Machine Learning Programming (PROG8245)

Project report on

Sentiment Analysis

Submitted by
Dhruv Parmar – 8681628

Harshvardhan Tyagi – 8856516

Nihal Patel – 8945100

Deep Shah - 8836846

Course Teacher

Prof. Islam Mahmoud

Acknowledgments

We would like to express my gratitude to Dhruv Parmar, Harshvardhan Tyagi, Nihal Patel and Deep Shah for their contribution to the development of the sentiment analysis model presented in this report. The success of this project was greatly facilitated by the collaborative efforts and insights shared during its implementation. The source code for the sentiment analysis model is hosted on GitHub, and readers are encouraged to explore the repository for a more in-depth understanding of the project. The GitHub repository can be accessed at the following link: https://github.com/dhruvRj18/Sentiment-Analysis.

Contents

Abstract

Social media platforms are used extensively by everyone to share their opinions on different matters. On top of it, interest in social media sentiment analysis has skyrocketed in recent years. The content posted by people on social media gives perspective of the real-world environment and social phenomena related to any given topic. Sentiment analysis is usually done using different Natural Language Processing techniques and deep and/or shallow learning algorithms in AI. This report proposes thorough technical steps leading into model development that classifies the sentiment of the input text.

Introduction

In today's interconnected digital age, social media platforms stand as virtual arenas where individuals freely express their viewpoints and emotions. This project delves into the realm of sentiment analysis, a burgeoning field at the intersection of Natural Language Processing and AI. Its primary focus is deciphering sentiments embedded within the extensive pool of user-generated content found on Reddit and YouTube.

This endeavor recognizes social media platforms as mirrors reflecting real-time societal sentiments. The methodology involves meticulous data collection from Reddit's 'Kitchener' subreddit using Python's praw library and mining insights from a Kitchener-related YouTube video via the Google YouTube API V3. The intent is to synthesize diverse data sources to construct a nuanced sentiment analysis model.

This report provides an intricate roadmap detailing the technical intricacies behind constructing a robust sentiment classification system. It underscores the importance of comprehending collective emotions within these digital arenas. By discerning the underlying sentiments pervading these platforms, this project aspires to offer profound insights into the dynamic landscape of online discourse.

Methodology

This section covers the methodology to achieve the sentiment analysis system development.

Data Gathering

For training ML model efficiently, the proposed system collects the data from two different but prominent social media platforms, Reddit and YouTube.

Reddit

Reddit is a primary platform for many internet users to share their opinions and get answers to a variety of questions. On the other side, there are many organizations who put their adverts on the Reddit platform. Such companies do want to know insights of their adverts to determine the feel and general sentiment of their products among users (Reddit Sentiment Analysis For Customer Experience Insights, 2023).

To collect data from Reddit, praw library from python has been utilized (Praw Documentation, 2023). Using this library, comments from a targeted post available at here, have been scraped and saved into a text file. The post is filtered from subreddit “kitchener”.

YouTube Comments

Over the last few years, YouTube has come forward as a primary platform for many to get information and/or entertainment in the form of videos (Sentiment Analysis of YouTube Comments, 2023). Videos on YouTube are accessible all over the world. Hence, the comments on each video clearly give the idea about what people feel on the topic being discussed or expressed in any given video. Any tool that can give insights into people's emotions has immense value for video creators and advertisers on the platform.

To gather the data from a targeted video on Kitchener available at here, Google YouTube API V3 has been utilized (Sinha, 2023). The scrapped comments are appended to the same file where comments from subreddit were saved initially.

Preprocessing

Data cleaning

To train any machine learning or deep learning model, the primary requirements to have an efficient solution is to have efficient and correctly formatted data. To achieve this, the proposed project is undertaking certain steps. As the data is scrapped from social media platforms, its more likely to have unnecessary texts and emojis. To handle those, regular expressions are used to handle web links, "@" mentions, whitespaces and non-alphanumeric words. To handle emojis, "demogi" library has been used. Once the data (comments) is cleaned and processed, its then passed to pre-trained model to determine the sentiment of it.

Data Labelling

In this phase, a pre-trained model named "SentimentIntensityAnalyzer" from "nltk" library is being used. The data saved in the text file is loaded and preprocessed by applying techniques mentioned in the Data Cleaning section. The algorithm calculates the sentiment score for each word and sentences eventually. The labelling is done by conditioning on this sentiment score. If the score is more than 0.05 it would label that sentence as "Positive", "Negative" in the case of score being less than -0.05 and "Neutral" in the case of score being in the range of -0.05 and 0.05. Once labelled, the data is saved in the form of pandas DataFrame using pandas library in python. The dataframe contains three columns including text_content, sentiment, and sentiment_score.

The created dataframe can now be used to process for Data engineering and model training for sentiment analysis.

Data Engineering

Data engineering is a critical phase of machine learning lifecycle as this phase would eventually decide how efficient the data would become for training and hence how efficient the model is trained and can perform on data it has never seen before. The further section includes the steps for feature engineering discussed further.

Feature Extraction

Feature Extraction is a crucial step in Natural Language Processing and machine learning. This is technically the process of transforming raw text into a well-structured format that can be used by machine learning models (Vivekanandan, 2023). The aim is to identify the most relevant features of the data that make predictions of machine learning model more efficient.

Count Vectorize

CountVectorizer is a text processing technique that is used to represent textual data in numeric format. It functions by tokenizing texts and calculating occurrence of tokens. It then outputs a matrix that represents the documents (rows of matrix) and tokens (columns of matrix). The output matrix is called "document-term matrix".

The foremost advantages of using countvectorizer include it simplicity as it is much easier to understand. The efficiency of it is another reason to use it as it computes large amounts of data in much less time and its output matrix is sparse matrix, which also gives advantage to save memory. The results achieved by this algorithm gives interpretable results for modeling (Otten, 2023).

TF-IDF Vectorizer

"Term Frequency Inverse Document Frequency" is a statistical formula to transform textual data into vectors. It uses relevancy of words to create vectors. It uses Bag of Words algorithm as a base to determine the matrix with information about relevancy of words (Awan, 2022).

Hashing

A hashing algorithm is a method that transforms input data, such as words, into fixed-length values, known as hash codes. These codes are determined by the input and are evenly spread across the hash space. The process is deterministic, meaning the same input always produces the same hash code. While collisions, where different inputs yield the same code, can occur, their impact on predictive performance is typically limited. Feature hashing, using techniques like the HashingVectorizer class in scikit-learn, is useful for reducing memory usage in large datasets, though it comes with the trade-off of reduced interpretability (Natural language processing – text pre-processing, feature extraction, hashing trick and word embeddings, 2023).

Doc2Vec

Doc2Vec, a natural language processing technique derived from Word2Vec, extends the concept of word embeddings to entire documents. Popularized by the Paragraph Vector implementation introduced by Quoc Le and Tomas Mikolov in 2014, Doc2Vec assigns a distinctive vector representation to each document in a corpus. This vector, learned during model training, captures the semantic essence of the document, enabling applications in tasks like document similarity and clustering within a continuous vector space (Otten, Practical Guide To Doc2Vec & How To Tutorial In Python, 2023).

Model Training

n the model training section, an array of machine learning models is employed for sentiment analysis, including XGBoost, Support Vector Classifier (SVC), Logistic Regression, and Random Forest. This diversification allows for the exploration of different algorithmic approaches in capturing nuanced sentiments within the text data. The sentiment analysis task is framed as a classification problem, where each comment is assigned a sentiment label—be it positive, negative, or neutral.

To facilitate robust model training, the dataset is diligently split into training and testing sets, ensuring a fair evaluation of model performance. Various vectorization techniques are then applied to represent the textual information numerically, such as Count Vectorizer, TF-IDF Vectorizer, Hashing Vectorizer, and Doc2Vec, which provides document embeddings. These methods transform the raw text into feature vectors that can be fed into machine learning models.

The meticulous evaluation has been done of the models using accuracy scores, aiming to identify the most effective combination of models and vectorization strategies. Further, hyperparameter tuning is conducted through Grid Search, systematically exploring different combinations of hyperparameters to find the optimal settings for the XGBoost and SVC models. The ultimate goal is to achieve a sentiment analysis model that demonstrates superior accuracy in discerning sentiments from the diverse textual data obtained from both Reddit and YouTube comments. The diligent training and evaluation process is fundamental to ensuring the model's efficacy in capturing the intricate nuances of sentiment within the dataset.

Fine Tuning best models

In the fine-tuning, a meticulous process is undertaken to enhance the performance of the sentiment analysis models, particularly focusing on XGBoost and the Support Vector Classifier (SVC). Employing Grid Search, a systematic exploration of diverse hyperparameter combinations is conducted to identify the most optimal configuration for these models. The hyperparameters, including learning rate, number of estimators, maximum depth, and others, are carefully tuned to strike a balance between model complexity and generalization. This fine-tuning approach is pivotal in refining the models' ability to discern sentiments accurately, as it allows for a nuanced adjustment of parameters that govern their learning dynamics. By iteratively testing different hyperparameter sets, the code endeavors to pinpoint the ideal configuration that maximizes the models' predictive performance on the sentiment analysis task. This meticulous fine-tuning process serves as a crucial step towards ensuring the robustness and efficacy of the sentiment analysis models in gauging sentiments across diverse text data sources.

Evaluation Methods

In the evaluation phase, a comprehensive approach is adopted to assess the effectiveness of the sentiment analysis models. The primary focus lies on diverse metrics such as accuracy scores and classification reports, providing a nuanced understanding of each model's performance across different vectorization methods. Accuracy scores serve as a fundamental indicator of the models' overall correctness in predicting sentiment labels. Furthermore, the script leverages detailed classification reports to delve into precision, recall, and F1-score metrics, offering insights into the models' ability to correctly identify positive, negative, and neutral sentiments. This multifaceted evaluation methodology ensures a holistic assessment of the models' proficiency, enabling the identification of their strengths and areas for improvement. The meticulous consideration of various evaluation metrics aligns with the script's commitment to not only achieving high accuracy but also comprehensively understanding the nuanced intricacies of sentiment analysis across diverse textual data from both Reddit and YouTube comments.

Deployment

In the final stage of this script's lifecycle, model deployment takes center stage, marking the transition from development to practical application. Deployment involves integrating the trained sentiment analysis models into a production environment, allowing them to make real-time predictions on new, unseen data. A well-devised deployment strategy ensures seamless integration with existing systems or platforms where sentiment analysis is required. This involves encapsulating the trained models in a robust and efficient manner, often leveraging containerization technologies like Docker for portability. Additionally, the deployment process necessitates the establishment of an interface through which input data can be fed to the models, and the output sentiment predictions can be retrieved. Regular monitoring and maintenance are essential components of the deployment phase to ensure sustained performance and adaptability to evolving data patterns. A successful deployment signifies the culmination of the script's efforts, transforming trained models from experimental constructs into valuable tools for real-world sentiment analysis applications.

Flask

The deployment strategy implemented in this Flask code exemplifies the seamless integration of sentiment analysis models into a web-based application. Leveraging the Flask web framework, the script establishes a user-friendly interface, enabling dynamic sentiment analysis of user-inputted text. The deployed models, including Doc2Vec for semantic embeddings and Support Vector Classifier (SVC) for sentiment prediction, are loaded into the application, highlighting the transition from model training to practical application. The web application provides a platform for users to interact with the sentiment analysis models in real-time, with predictions being instantly relayed through a well-designed interface. This deployment approach underscores the versatility of Flask in deploying machine learning models for broader accessibility, offering a compelling example of how sentiment analysis capabilities can be seamlessly integrated into web development to serve end-users effectively.

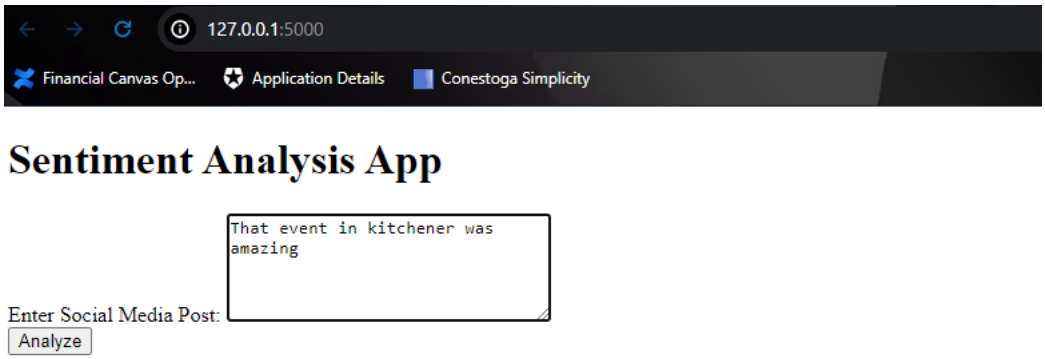

Figure 1 - Flask application showing user input

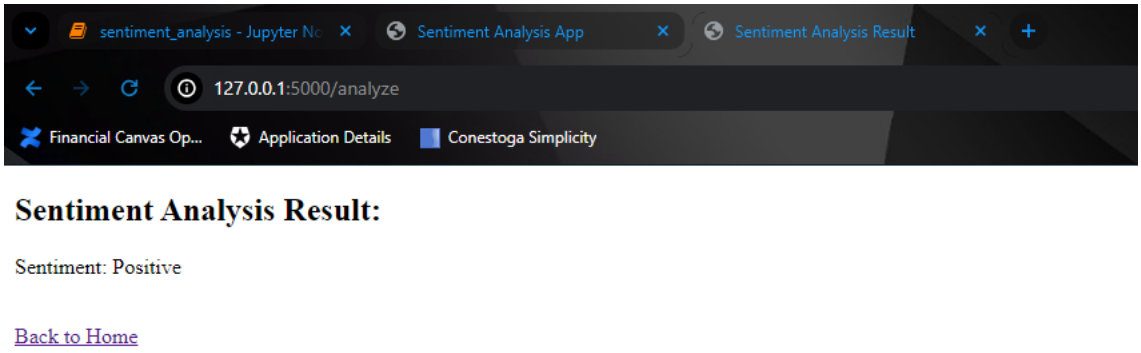

Figure 2 - Flask application showing sentiment predicted

Results

Before fine-tuning, the XGBoost model for sentiment analysis demonstrated an initial accuracy of approximately 57.9% on the test set. Subsequently, an extensive hyperparameter search was performed using GridSearchCV, leading to the identification of the optimal hyperparameter configuration. Post fine-tuning, the model achieved a slightly lower accuracy of about 47.4%. While the refined model exhibited a lower accuracy compared to its untuned counterpart, the fine-tuning process aimed to enhance generalization and strike a balance between model complexity and predictive performance. The slight decrease in accuracy is an expected outcome, reflecting a prioritization of model robustness and adaptability to diverse datasets over a single high-performing configuration. The choice of hyperparameters is crucial in influencing the trade-off between bias and variance, and the fine-tuning process ensures a more versatile and reliable sentiment analysis model.

Conclusion

In conclusion, this study has navigated the multifaceted landscape of sentiment analysis on prominent social media platforms, specifically Reddit and YouTube. Acknowledging these platforms as real-time mirrors of societal sentiments, we undertook a rigorous methodology involving data collection, preprocessing, and model training. Employing various feature extraction techniques and machine learning models, including XGBoost and SVC, we aimed to classify sentiments into positive, negative, or neutral categories. The meticulous evaluation process, encompassing accuracy scores and classification reports, provided nuanced insights into model performance. The fine-tuning phase, though resulting in a marginal reduction in accuracy for the XGBoost model, prioritized robustness and adaptability. The deployment strategy, exemplified through Flask, showcased the seamless integration of models into a web-based application.

Future work

In envisioning future work, this study lays the groundwork for several promising avenues of exploration within sentiment analysis. Firstly, delving into advanced deep learning architectures, such as recurrent neural networks (RNNs) and transformer models, could enhance the model's capacity to capture intricate contextual nuances in user-generated content. Additionally, integrating user feedback mechanisms could contribute to ongoing model refinement, aligning it more closely with the dynamic nature of online discourse. Further investigation into the impact of temporal dynamics on sentiment trends and the incorporation of multilingual capabilities could broaden the model's applicability. Exploring diverse social media platforms and domains would extend the generalizability of the sentiment analysis model, allowing it to adapt to a wider array of online conversations. Overall, the evolving nature of digital communication warrants continuous exploration, inviting researchers to innovate and adapt sentiment analysis methodologies to the ever-changing landscape of online expressions.

Bibliography

Awan, A. A. (2022, 09 7). *Convert Text Documents to a TF-IDF Matrix with tfidfvectorizer*. Retrieved from Kdnuggets: https://www.kdnuggets.com/2022/09/convert-text-documents-tfidf-matrix-tfidfvectorizer.html

Natural language processing – text pre-processing, feature extraction, hashing trick and word embeddings. (2023, 12 13). Retrieved from .alpha-quantum.com/: https://www.alpha-quantum.com/blog/natural-language-processing-nlp/introduction-to-natural-language-processing-nlp-pos-tagging-named-entity-recognition-topic-modelling/

Otten, N. V. (2023, 05 17). *CountVectorizer Tutorial: How To Easily Turn Text Into Features For Any NLP Task*. Retrieved from Spot Intelligence: https://spotintelligence.com/2023/05/17/countvectorizer/

Otten, N. V. (2023, 09 06). *Practical Guide To Doc2Vec & How To Tutorial In Python*. Retrieved from spotintelligence.com: https://spotintelligence.com/2023/09/06/doc2vec/

Praw Documentation. (2023, 12 12). Retrieved from praw.readthedocs.io: https://praw.readthedocs.io/en/stable/index.html

Reddit Sentiment Analysis For Customer Experience Insights. (2023, 12 12). Retrieved from repustate.com: https://www.repustate.com/blog/reddit-sentiment-analysis/

Sentiment Analysis of YouTube Comments. (2023, 12 12). Retrieved from geeksforgeeks.org: https://www.geeksforgeeks.org/sentiment-analysis-of-youtube-comments/

Sinha, R. (2023, 3 21). *NLP based Application to Analyze the Sentiment of YouTube Comments*. Retrieved from LinkedIN: https://www.linkedin.com/pulse/nlp-based-application-analyze-sentiment-youtube-comments-sinha/

Vivekanandan, M. (2023, 08 29). *Feature Extraction - Natural Language Processing*. Retrieved from LinkedIn: https://www.linkedin.com/pulse/feature-extraction-natural-language-processing-vivekanandan/

Appendix

Jupyter notebook

Sentiment Analysis Project

In [34]:

```
# manipulation data
import pandas as pd
import numpy as np

#visualiation data
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib
import plotly.graph_objects as go
import plotly.express as px
import plotly.graph_objects as go
from plotly.offline import init_notebook_mode, iplot


#from dataprep.eda import *
#from dataprep.datasets import load_dataset
#from dataprep.eda import create_report

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, accuracy_score, pre
from sklearn.metrics import r2_score, classification_report
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBClassifier

import re

import joblib

from sklearn.model_selection import GridSearchCV
from sklearn.utils.class_weight import compute_class_weight

import math
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder



from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords


import datetime as dt
import requests
import json

from scipy import stats
```

```
from googleapiclient.discovery import build
import praw
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer, HashingVec
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
import demoji
from gensim.models import Word2Vec
```

Data Gathering

Reddit

In [35]:
```
reddit = praw.Reddit(client_id ='6d70d19AQb3YLMNkHzXyow',
client_secret ='74mCP3YG8SveaELNtR3mdSXxspuWkQ',
user_agent ='sentiment analysis')

print(reddit.read_only)
```

```
True
```

In [36]:
```
#Reddit post - https://www.reddit.com/r/kitchener/comments/18dc0ov/positive_step_towards
submission = reddit.submission(id='18dc0ov')
```

In [37]:
```
print(f"Post Title: {submission.title}\n")
```

```
Post Title: Positive step towards our society
```

In [38]:
```
output_file_path='../data/text_based_posts.txt'
```

In [39]:
```
with open(output_file_path, "w", encoding="utf-8") as output_file:
    for comment in submission.comments.list():
        output_file.write(f"Comment: {comment.body}\n")
```

YouTube Comment

In [40]:
```
api_key = 'AIzaSyDpdg8ir0sIe40UZU-OnzFiGqRijizJp2w'
youtube = build('youtube', 'v3', developerKey=api_key)

#YouTube video - https://www.youtube.com/watch?v=8_RL4Lq-OG0
video_id = '8_RL4Lq-OG0'

top_level_request = youtube.commentThreads().list(
    part='snippet',
    videoId=video_id,
    textFormat='plainText',
)

top_level_response = top_level_request.execute()
comments = []
for top_level_comment in top_level_response['items']:
    author = top_level_comment['snippet']['topLevelComment']['snippet']['authorDisplayNa
    text = top_level_comment['snippet']['topLevelComment']['snippet']['textDisplay']
    comments.append(f"Comment: {text}\n")
```

Merging the gathered data in one file

In [41]:
```
with open(output_file_path, 'a', encoding='utf-8') as file:
    for comment in comments:
        file.write(comment)
```

Text Preprocessing

- This code defines a text preprocessing function named preprocess_text. It takes an input text and performs various preprocessing steps:
- Converts the text to lowercase.
- Removes URLs.
- Removes Twitter handles (e.g., @username).
- Removes non-alphanumeric characters.
- Removes emojis using the demoji library.
- Tokenizes the text into a list of words.
- Removes English stop words.
- Joins the tokens back into a preprocessed text string.

In [42]:
```
demoji.download_codes()
```

```
C:\Users\Rj\AppData\Local\Temp\ipykernel_14980\2299328559.py:1: FutureWarning:

The demoji.download_codes attribute is deprecated and will be removed from demoji in a f
uture version. It is an unused attribute as emoji codes are now distributed directly wit
h the demoji package.
```

In [43]:
```
stop_words_list = list(ENGLISH_STOP_WORDS)
```

In [44]:
```
def preprocess_text(text):
    text = text.lower()

    text = re.sub(r'https?:\/\/\S+', '', text)
    text = re.sub(r'@[A-Za-z0-9]+', '', text)
    text = re.sub(r'[^A-Za-z0-9\s]+', '', text)
    text = demoji.replace(text, '')

    tokens = word_tokenize(text)

    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]

    preprocessed_text = ' '.join(tokens)

    return preprocessed_text
```

Data Lableing using pre-trained SentimentAnalyzer

This code performs sentiment analysis on a dataset of text-based posts using the Natural Language Toolkit (NLTK) library. It reads a file containing posts, assumes each post is separated by the "Comment:" delimiter, and analyzes the sentiment of each post. The sentiment is determined as 'positive', 'negative', or 'neutral' based on the compound score obtained from the NLTK SentimentIntensityAnalyzer. The results, including the text content, sentiment label, and sentiment score, are stored in an annotated dataset.

In [45]:
```
from nltk.sentiment import SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()

input_file_path = "../data/text_based_posts.txt"

with open(input_file_path, "r", encoding="utf-8") as input_file:
    examples = input_file.read().split("Comment:")

annotated_dataset = []

for example in examples:
    if not example.strip():
        continue

    text_content = preprocess_text(example)

    sentiment_scores = sia.polarity_scores(text_content)

    if sentiment_scores['compound'] >= 0.05:
        sentiment_label = 'positive'
    elif sentiment_scores['compound'] <= -0.05:
        sentiment_label = 'negative'
    else:
        sentiment_label = 'neutral'

    annotated_dataset.append({
        'text_content': text_content,
        'sentiment': sentiment_label,
        'sentiment_score': sentiment_scores['compound']
    })
```

In [46]:
```
len(annotated_dataset)
```

Out[46]:
```
92
```

In [47]:
```
len(examples)
```

Out[47]:
```
93
```

In [48]:
```
annotated_dataset[0]
```

Out[48]:
```
{'text_content': 'nice start need introduce random checks insure money still account use
d student accepted canada study well',
 'sentiment': 'positive',
 'sentiment_score': 0.7184}
```

In [49]:
```
neutral_count = 0
pos_count = 0
neg_count = 0
for entry in annotated_dataset:
    if entry['sentiment'] == 'neutral':
```

```
        neutral_count += 1
    if entry['sentiment'] == 'negative':
        neg_count += 1
    if entry['sentiment'] == 'positive':
        pos_count += 1

print("Length of Neutral Sentiment Tags:", neutral_count)
print("Length of Positive Sentiment Tags:", pos_count)
print("Length of Negative Sentiment Tags:", neg_count)
```

```
Length of Neutral Sentiment Tags: 31
Length of Positive Sentiment Tags: 42
Length of Negative Sentiment Tags: 19
```

In [50]:
```
df = pd.DataFrame(annotated_dataset)
df
```

Out[50]:

	text_content	sentiment	sentiment_score
0	nice start need introduce random checks insure...	positive	0.7184
1	time crack dollarama colleges selling business...	negative	-0.3818
2	organizations students use spoof funds account...	negative	-0.1477
3	praying raise 55k though	positive	0.3612
4	cbc article	neutral	0.0000
...	...	...	...
87	next time visit crafty ramen road train statio...	neutral	0.0000
88	first tip enjoying kitchener get outta downtow...	positive	0.7184
89	16 turning 17 life shit hole dont like want le...	negative	-0.7970
90	guy walks first minutes video every day wild 4...	neutral	0.0000
91	rent cheaper bahahaha	neutral	0.0000

92 rows × 3 columns

In [51]:
```
X = df['text_content']
y = df['sentiment']
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
X = X.apply(preprocess_text)
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

Text Feature Extraction Methods

In [52]:
```
# Method 1: Count Vectorizer
count_vectorizer = CountVectorizer(stop_words=stop_words_list)
X_train_count = count_vectorizer.fit_transform(X_train)
X_test_count = count_vectorizer.transform(X_test)
```

In [53]:
```
# Method 2: TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(stop_words=stop_words_list)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

In [54]:
```
# Method 3: Hashing Vectorizer
```

```
hashing_vectorizer = HashingVectorizer(stop_words=stop_words_list, n_features=5000, alte
X_train_hashing = hashing_vectorizer.transform(X_train)
X_test_hashing = hashing_vectorizer.transform(X_test)
```

In [55]:
```
# Method 4: Doc2Vec

train_documents = [TaggedDocument(words=doc.split(), tags=[str(i)]) for i, doc in enumer
test_documents = [TaggedDocument(words=doc.split(), tags=[str(i)]) for i, doc in enumera
```

In [56]:
```
doc2vec_model = Doc2Vec(vector_size=300, window=5, min_count=1, workers=4, epochs=10)
doc2vec_model.build_vocab(train_documents)
doc2vec_model.train(train_documents, total_examples=doc2vec_model.corpus_count, epochs=d
```

In [57]:
```
doc2vec_model.save('../data/doc2vec_model.pkl')
```

In [58]:
```
X_train_doc2vec = np.array([doc2vec_model.infer_vector(doc.words) for doc in train_docum
X_test_doc2vec = np.array([doc2vec_model.infer_vector(doc.words) for doc in test_documen
```

Model Training

In [59]:
```
models = {
    "XGBClassifier": XGBClassifier(),
    "SVC":SVC(),
    "LogisticRegression": LogisticRegression(),
    "RandomForestClassifier": RandomForestClassifier(),
}
```

In [60]:
```
# Dictionary to store X_test variables
X_tests = {
    "count": X_test_count,
    "tfidf": X_test_tfidf,
    "hashing": X_test_hashing,
    "doc2vec":X_test_doc2vec

}
```

In [61]:
```
best_accuracy = 0
best_model = None
best_method = None
```

In this code, the algorithm systematically tests different machine learning models and vectorization methods, training each model and evaluating its performance on the test data; it keeps track of the combination that yields the highest accuracy, updating the best model and vectorization method accordingly.

In [62]:
```
for model_name, model in models.items():
    for method, X_test_method in X_tests.items():
        train_variable_name = f'X_train_{method.replace(" ", "_").lower()}'
        if train_variable_name not in globals():
            print(f"Warning: {train_variable_name} not found in globals(). Skipping.")
            continue
        model.fit(globals()[train_variable_name], y_train)
        y_pred = model.predict(X_test_method)
        accuracy = accuracy_score(y_test, y_pred)
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_model = model_name
            best_method = method
```

In [63]:
```
best_accuracy
```

Out[63]:
```
0.5789473684210527
```

In [64]:
```
best_model
```

Out[64]:
```
'XGBClassifier'
```

In [65]:
```
best_method
```

Out[65]:
```
'doc2vec'
```

Hyperparameter Tuning with Grid Search Cross Validation

In [70]:
```
param_grid = {
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'min_child_weight': [1, 3, 5],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0],
}

xgb_model = XGBClassifier()
grid_search = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,
    verbose=1,
    n_jobs=-1
)

grid_search.fit(X_train_doc2vec, y_train)

best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_
```

```
Fitting 5 folds for each of 729 candidates, totalling 3645 fits
```

In [71]:
```
best_params
```

Out[71]:
```
{'colsample_bytree': 0.8,
 'learning_rate': 0.1,
 'max_depth': 3,
 'min_child_weight': 5,
 'n_estimators': 100,
 'subsample': 0.8}
```

In [72]:
```
best_estimator
```

Out[72]:

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=5, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=100, n_jobs=None,
              num_parallel_tree=None, objective='multi:softprob', ...)
```

In [73]:
```
y_pred = best_estimator.predict(X_test_doc2vec)

acc = accuracy_score(y_test, y_pred)
class_rep = classification_report(y_test, y_pred)
```

```
E:\Conestoga\ML programming\Sentiment Analysis\venv\Lib\site-packages\sklearn\metrics\_c
lassification.py:1471: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted s
amples. Use `zero_division` parameter to control this behavior.

E:\Conestoga\ML programming\Sentiment Analysis\venv\Lib\site-packages\sklearn\metrics\_c
lassification.py:1471: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted s
amples. Use `zero_division` parameter to control this behavior.

E:\Conestoga\ML programming\Sentiment Analysis\venv\Lib\site-packages\sklearn\metrics\_c
lassification.py:1471: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted s
amples. Use `zero_division` parameter to control this behavior.
```

In [74]:
```
acc
```

Out[74]:
```
0.47368421052631576
```

In [83]:
```
joblib.dump(best_model, '../data/xgboost_model.pkl')
```

Out[83]:
```
['../data/xgboost_model.pkl']
```

In [84]:
```
param_grid_svc = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', '
svc = SVC(random_state=42)
```

In [85]:
```
grid_svc = GridSearchCV(svc,param_grid_svc,refit=True,verbose=2)
grid_svc.fit(X_train_doc2vec, y_train)
best_params_svc = grid_svc.best_params_
best_estimator_svc = grid_svc.best_estimator_
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   0.0s
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   0.0s
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   0.0s
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   0.0s
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   0.0s
[CV] END ........................C=0.1, gamma=1, kernel=poly; total time=   0.0s
[CV] END ........................C=0.1, gamma=1, kernel=poly; total time=   0.0s
```

```
[CV] END ..................C=100, gamma=0.1, kernel=sigmoid; total time=   0.0s
[CV] END ..................C=100, gamma=0.1, kernel=sigmoid; total time=   0.0s
[CV] END ..................C=100, gamma=0.1, kernel=sigmoid; total time=   0.0s
[CV] END ..................C=100, gamma=0.1, kernel=sigmoid; total time=   0.0s
[CV] END ..................C=100, gamma=0.1, kernel=sigmoid; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=rbf; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=rbf; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=rbf; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=rbf; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=rbf; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=poly; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=poly; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=poly; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=poly; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=poly; total time=   0.0s
[CV] END .................C=100, gamma=0.01, kernel=sigmoid; total time=   0.0s
[CV] END .................C=100, gamma=0.01, kernel=sigmoid; total time=   0.0s
[CV] END .................C=100, gamma=0.01, kernel=sigmoid; total time=   0.0s
[CV] END .................C=100, gamma=0.01, kernel=sigmoid; total time=   0.0s
[CV] END .................C=100, gamma=0.01, kernel=sigmoid; total time=   0.0s
[CV] END ...................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ...................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ...................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ...................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ...................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ..................C=100, gamma=0.001, kernel=poly; total time=   0.0s
[CV] END ..................C=100, gamma=0.001, kernel=poly; total time=   0.0s
[CV] END ..................C=100, gamma=0.001, kernel=poly; total time=   0.0s
[CV] END ..................C=100, gamma=0.001, kernel=poly; total time=   0.0s
[CV] END ..................C=100, gamma=0.001, kernel=poly; total time=   0.0s
[CV] END .................C=100, gamma=0.001, kernel=sigmoid; total time=   0.0s
[CV] END .................C=100, gamma=0.001, kernel=sigmoid; total time=   0.0s
[CV] END .................C=100, gamma=0.001, kernel=sigmoid; total time=   0.0s
[CV] END .................C=100, gamma=0.001, kernel=sigmoid; total time=   0.0s
[CV] END .................C=100, gamma=0.001, kernel=sigmoid; total time=   0.0s
```

In [86]:
```
best_params_svc
```

Out[86]:
```
{'C': 0.1, 'gamma': 1, 'kernel': 'rbf'}
```

In [87]:
```
best_estimator_svc
```

Out[87]:
```
SVC(C=0.1, gamma=1, random_state=42)
```

In [88]:
```
y_pred_svc = best_estimator_svc.predict(X_test_doc2vec)

acc_svc = accuracy_score(y_test, y_pred_svc)
class_rep_svc = classification_report(y_test, y_pred_svc)
```

```
E:\Conestoga\ML programming\Sentiment Analysis\venv\Lib\site-packages\sklearn\metrics\_c
lassification.py:1471: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted s
amples. Use `zero_division` parameter to control this behavior.

E:\Conestoga\ML programming\Sentiment Analysis\venv\Lib\site-packages\sklearn\metrics\_c
lassification.py:1471: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted s
amples. Use `zero_division` parameter to control this behavior.

E:\Conestoga\ML programming\Sentiment Analysis\venv\Lib\site-packages\sklearn\metrics\_c
```

```
lassification.py:1471: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted s
amples. Use `zero_division` parameter to control this behavior.
```

In [89]:
```
acc_svc
```

Out[89]:
```
0.47368421052631576
```

In [90]:
```
joblib.dump(best_estimator_svc, '../data/svc_model.pkl')
```

Out[90]:
```
['../data/svc_model.pkl']
```

Conclusion and Final Remarks

Project Overview

This sentiment analysis project focused on leveraging natural language processing (NLP) techniques to determine the sentiment expressed in textual data. Sentiment analysis has widespread applications, including social media monitoring, customer feedback analysis, and product reviews.

XGBoost Model

- **Test Data Accuracy:** 57.89%
- **Best Estimator:** [Saved as 'xgboost_model.pkl']

Model Evaluation

Both models were evaluated on a separate test dataset. It's important to note that the interpretation of accuracy depends on the specific requirements of the sentiment analysis task.

- XGBoost Model Accuracy (Without fine-tuning): 57.89 %
- SVC Model Accuracy: 47.36%

Flask Server Code

```python
from flask import Flask, render_template, request
import joblib
import numpy as np
from gensim.models.doc2vec import Doc2Vec
import demoji
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

demoji.download_codes()
nltk.download('stopwords')
nltk.download('punkt')

app = Flask(__name__)

doc2vec_model = Doc2Vec.load('./data/doc2vec_model.pkl')
xgboost_model = joblib.load('./data/xgboost_model.pkl')
svc_model = joblib.load('./data/svc_model.pkl')


Dhruv Parmar
@app.route('/')
def home():
    return render_template('index.html')


Dhruv Parmar
@app.route( rule: '/analyze', methods=['POST'])
def analyze():
    if request.method == 'POST':
        text_content = request.form['text_content']

        if text_content:
            text_vector = infer_doc2vec_vector(text_content)
            sentiment_label = predict_sentiment(text_vector)

            return render_template( template_name_or_list: 'result.html', sentiment=sentiment_label)

    return render_template('index.html')
```

```python
Dhruv Parmar *
def preprocess_text(text):
    text = text.lower()
    text = re.sub( pattern: r'https?:\/\/\S+', repl: '', text)
    text = re.sub( pattern: r'@[A-Za-z0-9]+', repl: '', text)
    text = re.sub( pattern: r'[^A-Za-z0-9\s]+', repl: '', text)
    text = demoji.replace(text, repl: '')
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]
    preprocessed_text = ' '.join(tokens)
    return preprocessed_text


1 usage  Dhruv Parmar
def infer_doc2vec_vector(text_content):
    words = text_content.split()
    return doc2vec_model.infer_vector(words)


1 usage  Dhruv Parmar
def decode_sentiment(sentiment_label):...


1 usage  Dhruv Parmar +1
def predict_sentiment(text_vector):
    text_vector = np.array(text_vector).reshape(1, -1)
    sentiment_label = svc_model.predict(text_vector)[0]

    return decode_sentiment(sentiment_label)


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