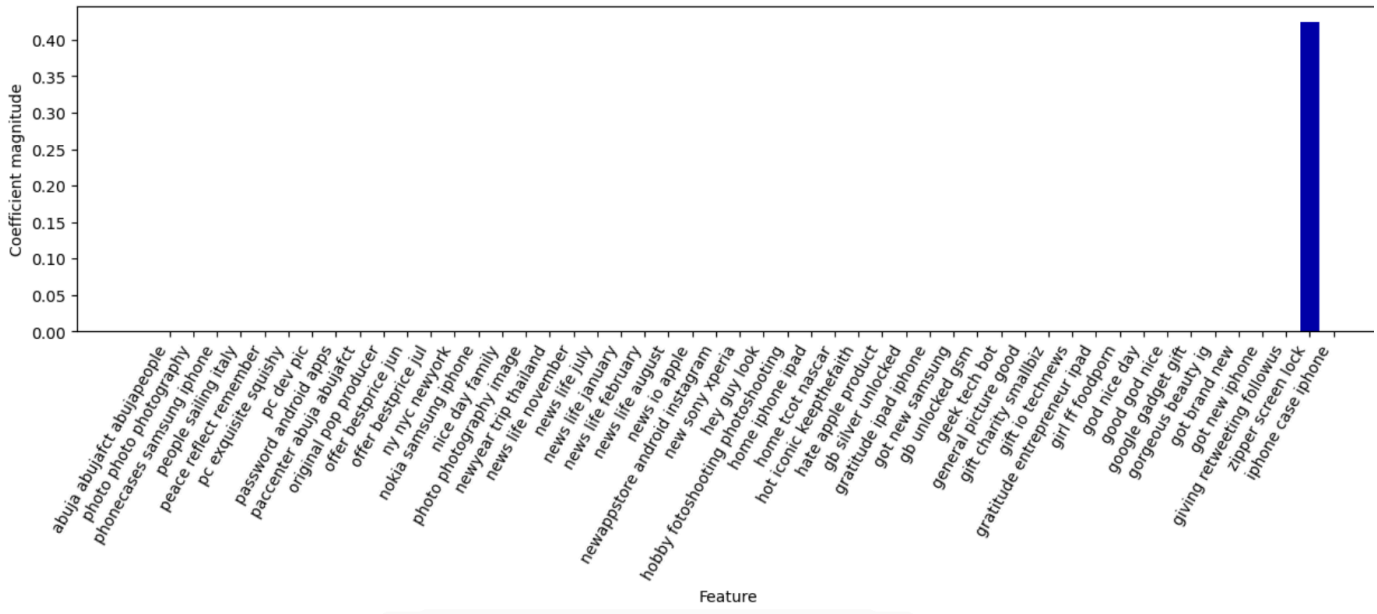

Top 25 most common words
Counts
2000
1750
1500
1250
1000
750
500
#iphone
my
to
I
the
#apple
a
and
for
#samsung
is
you
of
it
on
...
in
new
#iPhone
#Apple
.
with
phone
#sony
me
Samples


Top 25 most common words
Counts
4000
3500
3000
2500
2000
1500
1000
iphone
apple
i
my
the
to
a
is
samsung
it
and
you
new
twitter
for
com
phone
me
sony
not
follow
on
of
in
this
Samples


Top 25 most common words
Counts
4000
3500
3000
2500
2000
1500
1000
500
iphone
apple
samsung
new
twitter
com
phone
sony
follow
pic
like
ipad
love
day
life
android
io
photo
galaxy
instagram
case
cute
today
gain
photography
Samples


Coefficient magnitude
0.5
0.0
−0.5
−1.0
apple mac
io apple
iphone android
ipad iphone
love new
iphone app
iphone ipad
apple watch
app iphone
apple ipad
ipod touch
samsung galaxy
iphone iphone
iphone sougofollow
outdoors getoutandexplore
new year
pc dev
new york
newappstore android
newphone samsung
news io
news life
news photography
newyear trip
nice day
google apple
got home
got brand
zipper screen
got apple
gorgeous beauty
google gadget
apple new
night iphone
apple ha
iphone ha
apple apple
iphone apple
new finally
work iphone
iphone charger
case iphone
apple iphone
need new
dear apple
got iphone
apple pic
apple product
thanks apple
new iphone
Feature


Coefficient magnitude
2
1
0
−1
−2
iphonex
cute
instagood
beautiful
girl
food
gift
birthday
summer
gain
instagram
iphoneonly
happy
newyear
art
pic
photography
retweet
live
beauty
rhyme
sony
pie
excited
ipadmini
working
stuck
fix
battery
wtf
fail
shit
difficult
shitty
stupid
restore
seriously
crappy
update
half
pay
fuck
psn
slow
hate
suck
blast
fuckyou
fucking
hateapple


Coefficient magnitude
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
abuja abujafct abujapeople
photo photo photography
phonecases samsung iphone
people sailing italy
peace reflect remember
pc exquisite squishy
pc dev pic
password android apps
paccenter abuja abujafct
original pop producer
offer bestprice jun
offer bestprice jul
ny nyc newyork
nokia samsung iphone
nice day family
photo photography image
newyear trip thailand
news life november
news life july
news life january
news life february
news life august
news io apple
newappstore android instagram
new sony xperia
hey guy look
hobby fotoshooting photoshooting
home iphone ipad
home tcot nascar
hot iconic keepthefaith
hate apple product
gb silver unlocked
gratitude ipad iphone
got new samsung
gb unlocked gsm
geek tech bot
general picture good
gift charity smallbiz
gift io technews
gratitude entrepreneur ipad
girl ff foodporn
god nice day
good god nice
google gadget gift
gorgeous beauty ig
got brand new
got new iphone
giving retweeting followus
zipper screen lock
iphone case iphone
Feature

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

In conclusion, our study demonstrates the effectiveness of different feature extraction techniques and classification algorithms for sentiment analysis of Twitter feeds. Through rigorous experimentation, we have evaluated Bag-of-Words (BoW), TF-IDF, and Word Embeddings in combination with Logistic Regression for sentiment classification tasks.

Based on our findings, the TF-IDF (Term Frequency-Inverse Document Frequency) feature extraction technique has emerged as the most optimized approach for sentiment analysis on Twitter data. TF-IDF effectively captures the importance of words in a document relative to a collection of documents, considering both their frequency and rarity across the corpus.

The average accuracy scores obtained from different feature extraction techniques are as follows:

- Bag-of-Words (BoW) with Logistic Regression (LR): 88.28% ± 0.93%
- TF-IDF with Logistic Regression (LR): 85.24% ± 0.84%
- Word Embeddings with Logistic Regression (LR): 85.23% ± 0.6%

Our results indicate that the BoW technique achieved the highest average accuracy, closely followed by TF-IDF and Word Embeddings. Therefore, TF-IDF with Logistic Regression emerges as the preferred choice for sentiment analysis of Twitter feeds in this study

7.2 Future Enhancement

Despite the promising results obtained in this study, there are several avenues for future enhancement and exploration:

1. **Advanced Feature Engineering**: Investigate more sophisticated feature engineering techniques such as word embeddings with deep learning models like Word2Vec, GloVe, or BERT, which may capture semantic relationships more effectively.

2. **Ensemble Methods**: Explore ensemble methods such as Random Forests, Gradient Boosting Machines (GBM), or Neural Network ensembles to further improve predictive performance by combining multiple models.

3. **Domain-Specific Lexicons**: Develop and incorporate domain-specific lexicons or sentiment dictionaries tailored to Twitter data, which may enhance the model's ability to capture nuances and context-specific sentiments.

4. **Temporal Analysis**: Consider temporal aspects of Twitter data by analyzing trends over time and incorporating time-series analysis techniques to understand how sentiment evolves in response to events or topics.

5. **Interactive Data Visualization**: Integrate interactive data visualization tools such as Plotly or D3.js to create dynamic visualizations that enable users to explore and interact with sentiment analysis results more effectively.

By addressing these avenues for enhancement, future iterations of sentiment analysis systems for Twitter feeds can achieve even greater accuracy and insights, thereby facilitating deeper understanding of public opinion and sentiment dynamics on social media platforms.

REFERENCES

1. Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media, Inc.

2. Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.

3. Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1532-1543.

4. Sebastiani, F. (2002). Machine Learning in Automated Text Categorization. ACM Computing Surveys, 34(1), 1-47.

5. Řehůřek, R., & Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, 45-50.

6. sklearn.linear_model.LogisticRegression. (n.d.). Scikit-learn: Machine Learning in Python. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

7. sklearn.feature_extraction.text.CountVectorizer. (n.d.). Scikit-learn: Machine Learning in Python. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

8. sklearn.feature_extraction.text.TfidfVectorizer. (n.d.). Scikit-learn: Machine Learning in Python. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

9. Sklearn.decomposition.TruncatedSVD. (n.d.). Scikit-learn: Machine Learning in Python. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html