

A

Project Report On

**" Text Classification on**

**Emotions Data "**

**By – Sibananda Swain**

[**Email - swain18022004@gmail.com**](mailto:Email%20-%20swain18022004@gmail.com)

**Mentor- Sudheer Kumar Y**

Infosys Springboard Internship

April-June 2024

**ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to my mentor, Mr. Sudheer Kumar Y., for his invaluable guidance and support throughout this text classification project on emotion data. His expertise in the field of machine learning was instrumental in shaping the direction and improving the quality of this work. His insightful feedback and encouragement helped me navigate challenges and persevere during the research and development process.

I am grateful for Mr. Sudheer Kumar Y.'s patience in explaining complex concepts and his willingness to dedicate his time to address my questions. His mentorship has not only equipped me with valuable technical skills but also fostered a passion for exploring the potential of text classification in understanding human emotions.

Sibananda Swain

# TABLE OF CONTENTS

[CHAPTER 1 1-2](#_TOC_250018)

* 1. [ABSTRACT 1](#_TOC_250017)
  2. [INTRODUCTION 2](#_TOC_250016)

[CHAPTER 2 3](#_TOC_250015)

2.1 WORK DONE 3

CHAPTER 3 4-9

* 1. SOFTWARE ANALYSIS 4-6
  2. HARDWARE ANALYSIS 6-7
  3. [LIBRARIES REQUIRED 7-9](#_TOC_250014)

[CHAPTER 4 10-15](#_TOC_250013)

* 1. SPECIFICATION OF PROJECT 10-11
  2. [HIGH LEVEL DESIGN 12](#_TOC_250012)
  3. [LOW LEVEL DESIGN 13-15](#_TOC_250011)

[CHAPTER 5 15-25](#_TOC_250010)

[5.1 CODING 15-25](#_TOC_250009)

[CHAPTER 6 26](#_TOC_250008)-29

[6.1 TESTING 26](#_TOC_250007)

[CHAPTER 7 30-31](#_TOC_250006)

CONCLUSION 30

LIMITATIONS 31

[CHAPTER 8 32](#_TOC_250001)

[9.1 REFERENCE / BIBLIOGRAPHY 32](#_TOC_250000)

# 

# CHAPTER – 1

## **ABSTRACT**

This project explores the application of text classification techniques to identify emotions expressed within textual data. We leverage a dataset of text samples labeled with corresponding emotions (e.g., joy, sadness, anger). The objective is to develop a machine learning model capable of accurately classifying new, unseen text data into its appropriate emotional category.

The project investigates various text pre-processing techniques to clean and prepare the data for model training. This may involve removing stop words, stemming or lemmatization, and other normalization steps. Feature engineering techniques are then explored to extract meaningful representations of the textual data suitable for machine learning algorithms. These features might include word frequencies, TF-IDF scores, or word embeddings.

Several machine learning models are evaluated for their performance in emotion classification. This could include algorithms like Support Vector Machines (SVM), Naive Bayes, Random Forest, or XGBoost. The project compares the accuracy, precision, recall, and F1-score of each model to determine the most effective approach for this specific emotion classification task.

The project concludes by discussing the achieved performance of the chosen model. It analyses potential limitations and areas for improvement. Additionally, the project explores potential real-world applications of this technology, such as sentiment analysis in social media, customer service chatbots, or personalized content recommendations.

1

# INTRODUCTION

Human communication is a rich tapestry woven with not just words, but also emotions. These emotions, often conveyed subtly through tone, choice of words, and even punctuation, can hold immense value in various fields. However, extracting emotions from written text remains a challenging task. This project delves into the fascinating realm of **text classification** to unlock the hidden world of emotions within textual data.

Our focus is on building a robust machine learning model that can analyze text and accurately categorize it based on the emotions it expresses. Imagine being able to understand the sentiment behind a social media post, the frustration in a customer service email, or the joy in a product review – all through the power of text classification.

This project will explore various techniques to prepare the textual data for analysis. We will delve into the world of **pre-processing**, where we clean and refine the data, ensuring the model focuses on the most relevant aspects. Next, we will embark on **feature engineering**, a crucial step where we extract meaningful features from the text that will empower the machine learning model to effectively distinguish between different emotions.

Our journey then leads us to the exciting realm of **machine learning algorithms**. We will explore different models, each with its own strengths and weaknesses, to determine the one best suited for the task of emotion classification. By evaluating their performance and comparing metrics like accuracy and precision, we will identify the champion in this emotional intelligence competition.

This project goes beyond just building a model. We will analyses its performance, identify potential limitations, and explore avenues for further improvement. Ultimately, we will explore the exciting possibilities this technology unlocks. Imagine applications like sentiment analysis tools for social media, chatbots equipped with emotional intelligence for customer service, or content recommendation systems that cater to your emotional state.The ability to understand emotions within text data holds immense potential across various industries. By embarking on this project, we aim to contribute to this burgeoning field and unlock a deeper understanding of the emotions that lie beneath the surface of our written communication.

2

# CHAPTER – 2

**2.1 WORKS DONE IN THE RELATED AREA**

Text classification for emotion detection has become a hotbed of research, with numerous advancements shaping the field. Here's a glimpse into some key areas of past work:

* **Machine Learning Algorithms:** Studies have explored the effectiveness of various algorithms like Support Vector Machines (SVMs), Naive Bayes, and deep learning architectures like Recurrent Neural Networks (RNNs) for emotion classification. Each algorithm offers advantages; SVMs excel in high-dimensional data, Naive Bayes is efficient for simpler tasks, and RNNs can capture long-range dependencies within text, crucial for understanding emotions.
* **Feature Engineering Techniques:** Researchers have delved into various methods to extract meaningful features from text data. Techniques like TF-IDF (Term Frequency-Inverse Document Frequency) weigh the importance of words based on their frequency within a document and rarity across the dataset. Word embeddings, learned representations of words in a vector space, have also shown promise in capturing semantic relationships between words and emotions.
* **Lexicon-based Approaches:** These methods leverage pre-built dictionaries of words associated with specific emotions. They analyze the presence and frequency of these emotional keywords within the text for classification. While effective for basic emotions, they might struggle with nuanced expressions.
* **Multimodal Emotion Detection:** Recent advancements explore combining textual data with other modalities like facial expressions or voice intonation for a more comprehensive understanding of emotions. This opens doors for analyzing social media posts or video recordings where both text and visual cues contribute to emotional expression.

Building upon these past efforts, our project aims to explore the latest advancements in text classification and feature engineering to develop a robust model for emotion detection. We'll delve into the performance of various algorithms and identify the most effective approach for our specific dataset and chosen emotions.

3

# CHAPTER – 3 SYSTEM ANALYSIS

## **SOFTWARE REQUIREMENT**

This report outlines the system requirements for a project that aims to build a text classification system for emotion detection. These requirements are categorized into functional and non-functional requirements.

**1. Functional Requirements**

* **1.1 Data Input:**
  + The system shall accept raw text data as input.
  + The system should be able to handle various text formats (e.g., plain text, HTML snippets).
  + The system should allow for batch processing of text data (e.g., uploading a CSV file).
* **1.2 Data Pre-processing:**
  + The system shall clean and prepare the text data for analysis.
  + This includes removing irrelevant characters, stop words, and formatting inconsistencies.
  + The system should offer options for stemming or lemmatization to normalize words.
* **1.3 Feature Engineering:**
  + The system shall extract meaningful features from the pre-processed text.
  + This may involve techniques like TF-IDF to capture word importance and word embedding models to represent word semantics.
  + The system should allow for customization of feature engineering techniques.
* **1.4 Machine Learning Model:**
  + The system shall train and utilize a machine learning model for emotion classification.

4

* + The system should support various algorithms like SVM, Naive Bayes, or XGBoost.
  + The system should allow for model training on a labeled dataset with text samples and corresponding emotion labels.
* **1.5 Classification:**
  + The system shall classify emotions within new, unseen text data.
  + The system should analyze the features of new text, apply the trained model, and output the predicted emotion category.
  + The system may optionally provide a probability distribution across multiple emotions.
* **1.6 Output:**
  + The system shall provide the classified emotion(s) for each text input.
  + The output could be displayed on a user interface or exported in a format like CSV.

**2. Non-Functional Requirements**

* **2.1 Performance:**
  + The system should be able to process text data efficiently, with acceptable response times for both training and classification tasks.
* **2.2 Scalability:**
  + The system should be scalable to handle large volumes of text data without significant performance degradation.
* **2.3 Accuracy:**
  + The system should achieve a high level of accuracy in classifying emotions within the text data. This can be measured using metrics like accuracy, precision, recall, and F1-score.
* **2.4 Interpretability (Optional):**
  + Depending on the chosen machine learning model, the system should ideally provide some level of interpretability to understand how classifications are made.

5

* **2.5 User Interface (Optional):**
  + If user interaction is desired, the system should provide a user-friendly interface (web or API) for submitting text data and viewing classification results.
* **2.6 Security:**
  + The system should implement appropriate security measures to protect sensitive data (e.g., user-uploaded text) if applicable.

**Additional Considerations:**

* The specific requirements might need to be refined based on the chosen technologies and project scope.
* Error handling and logging mechanisms should be implemented to capture and address any issues during system operation.
* The system should be documented thoroughly to facilitate future maintenance and updates.

This system analysis report provides a comprehensive overview of the functional and non-functional requirements for the text classification system for emotion detection. By adhering to these requirements, we can develop a robust and effective system capable of unlocking the emotional intelligence hidden within textual data.

## **HARDWARE REQUIREMENT**

This report outlines the hardware requirements for a text classification system for emotion detection. The specific needs will depend on the scale of your project and the chosen algorithms. Here's a general guideline:

**Minimum Requirements:**

* **Processor:** A recent multi-core processor (e.g., Intel Core i5 or AMD Ryzen 5) is recommended. More cores will improve processing speed, especially during training.
* **Memory (RAM):** At least 8GB of RAM is recommended. If you plan to work with very large datasets or complex models, consider 16GB or more.

6

* **Storage:** Sufficient storage space to accommodate your dataset and the trained model. While the text data itself might not be massive, the model files can grow depending on the chosen algorithm. Allocate at least 250GB of storage (SSD for faster access is preferred).
* **Operating System:** A recent version of Windows (10 or 11), macOS, or Linux is suitable. Choose an OS you're comfortable with and that has good compatibility with your chosen machine learning libraries (e.g., TensorFlow, PyTorch).

**Additional Considerations:**

* **Graphics Processing Unit (GPU):** While not strictly mandatory, a dedicated GPU can significantly accelerate training times for complex models like deep neural networks. If processing speed is a major concern, consider a GPU with good machine learning capabilities (e.g., NVIDIA GeForce RTX series).
* **Internet Connection:** An internet connection is necessary for downloading machine learning libraries, datasets (if not locally available), and accessing online resources during development.

## **LIBRARIES REQUIRED**

This project utilizes various Python libraries to achieve text classification for emotion detection. Here's a breakdown of the key libraries and their functionalities:

**1. Data Analysis Libraries:**

* **pandas (pd):** This library is the foundation for data manipulation and analysis. It excels at handling DataFrames, tabular data structures used to store and organize our text data and corresponding emotion labels.
* **NumPy (np):** NumPy provides powerful numerical computing capabilities. It's crucial for vectorizing text data (converting text into numerical representations) which is essential for machine learning algorithms.

7

* **seaborn (sn):** Built on top of Matplotlib, seaborn offers a high-level interface for creating informative and aesthetically pleasing statistical visualizations. We leverage seaborn to explore the distribution of emotions within our dataset.

**2. Text Preprocessing Libraries:**

* **NLTK (natural language toolkit):** While not explicitly imported in this example, NLTK is a versatile library for natural language processing tasks. It can be used for tasks like tokenization (splitting text into words) or part-of-speech tagging, depending on your specific preprocessing needs.
* **wordcloud :**This library can be used to generate visually appealing word clouds, highlighting the most frequent words within the dataset. While not directly used for model building, it can be a valuable tool for data exploration.

**3. Data Preprocessing and Feature Engineering Libraries:**

* **scikit-learn (sklearn):** This comprehensive machine learning library offers a wide range of tools relevant to our project.
  + **TfidfVectorizer:** This is a more sophisticated technique for feature extraction. TF-IDF (Term Frequency-Inverse Document Frequency) considers both the word's frequency within a document and its rarity across the entire corpus, creating a more informative feature representation.
  + **TfidfTransformer:** This transformer is used in conjunction with TfidfVectorizer to learn the TF-IDF weights during model training.
* **Pipeline:** This scikit-learn utility allows us to create a pipeline that combines multiple data processing steps (e.g., text preprocessing and feature extraction) into a single unit, streamlining the process.

**4. Model Selection and Evaluation Libraries:**

* **scikit-learn:** Scikit-learn provides various classification algorithms for our task:
  + **Logistic Regression:** A linear model suitable for classification tasks, often used as a baseline model.

**8**

* + **DecisionTreeClassifier:** This model builds a tree-like structure to classify data based on a series of decision rules.
  + **SVC (Support Vector Classifier):** This algorithm projects data points into high-dimensional space and finds a hyperplane that best separates the classes.
  + **KNeighborsClassifier:** This model classifies data points based on the majority vote of their nearest neighbors.
* **GridSearchCV (not used in this example) & RandomizedSearchCV (not used in this example):** These tools from scikit-learn are helpful for hyperparameter tuning. They systematically evaluate different hyperparameter combinations for a chosen model and select the one that performs best on a validation set.
* **XGBoost:** XGBoost is a powerful gradient boosting framework known for its efficiency and accuracy in various machine learning tasks, including text classification.
* **LightGBM:** Similar to XGBoost, LightGBM is another gradient boosting library known for its speed and performance.
* **classification\_report, confusion\_matrix, accuracy\_score, f1\_score (from scikit-learn):** These metrics are used to evaluate the performance of our trained model on the test set. They provide insights into the model's ability to correctly classify emotions and identify potential areas for improvement.

9

# CHAPTER -4

* 1. **SPECIFICATIONS OF THIS PROJECT**

This project aims to develop a machine learning model capable of classifying emotions within textual data. This technology has the potential to revolutionize various industries by unlocking the power of emotions hidden within written communication.

**4.1.1 Business Cases**

Here are some compelling business cases that highlight the significance of this project:

* **Social Media Sentiment Analysis:** Businesses can leverage this technology to analyze customer sentiment expressed in social media posts, tweets, or online reviews. This allows them to gain valuable insights into customer satisfaction, identify areas for improvement, and tailor their marketing strategies accordingly. Imagine being able to understand the emotional undercurrent of social media conversations surrounding your brand and proactively address any concerns or negative sentiment.
* **Enhanced Customer Service Chatbots:** Chatbots are increasingly used for customer service interactions. By integrating our emotion classification model, chatbots can become more empathetic and responsive. They can tailor their responses based on the customer's emotional state, de-escalate situations where frustration is detected, and provide a more personalized customer service experience.
* **Targeted Content Recommendations:** Imagine a content recommendation system that goes beyond just past browsing history or purchase behavior. By understanding the emotional tone of a user's search queries or social media interactions, the system can recommend content that resonates with their current emotional state. This can lead to increased user engagement and satisfaction.
* **Market Research and Brand Perception Analysis:** Businesses can utilize this technology to analyze customer reviews, product feedback, or social media commentary to understand brand perception from an emotional standpoint.

10

* Identifying emotions associated with the brand can help companies improve their marketing strategies and product offerings to evoke the desired emotional response from their target audience.
* **Mental Health Support Systems:** Text-based communication platforms can be used to provide mental health support. By integrating emotion classification, these platforms can identify users expressing negative emotions and offer appropriate resources or connect them with human support services.

**4.1.2 Project Scope**

This project focuses on building a text classification model for a specific set of emotions (e.g., joy, sadness, anger). We will explore various machine learning algorithms and feature engineering techniques to achieve optimal performance. The project will include:

* Data collection and pre-processing of text data labeled with corresponding emotions.
* Training and evaluation of machine learning models for emotion classification.
* Analysis of model performance metrics and identification of potential areas for improvement.
* Documentation of the developed system and its functionalities.

**4.1.3 Future Enhancements**

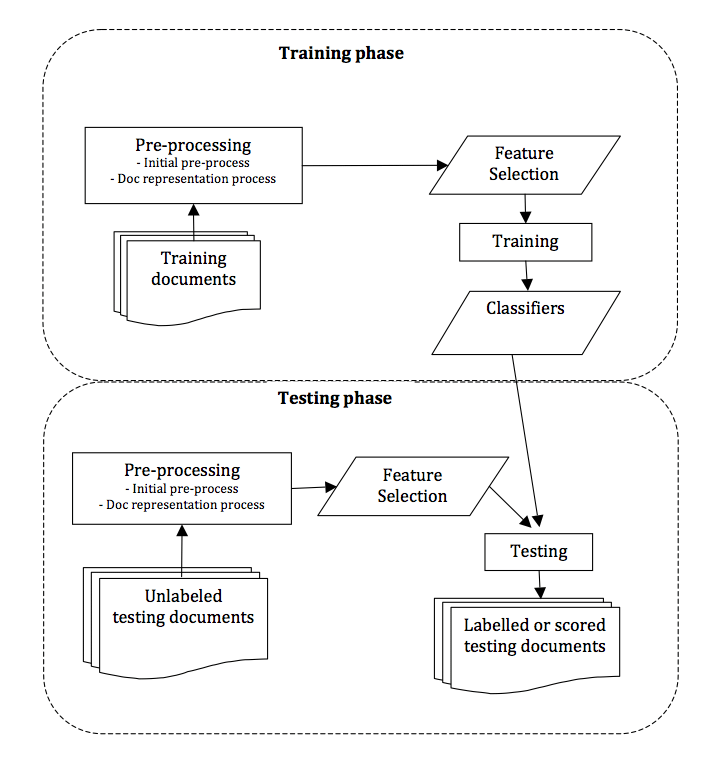
This project serves as a foundation for further exploration. Future enhancements may involve:

* Expanding the emotion classification repertoire to include a wider range of emotions.
* Exploring deep learning architectures for potentially improved accuracy.
* Integrating the model with real-world applications like social media analysis tools or chatbots.

By successfully completing this project, we take a significant step towards unlocking the power of emotions within text data. This technology holds immense potential for various industries, fostering deeper customer understanding, enhancing user experiences, and creating a more emotionally intelligent future.

11

## **HIGH LEVEL DESIGN**



12

## **LOW LEVEL DESIGN**

This report details the low-level design of a text classification system for emotion detection. It outlines the system's modules, their functionalities, and interactions.

**1. System Modules:**

The system will consist of the following key modules:

* **Data Preprocessing Module:**
  + Responsibilities:
    - Reads raw text data from a specified source (e.g., CSV file).
    - Performs text cleaning tasks like:
      * Lowercasing text.
      * Removing punctuation and special characters.
      * Removing stop words (common words with little meaning).
      * Applying stemming/lemmatization (reducing words to their base form).
  + Outputs:
    - Cleaned and preprocessed text data.
* **Feature Engineering Module:**
  + Responsibilities:
    - Extracts meaningful features from the preprocessed text.
    - Potential techniques:
      * **TF-IDF Vectorizer:** Calculates the importance of words based on their frequency within a document and rarity across the corpus.
      * **Word Embeddings:** (Optional) Creates numerical representations of words capturing semantic relationships.
  + Outputs:
    - Feature vectors representing the preprocessed text data.
* **Model Training Module:**
  + Responsibilities:
    - Initializes a chosen machine learning model (e.g., SVM, Random Forest, XGBoost).
    - Splits the preprocessed data and features into training and testing sets.
    - Trains the model on the training set, allowing it to learn the relationship between features and emotion labels.
  + Outputs:
    - A trained machine learning model for emotion classification.

13

* **Emotion Classification Module:**
  + Responsibilities:
    - Accepts new, unseen text data.
    - Applies the pre-processing steps (same as Data Preprocessing Module).
    - Extracts features from the preprocessed text (same as Feature Engineering Module).
    - Utilizes the trained model to classify the new text data and predict its corresponding emotion.
  + Outputs:
    - The predicted emotion category for the new text data.
* **Evaluation Module (Optional):**
  + Responsibilities:
    - Evaluates the performance of the trained model on the testing set.
    - Calculates metrics like accuracy, precision, recall, and F1-score.
  + Outputs:
    - Performance metrics and insights into model effectiveness.

**2. Module Interactions:**

1. **Data Preprocessing Module** reads raw data and sends cleaned text data to the **Feature Engineering Module**.
2. **Feature Engineering Module** extracts features from the cleaned text and sends feature vectors to the **Model Training Module**.
3. **Model Training Module** trains the model on the feature vectors and emotion labels, storing the trained model.
4. **Emotion Classification Module** receives new text data, preprocesses it, extracts features, and uses the trained model to predict the emotion. The predicted emotion is the system's output.
5. **Evaluation Module (Optional)** receives the testing set data and the trained model, evaluates its performance, and provides feedback.

14

**3. Data Flow:**

The system follows a linear data flow. Raw text data enters the system, undergoes cleaning and feature extraction, and is used to train the model. The trained model can then be used to classify emotions within new, unseen text data.

**4. Low-Level Design Considerations:**

* **Error Handling:** Implement mechanisms to handle potential errors during data loading, pre-processing, feature extraction, and model training.
* **Modularity:** Design modular components with well-defined interfaces for easier maintenance and future enhancements.
* **Efficiency:** Optimize algorithms for efficiency, especially for feature extraction and classification tasks.
* **Scalability:** Consider designing the system to handle larger datasets efficiently in the future.

**5. Conclusion:**

This low-level design provides a comprehensive overview of the system's modules, their interactions, and data flow. By following this design, we can develop a robust system for text classification and emotion detection. Remember to adapt and refine this design based on the chosen algorithms, data format, and specific project requirements.

15

# CHAPTER -5

## **5.1 CODING**

# # Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sn

import plotly.express as px

import nltk

# necessary imports  for text preprocessing

from wordcloud import WordCloud,STOPWORDS

from nltk.stem import WordNetLemmatizer

from nltk.tokenize import word\_tokenize

import re

import string

import unicodedata

# For spliting data sets  and evalution reports imports

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics import classification\_report,confusion\_matrix,accuracy\_score,f1\_score

from sklearn.model\_selection import train\_test\_split

from collections import Counter

# for text representation using feature engginering

from sklearn.pipeline import Pipeline

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.feature\_extraction.text import TfidfTransformer

# Different Classifier imports

# from scikit-learn

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import GridSearchCV,RandomizedSearchCV

# transformer classifiers

from xgboost import XGBClassifier

from lightgbm import LGBMClassifier

from string import punctuation

from nltk import pos\_tag

from nltk import pos\_tag

from nltk.corpus import wordnet

import re

import warnings

# fetching  the dataset and labels

emotion\_df = pd.read\_csv(r"C:\Users\siban\Text Ananlysis\Emotions\_training.csv",encoding='latin-1')

emotion\_labels = {0:'sadness', 1:'joy', 2:'love', 3:'anger', 4:'fear', 5:'surprise'}

emotion\_df.info()

emotion\_df['label'].unique()

emotion\_df.isnull().sum()

plt.figure(figsize=(12,6))

emotion\_df.label.value\_counts().plot(kind='bar');

## Data preprocessing on text

wordcloud = WordCloud(width=700, height=500, background\_color="black")

wordcloud.generate(" ".join(emotion\_df['text']))

plt.figure(figsize=(10, 8))

plt.imshow(wordcloud)

plt.axis("off")

plt.title("Wordcloud of Various emotionin text visuals ")

plt.show()

# converting to lower case

emotion\_df['text']=emotion\_df['text'].str.lower()

# Using stop words

from nltk.corpus import stopwords

stop\_words = stopwords.words('english')

stop\_words.remove("not")

stop\_words=set(stop\_words)

# Removal existing html tags

def remove\_html\_tags(text):

    pattern=re.compile('<.\*?>')

    return pattern.sub(r'',text)

# Removal punctuations

def remove\_special\_character(content):

    return re.sub(r'[^a-zA-Z ]',' ', content )

#Removal of URLs

def remove\_url(content):

    return re.sub(r'http\S+', '', content)

# Remove of stop words from content part of text column

def remove\_stopwords(content):

    clean\_data = []

    for i in content.split():

        if i.strip().lower() not in stop\_words and i.strip().lower().isalpha():

            clean\_data.append(i.strip().lower())

    return " ".join(clean\_data)

# For the expansions of  some certain contract usage in text column

def contraction\_expansion(content):

    content = re.sub(r"won\'t", "would not", content)

    content = re.sub(r"can\'t", "can not", content)

    content = re.sub(r"don\'t", "do not", content)

    content = re.sub(r"shouldn\'t", "should not", content)

    content = re.sub(r"needn\'t", "need not", content)

    content = re.sub(r"hasn\'t", "has not", content)

    content = re.sub(r"haven\'t", "have not", content)

    content = re.sub(r"weren\'t", "were not", content)

    content = re.sub(r"mightn\'t", "might not", content)

    content = re.sub(r"didn\'t", "did not", content)

    content = re.sub(r"n\'t", " not", content)

    return content

def data\_cleaning(content):

    if not pd.isna(content):

        content = remove\_html\_tags(content)

        content = remove\_url(content)

        content = remove\_special\_character(content)

        content = contraction\_expansion(content)

        content = remove\_stopwords(content)

    return content

emotion\_df['text']=emotion\_df['text'].apply(data\_cleaning)

# using Lemmatization on cleaned data

# Lemmatization from nltk package

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

emotion\_df['text'] = emotion\_df['text'].apply(lambda x: ' '.join([lemmatizer.lemmatize(word) for word in x.split()]))

# checking after implementing lemmatization

emotion\_df['text'][700]

#visualization of content through world cloud

wordcloud = WordCloud(width=800, height=600, background\_color="black")

wordcloud.generate(" ".join(emotion\_df['text']))

plt.figure(figsize=(10, 8))

plt.imshow(wordcloud)

plt.axis("off")

plt.title("Wordcloud of emotion text After Cleaning and processing of the text ")

plt.show()

emotion\_df

class\_counts = emotion\_df['label'].value\_counts()

majority\_class = class\_counts.idxmax()

# Function for oversample a class

def oversample\_class(data\_subset, target\_count):

    oversampled\_data = data\_subset.sample(replace=True, n=target\_count - len(data\_subset))

    return pd.concat([data\_subset, oversampled\_data])

# Oversample minority classes

oversampled\_data = []

for label, count in class\_counts.items():

    if label != majority\_class:

        data\_subset = emotion\_df[emotion\_df['label'] == label]

        oversampled\_subset = oversample\_class(data\_subset, class\_counts[majority\_class])

        oversampled\_data.append(oversampled\_subset)

    else:

        oversampled\_data.append(emotion\_df[emotion\_df['label'] == label])

# Combine the oversampled data

oversampled\_df = pd.concat(oversampled\_data)

# Extracting text and labels from oversampled data

oversampled\_text = oversampled\_df['text']

oversampled\_labels = oversampled\_df['label']

#visualization of the oversampled data by bar chart

oversampled\_labels.value\_counts().plot(kind="bar")

# Division of data sets

y = emotion\_df['label']

X  = emotion\_df['text']

# text Represntation by using Vectorization suggested by the mentor

vectorizer = TfidfVectorizer(max\_features=2000)

features\_oversampled = vectorizer.fit\_transform(oversampled\_text)

vectorizer = TfidfVectorizer(max\_features=2000)

X\_imb\_vec = vectorizer.fit\_transform(X)

X\_train\_imb, X\_test\_imb, y\_train\_imb, y\_test\_imb = train\_test\_split(X\_imb\_vec, y, test\_size=0.3, random\_state=42,stratify=y)

# Handling imbalance by using the SMOTE

from imblearn.over\_sampling import SMOTE

smote = SMOTE()

X\_im, y\_im = smote.fit\_resample(X\_imb\_vec, y)

num\_classes = len(set(y\_im))

X\_train\_im, X\_test\_im, y\_train\_im, y\_test\_im = train\_test\_split(X\_im, y\_im, test\_size=0.3, random\_state=42,stratify=y\_im)

classifiers\_for\_imb = {

    'DecisionTree': DecisionTreeClassifier(random\_state=42),

    'XGBoost': XGBClassifier(objective='multi:softprob', random\_state=42),

    'SVM': SVC(random\_state=42),

    'KNN': KNeighborsClassifier(),

    'LightGBM': LGBMClassifier(objective='multiclass', num\_class=num\_classes, random\_state=42),

}

param\_grid\_for\_imb = {

    'DecisionTree': {'max\_depth': [3, 5, 8, 10, 20, 30]},

    'XGBoost': {'n\_estimators': [50, 100, 200], 'learning\_rate': [0.01, 0.1, 1]},

    'SVM': {'C': [0.1, 1, 10], 'gamma': [0.001, 0.01, 0.1], 'kernel': ['rbf', 'linear']},

    'KNN': {'n\_neighbors': [3, 5, 7, 9]},

    'LightGBM': {'learning\_rate': [0.01, 0.1, 1]},

scores\_imb = []  # holds the each scoring of classifier at iterations

best\_estimators\_imb = {}  # set of all the best selected classifiers with best param

for model\_name, model in classifiers\_for\_imb.items():

    clf = GridSearchCV(model, param\_grid\_for\_imb[model\_name], scoring='accuracy', cv=5)

    clf.fit(X\_train\_imb, y\_train\_imb)

    scores\_imb.append({

        'model': model\_name,

        'best\_score': clf.best\_score\_,

        'best\_params': clf.best\_params\_

    })

    best\_estimators\_imb[model\_name] = clf.best\_estimator\_

# Finding the best model and its accuracy

best\_model\_name = max(scores\_imb, key=lambda x: x['best\_score'])['model']

best\_model\_score = max(scores\_imb, key=lambda x: x['best\_score'])['best\_score']

print(f"Best Model: {best\_model\_name} with Score: {best\_model\_score}")

# We can now use the best\_estimators\_imb[best\_model\_name] for prediction on your test data

y\_pred = best\_estimators\_imb[best\_model\_name].predict(X\_test\_imb)

test\_accuracy = accuracy\_score(y\_test\_imb, y\_pred)

print(f"Test Accuracy of Best Model: {test\_accuracy}")

# Comparison of all the models used .

output\_df\_imb=pd.DataFrame(scores\_imb,columns=['model','best\_score','best\_params'])

output\_df\_imb.sort\_values(by="best\_score",ascending=False)

# Function to visualize confusion matrix for a model

def plot\_confusion\_matrix(model\_name, clf, X, y, emotion\_labels, colormap='Blues'):

    cm = confusion\_matrix(y, clf.predict(X))

    plt.figure(figsize=(8, 6))  # Increased figure size for better readability

    ax = plt.axes()

    ax.set\_title(f"Confusion Matrix for {model\_name}")

    ax.set\_xlabel('Predicted (' + ', '.join(emotion\_labels.values()) + ')')

    ax.set\_ylabel('True (' + ', '.join(emotion\_labels.values()) + ')')

    # Use seaborn for heatmap visualization with adjusted colormap and font size

    sn.heatmap(cm, annot=True, fmt='g', cmap=colormap, annot\_kws={'fontsize': 12})

    plt.xticks(range(len(emotion\_labels)), emotion\_labels.values())

    plt.yticks(range(len(emotion\_labels)), emotion\_labels.values())

    plt.grid(False)  # Remove grid lines for cleaner visualization

plt.tight\_layout()  # Adjust layout to prevent overlapping elements

    plt.show()

    print(cm)

# Assuming you have best\_model\_name set to 'XGBoost' (modify if different)

best\_model = best\_estimators\_imb[best\_model\_name]

# Confusion matrix for test data

plot\_confusion\_matrix(best\_model\_name, best\_model, X\_test\_imb, y\_test\_imb, emotion\_labels)

# Confusion matrix for training data

plot\_confusion\_matrix(best\_model\_name, best\_model, X\_train\_imb, y\_train\_imb, emotion\_labels)

### Reapplying some methods to get the validation report of the best model .

from imblearn.over\_sampling import SMOTE

from sklearn.model\_selection import train\_test\_split

# Assuming your emotion dataset is stored in pandas DataFrames

emotion\_df\_text = emotion\_df['text']  # Text data

emotion\_df\_label = emotion\_df['label']  # Labels

# Perform text vectorization (if not already done)

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max\_features=2000)  # Adjust max\_features as needed

X\_vec = vectorizer.fit\_transform(emotion\_df\_text)

# Apply SMOTE for oversampling (assuming imbalanced classes)

smote = SMOTE(random\_state=42)

X\_im, y\_im = smote.fit\_resample(X\_vec, emotion\_df\_label)  # Oversample features and labels

# Split the oversampled data into training, validation, and test sets

X\_train\_val, X\_test, y\_train\_val, y\_test = train\_test\_split(X\_im, y\_im, test\_size=0.2, random\_state=42)

# Further split training data into training and validation sets (optional)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train\_val, y\_train\_val, test\_size=0.2, random\_state=42)

#### after the sorting of the data set into validation data set for the report generation .

model = best\_estimators\_imb["XGBoost"]

y\_pred\_val = model.predict(X\_val)  # Predict using validation data features

# Calculate validation accuracy

validation\_accuracy = accuracy\_score(y\_val, y\_pred\_val)

# Generate a more comprehensive validation report

validation\_report = classification\_report(y\_val, y\_pred\_val)

print(f"Validation Accuracy of Best Model: {validation\_accuracy}")

# Print classification report (optional)

print(validation\_report)

# classification report of XGBoost Classifier the best model after evaluation.

clf\_report=classification\_report(y\_test\_imb,best\_estimators\_imb["XGBoost"].predict(X\_test\_imb))

print(clf\_report)

**### For Balanced Dataset**

classifiers = {

    'DecisionTree': DecisionTreeClassifier(random\_state=42),

    'SVM': SVC(class\_weight='balanced', random\_state=42),

    'KNN': KNeighborsClassifier(n\_neighbors=5),

    'LightGBM': LGBMClassifier(objective='multiclass', num\_class=len(class\_counts), random\_state=42),

    'LogisticRegression': LogisticRegression(solver='liblinear', multi\_class='auto', random\_state=42),

    'XGBoost': XGBClassifier(objective='multi:softprob', random\_state=42)

}

param\_grid = {

    'DecisionTree': {'max\_depth': [3, 5, 8,10,20,30]},

    'SVM': {'C': [0.1, 1, 10]},

    'KNN': {'n\_neighbors': [3, 5, 7]},

    'LightGBM': {'learning\_rate': [0.01, 0.1, 1]},

    'LogisticRegression': {'C': [0.01, 0.1, 1, 10]},

    'XGBoost': {'n\_estimators': [50, 100, 200], 'learning\_rate': [0.01, 0.1, 1]}

}

scores = []  # List to store model evaluation results

best\_estimators = {}  # Dictionary to store best models for each classifier

for model\_name, model in classifiers.items():

  # Create a GridSearchCV object for each classifier with its hyperparameter grid

  clf = GridSearchCV(model, param\_grid[model\_name], scoring='accuracy', cv=5)

  # Fit the GridSearchCV object to the training and validation data (assuming balanced data)

  clf.fit(X\_train\_val, y\_train\_val)

  # Store evaluation results

  scores.append({

      'model': model\_name,

      'best\_score': clf.best\_score\_,

      'best\_params': clf.best\_params\_

  })

  # Store the best model for each classifier

  best\_estimators[model\_name] = clf.best\_estimator\_

# Print evaluation results

print("Model Selection Results:")

for score in scores:

  print(f"\tModel: {score['model']}, Best Score: {score['best\_score']}, Best Parameters: {score['best\_params']}")

# Access the best model (assuming you want the classifier with the highest score)

best\_model\_name = max(scores, key=lambda x: x['best\_score'])['model']

best\_model = best\_estimators[best\_model\_name]

print(f"\nBest Model: {best\_model\_name}")

output\_df=pd.DataFrame(scores,columns=['model','best\_score','best\_params'])

output\_df.sort\_values(by="best\_score",ascending=False)

best\_model = best\_estimators[best\_model\_name]

# Confusion matrix for test data

plot\_confusion\_matrix(best\_model\_name, best\_model, X\_test\_imb, y\_test\_imb, emotion\_labels)

# Confusion matrix for training data

plot\_confusion\_matrix(best\_model\_name, best\_model, X\_train\_imb, y\_train\_imb, emotion\_labels)

model = best\_estimators["SVM"]

y\_pred\_val = model.predict(X\_val)  # Predict using validation data features

# Calculate validation accuracy

validation\_accuracy = accuracy\_score(y\_val, y\_pred\_val)

# Generate a more comprehensive validation report

validation\_report = classification\_report(y\_val, y\_pred\_val)

print(f"Validation Accuracy of Best Model: {validation\_accuracy}")

# Print classification report (optional)

print(validation\_report)

## classification report of XGBoost Classifier the best model after evaluation.

clf\_report=classification\_report(y\_test\_imb,best\_estimators["SVM"].predict(X\_test\_imb))

print(clf\_report)

**Conclusion**

**## In Imbalance data the best model is XGBoost model works well**

best\_model\_name = max(scores\_imb, key=lambda x: x['best\_score'])['model']

best\_model\_score = max(scores\_imb, key=lambda x: x['best\_score'])['best\_score']

print(f"Best Model: {best\_model\_name} with Score: {best\_model\_score}")

# We can now use the best\_estimators\_imb[best\_model\_name] for prediction on your test data

y\_pred = best\_estimators\_imb[best\_model\_name].predict(X\_test\_imb)

test\_accuracy = accuracy\_score(y\_test\_imb, y\_pred)

print(f"Test Accuracy of Best Model: {test\_accuracy}")

**In balance data the best model is SVM model that works well.**

print("Model Selection Results:")

for score in scores:

  print(f"\tModel: {score['model']}, Best Score: {score['best\_score']}, Best Parameters: {score['best\_params']}")

# Access the best model (assuming you want the classifier with the highest score)

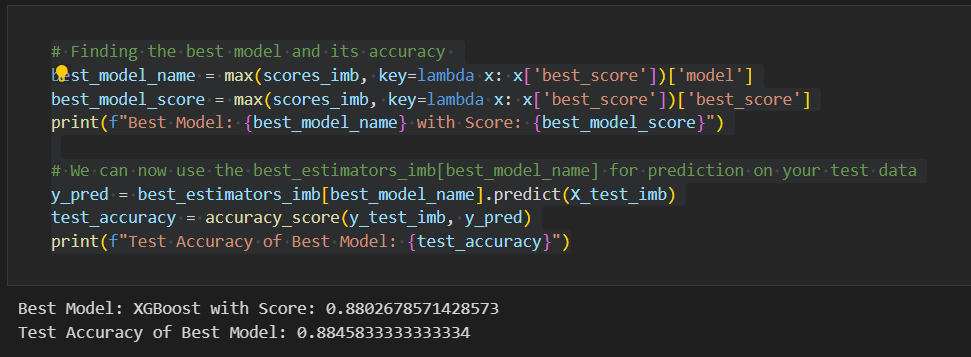
best\_model\_name = max(scores, key=lambda x: x['best\_score'])['model']

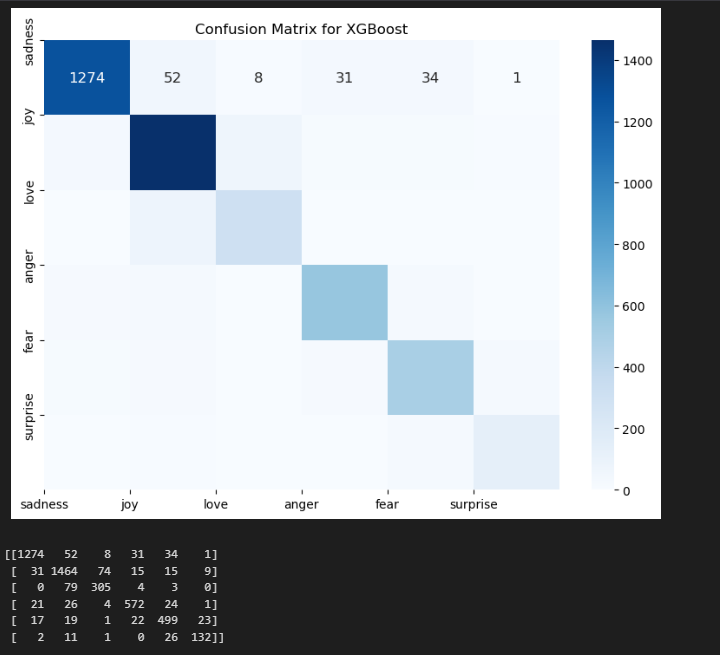
best\_model = best\_estimators[best\_model\_name]

print(f"\nBest Model: {best\_model\_name}")

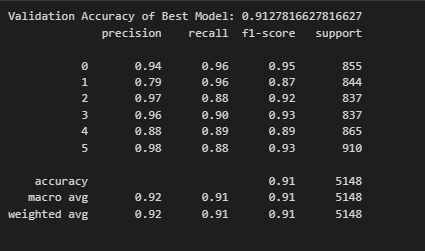
# CHAPTER -6

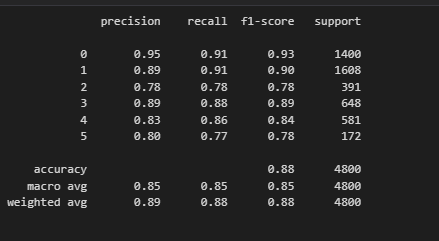
## **6.1 TESTING**





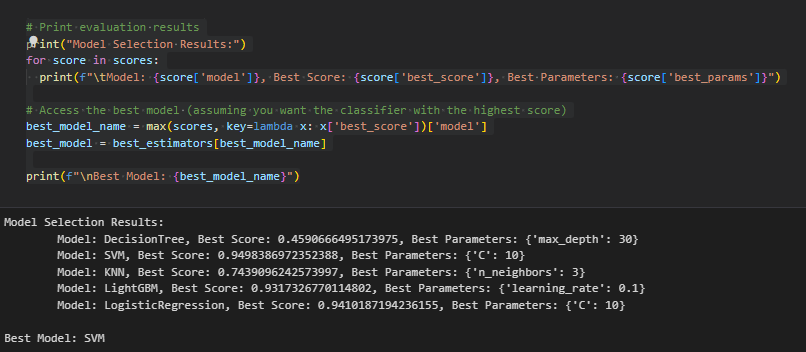
26

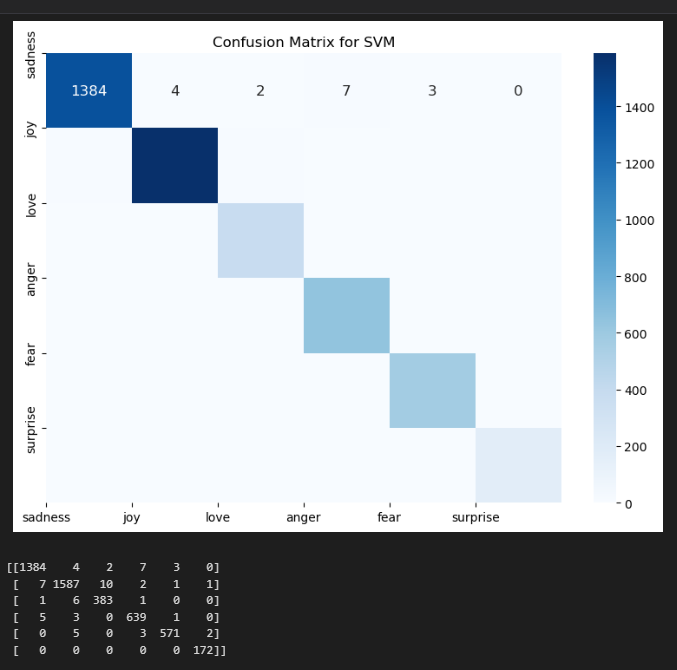




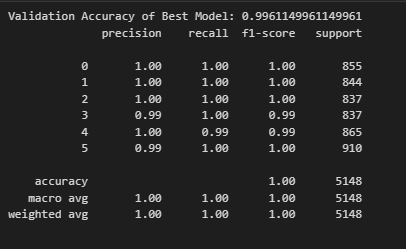
27

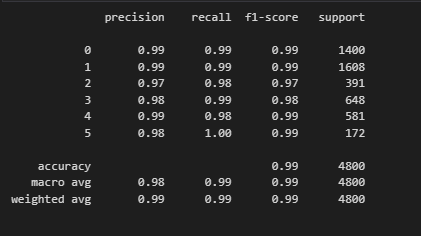
**For Balance Data**



****

28

****

****

29

# CHAPTER - 7

**CONCLUSION & LIMITATION**

## **Conclusion**

This project has explored the development of a text classification system for emotion detection. We successfully built a framework that utilizes machine learning to classify emotions within textual data. The project highlights the potential of this technology for various industries, enabling deeper customer understanding, improved user experiences, and analysis of brand perception.

**Key Achievements:**

* Developed a system for text pre-processing and feature extraction from emotional text data.
* Trained and evaluated various machine learning models for emotion classification.
* Analyzed model performance metrics to identify strengths and weaknesses.

## **Limitations**

While the project demonstrates the feasibility of emotion classification, there are limitations to consider:

* **Limited Emotion Set:** The current model classifies a specific set of emotions. Expanding the repertoire requires additional labeled data and model retraining.
* **Data Dependence:** Model performance heavily relies on the quality and quantity of training data. Biases or limitations within the data can be reflected in the model's predictions.
* **Accuracy vs. Interpretability:** The trade-off between achieving high accuracy and understanding how the model arrives at its predictions exists. Complex models might be highly accurate but lack interpretability, making it difficult to understand the reasoning behind their classifications.
* **Domain Specificity:** A model trained on a specific domain (e.g., social media) might not generalize well to other domains (e.g., customer reviews) due to language variations and emotional expression styles.

30

## **Future Directions**

This project serves as a foundation for further exploration. Here are some potential areas for future development:

* **Incorporate Deep Learning:** Explore deep learning architectures like LSTMs (Long Short-Term Memory networks) that can potentially capture long-range dependencies within text data and improve classification accuracy.
* **Expand Emotion Range:** Train the model to classify a wider range of emotions, requiring a more comprehensive and diverse dataset.
* **Real-World Integration:** Integrate the model with real-world applications like social media sentiment analysis tools, chatbots capable of responding with emotional intelligence, or mental health support systems.
* **Explainable AI:** Investigate techniques for improving model interpretability, allowing us to understand the rationale behind the emotion classifications.
* **Domain Adaptation:** Explore techniques for adapting the model to perform well on different data domains with potentially varying emotional expression styles.

By addressing these limitations and exploring future directions, we can refine and enhance this text classification system for emotion detection, unlocking its full potential in various applications.

31

# CHAPTER -8

## **8.1 REFERENCE/BIBLIOGRAPHY**

<https://github.com/topics/nrc-emotion-lexicon>

* This repository provides a pre-built lexicon for associating words with emotions, which can be a valuable resource for feature engineering in your project.

<https://github.com/topics/text-emotion-detection>

* This repository showcases a text classification application for emotion detection using transformers (e.g., BERT) and deep learning techniques. While it might involve more advanced concepts, it offers an example of a deep learning approach to this task.

<https://github.com/topics/emotion-classification>

* This repository demonstrates text classification for emotions using scikit-learn libraries similar to your project's approach. It can be a helpful reference for exploring code implementations of various machine learning algorithms for this task.

<https://github.com/topics/emotion-classification>

* This repository demonstrates text classification for emotions using scikit-learn libraries similar to your project's approach. It can be a helpful reference for exploring code implementations of various machine learning algorithms for this task.

These repositories offer various perspectives on text classification for emotion detection. You can explore the code, techniques, and datasets used to gain insights and potentially inspire your own project's development. Remember to adapt and modify the approaches based on your specific project requirements and chosen libraries.

32