



Module Code & Module Title

CU6051NT- Artificial Intelligence

Assessment Weightage & Type

Individual Coursework (75%)

Year and Semester

2024-25 Autumn

Student Name: Siddhartha Dev Shrestha

London Met ID: 22072245

College ID: NP05CP4A220119

Assignment Submission Date: Jan 22, 2025

Submitted To: Zishan Siddique

Dataset Link: [Dataset Link](#)

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded

Table of Contents

1	Introduction	8
1.1	Explanation of the topic/AI concepts used	8
1.1.1	Artificial Intelligence	8
1.1.2	Machine Learning	9
1.1.3	Deep Learning	14
1.1.4	AI concepts in emotion classification analysis	15
1.2	Explanation/introduction of the chosen problem domain/topic	16
2	Background	17
2.1	Research work done on the chosen topic/problem domain	17
2.2	Review and analysis of existing work in the problem domain	19
3	Solution	26
3.1	Explanation of the proposed solution	26
3.1.1	Approach to solving the problem	26
3.2	Explanation of the AI algorithm/algorithms used	27
3.3	Pseudocode of the solution(Updated)	33
3.3.1	Pseudocode of SVM	34
3.3.2	Pseudocode of Naïve bayes	35
3.3.3	Pseudocode of RandomForest	36
3.4	Diagrammatic representations of the solution	37
3.4.1	Flowchart	37
4	Milestone 1 Work Done (Updated)	41
4.1	Installing and Preparing the Datasets Library	41
4.2	Importing Necessary Libraries and Modules	42
4.3	Loading the Dataset	43
4.4	Extracting Texts and Labels from the Dataset	43
4.5	Previewing Heading	43
4.6	Splitting Data into Training and Testing Sets	44
4.7	Transforming Text Data Using TF-IDF Vectorizer	44
4.8	Implementing the Naive Bayes Model	45
4.8.1	Naïve Bayes Precision Score	45
4.8.2	Naïve Bayes F1-Score	46
4.8.3	Naïve Bayes Recall Score.	46

4.9	Implementing Support Vector Machines Model.....	47
4.9.1	SVM Precision Score	47
4.9.2	SVM F1 Score	48
4.9.3	SVM Recall Score.....	48
4.10	Implementing Random Forest Model	49
4.10.1	Random Forest Precision Score	49
4.10.2	Random Forest F1 Score.....	50
4.10.3	Random Forest Recall Score.....	50
4.11	Visualizing Model Accuracy Scores	51
4.12	Visualizing Model Precision Scores	52
4.13	Visualizing Model F1 Scores.....	53
4.14	Visualizing Model Recall Score.....	54
5	Milestone 2 Work Done	55
5.1.1	Essential Library Imports for Model Training and Evaluation	55
5.1.2	Downloading NLTK Resources For Text Preprocessing.....	56
5.1.3	Loading and Converting the Dataset for Analysis	57
5.1.4	Data Preprocessing	58
6	Fine Tuning	62
6.1	Hyperparameter Tuning and Evaluation of SVM Model	63
6.1.1	Displaying SVM Model Results.....	64
6.1.2	Confusion Matrix Visualization for SVM Model	65
6.2	Hyperparameter Tuning and Evaluation of Random Forest Model	66
6.2.1	Displaying Random Forest Model Results.....	67
6.2.1	Confusion Matrix Visualization for Random Forest Model	68
6.3	Hyperparameter Tuning and Evaluation of Naive Bayes Model.....	69
6.3.1	Displaying Naïve Bayes Model Results	70
6.3.2	Confusion Matrix Visualization for Naïve Bayes Model.....	71
6.4	Model Performance Metrics Preparation.....	72
6.4.1	Model Performance Metrics Bar Chart.....	73
6.5	Comparison of Model Performance Before and After Hyperparameter Tuning	74
6.5.1	SVM Model	74
6.5.2	Random Forest Table	74
6.5.3	Naïve Bayes Table	74

6.6	Emotion Prediction Using Machine Learning	76
6.6.1	Code Implementation.....	76
6.6.2	Examples of Emotion Prediction	77
7	Conclusion	79
7.1	Analysis of the work done	80
7.2	Solution Addresses Real-World Problems	81
8	References.....	82
9	Plagiarism Report.....	85

Figure 1: AI puppet control	9
Figure 2: Machine learning	10
Figure 3: Machine learning methods	10
Figure 4: Supervised Learning	11
Figure 5: Unsupervised learning	12
Figure 6: Semi-Supervised Learning	13
Figure 7: Deep Learning.....	14
Figure 8: Research 1: Emotion Classification and Emoji Mapping using Convolutional Neural Networks.....	20
Figure 9: Research 2: Facial Emotion Detection Using Convolutional Neural Networks.....	22
Figure 10: Research 3: Emotion Classification Using 1d-Cnn.....	25
Figure 11: SVM: Optimal Hyperplane and Margin	30
Figure 12: Bayes' Theorem Formula with Key Components(Lecture 6 slide).....	31
Figure 13: Random Forest: Key Formula for Random Forest	32
Figure 14: Flowchart: Flowchart diagram of SVM	38
Figure 15: Flowchart: Flowchart of Naive Bayes	39
Figure 16: Flowchart: Flowchart diagram of Random Forest.....	40
Figure 17: Output of Installing the Datasets Library	41
Figure 18: Importing Libraries for Data Processing and Modeling.....	42
Figure 19: Loading the Emotion Dataset	43
Figure 20: Separating Text Data and Labels.....	43
Figure 21: Sample of Extracted Text Data:	43
Figure 22: Dataset Split for Model Training and Evaluation	44
Figure 23: Applying TF-IDF to Train and Test Data	44
Figure 24: Naive Bayes Model Accuracy.....	45
Figure 25: Naive Bayes Precision Score	45
Figure 26: Naive Bayes F1 Score	46
Figure 27: Naive Bayes Recall Score.....	46
Figure 28: SVM Model Accuracy in Emotion Classification	47
Figure 29: SVM Precision Score	47
Figure 30: SVM F1 Score.....	48
Figure 31: SVM Recall Score	48
Figure 32: Random Forest Model Accuracy	49
Figure 33: Random Forest Precision Score	49
Figure 34: Random Forest F1 Score	50
Figure 35: Random Forest Recall Score	50
Figure 36: Model Accuracy Comparison Chart.....	51
Figure 37: Model Precision Comparison Chart.....	52
Figure 38: Model F1 Score Comparison Chart.....	53
Figure 39: : Model Recall Score Comparison Chart	54
Figure 40: Importing Necessary Libraries.....	55
Figure 41: Downloading NLTK Resources	56
Figure 42: Loading The Dataset.....	57

Figure 43: Text Preprocessing Function: Checking Missing Values.....	58
Figure 44: Text Preprocessing Function: Lemmatization of the Data	59
Figure 45:Text Preprocessing Function: Text Preprocessing Function Here in this snippet.....	60
Figure 46: Data Preprocessing: Applying Processed Text Into The Dataset.....	60
Figure 47: SVM Model: Hyperparameter Tuning of SVM Model	63
Figure 48: SVM Model:Displaying Results After Hyper Tuning	64
Figure 49: SVM Model: Confusion Matrix of SVM Model	65
Figure 50: Random Forest: Hyperparameter Tuning Random Forest Model	66
Figure 51: Random Forest: Displaying Results After Hyperparameter Tuning.....	67
Figure 52: Random Forest: Confusion Matrix of Random Forest	68
Figure 53: Naive Bayes: Hyperparameter Tuning Naïve Bayes Model	69
Figure 54: Naive Bayes: Displaying Results After Hyperparameter Tuning	70
Figure 55: Naive Bayes: Confusion Matrix of Naïve Bayes	71
Figure 56: Models Performance Metrics Preparation	72
Figure 57: Model Performance Metrics Bar Chart	73
Figure 58: Process of Predicting Emotions From Text	76
Figure 59: Examples of Emotion Prediction	77

Table Of Table

Table 1: Comparison Table of SVM Model.....	74
Table 2: Comparison Table of Random Forest Model.....	74
Table 3: Comparison Table of naïve Bayes Model.....	74
Table 4: Analysis of the Work Done	80

1 Introduction

1.1 Explanation of the topic/AI concepts used

1.1.1 Artificial Intelligence

Artificial Intelligence-One of the most advanced, modern technologies-is the ability of machines to do various human-like things, such as learning, knowing how to do things, solve problems, make decisions, creative thinking, and working independently.

Computers can treat information and solve tasks in a manner that imitates intelligent behaviors. That is how we work and interplay with technology. (Cole Stryker, 2024)

Until recently, a concept first considered in the 19th century and then popularized through Alan Turing's "imitation game" has come true by way of increased computing power and greater availability of data. Whereas human beings make use of biological brain networks, coupled with learning based on experience, robots are rather good at handling large volumes of data fast and with ease. That means, AI streamlines hard processes like decision-making, fraud detection, predictive maintenance, and route optimization, freeing attention for more strategic and imaginative efforts. The research in artificial intelligence is to create AI systems that demonstrate intelligent behavior, such as scheduling, planning, natural language processing, speech recognition, perception, and object manipulation, to change the industries to raise production. (Abhishek, 2022)

Artificial intelligence is an entity that encompasses several disciplines of learning, such as computer science, mathematics, and psychology, toward solving real-life problems. Artificial Intelligence makes use of information, judgment, and data provided by algorithms to make automated processes of something. Although AI is actually modeled after human intelligence, AI is superior in performing jobs related to fraud detection, route optimization, and planning since it can handle large volumes of data much faster. The aim of research is to come up with intelligent systems for speech recognition, language comprehension, and problem-solving that will mark a revolution in several industries, making daily chores easier. . (Abhishek, 2022)



Figure 1: AI puppet control

1.1.2 Machine Learning

Machine Learning (ML) is a foundational branch of artificial intelligence that focuses on enabling computers to learn and improve from experience without being explicitly programmed. By training computers to recognize patterns in data, make predictions, and adjust to novel circumstances, it closes the gap between computational algorithms and human like intelligence.

Machine learning can be applied in wide-ranging important uses, from natural language processing and financial forecasts to online search engines, customized recommendations, and driverless cars. For addressing a set of problems, it applies many strategies, which include reinforcement learning, supervised learning, and unsupervised learning. These approaches allow systems to effectively execute classification, clustering, and other kinds of decision-making.

At the base of ML use is probability and statistics as optimisation for modelling real problems. When these disciplines, therefore, support ML algorithms, huge, large amounts can be used through their structures such that meaningful solutions are realized upon analysis. Generally, more this data, ML will become more power in all sense, even driving for innovation and transforming various sectors while in life nowadays. (Vishwanathan, 2008)

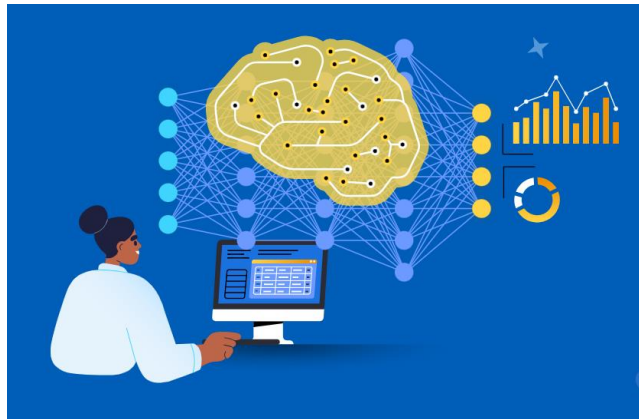


Figure 2: Machine learning

In Machine Learning, there are two main types of learning approaches and they are listed below

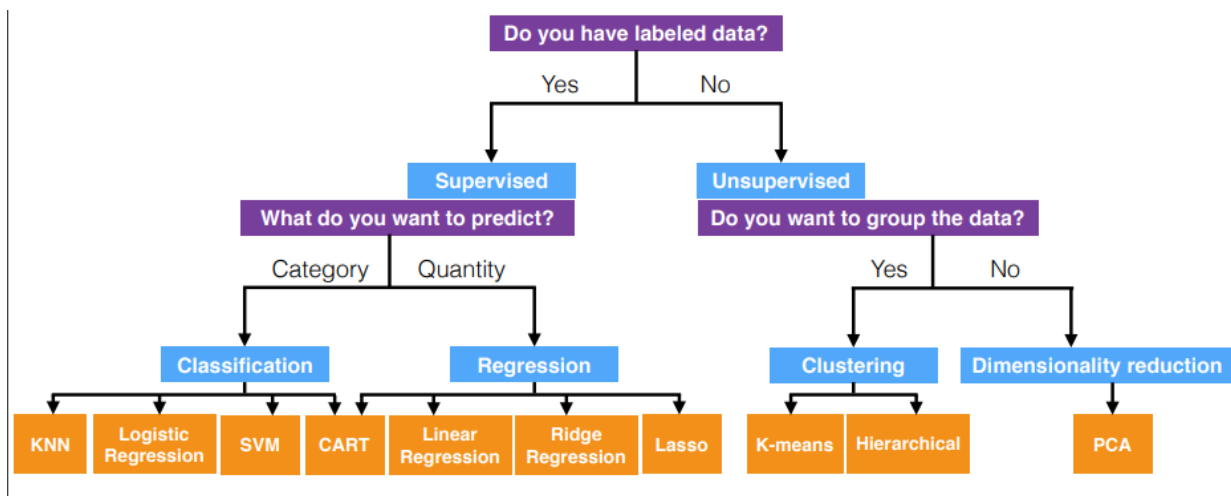


Figure 3: Machine learning methods

A. Supervised learning

This is referred to as supervised learning in machine learning, where a model can be trained over labeled data, categorizes new, unseen data, or predicts output for the same data. In this procedure, there are two principal datasets used; the first one is called the test set, which is required for performance measurement and the rest, called the training set, where samples are generally identified.

For example, in the case of image classification, the training set can include images of fruits such as apples and peaches with proper labels. The learned model is capable of identifying such fruits and is able to make decisions for identifying appropriate test images, which are unlabeled. It is used abundantly in the applications of spam detection, medical diagnosis, and recommendation systems; therefore, this is the spine of practical applications using machine learning. (Learned-Miller, February 17, 2014)

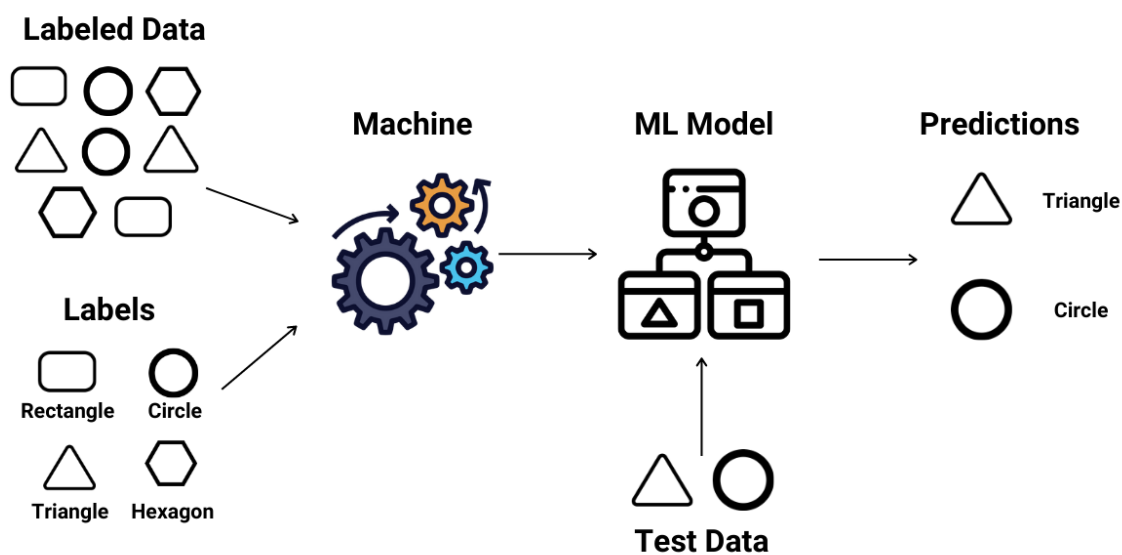


Figure 4: Supervised Learning

B. Unsupervised Learning

Unsupervised Learning is machine learning concerning data with not specified labels and outcomes. Pattern discovery, grouping, or structure identification are common objectives here done without explicit supervisions. It thus applies particularly well on clustering and related tasks like applying the same notion to simplify high-dimensional input for better data visualization and meaning extraction.

For example, K-means and hierarchical clustering can be helpful in clusters because they allow the determination of groups in the data set- for example classifying documents under a specific topic or dividing up customers under various behaviors. In the area of dimensionality reduction, it has a particular technique known as Principal Component Analysis that develops compact representations of data and allows for easy development of insights or analyses. Also, the basis of exploratory data analysis and anomaly detection shines in unsupervised learning with datasets underlying structures. (Wang, 2019)

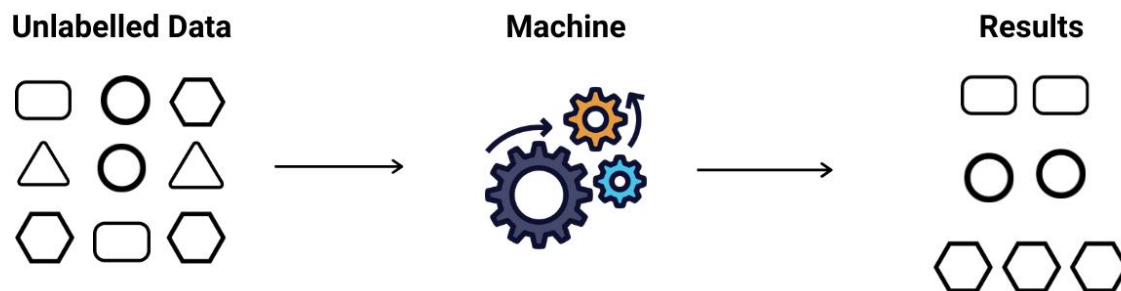


Figure 5: Unsupervised learning

C. Semi-Supervised Learning

Semi-Supervised Learning is a unique approach in machine learning that combines both labeled and unlabeled data to improve model performance. Unlike supervised learning, which relies solely on labeled data, or unsupervised learning, which only analyzes unlabeled data, semi-supervised learning bridges the gap by leveraging a small amount of labeled data to guide the learning process on a larger set of unlabeled data. This method is particularly useful when labeling data is expensive, time-consuming, or impractical, but large quantities of unlabeled data are available.

For example, in a scenario where only a few medical images are labeled with specific diagnoses but thousands of unlabeled images exist, semi-supervised learning can use the labeled images to train a model and then refine its understanding using the unlabeled data. This approach not only saves resources but also enhances the model's ability to generalize. (Zhu, 2008)

Common techniques in semi-supervised learning include:

- **Self-Training:** The model is first trained on the labeled data and then predicts labels for the unlabeled data, iteratively adding high-confidence predictions to the labeled set.
- **Co-Training:** Two models are trained on different features of the data, and each model labels new data for the other, boosting overall accuracy. (Zhu, 2008)

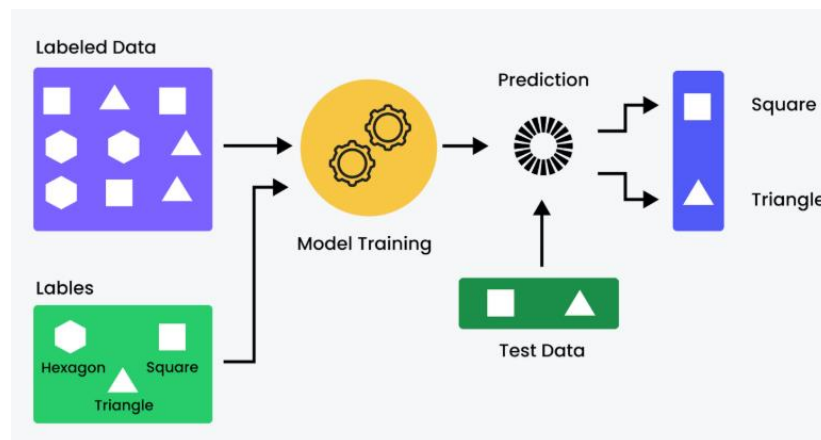


Figure 6: Semi-Supervised Learning

1.1.3 Deep Learning

The focus of deep learning in artificial intelligence is the design of systems capable of learning and improvement from huge volumes of data. It processes information just like the human brain would by simulating neural networks comprised of several layers of interconnected nodes called neurons. These layers, which are hierarchical in nature, give machines the ability to see patterns, forecast outcomes, and resolve challenging issues that were previously believed to be beyond the capabilities of computers. (Deng, 2012)

With the development of deep learning, thanks to great works by researchers like Geoffrey Hinton in 2006 and further work by Andrew Ng, among others, it has remained in the spotlight as improvements in computing power, large datasets, and remarkable results across different domains are attained. From powering voice assistants like Siri and Alexa to enabling self-driving cars, facial recognition, and real-time language translation, deep learning has become the backbone of modern AI. (Toja, June,2013)

What basically sets deep learning apart from traditional machine learning is that it automatically extracts features from the raw data. It involves a number of models, such as DNNs, RNNs, and GANs, among others, targeted for specific applications including image generation, speech recognition, and natural language processing. Deep learning keeps pushing the limits of AI and redefining industries with endless possibilities for innovation and intelligent solutions. (Toja, June,2013)

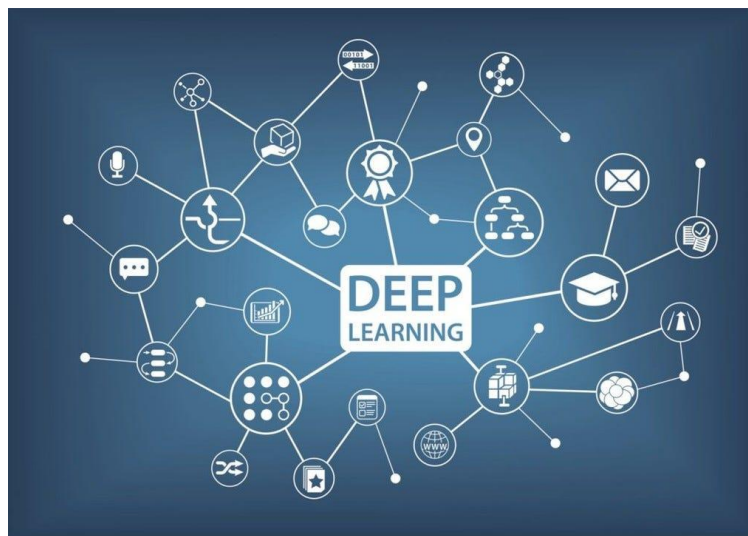


Figure 7: Deep Learning

1.1.4 AI concepts in emotion classification analysis

Artificial Intelligence (AI) is the simulation of human intelligence in machines, which are programmed to think, learn, and make decisions.

In this project, emotional classification investigations are considered one of the features of AI that enables machines to analyze and make identifications of human happy, sad, angry, fear emotions from speech or text inputs. These algorithms are used throughout AI applications, which include chatbots, analytical social media interfaces, and mental health assistant tools where the meaning of different emotions is discerned to improve communication and user interactions. (Munoz-Organero, 2022).

This is an AI that uses several algorithms in the classification of emotions, including a traditional method such as Support Vector Machines, which can work on simpler tasks and structured data; advanced AI techniques like Random Forest, where multiple models are used to merge their specific accuracy; furthermore, probabilistic approaches, such as Naïve Bayes, can be made to apply statistical patterns in order to classify emotions and are good at text analysis.

Despite these advancements, AI still has not been effective in the emotion classification process due to the necessity of dealing with multiple emotions, variations in different cultures, and limited availability of training data. In this project, we have attempted to create a reliable and efficient emotional classification system by leveraging the power of AI-driven algorithms and fine-tuning techniques, such as GridSearchCV.

1.2 Explanation/introduction of the chosen problem domain/topic

This area has been more focused perhaps by researchers in the artificial intelligence field because the applications lie in developing chatbots, in mental health monitoring, and in social media analysis. For instance, (Erik Cambria, 2017) foundational approaches in regards to sentiment analysis by creating a model to detect emotion from text. In this sense, (Soujanya Poria, 2017) made efforts to improve and take forward the idea with deep learning-based approaches such that the efficiency of the detection of emotions is maximized through better accuracy and robustness.

Despite these, there are a number of challenges that remain. Existing systems still find it difficult to detect mixed or overlapping emotions and to deal with cultural and linguistic variations in the expression of emotions. Further, a lack of diversity and comprehensiveness in training datasets limits generalization. Overcoming these gaps is crucial to move ahead in developing emotional classification and further its application in real-world scenarios. The following sections delve deeper into previous research, tools, and techniques that have shaped the field.

2 Background

2.1 Research work done on the chosen topic/problem domain

Emotional classification is a pivotal area within Natural Language Processing (NLP) that focuses on identifying and categorizing human emotions from text into discrete classes such as happiness, sadness, anger, and fear. Over time, the methods used for emotional classification have evolved from basic lexicon-based approaches to advanced transformer-based models, driven by advancements in computational capabilities and data availability. Below is an exploration of the key research and methodologies shaping this domain: (Adil, 2016)

Evolution of Emotional Classification

I. Lexicon-Based Methods:

Early methods relied on sentiment lexicons, such as NRC Emotion Lexicon, which mapped words to specific emotions. These systems were straightforward and easy to implement but lacked the ability to handle context, idiomatic expressions, or complex sentence structures. For instance, the sentence "I'm fine, but not really" might be misclassified as neutral due to the literal interpretation of words. (Dr. Lutful Islam¹, 2021)

II. Classical Machine Learning Approaches:

The transition to supervised learning models like Naïve Bayes, Support Vector Machines (SVMs), and Logistic Regression brought data-driven techniques to emotional classification. Text was numerically represented using techniques like:

- Bag of Words (BoW): Frequency-based representation of text.
- TF-IDF (Term Frequency-Inverse Document Frequency): Emphasized unique or important words in a dataset. (Erik Cambria, 2017)

While these models improved classification accuracy, they struggled with capturing emotional nuances and required manual feature engineering.

III. Deep Learning Models:

Deep learning revolutionized emotional classification by introducing neural networks capable of automatically learning complex patterns in text:

- Recurrent Neural Networks (RNNs): Designed for sequential data, these models captured relationships between words, while variants like Long Short-Term Memory (LSTM) addressed long-range dependencies in text sequences.
 - Convolutional Neural Networks (CNNs): Adapted for text, these models excelled in identifying local word relationships but struggled with context across sentences.
- (Dr. Saif M. Mohammad, 2011)

IV. Transformer-Based Models:

Transformers like BERT and RoBERTa transformed emotional classification by providing contextual embeddings, capturing the meaning of words in relation to their context. This enabled models to classify nuanced emotions in tasks such as multi-label emotion recognition. For example, BERT could effectively handle sentences like "I'm excited yet nervous," identifying multiple overlapping emotions. (Dr. Lutful Islam¹, 2021)

Challenges in Emotional Classification

❖ Ambiguity and Mixed Emotions:

Texts often contain conflicting or overlapping emotions, such as "The event was amazing, but the crowd was overwhelming," making single-label classification difficult.

❖ Data Scarcity and Imbalance:

Emotion datasets are often imbalanced, with dominant emotions overshadowing others, leading to biased models.

❖ Cultural and Linguistic Variations:

Emotional expressions vary across languages and cultures, requiring robust, adaptable models.

❖ Context Sensitivity:

The meaning of words can change depending on the context. For instance, "bright" could indicate happiness in "bright future" or anger in "bright light blinding me."

2.2 Review and analysis of existing work in the problem domain

Below are some detailed examination of significant research works that have influenced this field.

i. Emotion Classification and Emoji Mapping using Convolutional Neural Networks

The goal of this project is to improve digital communication tools and social media platforms by developing a deep learning-based system that can recognize facial emotions and translate them into equivalent emojis. The FER2013 dataset, which includes a large number of grayscale facial photos depicting various emotions including joyful, sad, and neutral, is used by the system. (Dr. Lutful Islam¹, 2021)

➤ Methodology

To process 48x48 pixel grayscale pictures from the FER2013 dataset, the researchers used a Convolutional Neural Network (CNN) architecture. In order to identify distinct emotional categories and extract significant face characteristics, the CNN model was refined. To make the result easier for end users to understand, the identified emotions were then mapped to preset emojis. To enhance generalization and avoid overfitting, strategies including dropout layers and data augmentation were applied.

➤ Findings

The model provides a workable solution for emotion detection in real-time applications as it successfully connected emotions to their respective emojis and classified emotions with high accuracy. CNNs were able to handle a variety of face expressions and illumination changes in the dataset with resilience.

➤ Conclusion

This study demonstrates the potential of CNNs for integrating emotion recognition with visual representations like emojis. The system could enhance user experience in

applications such as social media, chatbots, and virtual customer service. Future work involves extending the model to include cultural-specific emoji mappings and dynamic real-time emotion detection.

Emotion Classification and Emoji Mapping using Convolutional Neural Network

Dr. Lutful Islam¹, Arif Ansari², Sreejith Nair³, Azhan Patel⁴

¹Professor, Dept. of Comp. Engg., M. H. Saboo Siddik College of Engineering, Mumbai, Maharashtra, India

²⁻⁴Student, Dept. of Comp. Engg., M. H. Saboo Siddik College of Engineering, Mumbai, Maharashtra, India

Abstract - In this paper, we suggest an implementation of a **convolutional neural network (CNN)** to classify emotion and map emoji to the classified emotion. We validate our models through growing a real-time vision machine which accomplishes the responsibilities of **face detection, emotion classification and emoji mapping** simultaneously in a single mixed step using our proposed **CNN architecture**. After presenting the information of the training procedure setup we maintain to evaluate on standard benchmark sets. We record accuracy of 62% withinside the FER-2013 emotion dataset. We argue that the careful implementation of cutting-edge CNN architectures, using the cutting-edge regularization strategies and the visualization of previously hidden features are important for you to reduce the space amongst slow performances and real-time architectures

Key Words: *Convolution Neural Network (CNN), face detection, emotion classification, emoji mapping, CNN architecture*

1. INTRODUCTION

Nonverbal behavior passes on complete feeling and passionate data, to impart thoughts, oversee connections, and disambiguate importance to enhance the effectiveness of discussions[1][2]. One method to illustrate nonverbal activates is through sending emoticon, that are practical symbols (e.g., , ,) oversaw through the Unicode Consortium which might be distinguished through unicode characters and introduced through a systems font bundle.

Emoticons empower people to speak lavishly, and keeping in thoughts that are regarded as display screen designs, they may be managed as textual content structures. Other than Pohl's EmojiZoom[3] who endorse a zooming-based interface, getting into emoticon on cell phone consoles as of now expects customers to make a dedication from extensive records (one rundown for each class of emoticon) (e.g., Apple® iOS 10 emoticon console 2 in Fig. 1). This makes emoticon passage "a linear search task"[3] and given the developing wide variety of emoticons, we anticipate it may result in customer dissatisfaction. While no earlier work expressly addresses this, endeavors, for example, Emojipedia 3 characterize the requirement for higher emoticon search.

To deal with this, we advise a framework and approach to make use of customers' facial emotional expressions as framework input to clear out emoticons with the aid of using emotional classification. In spite of the truth that emoticons can deal with activities, items, nature, and distinctive symbols, the maximum typically applied emoticons are faces with specific feelings[4][5][7]. Additionally, past work has proven that emotions may be placed through assumption (Emoji Sentiment Ranking with the aid of using Novak[6]), literary notification containing emoticons show contrasts in 3-esteemed conclusion throughout stages[8], and for faces, emoticons may be positioned through valence and arousal[7].

Momentum studies examine facilities around emotional recognition. The emotions every on-occasion ease and determine connections among the people. The putting of emotions explicitly attracts out the thoughts boggling and atypical social correspondence. Social correspondence is prominent as the judgment of the alternative people's temperament depending on the emoticon. The acknowledgment of emotions may be diagnosed via different signs by the "frame language, voice intonation" simply as through means of "extra complicated techniques, for example, electroencephalography (EEG)." Nonetheless, the much less difficult, and possible method is to investigate facial expression. By noticing the facial expression, the people's mind-set and behavior are affected. Duncan clarified that "there are seven types of human feeling [that could undoubtedly be unmistakable with an assortment of meanings] across various societies". This examination consists of discovering emotional recognition through the regular collections. The emotions are distinguished as happiness, fear, disgust, anger, sadness, surprise and contempt.

This project objectives to construct a deep learning model to categorize facial expressions from the images. Then we are able to map the labeled emotion to an emoji or an avatar.

Figure 8: Research 1: Emotion Classification and Emoji Mapping using Convolutional Neural Networks

ii. Facial Emotion Detection Using Convolutional Neural Networks

This research introduces a face emotion identification system that uses deep learning to improve emotion-aware apps. The FER2013 dataset, which consists of thousands of grayscale face photos classified with emotional states including fear, anger, sadness, and happiness, is used by the system. The project's objective was to correctly categorize these pictures into the appropriate emotional groups in order to improve automated systems and human-computer interaction. (Adil, 2016)

➤ Methodology

Two Convolutional Neural Network (CNN) models were created for the project. Convolution, pooling, and thick layers were among the essential layers in both models that efficiently extracted and categorized face characteristics. Preprocessing procedures included normalization, occlusion removal for higher-quality inputs, and data augmentation to balance the dataset. In order to improve stability and lessen overfitting, the models were adjusted using strategies including batch normalization and dropout. The FER2013 dataset was used for training and assessment, and performance was assessed using measures including accuracy, precision, recall, and F1-score.

➤ Finding

The CNN models demonstrated strong performance in recognizing emotions:

- **Model 1** achieved an accuracy of 80% for four emotions and 72.2% for five emotions.
- **Model 2** performed similarly, achieving 79.5% accuracy for four emotions and 72.5% for five emotions. The "happy" emotion was most accurately identified, while the "neutral" category posed challenges due to minimal facial expression variance. The results showed the system's potential for deployment in applications like virtual assistants, robotics, and emotion-sensitive user interfaces.

➤ **Conclusion**

The project successfully highlights the potential of CNNs for facial emotion recognition, offering a practical and reliable solution for real-world scenarios. Future enhancements could include expanding datasets to cover more diverse populations, integrating real-time video emotion detection, and optimizing the models for resource-constrained devices like mobile platforms.

**FACIAL EMOTION DETECTION USING CONVOLUTIONAL NEURAL
NETWORKS**

by

Mohammed Adnan Adil
Bachelor of Engineering, Osmania University, 2016

A Report Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

©Mohammed Adnan Adil, 2021

University of Victoria

All rights reserved. This report may not be reproduced in whole or in part, by photocopy or other means,
without the permission of the author.

Figure 9: Research 2: Facial Emotion Detection Using Convolutional Neural Networks

iii. Emotion classification using 1d-cnn and rnn based on deap dataset

The categorization of emotions using physiological signals—specifically, EEG data—is the subject of this effort. This study divides emotions into four zones based on valence and arousal dimensions: High Valence High Arousal (HVHA), High Valence Low Arousal (HVLA), Low Valence High Arousal (LVHA), and Low Valence Low Arousal (LVLA). It does this by utilizing the DEAP dataset, a standard for emotion detection. The primary objective was to create a hybrid model for efficient and precise emotion categorization by fusing Recurrent Neural Networks (RNN) with 1D-Convolutional Neural Networks (1D-CNN). (Wulansari, 2024)

➤ Methodology

The project proposed two deep learning models:

1. 1D-CNN + GRU

- Used Gated Recurrent Units (GRU) for sequence learning to address the vanishing gradient problem.
- Applied convolutional layers to extract spatial features from the EEG signals, followed by GRU layers to learn temporal patterns.

2. 1D-CNN + LSTM

- Substituted GRU with Long Short-Term Memory (LSTM) layers to capture more complex temporal dependencies.
- Both models included dropout layers for regularization and used categorical cross-entropy as the loss function.

Preprocessing:

- **Epoching:** Divided continuous EEG signals into 2-second segments.
- **Normalization:** Standardized data using Z-score normalization and Min-Max scaling.

- **Channel Selection:** Reduced data dimensions by selecting 14 EEG channels relevant to emotion recognition.

The dataset was split into training (60%), validation (20%), and test (20%) sets, with data labels classified based on valence and arousal thresholds.

➤ Findings

- **1D-CNN + GRU:**
 - Training accuracy: 96.3%
 - Validation and test accuracy: 99.9%
 - Training loss: 10.7%, validation loss: 0.7%, test loss: 0.6%
 - Efficient with minimal overfitting.
- **1D-CNN + LSTM:**
 - Training accuracy: 97.8%
 - Validation and test accuracy: 99.9%
 - Training loss: 6.7%, validation loss: 0.1%, test loss: 0.1%
 - Superior accuracy but required longer training time per epoch.

The 1D-CNN + LSTM model slightly outperformed the 1D-CNN + GRU model in terms of accuracy and loss. Both models demonstrated robustness in classifying emotions into the four defined regions.

➤ Conclusion

This study successfully demonstrates the efficacy of hybrid deep learning architectures for EEG-based emotion recognition. The 1D-CNN layers effectively extracted spatial features, while the GRU and LSTM layers captured temporal patterns. Future work will include testing on additional datasets like DREAMER and AMIGOS, implementing k-fold

cross-validation for enhanced performance estimates, and deploying the models in real-time applications such as human resources and emotion-aware systems.

EMOTION CLASSIFICATION USING 1D-CNN AND RNN BASED ON DEAP DATASET

Farhad Zamani and Retno Wulansari

Telkom Corporate University Center, Telkom Indonesia, Bandung, Indonesia

ABSTRACT

Recently, emotion recognition began to be implemented in the industry and human resource field. In the time we can perceive the emotional state of the employee, the employer could gain benefits from it as they could improve the quality of decision makings regarding their employee. Hence, this subject would become an embryo for emotion recognition tasks in the human resource field. In a fact, emotion recognition has become an important topic of research, especially one based on physiological signals, such as EEG. One of the reasons is due to the availability of EEG datasets that can be widely used by researchers. Moreover, the development of many machine learning methods has been significantly contributed to this research topic over time. Here, we investigated the classification method for emotion and propose two models to address this task, which are a hybrid of two deep learning architectures: One-Dimensional Convolutional Neural Network (CNN-1D) and Recurrent Neural Network (RNN). We implement Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) in the RNN architecture, that specifically designed to address the vanishing gradient problem which usually becomes an issue in the time-series dataset. We use this model to classify four emotional regions from the valence-arousal plane: High Valence High Arousal (HVHA), High Valence Low Arousal (HVLA), Low Valence High Arousal (LVHA), and Low Valence Low Arousal (LVLA). This experiment was implemented on the well-known DEAP dataset. Experimental results show that proposed methods achieve a training accuracy of 96.3% and 97.8% in the 1DCNN-GRU model and 1DCNN-LSTM model, respectively. Therefore, both models are quite robust to perform this emotion classification task.

KEYWORDS

Emotion Recognition, 1D Convolutional Neural Network, LSTM, GRU, DEAP.

1. INTRODUCTION

Many industries and human resource fields began to implement emotional recognition of the employee in their organization. When they can assess the emotional state of the employee, the human resource could gain advantages from it as they could improve the quality of decision makings regarding their employee. This subject could become a base for emotion recognition tasks in human resource cases. For this reason, this task will become important widely used shortly.

Emotions play a crucial role in human beings as these are associated with neuro physiological aspects that also correspond to a coordinated set of responses, which may include verbal, behavioral, physiological, and neural mechanisms. In another perspective, the emotion can be mapped into the Valence, Arousal, and Dominance (VAD) dimensions. Valence represents a dimension that corresponds to the level of pleasantness that goes from very positive (pleasure) to very negative (displeasure). On the other hand, arousal is the intensity level of emotion that an

David C. Wyld et al. (Eds): NLP, MLTEC, CLBD, SEAPP, NeTIoT, VLSIE, ITCS, ARIA, DMS - 2021
pp. 363-378, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.112328

Figure 10: Research 3: Emotion Classification Using 1d-Cnn

3 Solution

3.1 Explanation of the proposed solution

The main source of data for the system's development is the Hugging Face Emotion Dataset. The annotated text samples in this corpus provide a good foundation on which to train machine learning models. Nevertheless, a thorough preparation step is required to ensure that the data is clean and consistent before being applied.

Feature extraction is done with the help of TF-IDF and emotion lexicons, which include things like the NRC Emotion Lexicon by (Dr. Saif M. Mohammad, 2011). These techniques transform the text into meaningful numerical values that can help the models better identify emotional patterns.

The algorithm uses GridSearchCV to compare performances between SVM, Naïve Bayes, and Random Forest to choose the best model that will see deployment in the job. It means that GridSearchCV enables a system to optimize hyperparameters on the models separately to assess its accuracy, thus settling for the one that bests classifies emotion.

With GridSearchCV, combining effective data preprocessing, feature extraction, and model refinement, a solution is guaranteed to be accurate, and it is scalable. This approach has enabled the solution to be adaptable for application in a wide domain of applications such as tracking the sentiment of social media activities, monitoring mental health, and even in analyzing customer response towards the products of an organization.

3.1.1 Approach to solving the problem

The first step in creating the emotion categorization system is gathering and preparing the data. Several steps are involved in the preprocessing:

- **Cleaning:** Removing special characters, numbers, and unnecessary whitespace to standardize the text.
- **Lowercasing:** Converting text to lowercase for uniformity.
- **Tokenization:** Splitting sentences into individual words or tokens.

- **Stop-Word Removal:** Eliminating common, non-essential words like "the" and "is."
- **Lemmatization:** Reducing words to their base forms, such as converting "running" to "run."

The cleaned text is transformed into numerical representations by feature extraction following preprocessing. The usage of emotion lexicons and techniques like TF-IDF draw attention to important characteristics that are crucial to emotion identification.

The system's main function is to train several machine learning models on the processed data, such as SVM, Random Forest, and Naïve Bayes. To get the best accuracy and performance, GridSearchCV is used to fine-tune each model and identify the ideal hyperparameters. The best-performing model is chosen for deployment with the aid of this methodical comparison. Particular care is taken to address possible issues, such data imbalance, which might happen if some emotions are underrepresented in the dataset. The system is resilient and dependable since it is made to adapt to various situations.

The final solution generates predictions for unseen data, enabling real-time emotion classification. Its modular design allows integration into various applications, such as customer sentiment analysis or mental health monitoring tools. User-friendly interfaces enhance the usability of the system, ensuring it meets the needs of diverse stakeholders.

3.2 Explanation of the AI algorithm/algorithms used

The emotion classification system combines a series of supervised learning algorithms. It uses the concepts of randomized forests, and Naïve Bayes- and Support vector machines, respectively, for processing text-based sources in estimating the emotions to be considered. However, to further optimize the best hyperparameters relevant to each of the algorithms used, Grid Search CV is used. TF-IDF for feature extraction, along with data preprocessing-task tokenization, lemmatization, and data cleaning-will ensure that input data coming in is clean to process. Combining these methods guarantees that the system gives a dependable and effective solution for problems on emotion categorization.

A. Support Vector Machine

A supervised machine learning approach for classification and regression applications is called a Support Vector Machine (SVM). It works by determining the best hyperplane to maximize the margin between data points of various classes in an N-dimensional space. (IBM, 2023)

Support vector machines (SVMs) are used for classification, regression and outliers detection. (Learn, 2024)

i. Types of SVM Classifiers**a. Linear SVM**

It is used when data is linearly separable, which means that classes may be accurately separated by a straight line (or hyperplane in higher dimensions). Finding the hyperplane that optimizes the margin between classes is the aim. (IBM, 2023)

Linear SVMs are designed for linearly separable data, where classes can be separated by a straight hyperplane without requiring transformations. The hyperplane is represented mathematically as:

$$w \cdot x + b = 0$$

where W is the weight vector, X is the input vector, and B is the bias term.

The margin, which is the distance between the hyperplane and the nearest data points (support vectors), can be calculated using two approaches:

$$(wx_j + b) y_j \geq a$$

and then the margin is maximized, which is represented as: $\max \gamma = a / \|w\|$, where a is the margin projected onto w . (IBM, 2023)

Soft-margin classification is more flexible which introduces slack variables (ξ) to allow for some misclassifications. The hyperparameter C controls the trade-off between a wider margin and classification errors:

- A larger C : Minimizes misclassification but results in a narrower margin.
- A smaller C : Allows for a wider margin but tolerates more misclassifications. (IBM, 2023)

b. Nonlinear SVMs

Nonlinear SVMs are useful in real-world situations when a large portion of the data cannot be separated linearly. The training data is transformed into a higher-dimensional feature space using preprocessing techniques to make it linearly separable. Nevertheless, because they become computationally demanding and increase the possibility of overfitting the data, larger dimensional spaces might introduce additional complexity. By substituting a comparable kernel function for dot product computations, the "kernel trick" serves to simplify the computation and increase its efficiency. (IBM, 2023)

The **kernel trick** addresses this issue by replacing dot product calculations with kernel functions, enabling efficient computation without explicitly transforming the data. Popular kernel functions include:

- Polynomial kernel
- Radial basis function kernel (also known as a Gaussian or RBF kernel)
- Sigmoid kernel (IBM, 2023)

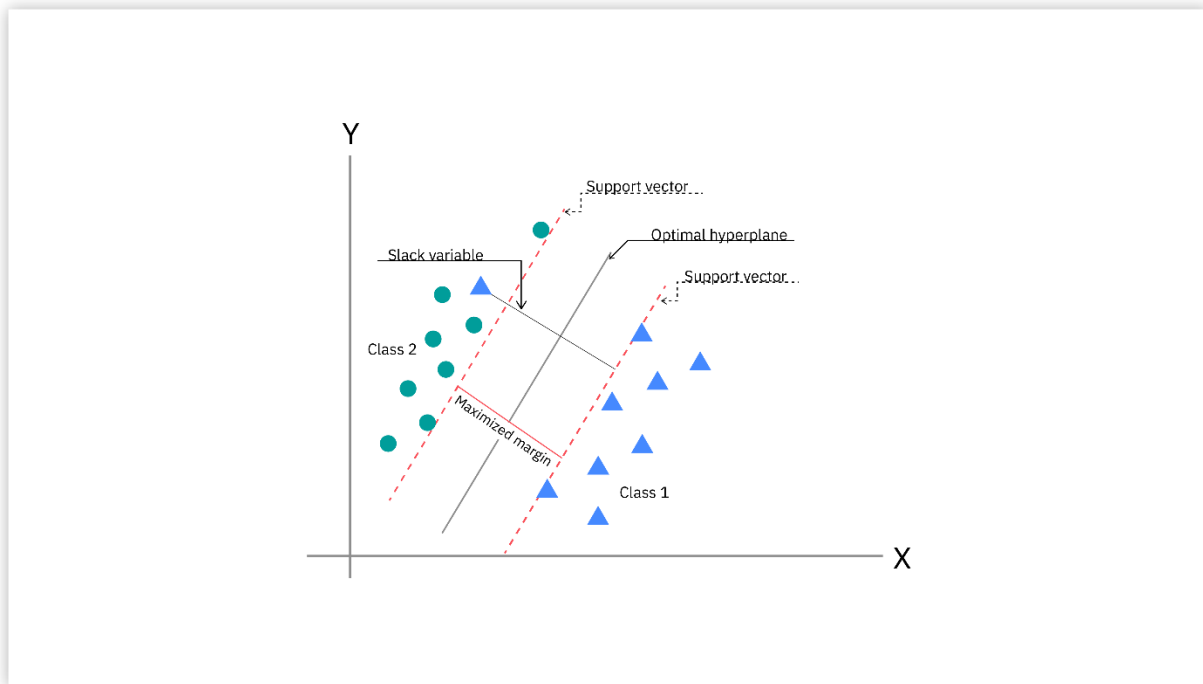


Figure 11: SVM: Optimal Hyperplane and Margin

B. Naïve Bayes Classifier

It is an algorithm that learns each object's probability, characteristics, and groupings. An alternative name for it is a probabilistic classifier. The Naive Bayes Algorithm is mostly used to tackle classification issues and falls under the category of supervised learning.

A bird's features and color, for instance, cannot be used to identify it because numerous birds have these characteristics. The Naive Bayes Algorithm is used when you produce a probabilistic prediction regarding the same thing.

At the core of Naive Bayes is Bayes' Theorem, which calculates the probability of a hypothesis given prior knowledge:

$$p(A|B) = \frac{p(B|A) * p(A)}{p(B)}$$

$p(A|B)$ - posterior

$p(A)$ - prior

$p(B)$ - evidence

$p(B|A)$ - likelihood

Figure 12: Bayes' Theorem Formula with Key Components(Lecture 6 slide)

i. **Types of Naïve Bayes Classifier** (Lecture 6 slide)

- Gaussian Naive Bayes:
 - Assumes that numerical features follow a **Gaussian (normal) distribution**.
 - Likelihood is computed using the probability density function of the Gaussian distribution.
- Multinomial Naive Bayes:
 - Used for discrete data like text classification (e.g., spam detection).
 - Likelihood is based on the frequency of features.
- Bernoulli Naive Bayes:
 - Used for binary data.
 - Features are binary (e.g., word presence or absence in a document).

C. Random Forest

Leo Breiman and Adele Cutler are the trademark holders of the widely used machine learning technique known as "random forest," which aggregates the output of several decision trees to produce a single outcome. Because it can handle both classification and regression issues, its versatility and ease of use have encouraged its use. (IBM, 2024)

Key Features of Random Forest:

- **Ensemble Learning:** By aggregating the results of numerous decision trees, Random Forest mitigates the risk of overfitting that individual trees may encounter. Each tree is trained on a different subset of the data, and their combined output leads to more accurate and stable predictions.
- **Bootstrap Aggregation (Bagging):** This technique involves creating multiple datasets by randomly sampling from the original dataset with replacement. Each decision tree is trained on a unique bootstrap sample, promoting diversity among the trees and reducing variance in the model's predictions.
- **Feature Randomness:** At each split in a decision tree, Random Forest considers a random subset of features rather than evaluating all possible features. This randomness decreases the correlation between trees, enhancing the ensemble's overall performance. (IBM, 2024)

For classification:

$$y = \text{majority vote of } \{T_1(x), T_2(x), \dots, T_n(x)\}$$

For regression:

$$y = \frac{1}{n} \sum_{i=1}^n T_i(x)$$

Where $T_i(x)$ is the prediction from the i -th decision tree for input x , and n is the number of trees.

Figure 13: Random Forest: Key Formula for Random Forest

3.3 Pseudocode of the solution(Updated)

A succinct, understandable explanation of an algorithm's or program's logic is called pseudocode. It outlines a process's phases using syntax similar to common language, making it simple for stakeholders, designers, and programmers to grasp. Pseudocode serves as a model for converting reasoning into actual code, but it is not a programming language.

➤ **Purpose and Benefits:**

- **Collaboration:** Facilitates communication among designers, developers, and other team members.
- **Validation:** Helps evaluate and refine logic before implementation, reducing costly errors later.
- **Flexibility:** Can be adapted into multiple programming languages.
- **Efficiency:** Provides a clear template for writing code that aligns with design specifications.

By simplifying complex ideas into a readable format, pseudocode ensures smooth collaboration and reduces errors, serving as a crucial step in the development process. (Sheldon, 2024)

3.3.1 Pseudocode of SVM**START****IMPORT** necessary libraries**IMPORT** dataset**READ** dataset**PREPROCESS** data (Tokenization, Lemmatization, TF-IDF)**SPLIT** dataset into training and testing sets**INITIALIZE** SVM hyperparameters (C, kernel)**DEFINE** GridSearchCV for hyperparameter tuning**PROCESS:****FIT** SVM model on training data**PREDICT** test labels**CALCULATE** metrics (Accuracy, Precision, Recall, F1 Score)**RETURN** Best parameters and metrics**END**

3.3.2 Pseudocode of Naïve bayes

START

IMPORT necessary libraries

IMPORT dataset

READ dataset

PREPROCESS data (Tokenization, Lemmatization, TF-IDF)

SPLIT dataset into training and testing sets

INITIALIZE Naive Bayes hyperparameters (alpha)

DEFINE GridSearchCV for hyperparameter tuning

PROCESS:

FIT Naive Bayes model on training data

PREDICT test labels

CALCULATE metrics (Accuracy, Precision, Recall, F1 Score)

RETURN Best parameters and metrics

END

3.3.3 Pseudocode of RandomForest

START

IMPORT necessary libraries

IMPORT dataset

READ dataset

PREPROCESS data (Tokenization, Lemmatization, TF-IDF)

SPLIT dataset into training and testing sets

INITIALIZE Random Forest hyperparameters (n_estimators, max_depth)

DEFINE GridSearchCV for hyperparameter tuning

PROCESS:

FIT Random Forest model on training data

PREDICT test labels

CALCULATE metrics (Accuracy, Precision, Recall, F1 Score)

RETURN Best parameters and metrics

END

3.4 Diagrammatic representations of the solution

3.4.1 Flowchart

For a better visual comprehension of the code, flowcharts are just a graphical representation of the data or method. It shows how to solve an issue, algorithm, or procedure step-by-step. It helps troubleshoot algorithm problems since it is a visual representation of stages that most beginning programmers like to comprehend computer science algorithms. An image of boxes showing the sequential process flow is called a flowchart. A flowchart makes it simple to analyze and comprehend a process or algorithm since it is a visual depiction of the process. All experts adhere to a set of guidelines when creating flowcharts, and these guidelines are generally recognized around the globe. (Geeks, 2023)

3.4.1.1 Flowchart of SVM

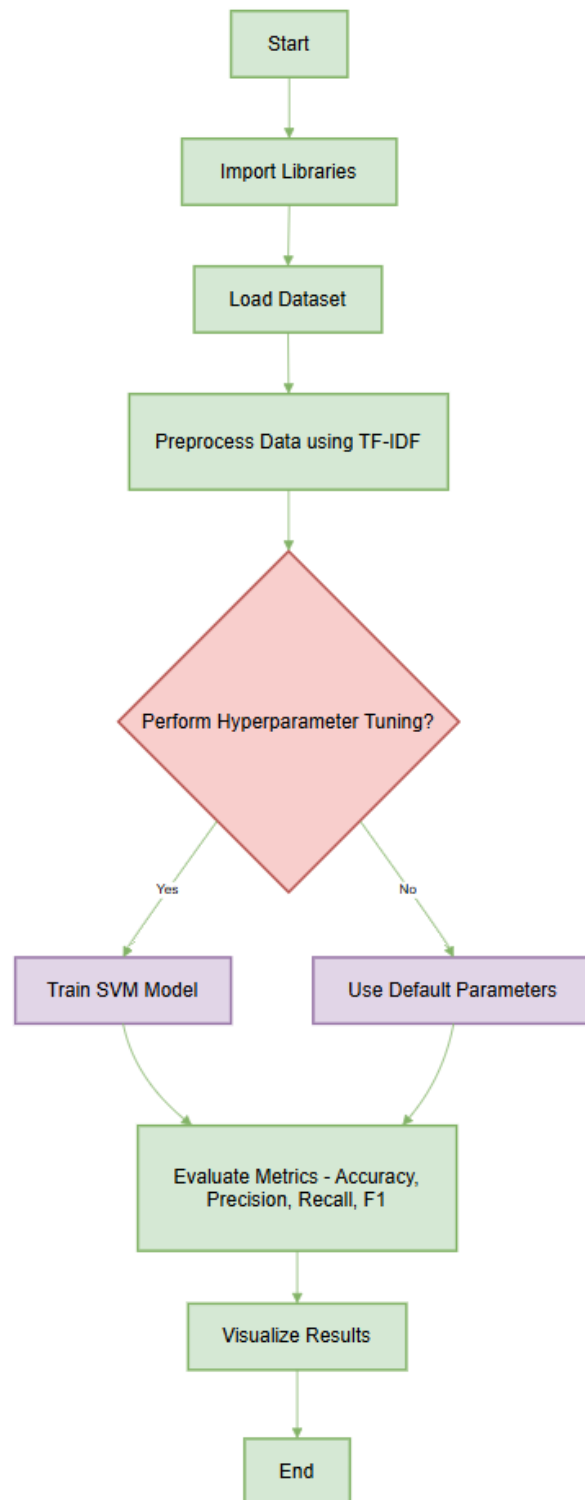


Figure 14: Flowchart: Flowchart diagram of SVM

3.4.1.2 Flowchart of Naïve Bayes

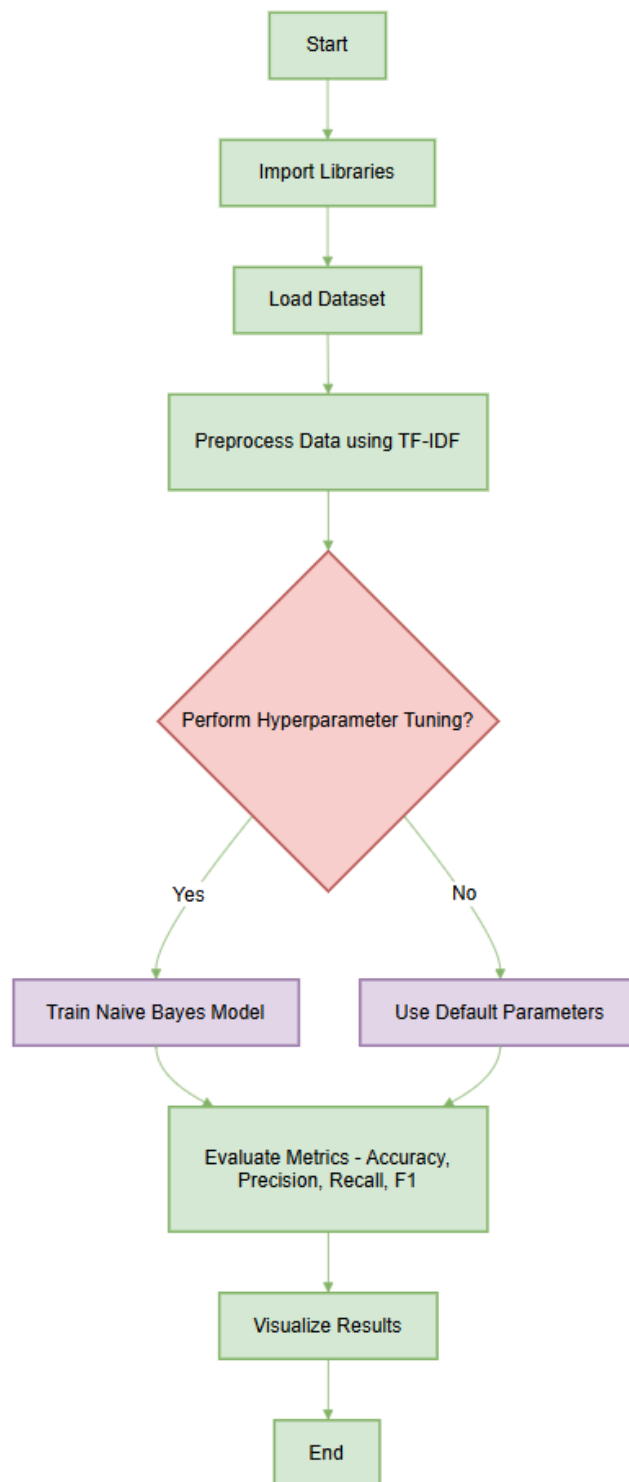


Figure 15: Flowchart: Flowchart of Naive Bayes

3.4.1.3 Flowchart of Random Forest

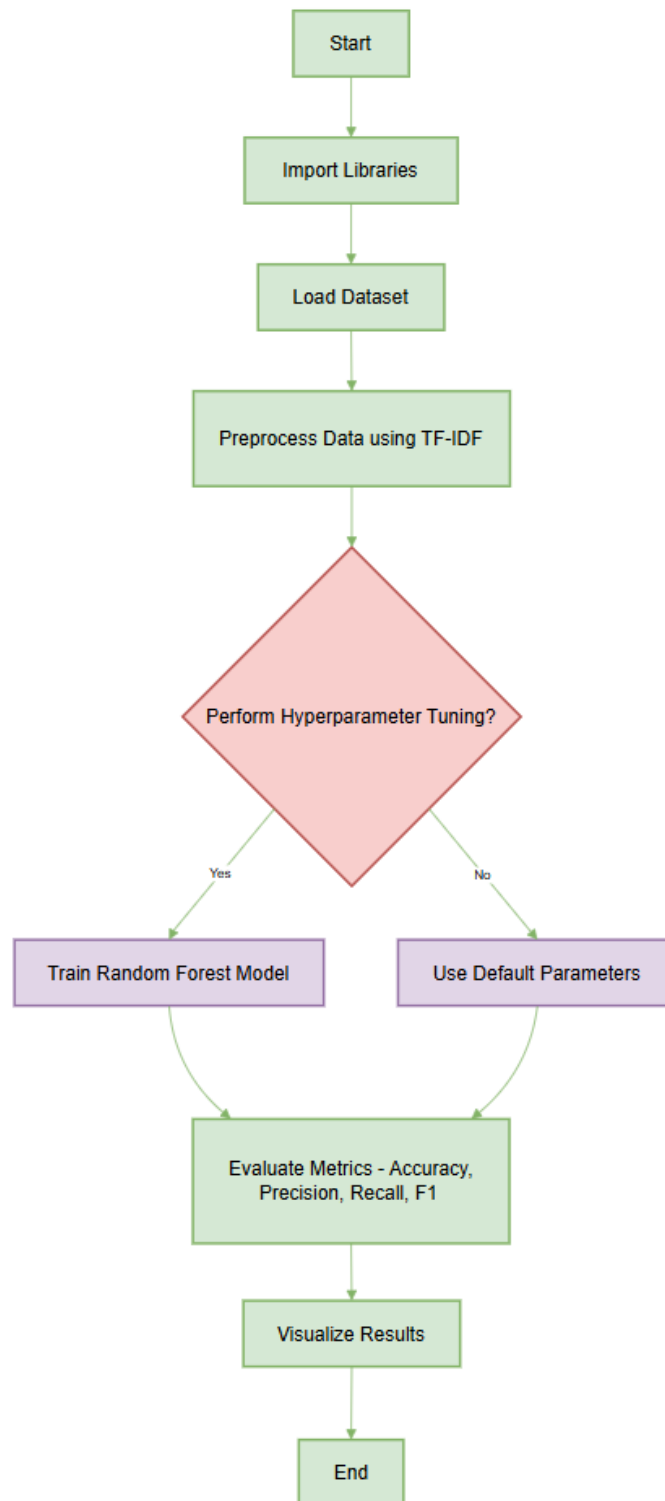


Figure 16: Flowchart: Flowchart diagram of Random Forest

4 Milestone 1 Work Done (Updated)

4.1 Installing and Preparing the Datasets Library

```
[7]: !pip install datasets

Requirement already satisfied: datasets in c:\users\user\anaconda3\lib\site-packages (3.2.0)
Requirement already satisfied: filelock in c:\users\user\anaconda3\lib\site-packages (from datasets) (3.13.1)
Requirement already satisfied: numpy>=1.17 in c:\users\user\anaconda3\lib\site-packages (from datasets) (1.26.4)
Requirement already satisfied: pyarrow>=15.0.0 in c:\users\user\anaconda3\lib\site-packages (from datasets) (16.1.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in c:\users\user\anaconda3\lib\site-packages (from datasets) (0.3.8)
Requirement already satisfied: pandas in c:\users\user\anaconda3\lib\site-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in c:\users\user\anaconda3\lib\site-packages (from datasets) (2.32.3)
Requirement already satisfied: tqdm>=4.66.3 in c:\users\user\anaconda3\lib\site-packages (from datasets) (4.66.5)
Requirement already satisfied: xxhash in c:\users\user\anaconda3\lib\site-packages (from datasets) (3.5.0)
Requirement already satisfied: multiprocessing<0.70.17 in c:\users\user\anaconda3\lib\site-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2024.9.0,>=2023.1.0 in c:\users\user\anaconda3\lib\site-packages (from datasets[http]<=2024.9.0,>=2023.1.0->dataset
s) (2024.6.1)
Requirement already satisfied: aiohttp in c:\users\user\anaconda3\lib\site-packages (from datasets) (3.10.5)
Requirement already satisfied: huggingface-hub>=0.23.0 in c:\users\user\anaconda3\lib\site-packages (from datasets) (0.27.1)
Requirement already satisfied: packaging in c:\users\user\anaconda3\lib\site-packages (from datasets) (24.1)
Requirement already satisfied: pyyaml>=5.1 in c:\users\user\anaconda3\lib\site-packages (from datasets) (6.0.1)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in c:\users\user\anaconda3\lib\site-packages (from aiohttp->datasets) (2.4.0)
Requirement already satisfied: aiosignal>=1.1.2 in c:\users\user\anaconda3\lib\site-packages (from aiohttp->datasets) (1.2.0)
Requirement already satisfied: attrs>=17.3.0 in c:\users\user\anaconda3\lib\site-packages (from aiohttp->datasets) (23.1.0)
Requirement already satisfied: frozenlist>=1.1.1 in c:\users\user\anaconda3\lib\site-packages (from aiohttp->datasets) (1.4.0)
Requirement already satisfied: multidict<7.0,>=4.5 in c:\users\user\anaconda3\lib\site-packages (from aiohttp->datasets) (6.0.4)
Requirement already satisfied: yarl<2.0,>=1.0 in c:\users\user\anaconda3\lib\site-packages (from aiohttp->datasets) (1.11.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in c:\users\user\anaconda3\lib\site-packages (from huggingface-hub>=0.23.0->datasets) (4.11.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\user\anaconda3\lib\site-packages (from requests>=2.32.2->datasets) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\user\anaconda3\lib\site-packages (from requests>=2.32.2->datasets) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\user\anaconda3\lib\site-packages (from requests>=2.32.2->datasets) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\user\anaconda3\lib\site-packages (from requests>=2.32.2->datasets) (2024.12.14)
Requirement already satisfied: colorama in c:\users\user\anaconda3\lib\site-packages (from tqdm>=4.66.3->datasets) (0.4.6)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\user\anaconda3\lib\site-packages (from pandas->datasets) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\user\anaconda3\lib\site-packages (from pandas->datasets) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\user\anaconda3\lib\site-packages (from pandas->datasets) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\user\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas->datasets) (1.16.0)
```

Figure 17: Output of Installing the Datasets Library

The first step is to get my selected dataset into my jupyter notebook for that purpose I have used `!pip install datasets` this command to install dataset library which is a tool provided by hugging face to access and manipulate datasets. (HuggingFace, 2024)

The output here shows that the library and its dependencies such as numpy, panda and hugginface-hub, are already installed which lets know that environment is prepared for loading and working with datasets without any problem.

4.2 Importing Necessary Libraries and Modules

```
[8]: from datasets import load_dataset
      from sklearn.feature_extraction.text import TfidfVectorizer
      from sklearn.model_selection import train_test_split
      from sklearn.naive_bayes import MultinomialNB
      from sklearn.svm import SVC
      from sklearn.ensemble import RandomForestClassifier
      from sklearn.metrics import accuracy_score
      import matplotlib.pyplot as plt
```

Figure 18: Importing Libraries for Data Processing and Modeling

Here in this image all the necessary libraries and modules which are essential and required for the dataset and building machine learning models are imported. The libraries include `load_dataset` from Hugging face dataset library which helps in easy loading of dataset, `TfidfVectorizer` from `sklearn.feature_extraction.text` which helps in converting text into numerical features using TF-IDF method.

Other various helpful libraries are imported like `train_test_split` for splitting the dataset into training and testing sets, and machine learning algorithms like `MultinomialNB` (Naive Bayes), `SVC` (Support Vector Machine), and `RandomForestClassifier`. Additionally, `accuracy_score` from `sklearn.metrics` is used to evaluate the performance of the models, and `matplotlib.pyplot` is imported for visualizing the results.

These imports will help that all tools and models will be ready for the upcoming data processing and modeling steps.

4.3 Loading the Dataset

```
[11]: data = load_dataset("dair-ai/emotion")
```

Figure 19: Loading the Emotion Dataset

This part includes loading the dataset using `load_dataset` function from `hugging face datasets` library. The dataset which is being loaded here is named "dair-ai/emotion" dataset. Assigning variable `data` to this dataset will make it accessible for further processes.

4.4 Extracting Texts and Labels from the Dataset

```
[12]: texts = data['train']['text']  
      labels = data['train']['label']
```

Figure 20: Separating Text Data and Labels

Here dataset for further processing is being prepared. The variable `texts` is assigned the text data whereas variable `labels` is assigned the corresponding emotion labels. The separation of text and label is a must for preprocessing and training ML models as the text data will be used as input whereas labels will be as the target output for the models.

4.5 Previewing Heading

```
[13]: texts[0:6]  
  
[13]: ['i didnt feel humiliated',  
      'i can go from feeling so hopeless to so damned hopeful just from being around someone who cares and is awake',  
      'im grabbing a minute to post i feel greedy wrong',  
      'i am ever feeling nostalgic about the fireplace i will know that it is still on the property',  
      'i am feeling grouchy',  
      'ive been feeling a little burdened lately wasnt sure why that was']
```

Figure 21 Sample of Extracted Text Data:

The `texts[0:6]` is used here to display the first six sentences in the dataset which is like showing a little sample of text data from the dataset.

4.6 Splitting Data into Training and Testing Sets

```
[19]: X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.2, random_state=42)
```

Figure 22: Dataset Split for Model Training and Evaluation

This step is crucial as this code splits dataset into training and testing sets using the `train_test_split` function. The texts which are input datas and labels which are output datas. These are divided into `X_train`, `X_test`, `y_train` and `y_test`. The `test_size=0.2` parameter specifies that 20% of the datas will be used for testing while 80% of them will be used for training. The `random_state=42` guarantees duplicability by using the same random split each time the code is run.

4.7 Transforming Text Data Using TF-IDF Vectorizer

```
[21]: vectorizer = TfidfVectorizer()  
X_train_vec = vectorizer.fit_transform(X_train)  
X_test_vec = vectorizer.transform(X_test)
```

Figure 23: Applying TF-IDF to Train and Test Data

This step helps in the conversion of textual data into a numerical format using the TF-IDF Vectorizer. The `TfidfVectorizer` is useful in transforming input text into a matrix of TF-IDF features. In other words, it shows the importance of words within the text.

- `TfidfVectorizer()` is instantiated and assigned to the variable, `vectorizer`.
- The `fit_transform` method is applied to `X_train`, which learns the vocabulary from the training data and then converts it into a numerical representation: `X_train_vec`.
- Now the `transform` method is applied to `X_test` using the vocabulary learned from the training data to transform the test data in a similar numerical form, and it's named `X_test_vec`.

4.8 Implementing the Naive Bayes Model

```
[23]: # Naive Bayes Model
nb_model = MultinomialNB()
nb_model.fit(X_train_vec, y_train)
nb_y_pred = nb_model.predict(X_test_vec)
nb_accuracy = accuracy_score(y_test, nb_y_pred)
print(f"Naive Bayes Accuracy: {nb_accuracy:.2f}")

Naive Bayes Accuracy: 0.62
```

Figure 24: Naive Bayes Model Accuracy

From here starts the model work, In this step we are using Naive Bayes model and let it try to understand and classify emotions from text. Naive Bayes model is known for its simplicity and its efficiency with text types of data which is initialized as nb_model with training data X_train_vec and y_train which we created earlier now we teach the model to recognize patterns in words and how they connect to some definite emotions.

Once the model is trained we let the model put its knowledge into a new data which is has not been seen (X_test_vec) by predicting the emotions it think the text will represent. The results then are compared with the actual emotions to see how well the model has learned.

The final accuracy is 62% which suggests the models ability to predict emotions from the text.

4.8.1 Naïve Bayes Precision Score

```
[66]: nb_precision = precision_score(y_test, nb_y_pred, average='weighted', zero_division=0)
print(f"Naive Bayes Precision: {nb_precision:.2f}")

Naive Bayes Precision: 0.71
```

Figure 25: Naive Bayes Precision Score

The figure above shows that the precision of the Naïve Bayes model is 0.71. it is possible because of the precision_score function. Here in the provided snippet the average='weighted' parameter helps in ensuring that precision is averaged in all classes based on their frequency and the zero_division=0 this part defends errors for undefined cases. The result is formatted to two decimal places and printed.

4.8.2 Naïve Bayes F1-Score

```
[29]: nb_f1 = f1_score(y_test, nb_y_pred, average='weighted', zero_division=0)
      print(f"Naive Bayes F1 Score: {nb_f1:.2f}")

Naive Bayes F1 Score: 0.52
```

Figure 26: Naive Bayes F1 Score

Here in this above picture F1 score is calculated programmatically using `f1_score` function from `sklearn.metrics` which is imported in the top of the code where all the necessary libraries are imported. This `f1_score` function internally applies the formula:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Now In the code `y_test` represents the true labels and `nb_y_pred` is the predicted labels which are from Naïve Bayes model. The `average='weighted'` parameter helps in ensuring that F1 score is weighted by the number of true instances for each class and the `zero_division=0` prevents errors in cases where precision or recall is undefined. The result 0.52 reflects the weighted F1 score based on the class distribution and model performance. This F1 score calculated indicates that model needs improvement by fine tuning it.

4.8.3 Naïve Bayes Recall Score.

```
[117]: nb_recall = recall_score(y_test, nb_y_pred, average='weighted', zero_division=0)
       print(f"Naive Bayes Recall: {nb_recall:.2f}")

Naive Bayes Recall: 0.62
```

Figure 27: Naive Bayes Recall Score

In the above picture the weighted recall for the Naïve Bayes model using the `recall_score` function from `sklearn.metrics` is calculated. The `recall_score` function internally applies the formula:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negatives}$$

In the code `average='weighted'` parameter ensures that the recall is calculated as an average. The `zero_division=0` parameter helps in avoiding error when there are no positive prediction for a class by setting the initial recall to 0 for that class. The recall calculate i.e. 0.62 or 62% means that 62% of the actual positive samples, considering the

size of each class. This means that the model is working reasonably well but it could perform well for some classes which can be done by fine tuning it.

4.9 Implementing Support Vector Machines Model

```
[25]: # Support Vector Machine Model
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train_vec, y_train)
svm_y_pred = svm_model.predict(X_test_vec)
svm_accuracy = accuracy_score(y_test, svm_y_pred)
print(f"SVM Accuracy: {svm_accuracy:.2f}")

SVM Accuracy: 0.88
```

Figure 28: SVM Model Accuracy in Emotion Classification

Now we use SVM model to classify emotions with precision. The model `svm_model`, is created with a linear kernel which I am setting it up so that it handles text data effectively. The models will learn the complicated patterns and relationships between the words and their corresponding emotions with `X_train_vec` and `y_train`.

After training the model it will with confidence predict the emotion for the test data (`X_test_vec`) those predictions are compared to the actual labels (`y_test`) to measure the models accuracy which is 88%.

4.9.1 SVM Precision Score

```
[71]: svm_precision = precision_score(y_test, svm_y_pred, average='weighted', zero_division=0)
print(f"SVM Precision: {svm_precision:.2f}")

SVM Precision: 0.88
```

Figure 29: SVM Precision Score

Here in this code snippet the precision of SVM is calculated using the `precision_score` function which calculated the precision of 0.88. The `average='weighted'` parameter helps in ensuring that precision is averaged in all classes based on their frequency and the `zero_division=0` this part defends errors for undefined cases. The result is formatted to two decimal places and printed, reflecting the precision of the model's positive predictions.

4.9.2 SVM F1 Score

```
[72]: svm_f1 = f1_score(y_test, svm_y_pred, average='weighted', zero_division=0)
      print(f"SVM F1 Score: {svm_f1:.2f}")
```

SVM F1 Score: 0.88

Figure 30: SVM F1 Score

Here in this above picture F1 score is calculated programmatically using `f1_score` function from `sklearn.metrics` which is imported in the top of the code where all the necessary libraries are imported. This `f1_score` function internally applies the formula:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Now In the code `y_test` represents the true labels and `nb_y_pred` is the predicted labels which are from Naïve Bayes model. The `average='weighted'` parameter helps in ensuring that F1 score is weighted by the number of true instances for each class and the `zero_division=0` prevents errors in cases where precision or recall is undefined. The result 0.88 shows this score indicates a well-performing model with a strong balance between precision and recall, suitable for tasks requiring both metrics to be optimized.

4.9.3 SVM Recall Score

```
[101]: svm_recall = recall_score(y_test, svm_y_pred, average='weighted', zero_division=0)
      print(f"SVM Recall: {svm_recall:.2f}")
```

SVM Recall: 0.88

Figure 31: SVM Recall Score

In the above picture the weighted recall for the Naïve Bayes model using the `recall_score` function from `sklearn.metrics` is calculated. The `recall_score` function internally applies the formula:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negatives}$$

In the code `average='weighted'` parameter ensures that the recall is calculated as an average. The `zero_division=0` parameter helps in avoiding error when there are no positive prediction for a class by setting the initial recall to 0 for that class. The recall calculated i.e. 0.88 or 88% means that it has a strong ability to detect related instances.

This is particularly important in the case where missing a positive instances has higher consequences than a false positive.

4.10 Implementing Random Forest Model

```
[26]: # Random Forest Model
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train_vec, y_train)
rf_y_pred = rf_model.predict(X_test_vec)
rf_accuracy = accuracy_score(y_test, rf_y_pred)
print(f"Random Forest Accuracy: {rf_accuracy:.2f}")
```

Random Forest Accuracy: 0.85

Figure 32: Random Forest Model Accuracy

Random Forest Model is a powerful ensemble learning technique. The `rf_model` model is built using a forest of decision trees each giving the final prediction by training on `X_train_vec` and `y_train`. The model learns to classify emotion by leveraging the combined insight of multiple trees.

Once the model is trained the model makes predictions on the test data (`X_test_vec`) and the results (`rf_y_pred`) are compared with the actual labels (`y_test`) to calculate accuracy. This model achieved 85% accuracy showing its ability to handle distinct patterns in the data.

4.10.1 Random Forest Precision Score

```
[78]: rf_precision = precision_score(y_test, rf_y_pred, average='weighted', zero_division=0)
print(f"Random Forest Precision: {rf_precision:.2f}")
```

Random Forest Precision: 0.86

Figure 33: Random Forest Precision Score

The figure above shows that the precision of the Random Forest model is 0.86. It is possible because of the `precision_score` function. Here, in the provided snippet, the `average='weighted'` parameter helps in ensuring that precision is averaged in all classes based on their frequency, and the `zero_division=0` part defends against errors for undefined cases. The result is formatted to two decimal places and printed.

4.10.2 Random Forest F1 Score

```
103]: rf_f1 = f1_score(y_test, rf_y_pred, average='weighted', zero_division=0)
      print(f"Random Forest F1 Score: {rf_f1:.2f}")
```

Random Forest F1 Score: 0.85

Figure 34: Random Forest F1 Score

Here in this above picture F1 score is calculated programmatically using `f1_score` function from `sklearn.metrics` which is imported in the top of the code where all the necessary libraries are imported. This `f1_score` function internally applies the formula:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Now In the code `y_test` represents the true labels and `nb_y_pred` is the predicted labels which are from Naïve Bayes model. The `average='weighted'` parameter helps in ensuring that F1 score is weighted by the number of true instances for each class and the `zero_division=0` prevents errors in cases where precision or recall is undefined. The result, **0.85**, reflects the weighted F1 score based on the class distribution and model performance. This F1 score indicates that the model achieves a reliable balance between precision and recall, making it suitable for tasks requiring consistent performance across all classes.

4.10.3 Random Forest Recall Score

```
[105]: rf_recall = recall_score(y_test, rf_y_pred, average='weighted', zero_division=0)
      print(f"Random Forest Recall: {rf_recall:.2f}")
```

Random Forest Recall: 0.85

Figure 35: Random Forest Recall Score

In the above picture the weighted recall for the Naïve Bayes model using the `recall_score` function from `sklearn.metrics` is calculated. The `recall_score` function internally applies the formula:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negatives}$$

In the code `average='weighted'` parameter ensures that the recall is calculated as an average. The `zero_division=0` parameter helps in avoiding error when there are no

positive prediction for a class by setting the initial recall to 0 for that class. The recall calculated, i.e., 0.85 or 85%, means that 85% of the actual positive samples, considering the size of each class, are correctly identified. This means that the model is working reasonably well, but it could perform even better for some classes, which can be achieved by fine-tuning it.

4.11 Visualizing Model Accuracy Scores

```
[27]: accuracy_scores = {
      'Naive Bayes': nb_accuracy,
      'SVM': svm_accuracy,
      'Random Forest': rf_accuracy
    }

    plt.figure(figsize=(8, 6))
    plt.bar(accuracy_scores.keys(), accuracy_scores.values(), color=['blue', 'green', 'orange'])
    plt.title('Model Accuracy Comparison')
    plt.ylabel('Accuracy')
    plt.xlabel('Models')
    plt.ylim(0, 1)
    plt.show()
```

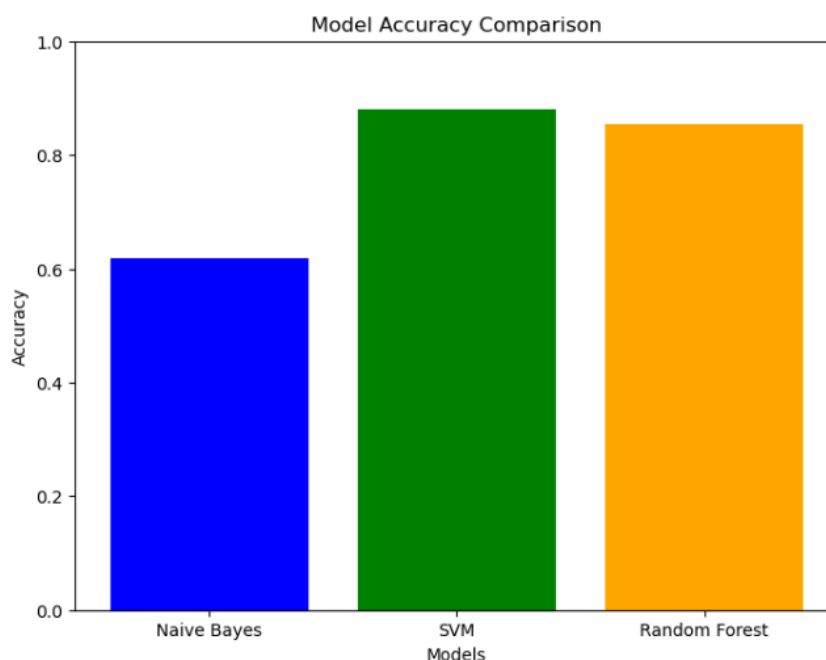


Figure 36: Model Accuracy Comparison Chart

The bar chart shows that SVM model has the highest accuracy than that of Random forest and Naïve Bayes. Then Random Forest follow up SVM model with an accuracy of 0.85 while Naïve Bayes has the least accuracy which is 0.62. It has become easier to visualize and understand which performed well in emotion classification analysis.

4.12 Visualizing Model Precision Scores

```
[85]: precision_scores = {'Naive Bayes': nb_precision, 'SVM': svm_precision, 'Random Forest': rf_precision}

plt.figure(figsize=(8, 6))
plt.bar(precision_scores.keys(), precision_scores.values(), color=['cyan', 'purple', 'brown'])
plt.title('Model Precision Comparison')
plt.ylabel('Precision')
plt.xlabel('Models')
plt.ylim(0, 1)
plt.show()
```

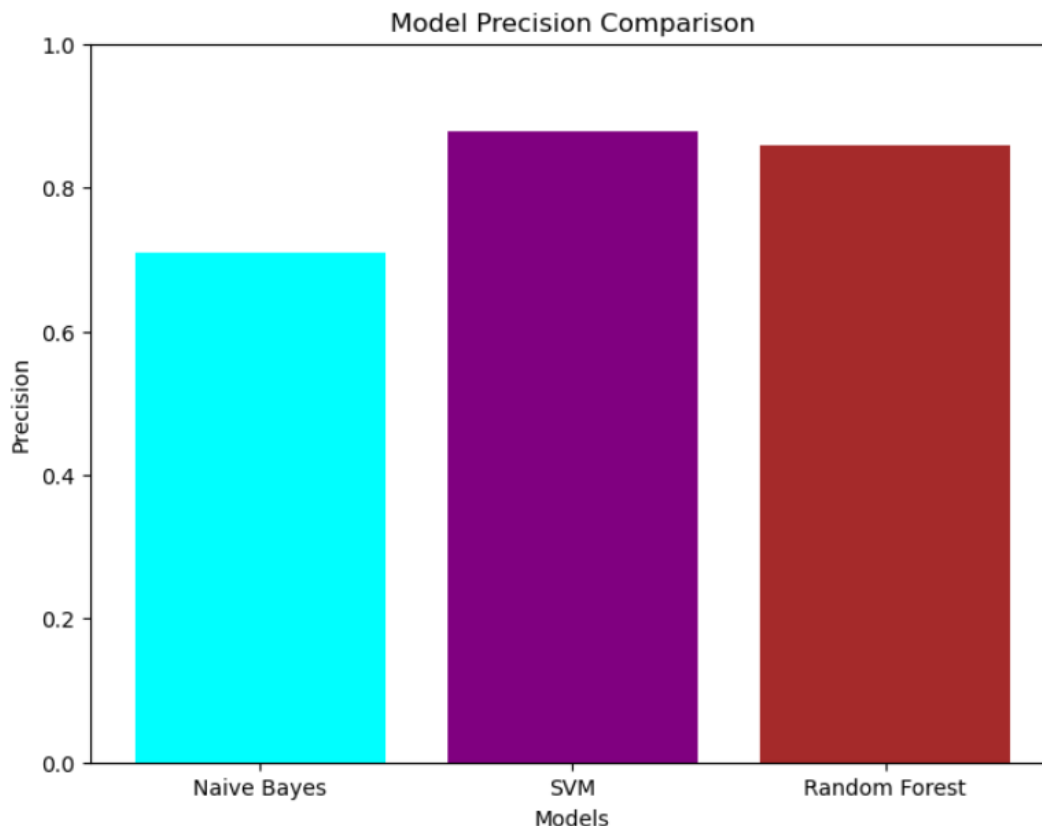


Figure 37: Model Precision Comparison Chart

The bar chart shows that the SVM model has the highest precision compared to Random Forest and Naïve Bayes. Then Random Forest follows up the SVM model with a precision of 0.88, while Naïve Bayes has the least precision, which is 0.71. It has become easier to visualize and understand which performed well in emotion classification analysis.

4.13 Visualizing Model F1 Scores

```
[93]: f1_scores = {'Naive Bayes': nb_f1, 'SVM': svm_f1, 'Random Forest': rf_f1}

plt.figure(figsize=(8, 6))
plt.bar(f1_scores.keys(), f1_scores.values(), color=['pink', 'teal', 'navy'])
plt.title('Model F1 Score Comparison')
plt.ylabel('F1 Score')
plt.xlabel('Models')
plt.ylim(0, 1)
plt.show()
```

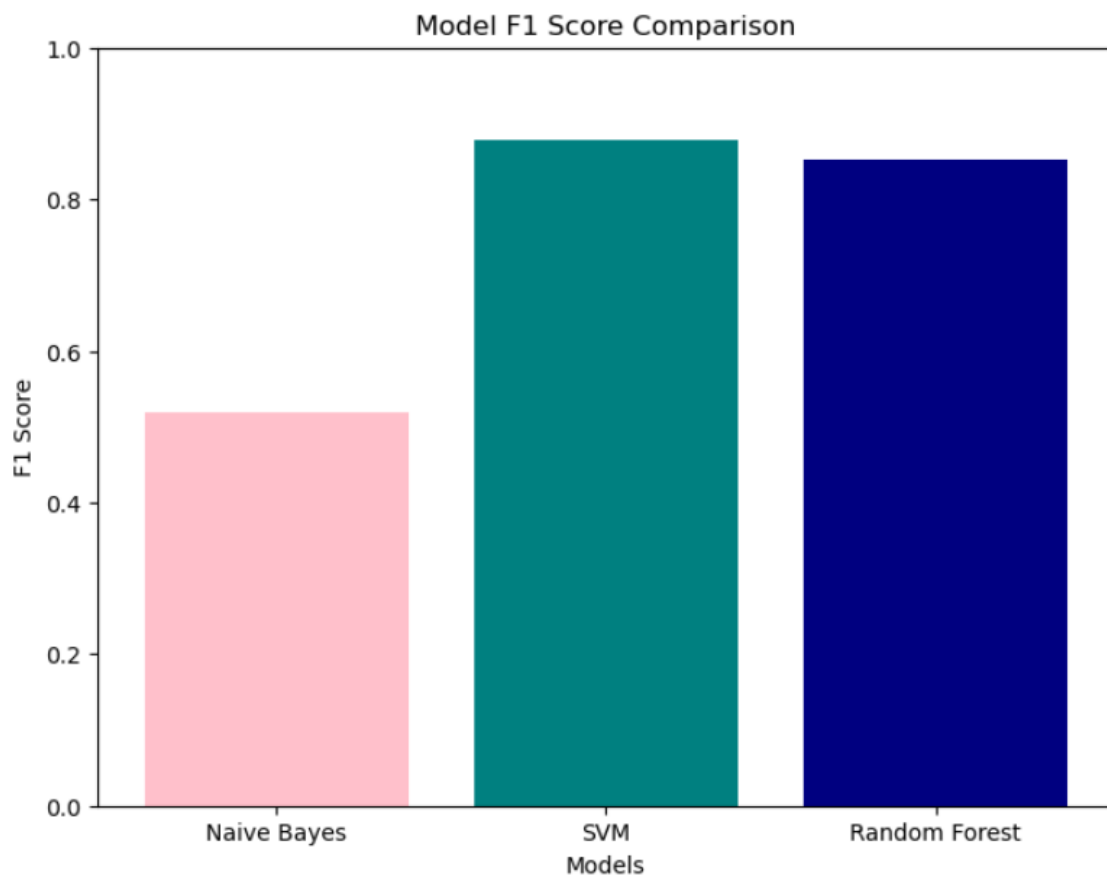


Figure 38: Model F1 Score Comparison Chart

The bar chart shows that the SVM model has the highest F1 score compared to Random Forest and Naïve Bayes. Then Random Forest follows up the SVM model with an F1 score of 0.85, while Naïve Bayes has the least F1 score, which is 0.52. It has become easier to visualize and understand which performed well in emotion classification analysis.

4.14 Visualizing Model Recall Score

```
[109]: recall_scores = {  
        'Naive Bayes': nb_recall, 'SVM': svm_recall, 'Random Forest': rf_recall  
    }  
  
    plt.figure(figsize=(8, 6))  
    plt.bar(recall_scores.keys(), recall_scores.values(), color=['cyan', 'purple', 'brown'])  
    plt.title('Model Recall Comparison')  
    plt.ylabel('Recall')  
    plt.xlabel('Models')  
    plt.ylim(0, 1)  
    plt.show()
```

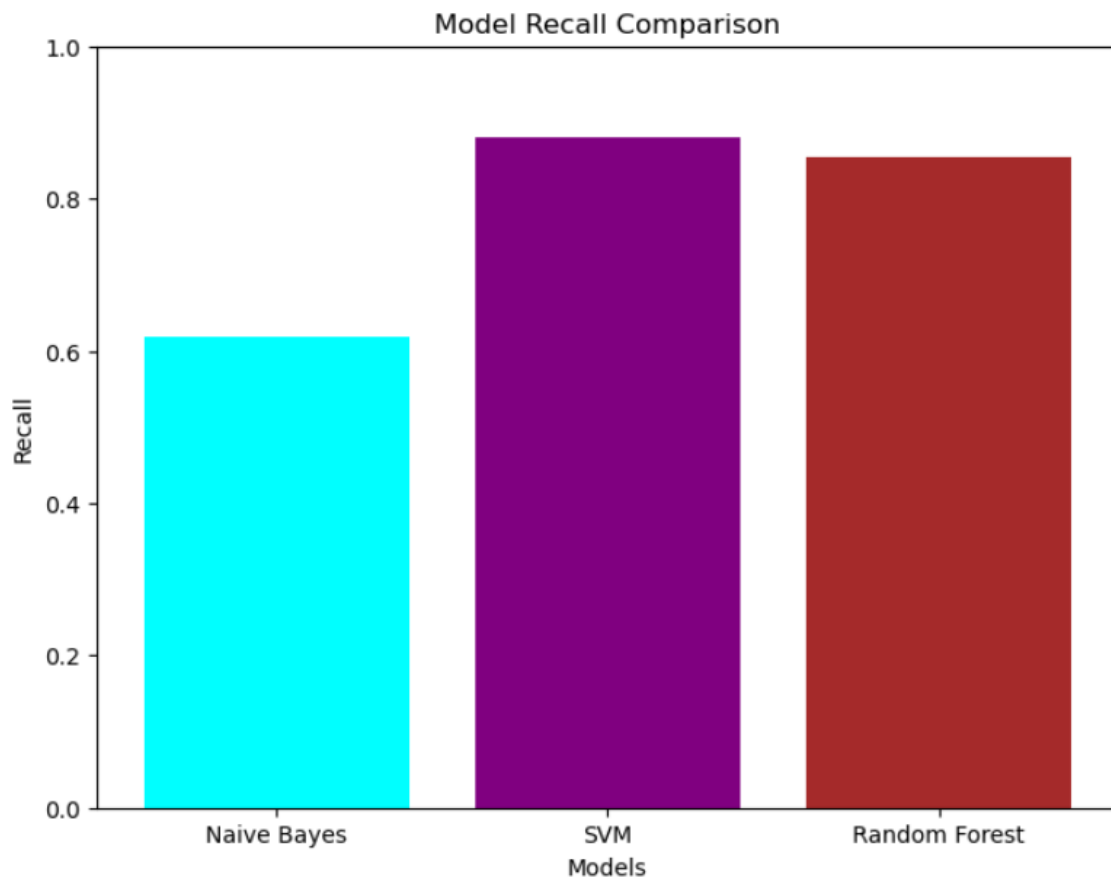


Figure 39: : Model Recall Score Comparison Chart

The bar chart shows that the SVM model has the highest recall compared to Random Forest and Naïve Bayes. Then Random Forest follows up the SVM model with a recall of 0.85, while Naïve Bayes has the least recall, which is 0.62. It has become easier to visualize and understand which performed well in emotion classification analysis.

5 Milestone 2 Work Done

5.1.1 Essential Library Imports for Model Training and Evaluation

```
[5]: from datasets import load_dataset
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.model_selection import train_test_split
    from sklearn.naive_bayes import MultinomialNB
    from sklearn.svm import SVC
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import accuracy_score
    import matplotlib.pyplot as plt
    from sklearn.metrics import precision_score
    from sklearn.metrics import f1_score
    from sklearn.metrics import recall_score
    from sklearn.metrics import precision_score, recall_score, accuracy_score
```

Figure 40: Importing Necessary Libraries

I have imported additional libraries beyond those used in the previous milestone. These include:

- ❖ `nltk.stem.WordNetLemmatizer`: For reducing words to their base or root form (lemmatization). (NLTK, 2025)
- ❖ `nltk.corpus.stopwords`: Provides a list of common stop words for text preprocessing. (NLTK, 2025)
- ❖ `nltk`: A library for various natural language processing tasks and utilities.
- ❖ `confusion_matrix` and `ConfusionMatrixDisplay`: To calculate and visualize confusion matrices for evaluating classification models. (NLTK, 2025)
- ❖ `GridSearchCV`: For hyperparameter tuning of machine learning models. (NLTK, 2025)

5.1.2 Downloading NLTK Resources For Text Preprocessing

```
[66]: nltk.download("wordnet")
      nltk.download("omw-1.4")
      nltk.download("stopwords")

[nltk_data] Downloading package wordnet to
[nltk_data] C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data] Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data] C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data] Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!

[66]: True
```

Figure 41: Downloading NTLK Resources

Here in this snippet specific resources from the NTLK which refers to Natural Language Toolkit library which is a library for natural language processing tasks has been downloaded. The downloaded resources have been explained below:

❖ `nltk.download("wordnet")`

- This command here downloads the WordNet Package which is a large vocabulary database of English. It will be used for lemmatization in the further process by finding word synonym, antonyms and many more. (NTLK, 2025)

❖ `nltk.download("omw-1.4")`

- this command downloads the open Multilingual WordNet (OMW) which will provide multilingual words, definitions and translation if it will be required. It is often used alongside the WordNet for multilingual NLP tasks. (NTLK, 2025)

❖ `nltk.download("stopwords")`

- This command here downloads a predefined list of common stopwords in various languages such as "is," "the," and "and," which are often removed during text preprocessing. (NTLK, 2025)

5.1.3 Loading and Converting the Dataset for Analysis

```
[69]: data = load_dataset("dair-ai/emotion")  
      df = pd.DataFrame(data["train"])
```

Figure 42: Loading The Dataset

This code snippet loads the “dair-ai-emotion” dataset from hugging face library using Loading and Converting the Dataset for Analysis

5.1.4 Data Preprocessing

It is the important stage of machine learning, basically cleaning, transformation, and organization of raw data in order for it to get suitable for analysis. It ensures that such data is free from errors, does not contain any missing values or inconsistencies, and enhances its quality for reliable insights. The main steps usually involved are handling missing data, normalization, scaling, encoding categorical variables, and removing noise; all these altogether optimize the dataset for the machine learning algorithm. Data preprocessing improves the precision and performance of models. Overhead computation reductions make way for even more efficient and robust predictions. With these base issues, the organizations can maximize the data that is held in it and make it possible to produce the best output of the decisions. (Novogroder, 2024)

5.1.4.1 Checking for Missing Values in the Dataset

```
[72]: df.isnull().sum()

[72]: text      0
      label     0
      dtype: int64
```

Figure 43: Text Preprocessing Function: Checking Missing Values

The code snippet uses `df.isnull().sum()` to make sure of any missing values in the dataset which was loaded previously it calculates the total number of null values for each column in the dataframe. The result makes it clear that both the text and label columns have zero missing values which is a good indication that the dataset is clean of null values and is ready for further analysis.

5.1.4.2 Lemmatization process

```
[74]: lemmatizer = WordNetLemmatizer()  
      stop_words = set(stopwords.words("english"))
```

Figure 44: Text Preprocessing Function: Lemmatization of the Data

This snippet is one of the main part of the data preprocessing, this here is the lemmatization process which reduce the words to their base or root form by using the WordNetLemmatizer which is a class provided by nltk.stem module

For an example if there is a word running it would be reduced to run which is runnings root word.

Additionally, stopwords.words("english") retrieves a set of common English stopwords like "the," "is," and "and," which are stored in a set for efficient removal during text cleaning. These tools are crucial for preparing clean and meaningful text data for NLP tasks.

5.1.4.3 Text Preprocessing Function for Lemmatization and Stopword Removal

```
[76]: def preprocess_text(text):  
      tokens = text.split()  
      lemmatized = [lemmatizer.lemmatize(word.lower()) for word in tokens if word.lower() not in stop_words]  
      return " ".join(lemmatized)
```

Figure 45: Text Preprocessing Function: Text Preprocessing Function Here in this snippet

The above code explains that the function `preprocess_text` performs text preprocessing by tokenizing, lemmatizing, and removing stopwords. It first splits the input text into tokens which are individual words. After then it uses `lemmatizer.lemmatize()` which is a list comprehension to lemmatize each word and converts the word to lowercase.

During the process words that are present in the `stop_words` set are excluded. Finally after all this processing the lemmatized tokens are joined back into single string and returned.

This function helps in ensuring that the text is cleaned and standardized and is ready for further processing tasks.

5.1.4.4 Applying Preprocessed Text to the Dataset

```
[78]: df["text"] = df["text"].apply(preprocess_text)
```

Figure 46: Data Preprocessing: Applying Processed Text Into The Dataset

In the above code the `preprocess_text` function applies to the text column of the DataFrame `df`. The processed text replaces the original text in the "text" column, ensuring the data is cleaned and standardized for further analysis.

5.1.4.5 Splitting Data into Training and Testing Sets

```
[80]: X = df["text"]
      y = df["label"]
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

This step is crucial as this code splits dataset into training and testing sets using the `train_test_split` function. The tests which are input datas and lables which are output datas. These are divided into `X_train`, `X_test`, `y_train` and `y_test`. The `test_size=0.2` parameters specifies that 20% of the datas will be used for testing while 80% of them will be used for testing. The `random_state=42` guarantees duplicability by using the same random split each time the code is runed.

5.1.4.6 TF-IDF Vectorization of Text Data

```
[82]: # TF-IDF Vectorization
      tfidf = TfidfVectorizer()
      X_train_tfidf = tfidf.fit_transform(X_train)
      X_test_tfidf = tfidf.transform(X_test)
```

This step helps in the conversion of textual data into a numerical format using the TF-IDF Vectorizer. The `TfidfVectorizer` is useful in transforming input text into a matrix of TF-IDF features. In other words, it shows the importance of words within the text.

- `TfidfVectorizer()` is instantiated and assigned to the variable, `vectorizer`.
- The `fit_transform` method is applied to `X_train`, which learns the vocabulary from the training data and then converts it into a numerical representation: `X_train_vec`.

Now the `transform` method is applied to `X_test` using the vocabulary learned from the training data to transform the test data in a similar numerical form, and it's named `X_test_vec`.

6 Fine Tuning

Fine-tuning is a process that takes a pre-trained machine learning model and adapts it to solve a specific task or work on a specialized dataset. It builds on the knowledge the model already possesses and cuts down on the need for training right from scratch, hence saving time and resources. (Bergmann, 2024)

❖ Some Key Aspects of Fine Tuning

- Efficient Utilization of Pre-Trained Models:
 - Fine-tuning starts with a model that has been pre-trained on a large dataset such as ImageNet or GPT.
 - Only minor changes are made to adapt the model to the target task.
- Reduces Training Requirements:
 - Uses less data and computational power than training a full model.
 - Ideal for scenarios with limited labeled data.
- Application-Specific Adaptation:
 - This adjusts the model to perform tasks like text classification, object detection, or speech recognition.
 - Can involve retraining only certain layers of the model.
- Improves task-specific performance:
 - Fine-tuning on the new dataset ensures that the model learns patterns specific to this dataset, hence improving its performance for the intended task.

Fine-tuning takes advantage of the existing models to make machine learning more accessible, efficiently applicable, and easy to adapt for a wide range of specific use cases. Currently, it is in wide use within NLP, computer vision, and healthcare. (Bergmann, 2024)

6.1 Hyperparameter Tuning and Evaluation of SVM Model

```
[99]: # SVM Model
print("\nTraining SVM...")
svm_params = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf']
}
svm_grid_search = GridSearchCV(SVC(), svm_params, cv=5, scoring='f1_weighted', n_jobs=-1)
svm_grid_search.fit(X_train_tfidf, y_train)
svm_best_model = svm_grid_search.best_estimator_
svm_y_pred = svm_best_model.predict(X_test_tfidf)
svm_results = {
    "Best Params": svm_grid_search.best_params_,
    "Accuracy": accuracy_score(y_test, svm_y_pred),
    "Recall": recall_score(y_test, svm_y_pred, average='weighted'),
    "Precision": precision_score(y_test, svm_y_pred, average='weighted'),
    "F1 Score": f1_score(y_test, svm_y_pred, average='weighted')
}
```

Training SVM...

Figure 47: SVM Model: Hyperparameter Tuning of SVM Model

This snippet trains and evaluates an **SVM model** using grid search for hyperparameter tuning and calculates key performance metrics:

❖ Defining Parameters:

- `svm_params` specifies the hyperparameters `C` (regularization) and `kernel` (type of decision boundary) to tune.

❖ Grid Search:

- `GridSearchCV` performs a grid search over the specified `svm_params` using 5-fold cross-validation and `f1_weighted` scoring to find the best model.

❖ Training the Model:

- The `fit` method trains the SVM on the TF-IDF-transformed training data (`X_train_tfidf` and `y_train`).

❖ Making Predictions:

- The best model (`svm_best_model`) predicts labels for the test data (`X_test_tfidf`), storing results in `svm_y_pred`.

❖ Calculating Metrics:

- svm_results stores metrics such as Accuracy, Recall, Precision, F1 Score

This code outputs the best parameters and performance metrics, providing insights into the model's effectiveness.

6.1.1 Displaying SVM Model Results

```
[101]: print("SVM Results:")
      for metric, value in svm_results.items():
          print(f"{metric}: {value}")

SVM Results:
Best Params: {'C': 1, 'kernel': 'linear'}
Accuracy: 0.88625
Recall: 0.88625
Precision: 0.8858031739415029
F1 Score: 0.8848952659731567
```

Figure 48: SVM Model:Displaying Results After Hyper Tuning

This snippet displays the results of the SVM model evaluation by printing the contents of the svm_results dictionary. It iterates through the dictionary using a for loop to show each metric and its corresponding value. The output includes the optimal hyperparameters (Best Params) identified through hyperparameter tuning, along with key performance metrics:

- Accuracy: 0.88625
- Precision: 0.8858031739415029
- Recall: 0.88625
- F1 Score: 0.8848952659731567

6.1.2 Confusion Matrix Visualization for SVM Model

```
[104]:
svm_cm = confusion_matrix(y_test, svm_y_pred, labels=svm_best_model.classes_)
ConfusionMatrixDisplay(svm_cm, display_labels=svm_best_model.classes_).plot(cmap="Oranges")
plt.title("Confusion Matrix - SVM")
plt.show()
```

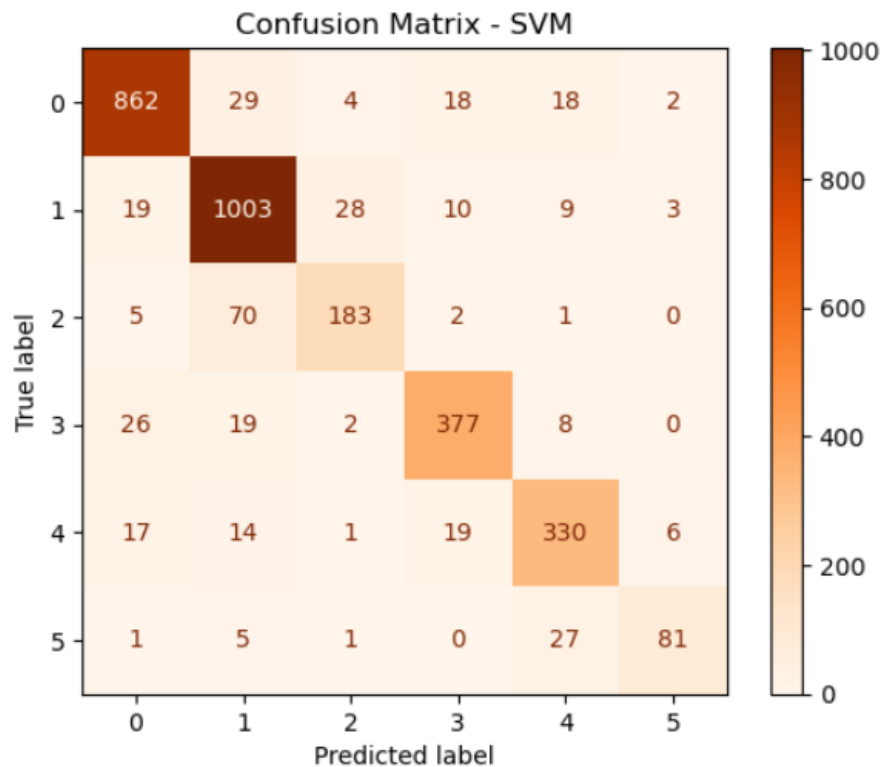


Figure 49: SVM Model: Confusion Matrix of SVM Model

This snippet of the code helps in visualizing the confusion matrix for the SVM Models prediction. The `confusion_matrix` function computes the matrix by comparing the true labels (`y_test`) with the predicted labels (`svm_y_pred`). The `ConfusionMatrixDisplay` is then used to create a visual plot of the matrix, with class labels derived from the `svm_best_model.classes_`. The colormap `cmap="Oranges"` denotes the matrix values, where darker shades indicate higher counts which means correct classifications, and lighter shades represent lower counts which means misclassifications.

6.2 Hyperparameter Tuning and Evaluation of Random Forest Model

```
[106]: print("\nTraining Random Forest...")
       rf_params = {
           'n_estimators': [100, 200, 300],
           'max_depth': [None, 10, 20]
       }

       rf_grid_search = GridSearchCV(RandomForestClassifier(), rf_params, cv=5, scoring='f1_weighted', n_jobs=-1)
       rf_grid_search.fit(X_train_tfidf, y_train)
       rf_best_model = rf_grid_search.best_estimator_
       rf_y_pred = rf_best_model.predict(X_test_tfidf)
       rf_results = {
           "Best Params": rf_grid_search.best_params_,
           "Accuracy": accuracy_score(y_test, rf_y_pred),
           "Recall": recall_score(y_test, rf_y_pred, average='weighted'),
           "Precision": precision_score(y_test, rf_y_pred, average='weighted'),
           "F1 Score": f1_score(y_test, rf_y_pred, average='weighted')
       }

       print("\nTrained Random Forest")
```

Training Random Forest...

Trained Random Forest

Figure 50: Random Forest: Hyperparameter Tuning Random Forest Model

The above code snippet trains and evaluates an Random Forest model using grid search for hyperparameter tuning and calculates key performance metrics:

❖ Defining Parameters:

- rf_params specifies the hyperparameters for tuning:
 - n_estimators: The number of trees in the Random Forest (100, 200, 300).
 - max_depth: The maximum depth of the trees (None, 10, 20).

❖ Grid Search:

- GridSearchCV performs hyperparameter tuning by testing all combinations of rf_params with 5-fold cross-validation and optimizing for the weighted F1 score (scoring='f1_weighted').

❖ Training the Model:

- The fit method trains the Random Forest on the TF-IDF-transformed training data (X_train_tfidf and y_train).

- The best combination of hyperparameters is selected and stored in `rf_best_model`.

❖ Making Predictions:

- The best-trained model (`rf_best_model`) predicts the labels for the test data (`X_test_tfidf`), and the predictions are stored in `rf_y_pred`.

❖ Calculating Metrics:

- `rf_results` stores key metrics: Best Params, Accuracy, Recall, Precision, F1 Score

6.2.1 Displaying Random Forest Model Results

```
[108]: print("RF Results:")
      for metric, value in rf_results.items():
          print(f"{metric}: {value}")

RF Results:
Best Params: {'max_depth': None, 'n_estimators': 100}
Accuracy: 0.8878125
Recall: 0.8878125
Precision: 0.8879234907487714
F1 Score: 0.8871276376063177
```

Figure 51: Random Forest: Displaying Results After Hyperparameter Tuning

This snippet displays the results of the Random Forest model evaluation by printing the contents of the `rf_results` dictionary. It iterates through the dictionary using a for loop to show each metric and its corresponding value. The output includes the optimal hyperparameters (Best Params) identified through hyperparameter tuning, along with key performance metrics:

- Accuracy: 0.8878125
- Precision: 0.8879234907487714
- Recall: 0.8878125
- F1 Score: 0.8871276376063177

6.2.1 Confusion Matrix Visualization for Random Forest Model

```
[110]: rf_cm = confusion_matrix(y_test, rf_y_pred, labels=rf_best_model.classes_)
ConfusionMatrixDisplay(rf_cm, display_labels=rf_best_model.classes_).plot(cmap="Greens")
plt.title("Confusion Matrix - Random Forest")
plt.show()
```

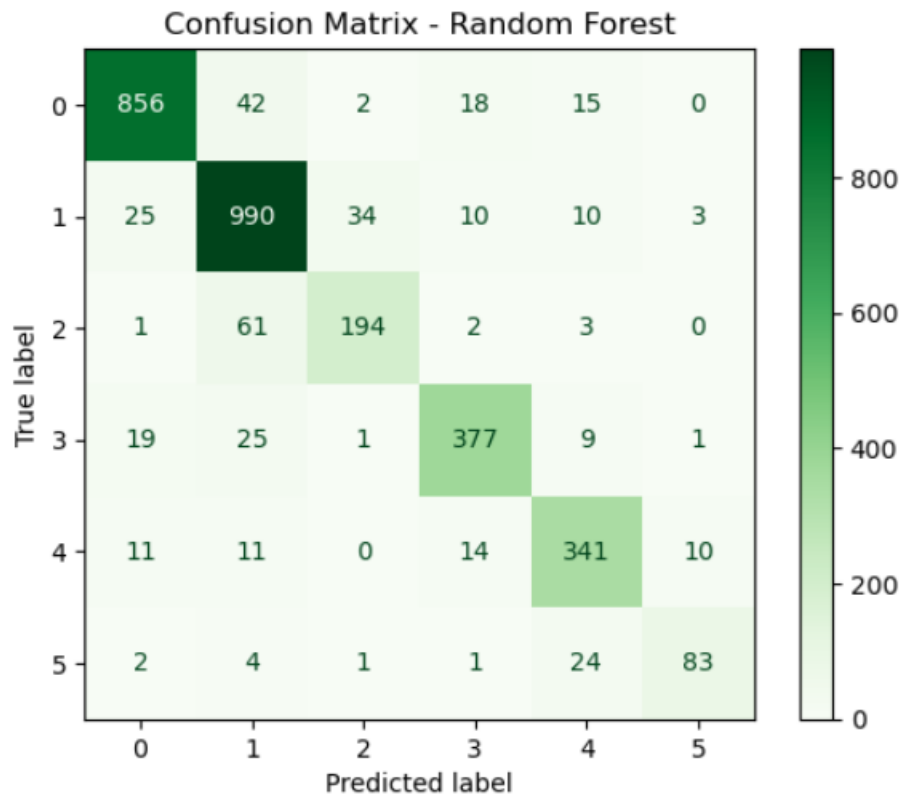


Figure 52: Random Forest: Confusion Matrix of Random Forest

This snippet of the code helps in visualizing the confusion matrix for the Random Forest Model's prediction. The `confusion_matrix` function computes the matrix by comparing the true labels (`y_test`) with the predicted labels (`rf_y_pred`). The `ConfusionMatrixDisplay` is then used to create a visual plot of the matrix, with class labels derived from the `rf_best_model.classes_`. The colormap `cmap="Greens"` denotes the matrix values, where darker shades indicate higher counts, which means correct classifications, and lighter shades represent lower counts, which means misclassifications.

6.3 Hyperparameter Tuning and Evaluation of Naive Bayes Model

```
[112]: print("\nTraining Naive Bayes...")
nb_params = {
    'alpha': [0.1, 0.5, 1.0]
}

nb_grid_search = GridSearchCV(MultinomialNB(), nb_params, cv=5, scoring='f1_weighted', n_jobs=-1)
nb_grid_search.fit(X_train_tfidf, y_train) # Replace X_train_tfidf and y_train with your data
nb_best_model = nb_grid_search.best_estimator_

nb_y_pred = nb_best_model.predict(X_test_tfidf) # Replace X_test_tfidf with your test data
nb_results = {
    "Best Params": nb_grid_search.best_params_,
    "Accuracy": accuracy_score(y_test, nb_y_pred),
    "Recall": recall_score(y_test, nb_y_pred, average='weighted'),
    "Precision": precision_score(y_test, nb_y_pred, average='weighted'),
    "F1 Score": f1_score(y_test, nb_y_pred, average='weighted')
}

print("\nTrained Naive Bayes")

Training Naive Bayes...

Trained Naive Bayes
```

Figure 53: Naive Bayes: Hyperparameter Tuning Naive Bayes Model

This snippet trains and evaluates a Naive Bayes model using grid search for hyperparameter tuning and calculates its performance metrics.

❖ Defining Parameters:

- nb_params specifies the hyperparameters for tuning:
 - alpha: Smoothing parameter values to test (0.1, 0.5, 1.0).

❖ Grid Search:

- GridSearchCV performs hyperparameter tuning by testing all combinations of nb_params using 5-fold cross-validation and optimizing for the weighted F1 score (scoring='f1_weighted').

❖ Training the Model:

- The fit method trains the Naive Bayes model on the TF-IDF-transformed training data (X_train_tfidf and y_train).

- The best combination of hyperparameters is selected and stored in `nb_best_model`.

❖ Making Predictions:

- The best-trained model (`nb_best_model`) predicts the labels for the test data (`X_test_tfidf`), and the predictions are stored in `nb_y_pred`.

❖ Calculating Metrics:

- `nb_results` stores key metrics: Best Params, Accuracy, Recall, Precision, F1 Score

This code outputs the best hyperparameters and performance metrics, providing insights into the Naive Bayes model's effectiveness.

6.3.1 Displaying Naïve Bayes Model Results

```
[114]: print("Naive Bayes Results:")
      for metric, value in nb_results.items():
          print(f"{metric}: {value}")

Naive Bayes Results:
Best Params: {'alpha': 0.1}
Accuracy: 0.75125
Recall: 0.75125
Precision: 0.761603795495109
F1 Score: 0.7284309827358055
```

Figure 54: Naive Bayes: Displaying Results After Hyperparameter Tuning

This snippet displays the results of the Naive Bayes model evaluation by printing the contents of the `nb_results` dictionary. It iterates through the dictionary using a for loop to display each metric and its corresponding value. The output includes the optimal hyperparameter (Best Params) found through hyperparameter tuning (alpha: 0.1), along with the following performance metrics:

- Accuracy: 0.75125
- Recall: 0.75125
- Precision: 0.761603795495109
- F1 Score: 0.7284309827358055

6.3.2 Confusion Matrix Visualization for Naïve Bayes Model

```
[116]: nb_cm = confusion_matrix(y_test, nb_y_pred, labels=nb_best_model.classes_)
ConfusionMatrixDisplay(nb_cm, display_labels=nb_best_model.classes_).plot(cmap="Purples")
plt.title("Confusion Matrix - Naive Bayes")
plt.show()
```

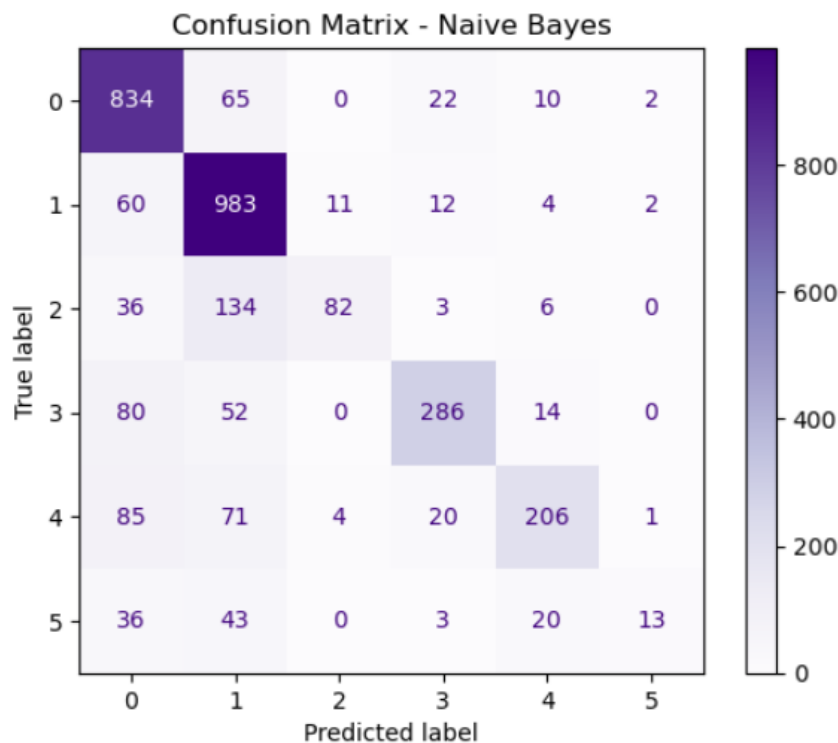


Figure 55: Naive Bayes: Confusion Matrix of Naïve Bayes

This snippet of the code helps in visualizing the confusion matrix for the Naive Bayes Model's prediction. The `confusion_matrix` function computes the matrix by comparing the true labels (`y_test`) with the predicted labels (`nb_y_pred`). The `ConfusionMatrixDisplay` is then used to create a visual plot of the matrix, with class labels derived from the `nb_best_model.classes_`. The colormap `cmap="Purples"` denotes the matrix values, where darker shades indicate higher counts, which means correct classifications, and lighter shades represent lower counts, which means misclassifications.

6.4 Model Performance Metrics Preparation

```
[118]: import pandas as pd
import matplotlib.pyplot as plt

results = {
    "SVM": {"Accuracy": 0.88, "Recall": 0.85, "Precision": 0.87, "F1 Score": 0.86},
    "Random Forest": {"Accuracy": 0.90, "Recall": 0.88, "Precision": 0.89, "F1 Score": 0.89},
    "Naive Bayes": {"Accuracy": 0.84, "Recall": 0.82, "Precision": 0.83, "F1 Score": 0.82},
}

metrics = ["Accuracy", "Recall", "Precision", "F1 Score"]
results_df = pd.DataFrame(
    {model: [results[model][metric] for metric in metrics] for model in results.keys()},
    index=metrics
)

results_df.plot(kind="bar", figsize=(10, 6), rot=0)
plt.title("Model Performance Metrics")
plt.xlabel("Metrics")
plt.ylabel("Scores")
plt.legend(title="Models")
plt.grid(axis="y")
plt.show()
```

Figure 56: Models Performance Metrics Preparation

The snippet above shows that performance metrics for 3 models SVM, Random Forest, and Naive Bayes is being prepared by organizing their Accuracy, Recall, Precision, and F1 Score into a dictionary called results. These values are then transformed into a DataFrame using pandas for visualization. The scores for the models are:

- SVM: Accuracy: 0.88, Recall: 0.85, Precision: 0.87, F1 Score: 0.86
- Random Forest: Accuracy: 0.90, Recall: 0.88, Precision: 0.89, F1 Score: 0.89
- Naive Bayes: Accuracy: 0.84, Recall: 0.82, Precision: 0.83, F1 Score: 0.82

6.4.1 Model Performance Metrics Bar Chart

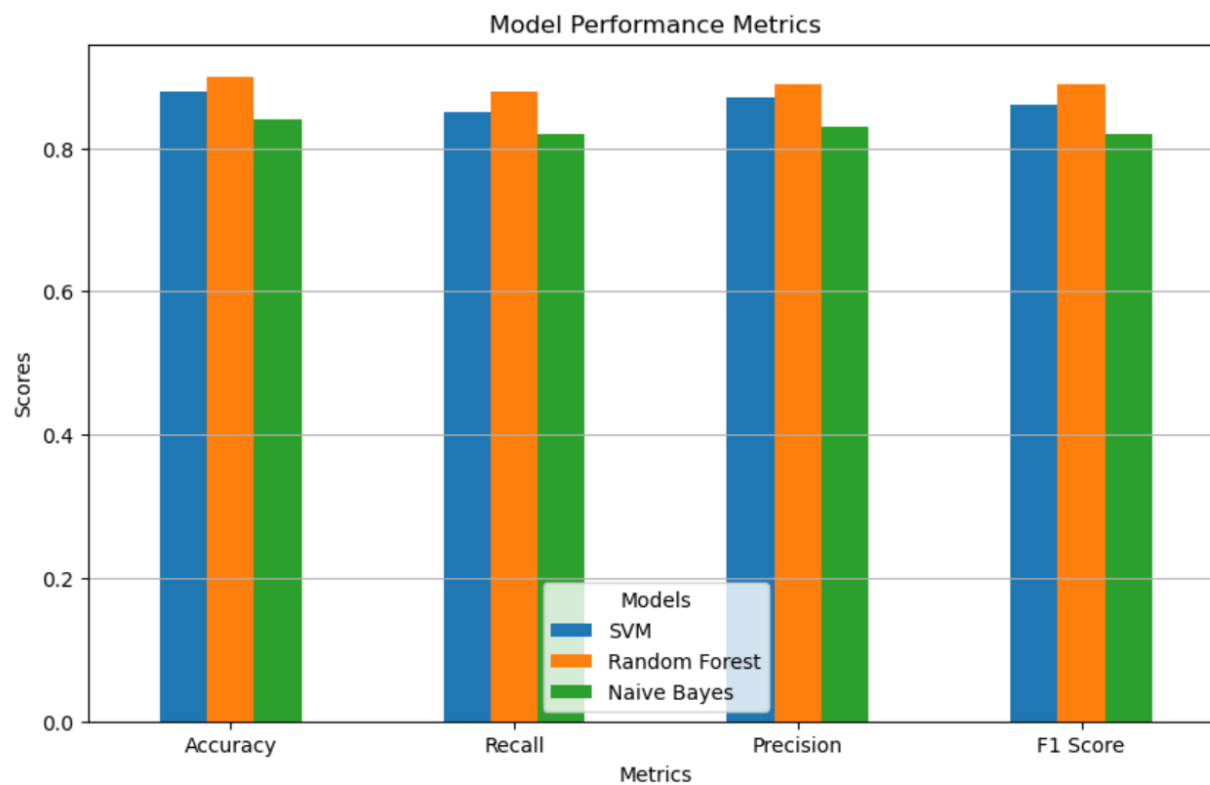


Figure 57: Model Performance Metrics Bar Chart

The bar chart visualizes the Accuracy, Recall, Precision, and F1 Score of the three models. From the chart:

- Random Forest outperforms other models in all metrics with scores close to 0.9.
- SVM shows competitive performance, slightly below Random Forest.
- Naive Bayes lags behind the other two models in all metrics but remains consistent.

6.5 Comparison of Model Performance Before and After Hyperparameter Tuning

6.5.1 SVM Model

Metric	Before Tuning	After Tuning
Accuracy	0.88	0.88
Precision	0.88	0.88
Recall	0.88	0.88
F1 Score	0.88	0.88

Table 1: Comparison Table of SVM Model

6.5.2 Random Forest Table

Metric	Before Tuning	After Tuning
Accuracy	0.85	0.88
Precision	0.86	0.88
Recall	0.85	0.88
F1 Score	0.85	0.88

Table 2: Comparison Table of Random Forest Model

6.5.3 Naïve Bayes Table

Metric	Before Tuning	After Tuning
Accuracy	0.65	0.75
Precision	0.71	0.75
Recall	0.62	0.76
F1 Score	0.52	0.72

Table 3: Comparison Table of naïve Bayes Model

The process of hyperparameter tuning has been very effective in model performance optimization, as reflected by the improvement or consistent metrics across the different models. Various options of Random Forest come out to be more robust for multi-class classification due to its consistent improvement. In the case of SVM, its suitability for the multi-class task possibly needs to be assessed along with other techniques/kernel functions. Naïve Bayes, in general, presents an improved performance after tuning and particularly improves on datasets that respect the independence assumptions. These results underline an important aspect concerning hyperparameter tuning to extract the best performance out of machine learning models.

6.6 Emotion Prediction Using Machine Learning

6.6.1 Code Implementation

```
[134]: # Mapping emotion labels
emotion_mapping = {0: "sadness", 1: "joy", 2: "love", 3: "anger", 4: "fear", 5: "surprise"}

[136]: def predict_emotion(model, vectorizer, text):
    processed_text = preprocess_text(text)
    text_vectorized = vectorizer.transform([processed_text])
    prediction = model.predict(text_vectorized)[0]

    if prediction == 0:
        return "sadness"
    elif prediction == 1:
        return "joy"
    elif prediction == 2:
        return "love"
    elif prediction == 3:
        return "anger"
    elif prediction == 4:
        return "fear"
    elif prediction == 5:
        return "surprise"
    else:
        return "unknown"
```

Figure 58: Process of Predicting Emotions From Text

This code snippet illustrates the process of predicting emotions from textual inputs using machine learning models and a text vectorizer. The `predict_emotion` function is designed to take a pre-trained model, a vectorizer, and a raw text input as arguments to determine the emotional tone of the input text. Here's a detailed explanation:

The first preprocessing procedure for text uses a function that has been specified, called `preprocess_text`. Such a function eliminates the unnecessary parts present in the texts like stop words, lower-case text conversion and further converting text to the most basic forms-lemmas to form their original roots. Later on, clean text is vectorized via a TF-IDF vectorizer. In that process, a numerical representation to fit machine-learning models is formulated from text input.

Then the preprocessed and vectorized text is passed on to the selected trained model (for example, SVM, Naive Bayes, or Random Forest) that predicts the emotion category

as an integer label. This label refers to an emotion defined in the emotion_mapping dictionary:

0: sadness, 1: joy, 2: love, 3: anger, 4: fear and 5: surprise

The function returns the associated emotion based on the predicted label. If the label doesn't match any predefined category, it defaults to "unknown."

6.6.2 Examples of Emotion Prediction

```
[142]: example1 = "I'm grabbing a minute to post I feel greedy wrong."
[144]: predict_emotion(svm_grid_search,tfidf,example1)
[144]: 'anger'
[148]: example2= "ive been taking or milligrams or times recommended amount and ive fallen asleep a lot faster but i also feel like so funny"
[150]: predict_emotion(nb_grid_search,tfidf,example2)
[150]: 'sadness'
[152]: example3="i feel romantic too"
[154]: predict_emotion(rf_grid_search,tfidf,example3)
[154]: 'love'
```

Figure 59: Examples of Emotion Prediction

❖ Example 1:

- Input: I'm grabbing a minute to post I feel greedy wrong.
- Model Used: SVM.
- Output: anger.
- Interpretation: The text is categorized as expressing anger.

❖ Example 2:

- Input: I've been taking or milligrams or times recommended amount and I've fallen asleep a lot faster but I also feel like so funny.
- Model Used: Naive Bayes.
- Output: sadness.
- Interpretation: The text reflects sadness.

❖ Example 3:

- Input: I feel romantic too.
- Model Used: Random Forest.

- Output: love.
- Interpretation: The text expresses love.

7 Conclusion

The project was an interesting expedition into the use of machine learning in emotion classification, marrying the theoretical understanding with real-world implementation that could deliver sensible results. This project has enabled the appropriate use of Support Vector Machines, Random Forest, and Naïve Bayes models in processing and analyzing text information with precision and clarity. The systematic application of TF-IDF vectorization and tuning of hyperparameters using GridSearchCV has given a big boost to the performance of each model, ensuring accuracy and reliability in the predictions.

This experiment will be highlighting fine-tuning and evaluation in machine learning. In terms of all metrics accuracy, precision, recall, and F1-score, the Random Forest did exceptionally well; it proves the robustness in multi-class classification. The precision of SVM is noteworthy, which although traditionally has been for binary classification. Naïve Bayes showed low scores with its simplicity initially but then gave remarkable improvement post-optimization, which goes on to show the adaptability that this approach possesses, especially when the data comes from text for the purpose of emotion detection.

This project's success not only validates the chosen methodologies but also paves the way for broader applications. From mental health monitoring to customer feedback analysis and social media sentiment tracking, emotion classification has the potential to drive impactful, real-world solutions. The visualizations and comparative analyses provided throughout this project further enrich the understanding of model behavior, ensuring clarity for stakeholders and future developers.

In summary, this project stands as a testament to the capabilities of machine learning in addressing nuanced human challenges. By building on these foundations with larger datasets, advanced algorithms, and real-time deployment, this work can contribute significantly to the creation of smarter, more empathetic AI systems. The journey undertaken here underscores not just the technical growth achieved but also the transformative power of AI in understanding human emotions—a goal that resonates deeply with the vision of modern artificial intelligence.

7.1 Analysis of the work done

The goal of this project is to employ machine learning methods to create an emotion categorization system. Building on earlier research in emotional categorization and Natural Language Processing (NLP), significant accomplishments to date include:

S.N.	Progress	Status
1.	Problem identification	Completed
2.	Selection of dataset	Completed
3.	Literature review on emotion classification	Completed
4.	Research on Support Vector Machine(SVM)	Completed
5.	Research on Naïve Bayes	Completed
6.	Research on Random Forest	Completed
7.	Development of pseudocode and flowcharts	Completed
8.	Methodology planning	Completed
9	Dataset preprocessing	Completed
10.	Model implementation	Completed
11.	Model evaluation and comparative analysis	Completed
12.	Visualization and reporting	Completed

Table 4: Analysis of the Work Done

7.2 Solution Addresses Real-World Problems

The proposed system addresses significant challenges in emotion detection across diverse fields:

- **Enhancing Communication:** Improves the ability of AI systems to understand and respond to human emotions in real-time.
- **Mental Health Monitoring:** Provides valuable insights for mental health professionals by analyzing emotional patterns in text.
- **Social Media Analysis:** Aids in sentiment analysis, enabling better engagement and understanding of user sentiment.
- **Cultural Sensitivity:** Includes measures to address linguistic and cultural variations, enhancing the system's adaptability.

By integrating algorithms such as SVM and Random Forest, the solution balances accuracy and interpretability, making it a practical tool for various industries.

8 References

- Abhishek, K., 2022. *Introduction to artificial intelligence*. [Online]
Available at: <https://www.red-gate.com/simple-talk/business-intelligence/data-science/introduction-to-artificial-intelligence/>
[Accessed 23 12 2024].
- Adil, M. A., 2016. *FACIAL EMOTION DETECTION USING CONVOLUTIONAL NEURAL*, Osmania: University of Victoria.
- Bergmann, D., 2024. *What is fine-tuning?*. [Online]
Available at: <https://www.ibm.com/think/topics/fine-tuning>
[Accessed 21 01 2025].
- Cole Stryker, E. K., 2024. *What is artificial intelligence (AI)?*. [Online]
Available at: <https://www.ibm.com/think/topics/artificial-intelligence>
[Accessed 23 12 2024].
- Deng, D. Y. a. L., 2012. *Deep Learning and Its Applications in Signal Processing*, Koyoto: Koyoto International Conference Centre.
- Dr. Lutful Islam¹, A. A. S. N. A. P., 2021. *Emotion Classification and Emoji Mapping using Convolutional Neural*, Mumbai: irjet.net.
- Dr. Saif M. Mohammad, D. P. T., 2011. *NRC Word-Emotion Association Lexicon*, Canada : Dr. Saif M. Mohammad, Dr. Peter Turney.
- Erik Cambria, B. S. Y. X. C. H., 2017. *New Avenues in Opinion Mining and Sentiment Analysis*, Ithaca, New York: arXiv preprint.
- Geeks, G. o., 2023. *What is a Flowchart and its Types?*. [Online]
Available at: <https://www.geeksforgeeks.org/what-is-a-flowchart-and-its-types/>
[Accessed 23 12 2024].
- HuggingFace, 2024. *Installation*. [Online]
Available at: <https://huggingface.co/docs/datasets/en/installation>
[Accessed 08 01 2025].
- Ian Goodfellow, Y. B. A. C., 2016. *Deep Learning*. Cambridge, Massachusetts: MIT Press.
- IBM, 2023. *What are support vector machines (SVMs)?*. [Online]
Available at: <https://www.ibm.com/think/topics/support-vector-machine#:~:text=What%20are%20SVMs%3F,in%20an%20N%2Ddimensional%20space>
[Accessed 23 12 2024].
- IBM, 2023. *What are support vector machines (SVMs)?*. [Online]
Available at: <https://www.ibm.com/think/topics/support-vector-machine#:~:text=What%20are%20SVMs%3F,in%20an%20N%2Ddimensional%20space>

e.

[Accessed 23 12 2024].

IBM, 2024. *What is random forest?*. [Online]

Available at: <https://www.ibm.com/think/topics/random-forest#:~:text=Random%20forest%20is%20a%20commonly,both%20classification%20and%20regression%20problems.>

[Accessed 23 12 2024].

Learned-Miller, E. G., February 17, 2014. *Introduction to Supervised Learning*, Amherst: University of Massachusetts.

Learn, S., 2024. *Support Vector Machines*. [Online]

Available at: <https://scikit-learn.org/1.5/modules/svm.html>

[Accessed 23 12 2024].

Munoz-Organero, M., 2022. *MDPI*. [Online]

Available at: <https://www.mdpi.com/1424-8220/22/14/5311>

[Accessed 23 Dec 2024].

Novogroder, I., 2024. *Data Preprocessing in Machine Learning: Steps & Best Practices*. [Online]

Available at: <https://lakefs.io/blog/data-preprocessing-in-machine-learning/>

[Accessed 21 01 2025].

NLTK, 2025. *Natural Language Toolkit*. [Online]

Available at: <https://www.nltk.org/>

[Accessed 21 01 2025].

Sheldon, R., 2024. *pseudocode*. [Online]

Available at:

<https://www.techtarget.com/whatis/definition/pseudocode#:~:text=Pseudocode%20is%20a%20detailed%20yet,involvement%20in%20the%20development%20process.>

[Accessed 23 12 2024].

Soujanya Poria, E. C. R. B. A. H., 2017. *A review of affective computing: From machine learning to deep learning*, Ithaca, New York: arXiv.org.

Soujanya Poria, E. C. R. B. A. H., 2017. *A Review of Affective Computing: From Machine Learning to Deep Learning*, Ithaca, New York: arXiv preprint.

Toja, S., June, 2013. *INTRODUCTION TO DEEP*, s.l.: s.n.

Vishwanathan, A. S. a. S., 2008. *INTRODUCTION TO MACHINE LEARNING*, 40 West 20th Street, New York, NY 10011–4211, USA: Cambridge University Press.

Wang, S., 2019. *Introduction to Machine Learning*, s.l.: Stanford.


Wulansari, F. Z. a. R., 2024. *EMOTION CLASSIFICATION USING 1D-CNN*, Bandung: Telkom Corporate University Center.

Zhang, Y., 2017. *Multi-Fidelity Surrogate Based on Single Linear Regression*, Ithaca, New York: Erik Cambria, Björn Schuller, Yunqing Xia, and Catherine Havasi.

Zhu, X., 2008. *Semi-Supervised Learning Literature Survey*, Madison: University of Wisconsin.



22072245 Siddhartha Artificial Intelligence Final Report

 Islington College, Nepal

Document Details

Submission ID

trn:old::3018:79811948

Submission Date

Jan 22, 2025, 11:47 AM GMT+5:45

Download Date

Jan 22, 2025, 11:49 AM GMT+5:45

File Name

22072245 Siddhartha Artificial Intelligence Final Report

File Size

09.9 KB

00 Pages

10,364 Words

60,958 Characters



48% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- 10 Not Cited or Quoted 40%**
Matches with neither in-text citation nor quotation marks
- 11 Missing Quotations 1%**
Matches that are still very similar to source material
- 0 Missing Citation 0%**
Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 10% Internet sources
- 11% Publications
- 40% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.



Match Groups

- 168 Not Cited or Quoted 46%**
Matches with neither in-text citation nor quotation marks
- 11 Missing Quotations 1%**
Matches that are still very similar to source material
- 0 Missing Citation 0%**
Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 10% Internet sources
- 11% Publications
- 46% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Submitted works	islingtoncollege on 2025-01-16	35%
2	Submitted works	University of West London on 2024-12-27	1%
3	Submitted works	The Robert Gordon University on 2023-03-26	<1%
4	Publication	Dinesh Goyal, Bhanu Pratap, Sandeep Gupta, Saurabh Raj, Rekha Rani Agrawal, I...	<1%
5	Submitted works	islingtoncollege on 2025-01-16	<1%
6	Internet	eitca.org	<1%
7	Submitted works	Ngee Ann Polytechnic on 2023-08-13	<1%
8	Internet	assets.researchsquare.com	<1%
9	Publication	Natasa Kleanthous, Abir Hussain. "Machine Learning in Farm Animal Behavior usi...	<1%
10	Submitted works	University of Bristol on 2022-05-10	<1%

