Emotionally Aware Chatbot
EDAIC

A Thesis Report
Submitted in full requirements for the Degree of
Bachelor of Science in Computer Science and Engineering

Submitted by

Nowshin Rumali Tisha	**160204107**
Amin Ahmed Toshib	**170104061**
Rejone-E-Rasul Redoy	**170104116**
Mehedi Hassan Sami	**170104118**

Supervised by

Mr. Md. Khairul Hasan
Associate Professor

Department of Computer Science and Engineering
Ahsanullah University of Science and Technology

Dhaka, Bangladesh

January 2, 2022

CANDIDATES' DECLARATION

We, hereby, declare that the Thesis presented in this report is the outcome of the investigation performed by us under the supervision of Mr. Md. Khairul Hasan, Associate Professor, Department of Computer Science and Engineering, Ahsanullah University of Science and Technology, Dhaka, Bangladesh. The work was spread over two final year courses, CSE4100: Project and Thesis I and CSE4250: Project and Thesis II, in accordance with the course curriculum of the Department for the Bachelor of Science in Computer Science and Engineering program.

It is also declared that neither this Thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Nowshin Rumali Tisha
160204107

Amin Ahmed Toshib
170104061

Rejone-E-Rasul Redoy
170104116

Mehedi Hassan Sami
170104118

CERTIFICATION

This Thesis titled, **"Emotionally Aware Chatbot EDAIC"**, submitted by the group as mentioned below has been accepted as satisfactory in full requirements for the degree B.Sc. in Computer Science and Engineering in January 2, 2022.

Group Members:

Nowshin Rumali Tisha	**160204107**
Amin Ahmed Toshib	**170104061**
Rejone-E-Rasul Redoy	**170104116**
Mehedi Hassan Sami	**170104118**

Mr. Md. Khairul Hasan
Associate Professor & Supervisor
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology

Prof. Dr. Mohammad Shafiul Alam
Professor & Head
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology

ACKNOWLEDGEMENT

First and foremost, we are very grateful to Almighty Allah (SWT) for helping us to complete our thesis work. Then we would like to express our gratitude to our respected thesis supervisor Mr. Md. Khairul Hasan for his valuable time and guidance from the beginning of our work for a whole year. His motivation has always encouraged us to gather knowledge. His supervision and experiences have helped us all the time to enrich our work. Whenever we needed his help, he was always there to assist and guide us. So, we are very grateful to him in many ways that can't be described in words.

Again, we would like to thank our respected department head Prof. Dr. Mohammad Shafiul Alam, and all our respected faculty members of the Computer Science and Engineering department of Ahsanullah University of Science and Technology. All of our teachers have helped us in many ways through their knowledge and teaching. So, we really appreciate their help and expertise.

Lastly, we would like to give our warmest tribute to our family and friends who have constantly supported us with their care and blessings in many difficult situations of life. Therefore, we could complete our undergraduate thesis work due to the help of all the respected and loving people around us.

Dhaka
January 2, 2022

Nowshin Rumali Tisha

Amin Ahmed Toshib

Rejone-E-Rasul Redoy

Mehedi Hassan Sami

ABSTRACT

Emotion is one of the basic instincts of a human being. Emotion detection plays a vital role in the field of textual analysis. At present, people's expressions and emotional states have turned into the leading topic for research works. People train chatbot that is a software program with artificial intelligence for detecting emotions and many other purposes. In this book, a chatbot is created to converse with humans to find their emotional states through machine learning techniques.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A chatbot is a software application that uses Artificial Intelligence and generates human-like conversations. In this book, EDAIC is a chatbot capable of textual interactions and detecting emotions from the text. EDAIC stands for Emotion Detector and Artificially Intelligent Chatbot. EDAIC will answer the question of a user after analyzing that question's keywords. Many apps use this kind of chatbot in the FAQ and Help sections so that users can converse with assigned chatbots and find answers to their queries. Instead of interacting with real humans, users converse with a human-like chatbot that acts like an actual human. In this case, chatbots are very helpful that can assist in different applications and websites.

Artificial Intelligence is the latest technology. Creating a chatbot using this technology can make anyone's professional profile stand out in this competitive era. As Python language is used to create this chatbot, learning this language can also enhance anyone's professional profile.

EDAIC, a chatbot that uses Artificial Intelligence to detect emotions from text, is introduced in this book. Emotions refer to various types of consciousness or states of mind that are shown as feelings. They can be expressed through facial expressions, gestures, texts, and speeches. Emotion detection through chatbots also helps to understand humans more precisely while conversing with them. Emotions like Joy, Love, Surprise, Neutral, Sadness, Fear and Anger are considered while creating this chatbot. This chatbot will communicate with people using the help of natural language processing (NLP) and classification algorithms.

In this book, EDAIC is more likely to work as an emotion detector. By detecting emotions, it can understand the customers' likes or dislikes for the products and services of a company more accurately. It can assist users with swift responses that will reduce time and carry out the service faster. Therefore, this chatbot helps to reduce the workload. It aids the business teams in communicating with customers to resolve their queries in a faster and optimized way.

1.1 Motivation

Emotion Detection and Recognition from texts are recent fields of research that are closely related to Emotion Analysis. Emotion Analysis aims at detecting and recognizing feelings through the expressions from sentences, such as Joy, Love, Surprise, Neutral, Sadness, Fear, and Anger. Motivation comes from many aspects.

- Depressed or isolated humans can interact with the chatbot. The chatbot can detect their emotions and come up with a reply that can console or motivate them.
- A chatbot can assist in responding to users' queries any time of the day without taking any breaks. It can also save time with quick responses.
- If a company launches a new product, it is essential to get feedback from the customers. A company can get feedback from the expression of their customers that needs emotion detection through conversations between chatbot and customer.
- A chatbot can help to get the democratic opinion regarding election results. Election results can be estimated through people's expressions or feelings about a particular candidate.

1.2 Purpose

EDAIC is an artificially intelligent chatbot created using machine learning techniques. It is designed to serve some purposes.

- The primary purpose is that the chatbot can detect emotions while interacting with humans. It can generate human-like conversations. As it can interact with humans, if our model obtains high accuracy, it can predict a person's emotional state or mood more accurately.
- A real-time emotion detection can make a chatbot perceive a person's current emotions while generating conversations.
- Our model can be served in many cases like business, gaming, and messaging apps. As it can detect emotions, it can assist in psychological sessions too.

Chapter 2

Background Studies

Going through some background research will help to grasp the technique before describing our implemented models. Some topics which are stated below will be the focus of these studies.

2.1 Chatbot

A chatbot is a software application that generates human-like conversations. It is an automated program that serves the customers and also assists in business criteria. There are many apps like messaging apps, gaming apps that use chatbots as a convenient tool to chat with users instead of an actual human. In this case, a chatbot helps to create a digital platform where a chatbot can converse with a user and send responses faster than a human that can save time (figure 2.1).

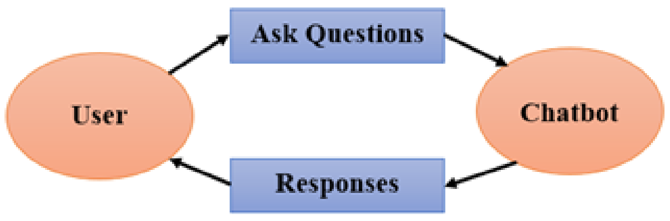


Figure 2.1: User and Chatbot Interaction

Some high-tech chatbot uses artificial intelligence that can detect a user's emotions through chats. A chatbot can work all time without taking any breaks and this makes a chatbot more appealing in the business circle and to the customers. There are times when a company doesn't have enough staff, faces financial problems, or lacks resources. In those times, it can simply use a chatbot to manage certain sections like dealing with users, helping with user queries, or taking feedback from users.

2.2 Natural Language Processing (NLP)

Natural Language Processing is a method that helps to set up an interaction between a computer and human language. It helps a computer with the processing of enormous amounts of natural language data and analyzing them. A computer only understands machine language that is incomprehensible to most people because machine language consists of binary digits (ones and zeroes). For this particular reason, NLP was introduced. NLP helps computers to analyze text, hear and recognize speech, overall understand human language.

A few examples of NLP that is used worldwide are:

- Spell Checker
- Grammar Checker
- Typing Suggestions
- Auto-correct
- Virtual Assistants

2.3 Artificial Intelligence Markup Language (AIML)

Artificial Intelligence Markup Language is an XML-based Markup Language that creates natural language software agents. It is used for pattern matching models. In AIML, a specified domain and related queries about that domain are used. A Chatbot using AIML files can converse with a human only on that fixed topic. More precisely, the chatbot offers a fixed response to some fixed or similar questions. We initially used this but later we changed our approach.

AIML Example:

```
<category>
    <pattern>Who is Larry?</pattern>
    <template>He is a painter.</template>
</category>
```

2.4 CountVectorizer

It helps to count the words in a sentence. More specifically, if there are multiple sentences and a need to convert each word in those sentences into vectors, CountVectorizer is used.

CountVectorizer [1] converts a collection of text documents to a matrix of token counts. If a dictionary is not provided and an analyzer that does some kinds of feature selection is not used, the number of features will be equal to the vocabulary size found by analyzing the data.

For Example:

Data=[‘The’, ‘Italaian’, ‘cuisine’, ‘and’, ‘the’, ‘Chinese’, ‘cuisine’, ‘has’, ‘fine’, ‘delicacies’]

$$\Downarrow$$

	The	Italian	cuisine	and	Chinese	has	fine	delicacies
Data	2	1	1	1	1	1	1	1

2.5 Necessary Libraries

People used to conduct Machine Learning tasks by manually coding all of the algorithms, mathematics, and statistical formulas, back in the day. The procedure became time consuming, laborious, and inefficient as a result of this. However, many Python libraries, frameworks, and modules have made it much easier and efficient in contemporary times compared to the previous days.

For the implementation, we have used some Python libraries.

- **Numpy:** NumPy is a popular Python package that allows you to handle massive multidimensional arrays and matrices using a wide range of high-level mathematical functions. It’s especially useful for linear algebra, the Fourier transform, and random number generation.
- **Pandas:** Pandas is a well-known Python data analysis library that provides high-level data structures as well as a comprehensive range of data analysis capabilities. It possesses a lot of built-in data grabbing, combining, and filtering techniques.
- **Matplotlib:** Matplotlib is a well-known Python data visualization toolkit. It’s a 2D plotting library for generating graphs and plots in 2D space. Python’s Pyplot package makes charting simple for programmers by allowing them to manage line styles, font attributes, axis formatting among other things. It has a variety of graphs and plots for data visualization, including histograms, error charts, bar charts, etcetera.

- **NLTK (Natural Language Toolkit):** It's a collection of statistical language processing libraries and applications. It's one of the most effective NLP libraries, including packages for training machines to comprehend human language and respond appropriately.
- **Seaborn:** Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- **Tkinter:** Tkinter is Python's standard GUI library [2]. If we combine Python and Tkinter, it can create quick and simple graphical user interfaces. Tkinter gives us a range of typical GUI elements, such as buttons, menus, and other types of entry fields and display spaces, that we can utilize to design our interface. These components are referred to as widgets. For our GUI, we'll create a tree of widgets, with each widget having a parent widget all the way up to the application's root window. For example, a radio button or a text field must be contained within some form of containing window.
- **Sklearn:** In Python, Scikit-learn (Sklearn) is the most usable and robust machine learning library [3]. It uses a Python consistency interface to give a set of efficient tools for machine learning and statistical modelings, such as classification, regression, clustering, and dimensionality reduction.

2.6 Feature Extraction

The process of extracting and selecting features refers to feature extraction. Characteristics are typically quantitative in nature and can be absolute numeric values or category features that can be encoded.

2.6.1 Bag of Words

The term "bag of words" refers to a text modeling approach used in Natural Language Processing. In other words, it's a method for extracting features from text data. This method of extracting characteristics from documents is easy and versatile. The number of times a word appears in a document is represented by a bag of words. Figure 2.2 shows the steps of the Bag of Words model.

2.6.2 Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF is an acronym for Term Frequency — Inverse Document Frequency. It is a strategy for quantifying a word in a document. A weight is assigned to each word that represents

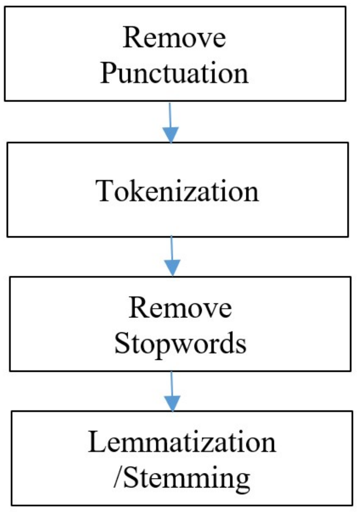


Figure 2.2: Process of Creating Bag of Words

its value in the document and corpus. In the fields of information retrieval and text mining, this method is frequently applied.

$$\text{TF}(t,d) = \frac{\text{Number of times term } t \text{ appears in } d}{\text{Total number of terms in } d} \tag{2.1}$$

$$\text{IDF}(t) = \log(\frac{\text{Total number of documents}}{\text{Number of documents with term } t}) \tag{2.2}$$

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) * \text{IDF}(t) \tag{2.3}$$

Terminology

t — a single term (word)

d — document (set of words in a sentence)

corpus — the total document set

2.7 Ensemble Learning

Ensemble learning is the process of systematically generating and combining many models, such as classifiers or experts, to handle a specific computational intelligence problem. Ensemble learning is largely used to improve a model's performance (classification, prediction, function approximation, etc.) or to lessen the risk of an unintentional poor model selection. Ensemble learning can also be used to assign a confidence level to the model's choice, choosing optimal (or near ideal) features, data fusion, incremental learning, non-stationary learning, and error-correcting [4].

Combining various models yields an ensemble-based system (henceforth classifiers). As a result, these systems are sometimes referred to as multiple classifier systems or simply ensemble systems. Using an ensemble-based system makes statistical sense in a variety of situations.

Unfortunately, the most widely used method of selecting the classifiers with the minimum error on training data is inaccurate. Even when computed using a cross-validation approach, performance on a training dataset can be misleading in terms of classification performance on previously unknown data. Then, among all (potentially infinite) classifiers with the same training - or even the same (pseudo) generalization performance as computed on the validation data (part of the training data left unused for evaluating the classifier performance). If everything else is equal, one might be tempted to pick at random, but this selection carries the possibility of picking a particularly bad model. Using an ensemble of such models, rather than just one, and integrating their results by averaging them, for example - can lessen the danger of selecting a particularly low performing classifier by accident. It's vital to note that there's no guarantee that a combination of many classifiers will always outperform the ensemble's best individual classifier.

2.7.1 Bagging

Bagging (bootstrap aggregation) is one of the earliest, most straightforward, and arguably simplest ensemble-based algorithms with surprisingly good performance (Breiman 1996). Bootstrapped clones of the training data are used to get the diversity of classifiers in bagging. For example, various training data subsets are picked up randomly with replacement from the full training dataset. Each subset of training data is used to train the same type of but different classifiers. Individual classifiers are then integrated using a simple majority vote. The ensemble decision is the class chosen by the most number of classifiers for each given instance. Because the training datasets may overlap significantly, extra methods such as the usage of a part of the training data to train each classifier or using relatively weak classifiers (e.g. decision stumps) can be used to promote diversity.

2.7.2 Boosting

Boosting, like bagging, generates an ensemble of classifiers by resampling the data and then combining them via majority voting. On the other hand, in boosting, resampling offers the most informative training data for each subsequent classifier. In essence, each boosting cycle produces three weak classifiers: the first, C1, is trained with a random subset of the available training data. Given C1, the most informative subset of training data for the second classifier, C2 is chosen. C2 is trained on a training set in which half of the data is correctly classified by C1 and the other half is incorrectly classified. C3 is a third classifier that is trained on cases where C1 and C2 disagree. A three-way majority vote is used to combine the three classifiers.

2.8 Classification Algorithms

Machine learning algorithms that classify, categorize or label data items based on what they've distinguished in the past are known as classification algorithms. Some classification algorithms are used to classify text.

2.8.1 Support Vector Machine (SVM)

A support vector machine (SVM) refers to a supervised machine learning model that employs a classification method for two-group classification issues that can categorize clean text after being given a set of labeled training data for each category [5].

They offer two primary benefits over other algorithms, such as neural networks, in terms of speed and accuracy with a limited number of samples (in the thousands). These make the method particularly well suited to text classification issues, where it's usual to have access to only a few thousand labeled samples.

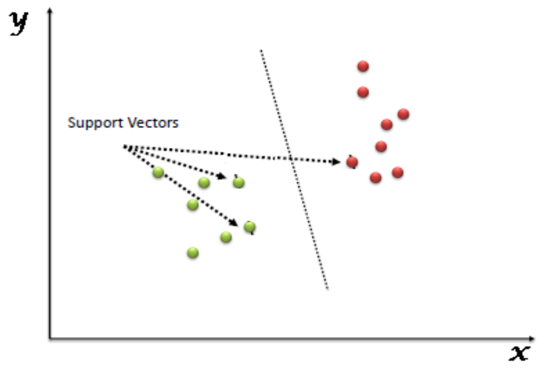


Figure 2.3: Classification by Support Vector Machine

The SVM algorithm's objective is to detect the optimal line or decision boundary for categorizing n-dimensional space into classes so that in the future additional data points may be readily placed in the proper category. The hyperplane is the optimum choice boundary. It perfectly distinguishes two classes (figure 2.3).

For example, to help grasp these ideas, a population split of 50/50 between males and females is considered. A set of criteria needs to be established based on a sample of this population that will help determine the gender class for the remainder of the people. Therefore, a plan is formulated to implement this algorithm for designing a robot that can tell if a person is male or female. Here is an example of a classification analysis issue. The population is divided into two halves implementing a set of rules [6].

For the sake of simplicity, the individual's height and hair length are viewed as the two distinguishing characteristics.

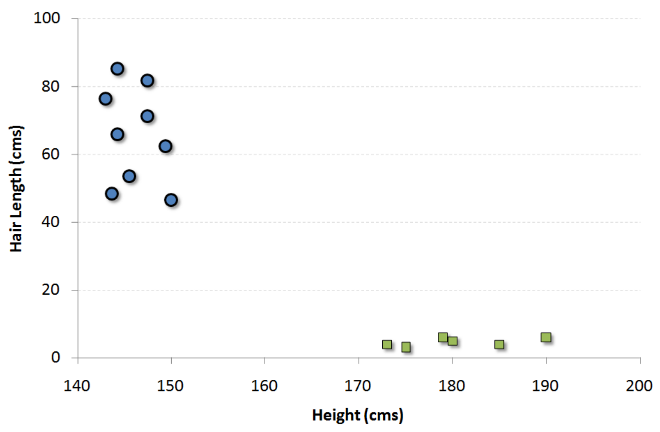


Figure 2.4: Scatter Plot of the Sample

Figure 2.4 shows a scatter plot of the sample. In the plot, females are represented by blue circles while males are represented by green squares. The following are a few predicted conclusions from the graph:

1. Males in our population are taller on average.
2. Females in our population have longer scalp hairs.

For example, if a person with a height of 180 cms and a hair length of 4 cms is observed, the most valid judgment is that this person is a male. That is how a classification analysis is done.

The Support Vector Machine (SVM) is a frontier that best separates men from females. Finding SVM is a lot easier because the two classes are properly divided in this example.

For categorizing the given situation, several frontiers are probably used. Figure 2.5 shows the three possible frontiers.

Determining the most minimal distance between the frontier and the nearest support vector is the simplest method to understand the objective function in SVM (this can belong to any

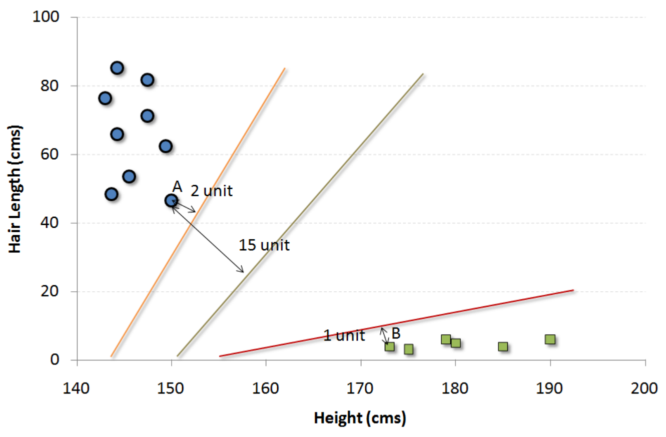


Figure 2.5: Three Possible Frontiers

class). For example, the orange frontier is the closest to the blue circles. The nearest blue circle is 2 units away from the orange frontier. The frontier with the vastest distance is selected after knowing the distances for all of the frontiers (from the closest support vector). Out of the three depicted, the black frontier is the furthest away from the nearest support vector that is 15 units.

If a clear line that divides the classes in the way shown in figure 2.5 is undiscovered, that means if the data points (figure 2.6) are scattered all over the domain, a straight-line frontier directly in the present plane that can act as the SVM in such instances won't be perceived.

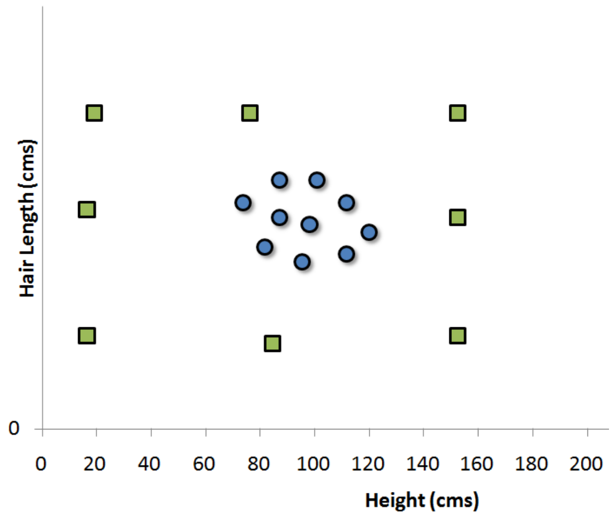


Figure 2.6: Scenerio Without Clean Frontier

In such instances, these vectors must be mapped to a higher-dimensional plane to separate them from one another. Once formulating SVM begins, such scenarios will be covered. For the time being, it may be imagined that such a change will result in the sort of SVM shown in figure 2.7.

In the original distribution, each green square is mapped on a modified scale. And the

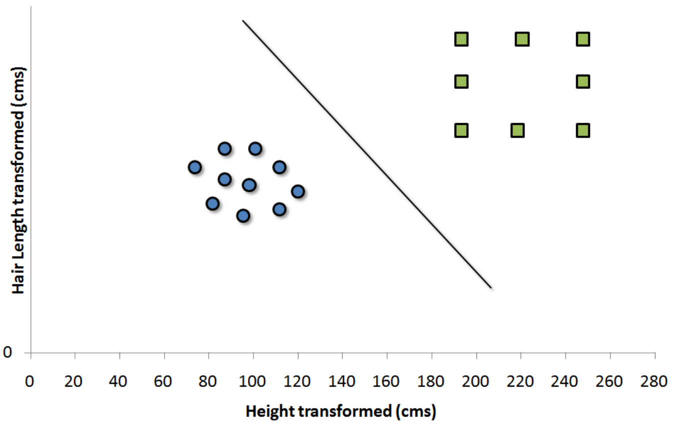


Figure 2.7: Original Distribution Mapped on a Transformed Scale

modified scale clearly distinguishes between classes.

SVM Kernels

A kernel is used to implement the SVM algorithm in practice. An input data space is transformed into the necessary form using a kernel. The kernel trick is a method used by SVM. The kernel converts a low-dimensional input space into a higher-dimensional space in this case. In another way, it turns non-separable issues into separable issues by including extra dimensions to them. It is especially effective in problems with non-linear separation. Kernel technique aids in the development of a more accurate classifier.

- **Linear Kernel:** Any two observations can be a regular dot product using a linear kernel. The total of the multiplication of each pair of input values is the product of two vectors.
- **Polynomial Kernel:** The linear kernel is a more extended version of the polynomial kernel. The polynomial kernel can distinguish if the input space is curved or nonlinear.

Difference between SVC & SVM

They are just different implementations of the same algorithm. The SVM module (SVC, NuSVC, etcetera) is a wrapper around the LibSVM library and supports various kernels, while LinearSVC is based on LibLinear and only supports a linear kernel. Hence, SVC (kernel = 'linear') is in theory "equivalent" to LinearSVC(). Various results will be produced because the implementations are different in practice. The most significant ones being that LinearSVC only supports a linear kernel that is faster and can scale a lot better.

2.8.2 Logistic Regression

The logistic model is used to decide the probability of a particular class or event that has a win/lose, pass/fail outcome. Logistic regression, a statistical model, uses a logistic function in its basic form that models a binary dependent variable, though there exist many complex extensions [7]. In regression analysis, logistic regression estimates the parameters of a logistic model that is a form of binary regression. A binary logistic model has a dependent variable that possesses two possible values mathematically such as pass/fail. An indicator variable represents those two values where the two values are labeled as "0" and "1". Logistic regression is used in numerous fields, including medical fields, machine learning, and social sciences. Logistic regression can be explained with the standard logistic function. The logistic function is a sigmoid function. It takes any real input t, and gives an output between zero and one. This is explained as having output probability while taking input log-odds for the logit. The standard logistic function is $\sigma : \mathbb{R} \rightarrow (0,1)$ that is defined as:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}} \tag{2.4}$$

Here, t is a linear function of a single exponential variable x (the case where t is a linear combination of multiple exponential variables is treated in a similar way). So, t can be expressed as follows:

$$t = \beta_0 + \beta_1 x \tag{2.5}$$

The general logistic function is $p : \mathbb{R} \rightarrow (0,1)$ that can be written as:

$$p(x) = \sigma(t) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{2.6}$$

In the logistic model, $p(x)$ is illustrated as the probability of the dependent variable and Y is illustrated as a success/case rather than a failure/non-case. Clearly, the response variables Y_i are not identically distributed. $P(Y_i = 1 \mid X)$ differs from one data point X_i to another, although they are not dependent given shared parameters β and design matrix X.

2.8.3 Random Forest Classifier

A random forest is a machine learning technique for classifying and predicting outcomes. It employs ensemble learning, a method for resolving complex problems by integrating many classifiers [8]. A random forest algorithm is made up of many decision trees. The 'forest' created by the random forest technique is trained using bagging or bootstrap aggregation.

Bagging is a meta-algorithm that groups machine learning algorithms together to improve accuracy. Based on decision tree predictions, the (random forest) algorithm determines the outcome. It makes predictions by averaging or averaging the output of different trees. As the number of trees grows, the precision of the result improves.

To get the required outcome, random forest classification employs an ensemble methodology. The training data is used to train various decision trees. When nodes are divided, this dataset contains observations and features that will be chosen at random. A rain forest system employs a variety of decision trees. A decision tree has three sorts of nodes: decision nodes, leaf nodes, and the root. The leaf node of each decision tree represents the decision tree's final output. A majority-voting technique is used to select the final product. In this case, the rain forest system's final output is the output chosen by the majority of decision trees. A simple random forest classifier is shown in the diagram (figure 2.8) below.

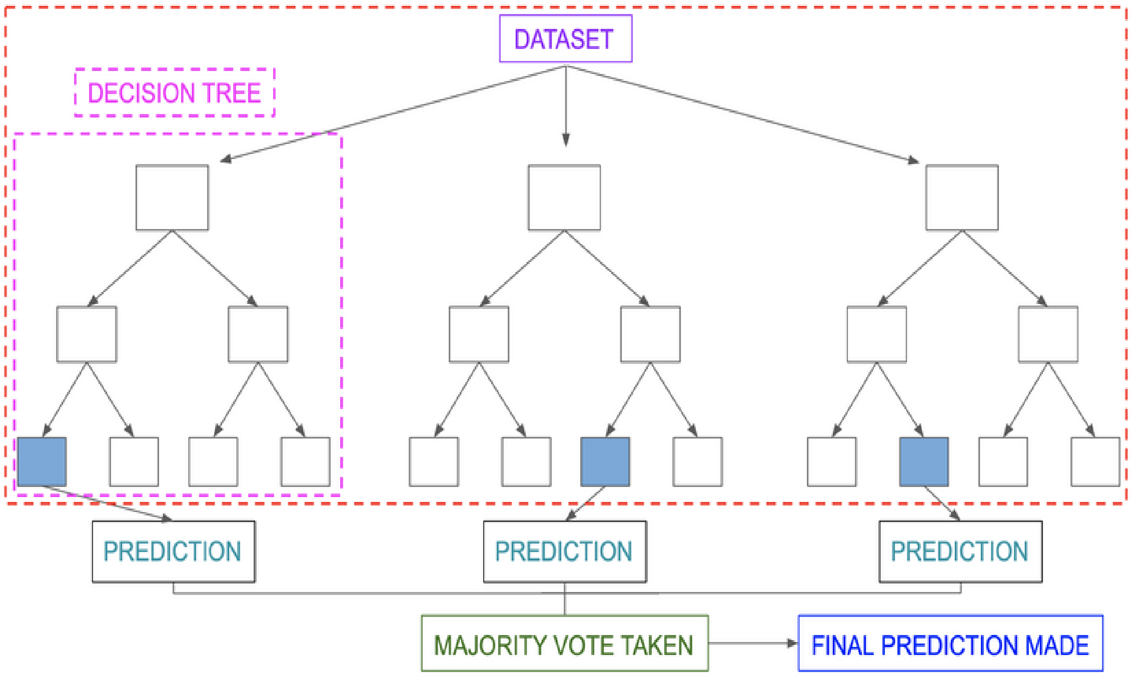


Figure 2.8: Simple Random Forest Classifier diagram

For example, in figure 2.9, a training dataset referred to as "Fruits" includes bananas, pineapples, apples, and mangoes. This dataset "Fruits" is divided into three subsets by the random forest classifier. In the random forest system, every decision tree is given these subsets. Each decision tree gives its output.

For example, The predictions for Tree-1 and Tree-2 are Apple and Tree-n indicated that the output would be Mango. Using the majority voting gathered by the random forest classifier, the final prediction is made.

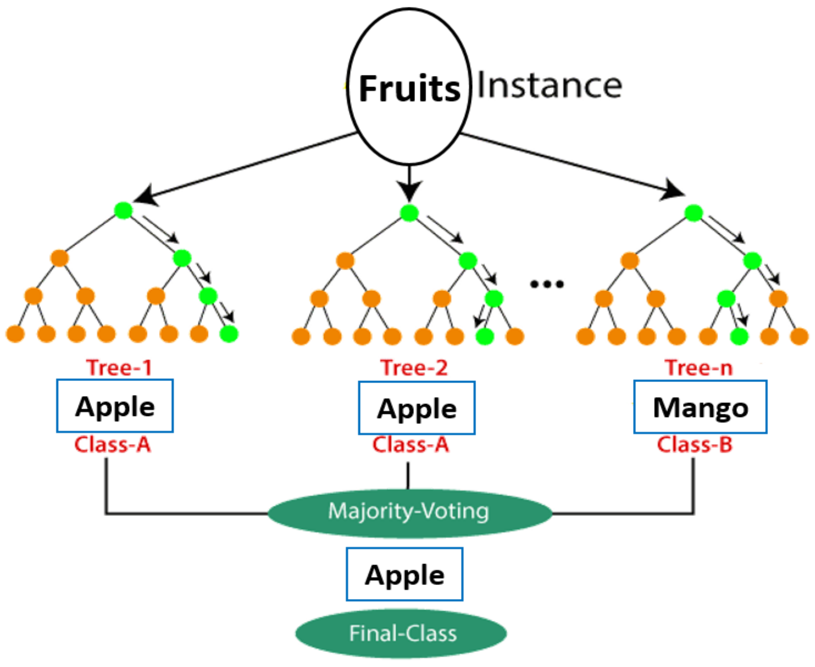


Figure 2.9: Random Forest Classifier Example

So, "Apple" has been chosen as the prediction by the majority of decision trees. As a result, the classifier selects "Apple" as its final prediction.

2.8.4 XGBoost Classifier

Extreme gradient boosting, often known as XGBoost, is a well-known gradient boosting technique(ensemble) that improves the performance and speed of tree-based (sequential decision trees) machine learning algorithms. Tianqi Chen designed XGBoost, which was first maintained by the Distributed (Deep) Machine Learning Community (DMLC). It is the most widely used algorithm for applied machine learning in competitions, and it has grown in popularity as a result of winning solutions in structured and tabular data. Previously, just Python and R packages were available for XGBoost, but it has since been expanded to include Java, Scala, Julia, and more languages [9].

In Ensemble Learning, XGBoost is considered as a boosting technique. Ensemble learning combines different models into a collection of predictors to improve prediction accuracy. The faults created by prior models are attempted to be repaired by subsequent models by adding weights to the models in the boosting technique. XGBoost has the following features:

1. It’s possible to use it on both single and distributed systems (Hadoop, Spark).
2. In supervised learning, XGBoost is employed (regression and classification problems).
3. Parallel processing is supported.

4. Optimization of the cache.
5. Memory management - that is efficient for large datasets that exceed RAM.
6. Has several regularizations that aid in the reduction of overfitting.
7. Auto tree trimming - After reaching specific internal restrictions, the decision tree will stop growing.
8. Missing values are not a problem.
9. Cross-Validation is built-in.
10. Outliers are dealt with to some extent.

The most effective Scalable Tree Boosting Method has been proved to be XGboost. It has displayed remarkable results in a variety of applications, including motion detection, stock sales forecasting, virus classification, consumer behavior analysis, and many others. The system runs far quicker on a single machine with efficient data and memory handling than any other machine learning technique. The optimization approaches used by the algorithm can improve performance and provide speed while using the fewest resources possible.

2.8.5 Multinomial Naïve Bayes

The Multinomial Naïve Bayes algorithm is a probabilistic learning approach that is used in Natural Language Processing (NLP) [10]. The Bayes theorem-based system predicts a text's tag, such as an email or a newspaper article. It calculates each tag's likelihood for each sample and outputs the tag with the highest probability. Computing class probabilities by multinomial Naïve Bayes for a document is discussed below.

Here, **C** is referred to as the set of classes. Let, the size of the vocabulary be **N**. Then MNB assigns a test document t_i to the class that has the highest probability Pr(c | t_i). Using Bayes' rule, the equation can be written as:

$$Pr(c|t_i) = \frac{Pr(c)Pr(t_i|c)}{Pr(t_i)}, \quad c \in C \tag{2.7}$$

The class prior is Pr(c). It can be calculated by dividing the number of documents that belong to class **c** by the total number of documents. Pr(t_i | c) is the probability of getting a document similar to t_i in class **c**. It is calculated as:

$$Pr(t_i|c) = (\textstyle\sum_n f_{ni})! \prod_n \frac{Pr(w_n|c)^{f_{ni}}}{f_{ni}!} \tag{2.8}$$

Here, f_{ni} is the count of word (n) in the test document t_i. $\Pr(w_n|c)$ is the probability of word (n) given by class c. The latter probability is calculated from the training documents as:

$$\widehat{Pr}(w_n|c) = \frac{1 + F_{nc}}{N + \sum_{x=1}^{N} F_{xc}} \tag{2.9}$$

Here, F_{xc} is the count of word **x** in all the training documents that belong to class **c**. To avoid the zero-frequency problem, the Laplace estimator is used to prime each word's count with one. The normalization factor $\Pr(t_i)$ in Equation 2.7 can be computed using:

$$Pr(t_i) = \sum_{k=1}^{|c|} Pr(k)Pr(t_i|k) \tag{2.10}$$

Here, the computationally expensive terms $(\sum_n f_{ni})!$ and $(\prod_n f_{ni})!$ in Equation 2.8 can be deleted or avoided without any change in the results because these two terms does not depend on the class **c**. Equation 2.8 can be written as:

$$Pr(t_i|c) = \alpha \prod_n Pr(W_n|c)^{f_{ni}} \tag{2.11}$$

Here α is a constant. It drops out because of the normalization process.

2.8.6 Decision Tree Classifier

In the field of data science, decision trees are commonly used. It's a tested method for making decisions in complex situations [11]. Ensemble approaches such as decision trees and random forests are commonly utilized in machine learning. Decision trees are a supervised learning method in which input is constantly split into distinct groups based on specified parameters. As the name implies, a decision tree is a flow-like tree structure that works on the notion of conditions. It is effective and employs powerful algorithms for predictive analysis. Internal nodes, branches, and a terminal node are the main characteristics. Every internal node represents a "test" on an attribute, branching represents the test's conclusion, and leaf nodes represent the class label. When it comes to supervised learning techniques, this is the most frequently used algorithm. Let's look at some of the terminologies utilized in Decision Tree, which are listed below:

Branches: The divisions of the tree as a whole.

Root: It represents the entire sample before it is subdivided.

Splitting: It is the process of dividing nodes.

Terminal node: It refers to a node that does not split further.

Decision Node: A decision node is a node that is further subdivided into different sub-nodes.

Pruning: The process of removing subnodes from a decision node.

When a node is further divided, it is referred to as a parent node, whereas the divided nodes, or sub-nodes, are referred to as a child node of the parent node.

It works with both categorical and continuous input and output types. In classification problems, the decision tree asks questions and divides data into sub-branches based on the replies of yes/no.

The algorithm in a decision tree begins with the root of the tree, compares the values of various attributes, and then proceeds to the next branch until it reaches the final leaf node. It employs various techniques to examine the split and variables that allow for the most homogeneous population groups. Figure 2.10 shows an example of the decision tree.

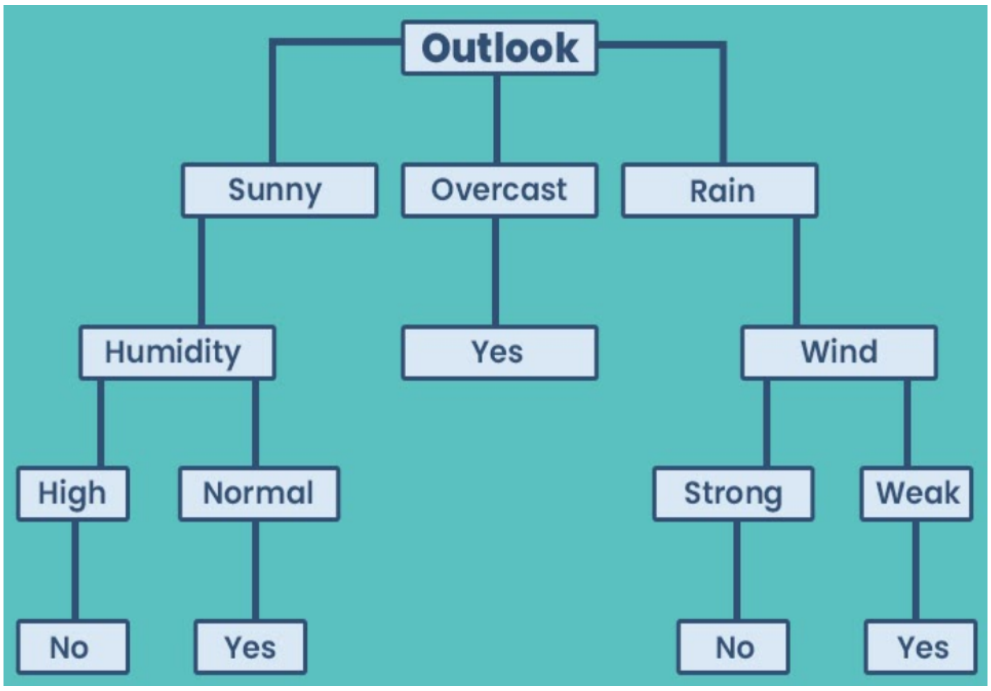


Figure 2.10: Weather Example of a Decision Tree

2.8.7 Multilayer Perceptron

The MLP is the most extensively used neural network structure, especially the 2-layer structure, which connects the input units and output layer with an intermediary hidden layer. Each neuron in the network has a differentiable nonlinear activation function, which allows the network to conduct static mapping between input and output spaces. Mathematically this can be written as:

$$y = \varphi(\sum_{i=1}^{n} w_i x_i + b) = \varphi(w^T x + b) \tag{2.12}$$

where w denotes the vector of weights, x is the vector of inputs, b is the bias, and φ is the activation function. The original perceptron of Rosenblatt used a Heaviside step function as the activation function φ. Nowadays, especially in multilayer networks, the activation function is often chosen to be the logistic sigmoid $1/1 + e^{-x}$ or the hyperbolic tangent $\tanh(x)$. They are related by $(\tanh(x) + 1)/2 = 1/(1 + e^{-2x})$. These functions are used because they are mathematically convenient and are close to linear near origin while saturating rather quickly when getting away from the origin. This allows MLP networks to model well both strongly and mildly nonlinear mappings.

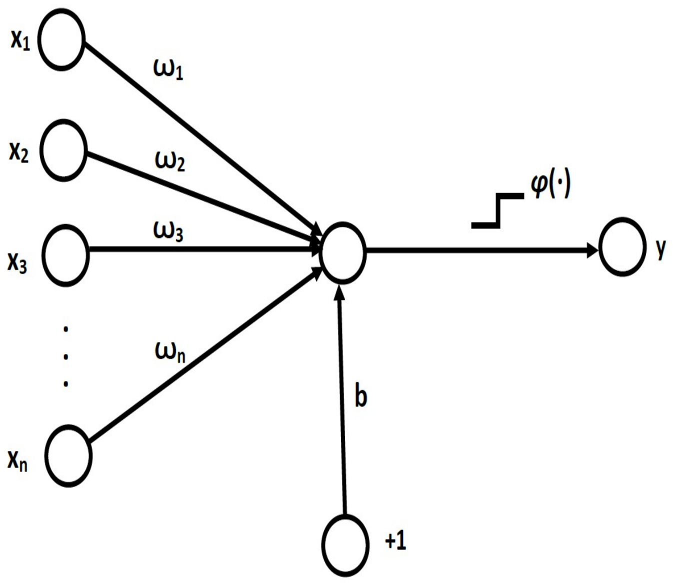


Figure 2.11: Signal-Flow Graph of the Perceptron

Because of its poor mapping capacity, a single perceptron isn't very useful. The perceptron can only represent an oriented ridge-like function, regardless of the activation function utilized. The perceptrons, on the other hand, can be employed as components of a bigger, more practical framework. An input layer of source nodes, one or more hidden layers of computation nodes, and an output layer of nodes make up a standard multilayer perceptron (MLP) network. Layer by layer, the input signal travels through the network. The mathematical expressions for a feedforward network with a single hidden layer, nonlinear activation functions, and a linear output layer are as follows:

$$x = f(s) B\varphi(As + a) = b \tag{2.13}$$

where s is an input vector and x is an output vector. The first layer's weight matrix is A, while the first layer's bias vector is a. The second layer's weight matrix and bias vector are denoted by B and b, respectively. An element-wise non-linearity is denoted by the function φ. The model's generalization to more hidden layers is clear.

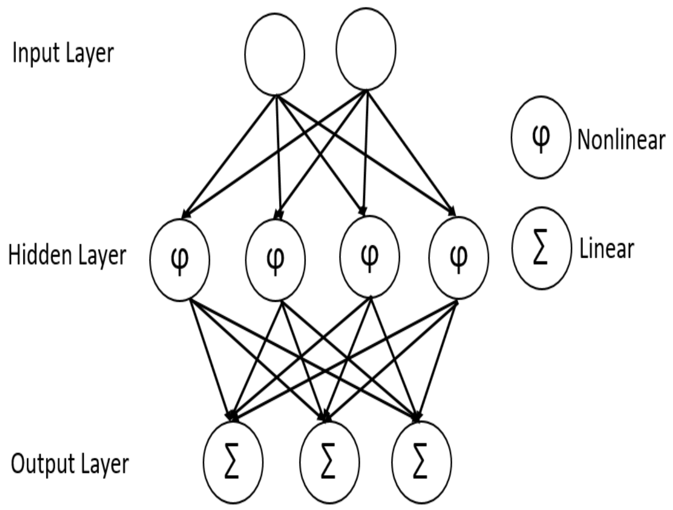


Figure 2.12: Signal-Flow Graph of the MLP

Chapter 3

Implemented Models

This chapter focuses on the complete workflow of creating a conversational chatbot and an emotion detector for text data. A series of steps must be followed to accomplish this goal. The method for detecting emotions from texts consists of two approaches which are Word-based and Learning-based. The method to create a chatbot and get responses from it consists of two approaches which are Keyword-based Search and Retrieval Based Approach.

3.1 Emotion Detection

3.1.1 Word-Based Approach

We used the NLTK (Natural Language Toolkit) package in the word-based approach [12] because it is the best for human language data. We begin by assigning all of the Emotion Labels (joy, love, fear, anger, neutral, sadness, and surprise) that we will detect from the text. Then we take the textual input and remove any unnecessary characters from the sentences in the textual data to be detected.

Dataset

The initial dataset came from a research of the Word-Emotion Association (a.k.a. NRC Emotion Lexicon) [13]. We utilized this dataset because it is comparable to this study subject.

Figure 3.2 shows the distribution of emotion classes.

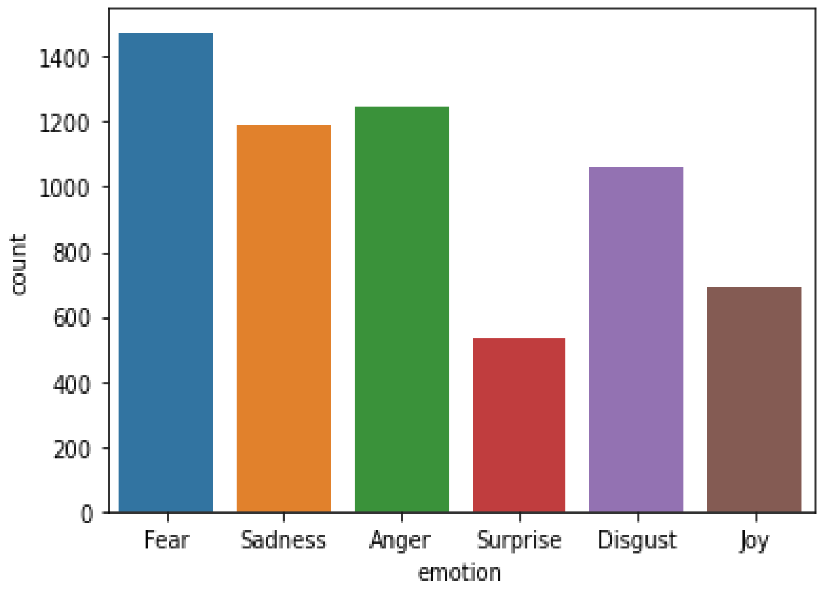


Figure 3.1: Distribution of Emotions

Algorithm and Flow Chart

The algorithm of word-based approach is shown below:

Algorithm 1: Word-Based Approach

```
1 Step-1: Collect Text from dataset.
2 Step-2: Then clean the collected text.
3     Step-2.1: After cleaning, Stopwords need to be removed.
4     Step-2.2: Tokenize the text.
5     Step-2.3: Then stem and lemmatize the tokenized words.
6 Step-3: Take each token from sentence and match the token with every word of
    dataset. If matches then increment the emotion corresponding to that word. Do
    this procedure until the last token of the sentence found.
```

The graphical representation(a.k.a flow chart) is shown in figure 3.2. It will show the flow of information. Which will make an easy understanding of the algorithm which is very helpful for understanding the flow of information.

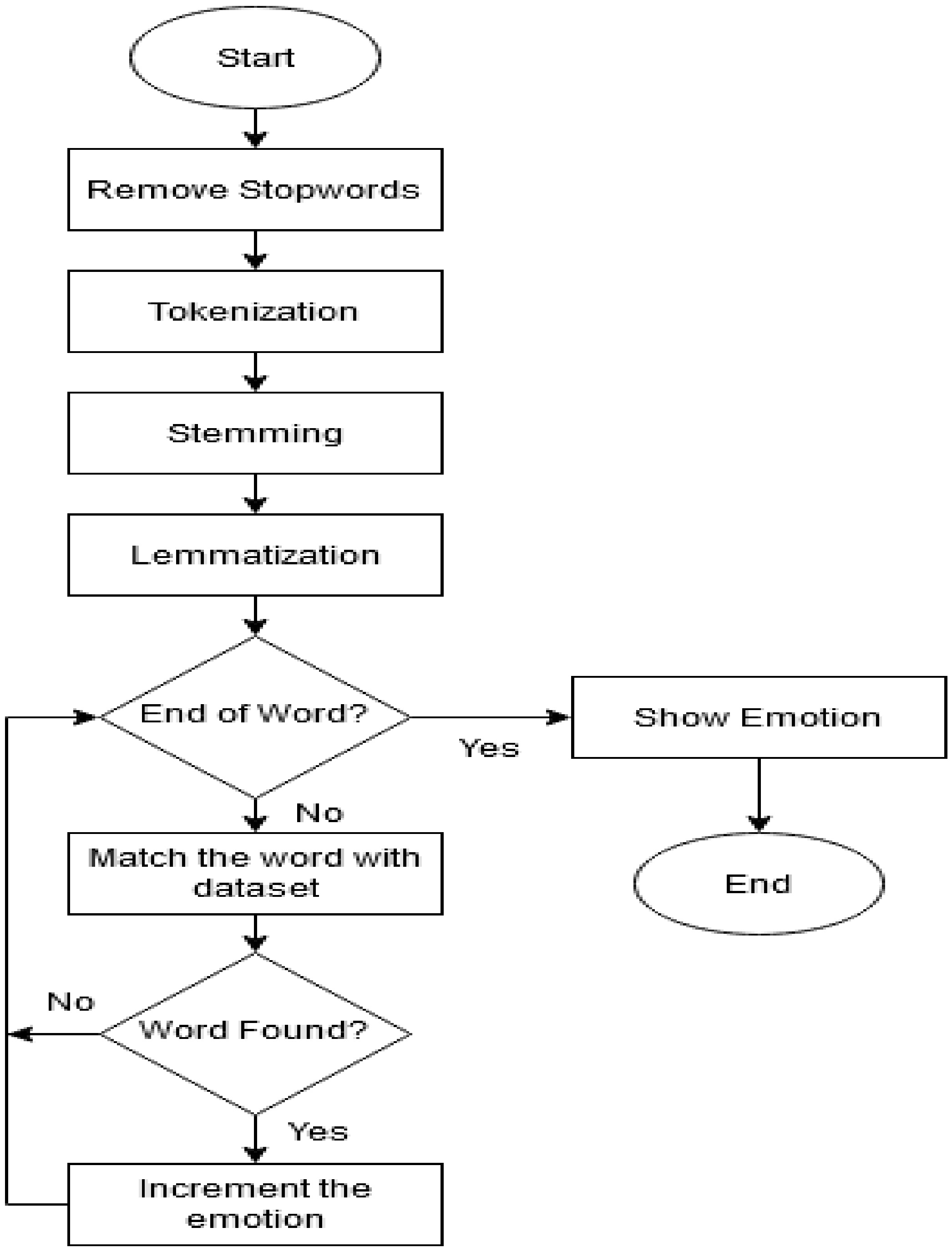


Figure 3.2: Flow Chart of Word-Based Approach

Data Cleanning

Text cleaning is defined as a set of procedures for cleaning and standardizing text data to convert it into a format that can be consumed as input by a classification system. The main reason is that punctuation and other phrases have minimal relevance when the text is assessed and utilized to extract attributes. The cleaning system can use a variety of approaches, including:

- **Removing stopwords**
  Stopwords are words that don't add much to a sentence's meaning. They may be safely ignored without jeopardizing the sentence's meaning. As an example: And, but, or , ! , #, $ etc.

- **Stemming and Lemmatization**
  The basic or root form of a word is known as the word stem, and it may be changed or created by adding prefixes or suffixes to it. Stemming is the process of extracting the word stem from a word. As an example: stemming for root word "like" include:

  ->"like": "likes"
  ->"like": "liked"

  Lemmatizing is a process that is quite similar to stemming. The main distinction is that, as previously stated, stemming frequently produces non-existent words, whereas lemmas are genuine words. So, while you can't look up your root stem (the word you end up with) in a dictionary, you can look up a lemma. As an example: Examples of lemmatization:

  -> rocks: rock
  -> better: good

- **Replace Emoji**
  Sometimes in a chat people use emoji to express their emotion. So if emoji can be replaced with meaningful text that will be easier to implement in our model.
  For example:
  : (= sad
  ;-) = affection
  xD = laugh
  XD = laugh

3.1.2 Learning-Based Approach

In the Learning-based approach [12] we have utilized Twitter tweets. Extracted Tweets will be kept in an excel file with two columns: one for the tweets, another column for the emotion label. That will contain emotions that we have already established. We split the training dataset and the test dataset in this way. The computer is trained and tested using the training dataset.

Dataset

The dataset is collected from tweeter text from Kaggle [14]. We choose this dataset since it is a similar sort of dataset to the one we used for this research. Table 3.1 shows the summary details of the dataset that we used for our emotion detector.

Table 3.1: Summary Details of the Dataset

Association	Number of Sentences	Number of Emotions	Categories (Emotions)
Emotions in Text	25000	7	joy, anger, love, fear, sadness, surprise, neutral

Figure 3.3 shows the distribution of emotions in the original dataset.

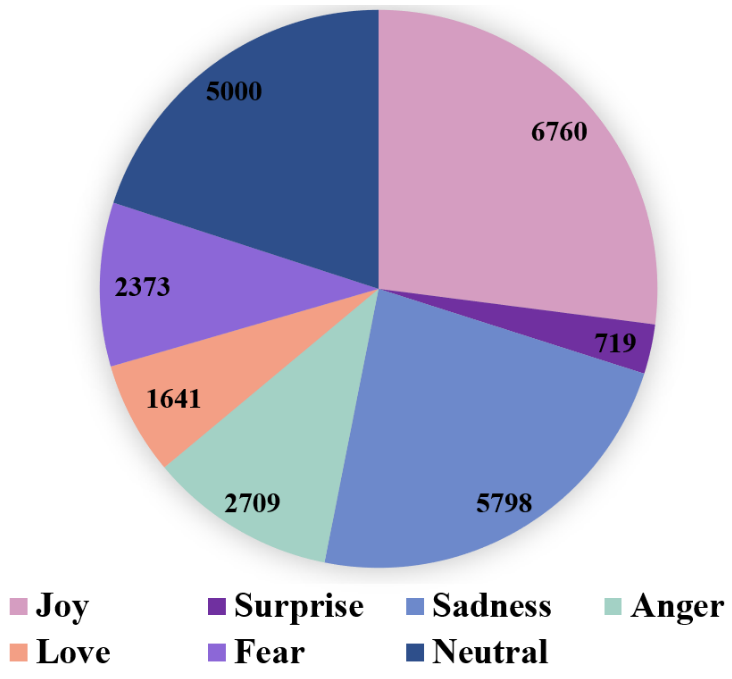


Figure 3.3: Distribution of Emotions

Working Procedure

For detecting emotion, some necessary steps should be taken into consideration. Firstly, user input will be needed and then the text needs to be cleaned to detect the emotion. After the cleaning process, feature extraction needs to be done. Then the data is split into the test-train. Then various supervised classification algorithm is used to detect the emotion label. This whole work procedure is shown in a graphical manner (figure 3.4) for an easy understanding.

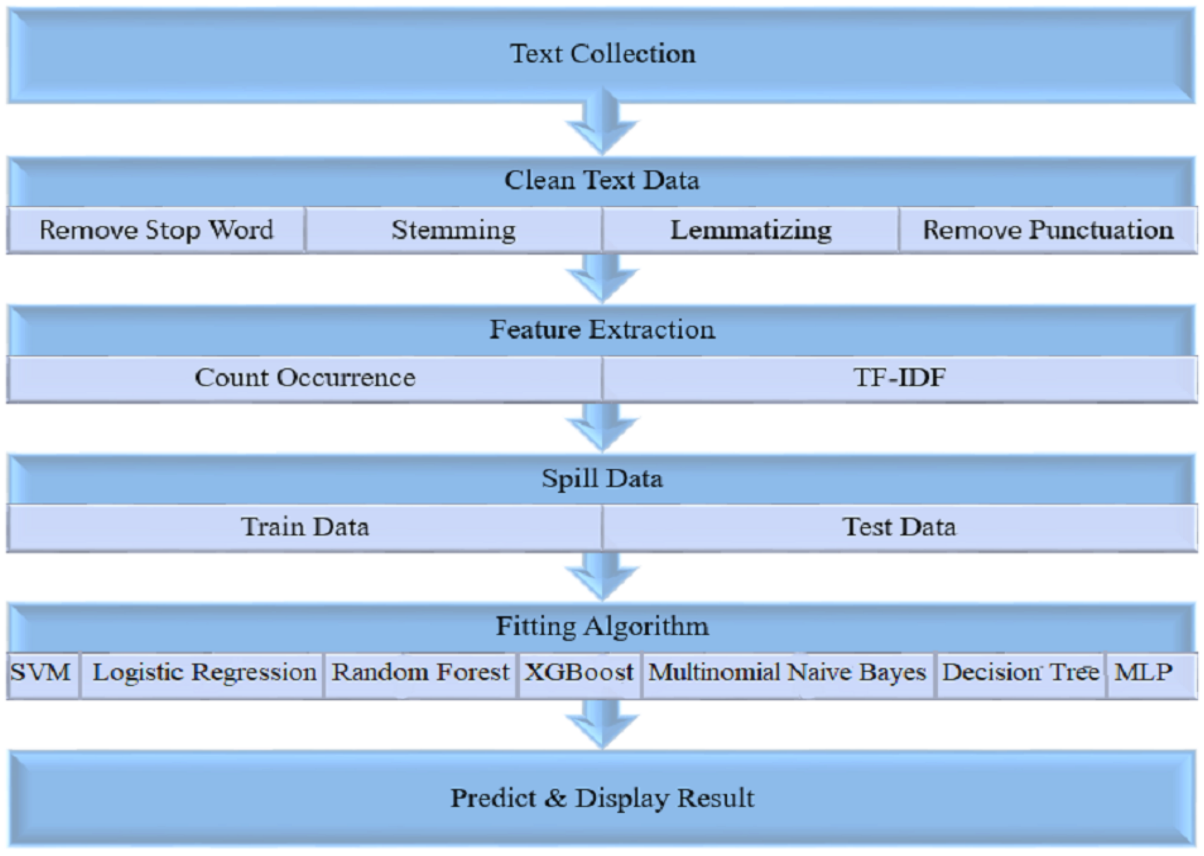


Figure 3.4: Working Procedure of Emotion Detection

Feature Extraction:

1. **Count Occurrence:** It Simply helps to determine the frequency of a word in a document This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors. For an example:

   Doc 1: I didn't feel humiliated.

   Doc 2: I'm grabbing a minute to post I feel greedy wrong.

   Doc 3: I have been feeling the need to be creative.

Firstly, we will Clean the documents such as preprocessing, remove punctuation, stemming, lemmatizing and next we will tokenize the words with frequency (Table 3.2).

Table 3.2: Example of Count Occurrence for Three Documents

	creative	feel	grab	greedy	humiliate	minute	need	post	wrong
Doc 1	0	1	0	0	1	0	0	0	0
Doc 2	0	1	1	1	0	1	0	1	1
Doc 3	1	1	0	0	0	0	1	0	0

2. **TF-IDF:** It stands for Term Frequency Inverse Document Frequency of records. It can be defined as the calculation of how relevant a word in a series or corpus is to a text.

   For an example [15]:

   Document 1: It is going to rain today.

   Document 2: Today I am not going outside.

   Document 3: I am going to watch the season premiere.

   Step 1: Find TF

   We will determine the value of TF according the following formula -

$$\text{TF}(t,d) = \frac{\text{Number of times term } t \text{ appears in } d}{\text{Total number of terms in } d} \tag{3.1}$$

   Table 3.3 shows the TF values for the three documents -

Table 3.3: Term Frequency for Documents

Words/Documents	**Document 1**	**Document 2**	**Document 3**
going	0.16	0.16	0.12
to	0.16	0	0.12
today	0.16	0.16	0
I	0	0.16	0.12
am	0	0.16	0.12
it	0.16	0	0
is	0.16	0	0
rain	0.16	0	0

   Step 2: Find IDF

   We will determine the value of IDF according the following formula -

$$\text{IDF}(t) = \log(\frac{\text{Total number of documents}}{\text{Number of documents with term } t}) \tag{3.2}$$

Table 3.4 shows the TF values for the three documents -

Table 3.4: Inverse Document Frequency for Documents

Words/ Documents	**IDF Values**
going	$\log_e(3/3)$
to	$\log_e(3/2)$
today	$\log_e(3/2)$
I	$\log_e(3/2)$
am	$\log_e(3/2)$
it	$\log_e(3/1)$
is	$\log_e(3/1)$
rain	$\log_e(3/1)$

Step 3: Find TF-IDF

By using the following formula, we can determine the value of TF-IDF of every word in the documents -

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) * \text{IDF}(t) \tag{3.3}$$

Table 3.5 shows the TF values for the three documents -

Table 3.5: Term Frequency-Inverse Document Frequency for Documents

	going	to	today	I	am	it	is	rain
Doc 1	0	0.07	0.07	0	0	0.17	0.17	0.17
Doc 2	0	0	0.07	0.07	0.07	0	0	0
Doc 3	0	0.05	0	0.05	0.05	0	0	0

3.2 Conversational Chatbot

A conversation chatbot will perform better if we apply a perfect model to train with. Here two approaches are proposed one is keyword-based and another one is Retrieval based Approach

3.2.1 Keyword-Based Search

In Keyword-Based Search, the user will give an input sentence, and then keywords in that sentence will be recognized. Then the intent score will be determined by matching those keywords with the existing keywords in the dataset. In this way, the chatbot will be capable of classifying the sentence according to the intent score. The chatbot will give the user a suitable response according to the sentence classifier.

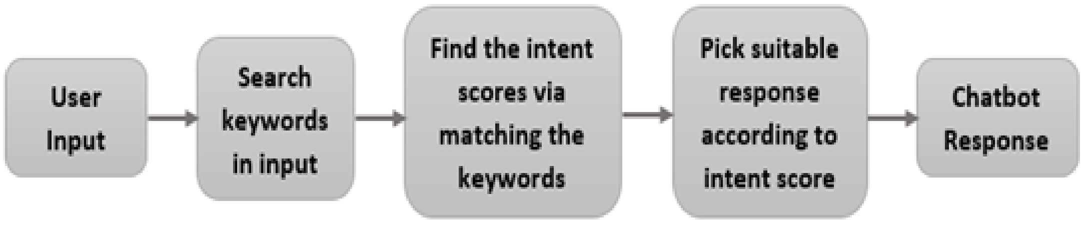

User Input
Search keywords in input
Find the intent scores via matching the keywords
Pick suitable response according to intent score
Chatbot Response

Figure 3.5: Keyword-Based Search Procedure

For example:
Input Text: Hi Good Evening
Term-1: Hi (intent: greeting)
Term-2: Good (intent: compliment)
Term-3: Evening (intent: greeting)
Classification Score: greeting = 2, compliment = 1
Response: Hello, Good Evening. (According to high score of greeting intent)

This is not an efficient approach. Therefore, the second approach which is Retrieval Based Approach will be used to implement our model.

3.2.2 Retrieval Based Approach

In Retrieval based approach, the user will give an input sentence, then using **Ensemble Learning** approach best intent is filtered out. After that, we calculate the cosine distance so that chatbot can give its best reply. The less value of cosine distance, the more suitable the response is.

Ensemble Learning

Ensemble learning works on the principle that a weak learner predicts poorly when alone. But when combined with other weak learners, they create a strong learner. The result obtained from an ensemble, a combination of machine learning models, can be more accurate than any single member of the group.

For Example:
Input Text: Hello!
Intent using SVM: Greeting
Intent using Naive Bayes: Sad
Intent using XGBoost: Greeting
Intent using Decision Tree: Greeting
Intent using Random Forest: Greeting
Intent using Logistic Regression: Greeting
Intent using Multi Layer Perceptron: Greeting

Six of the seven classifiers judged this input text to be a 'Greeting,' while just one predicted it to be a 'Sad.' From the working procedure of ensemble learning, several models make predictions. Those are known as votes; each prediction counts as a vote. As is often the case with voting, a decision is often in favor of most votes. The same applies here. The prediction made by the majority of classifiers becomes the final prediction. Now EDAIC will choose a reply according to cosine distance within the best intent.

Here,

Best Intent: Greeting
EDAIC: Hi, how are you?

Cosine Distance Determination

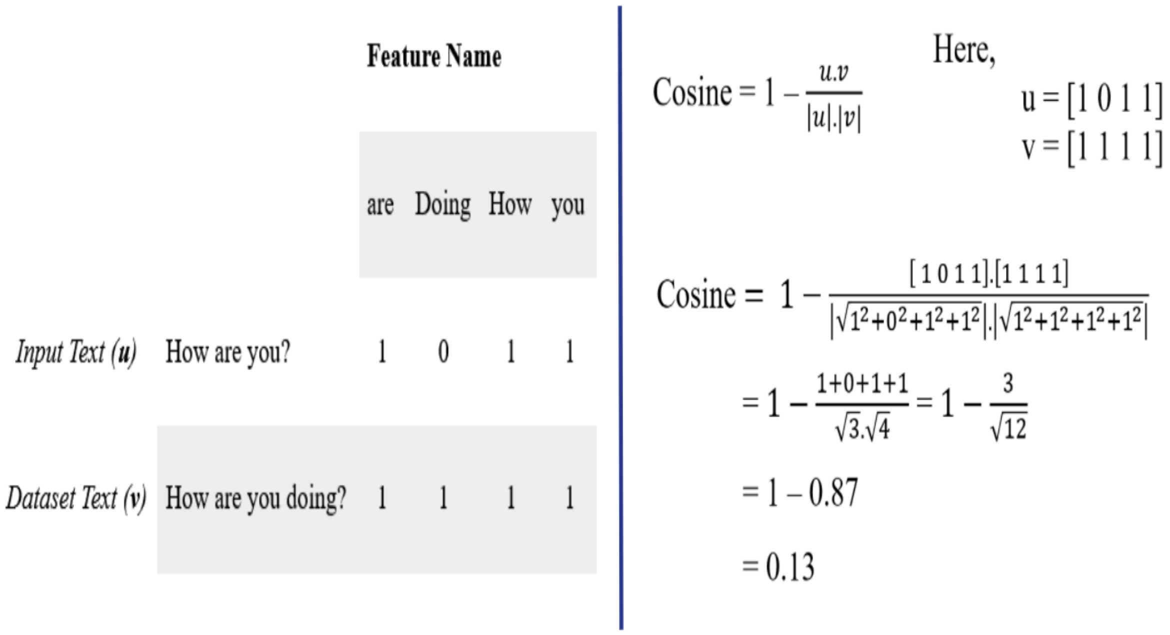
At first, the total number of words in a sentence needs to count through CountVectorizer(). Next, we need to find the cosine distance to identify which response is more suitable. To measure the cosine distance, the formula is:

$$\text{cosine distance} = 1 - \frac{u.v}{|u|.|v|} \tag{3.4}$$

Terminology
u = Feature vector of input text
v = Feature vector of each user's text from Dataset

The less the value of cosine distance, the more suitable the response is.

For example:



		Feature Name			
		are	Doing	How	you
*Input Text (**u**)*	How are you?	1	0	1	1
*Dataset Text (**v**)*	How are you doing?	1	1	1	1

$$\text{Cosine} = 1 - \frac{u.v}{|u|.|v|}$$

Here,

$u = [1\ 0\ 1\ 1]$

$v = [1\ 1\ 1\ 1]$

$$\text{Cosine} = 1 - \frac{[1\,0\,1\,1].[1\,1\,1\,1]}{\left|\sqrt{1^2+0^2+1^2+1^2}\right|.\left|\sqrt{1^2+1^2+1^2+1^2}\right|}$$

$$= 1 - \frac{1+0+1+1}{\sqrt{3}.\sqrt{4}} = 1 - \frac{3}{\sqrt{12}}$$

$$= 1 - 0.87$$

$$= 0.13$$

Here, on the left side, we can see that - the input text is "How are You" and the Dataset text is "How are you doing?". Our Feature names are "are", "doing", "how" and "you". Now if we match the feature names with the words of input and dataset text then we can see that the words in the dataset text are similar to the feature names and so we got "1111" as our v vector. But in the input text, there is no word that matches with the feature name "Doing" and so the value under "Doing" is 0 and so we got "1011" as our u vector.

on the right side, We will now calculate the cosine distance. Here, the u vector is "1011", and the v vector is "1111". Now after calculating, we found that 0.13 is the cosine distance.

Psuedo Code

If multiple cosine distance values are same then randomly a value is chosen and returned. Visualization will be easy through the pseudo code for generating reply from chatbot.

The pseudo code of extracting the best intent is shown below (Function 1):

```
Function 1: extract_best_intent (Predicted Intents)
  Input: A list of intents predicted by seven models, model_weight
  Output: bestIntent
1 $predList \leftarrow$ all predicted intent list
2 for each $intent$ in $predList$ do
3     $Intent\_scores[intent] \leftarrow Intent\_scores[intent] + model\_weight$
4 $bestIntent \leftarrow$ intent with the highest score
5 return $bestIntent$
```

The pseudo code of calculating the cosine distance is shown below (Function 2):

```
Function 2: cosine_distance(s1, s2)
  Input: s1, s2
  Output: similarity
1 $s1$ ← user input
2 $s2$ ← text of the user column in the dataset
3 convert $s1$, $s2$ into feature vector using countVectorizer() object
4 find the vocabulary matrix and the document term matrix
5 measure the cosine distance of the vocabulary matrix and the document term matrix
6 $similarity$ ← (1 - cosine distance)
7 return $similarity$
```

The pseudo code of getting the the reply from the chatbot is shown below (Function 3):

```
Function 3: Respond(text, Intent)
  Input: user input, best intent
  Output: reply
1 $data$ ← dataset carrying best intent
2 for each $sentence$ in $data$ do
3     $similarity$ ← $cosine\_distance(text, sentence)$
4     append the reply associated with the similarity to a list
5 $reply$ ← the reply which has maximum similarity from that list
6 return $reply$
```

Dataset

A chatbot is used for a variety of functions including customer support and queries, request routing, and information collecting. The Datasets used for conversational chatbot are "Intent Classification" [16], "context-free-dataset" [17]. We have made our dataset from these two datasets. Table 3.6 shows the summary details of the intents.

Table 3.6: Summary Details of the Dataset Intents

Association	Number of Intent	Categories
Intent Classification	27	Happy_Excited_Joy, Info, Confirm, Health, Love, Thanks, Greeting, Joke, Time, etc.

There are a total number of twenty-seven intents. These intents are selected according to the daily life conversation. So, it's a closed domain dataset. Here, figure 3.6 shows the distribution of intents.

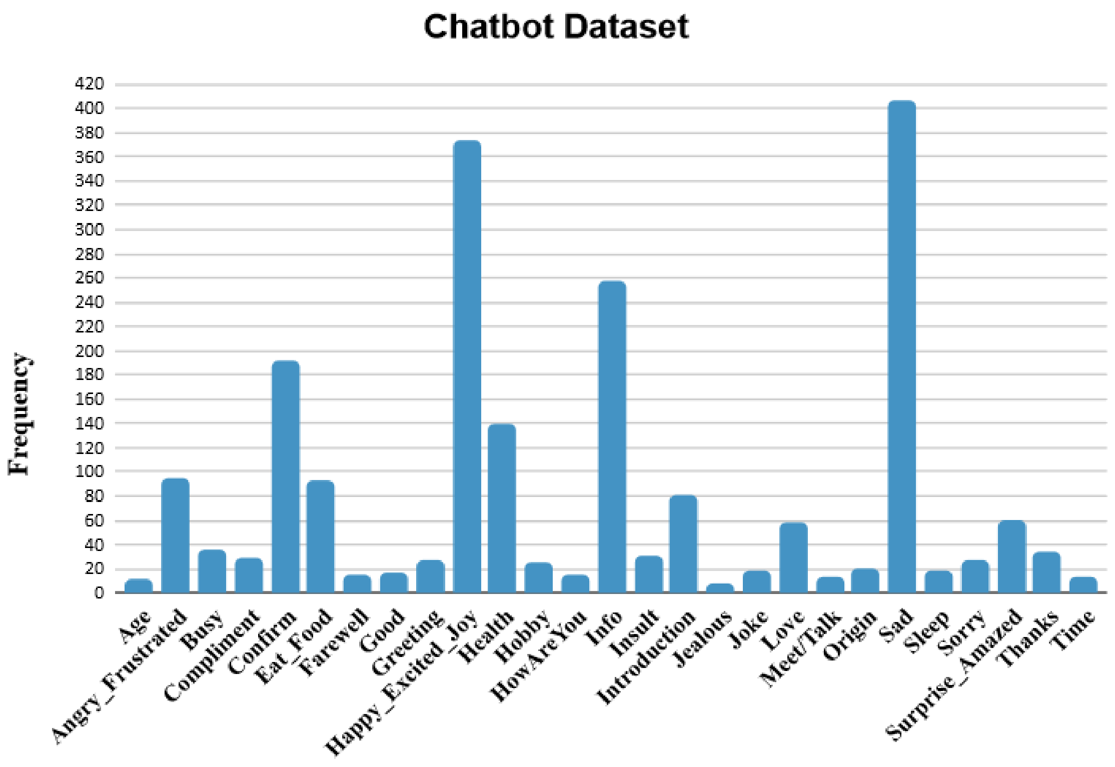


Figure 3.6: Distribution of Intents

3.3 Emotion Feedback

We have implemented a simple application using Python language and PyCharm software where users can chat and interact with EDAIC. Mainly, we created a chatbot that can chat with users. We created an emotion detector that can give us real-time emotion. Then we merged these two things to make one thing that is our EDAIC which is an emotionally aware chatbot.

At the time of chatting with EDAIC, users can rate the emotion based on how good or bad the predicted emotion is. The rating will be from 1 to 5. Here the numbering indicates 1(Very Bad), 2(Bad), 3(Average), 4(Good), and 5(Very Good). Users can also choose the right emotion if the predicted emotion is not up to the users' expectations. The options for emotions are Joy, Love, Surprise, Neutral, Sadness, Fear, and Anger. After receiving the feedbacks, we are storing them in a CSV file. This CSV file stores other data too. Mainly, we are keeping this feedbacks for our future work that is the self-learning feature of our chatbot.

Chapter 4

Simulation and Experimental Results

4.1 Emotion Detection

Pandas has been used to analyze the data set. After analyzing the data, Scikit-learn is used to extract features. This library is also used for building linear models and evaluating the accuracy metrics. Matplotlib library is used to visual the data.

From Work procedure step by step simulation is given below:

4.1.1 Text Collection

Text is collected from the Dataset [14]. For an example:

Doc 1: I didn't feel humiliated.

Doc 2: I'm grabbing a minute to post I feel greedy wrong.

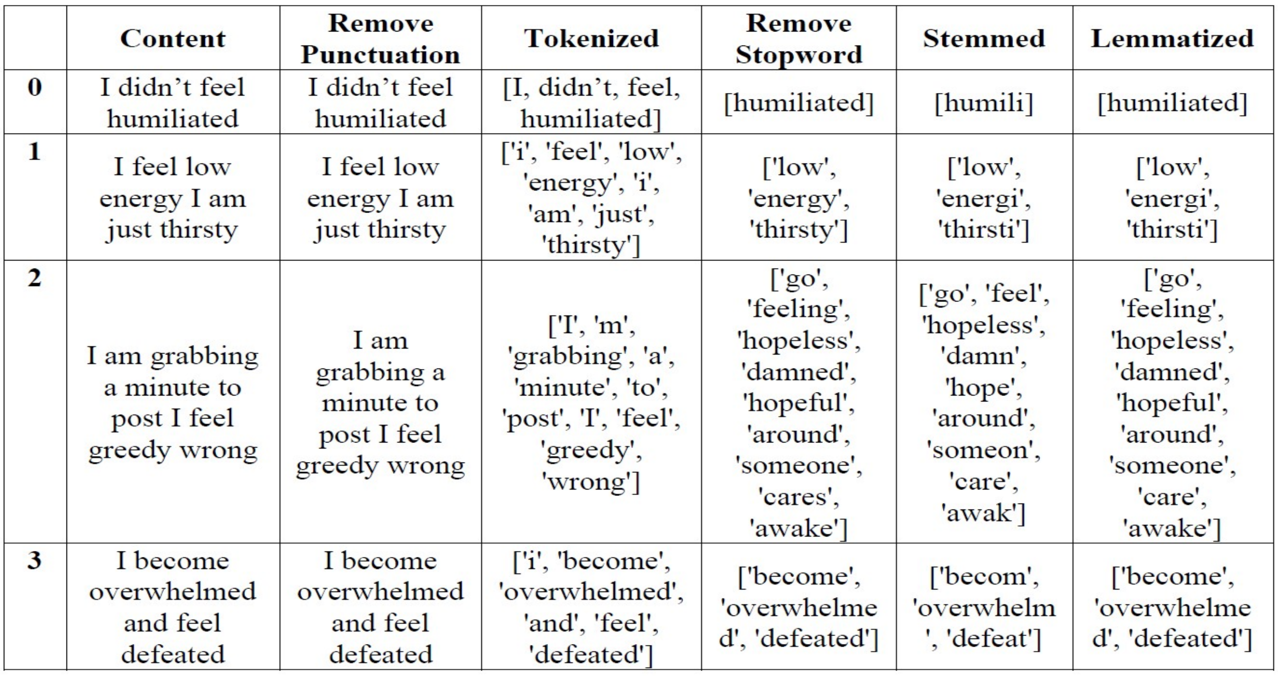
Doc 3: I have been feeling the need to be creative.

4.1.2 Text Cleaning

For cleaning text, we approached these methods - remove stop word, replace Emoji with text, remove punctuation, lemmatizing. The description of these methods was previously stated in the implemented model chapter [Reference: Data Cleaning 6].

An example in table 4.1 shows the steps of clean text.

Table 4.1: Steps of Text Cleaning



	Content	Remove Punctuation	Tokenized	Remove Stopword	Stemmed	Lemmatized
0	I didn't feel humiliated	I didn't feel humiliated	[I, didn't, feel, humiliated]	[humiliated]	[humili]	[humiliated]
1	I feel low energy I am just thirsty	I feel low energy I am just thirsty	['i', 'feel', 'low', 'energy', 'i', 'am', 'just', 'thirsty']	['low', 'energy', 'thirsty']	['low', 'energi', 'thirsti']	['low', 'energi', 'thirsti']
2	I am grabbing a minute to post I feel greedy wrong	I am grabbing a minute to post I feel greedy wrong	['I', 'm', 'grabbing', 'a', 'minute', 'to', 'post', 'I', 'feel', 'greedy', 'wrong']	['go', 'feeling', 'hopeless', 'damned', 'hopeful', 'around', 'someone', 'cares', 'awake']	['go', 'feel', 'hopeless', 'damn', 'hope', 'around', 'someon', 'care', 'awak']	['go', 'feeling', 'hopeless', 'damned', 'hopeful', 'around', 'someone', 'care', 'awake']
3	I become overwhelmed and feel defeated	I become overwhelmed and feel defeated	['i', 'become', 'overwhelmed', 'and', 'feel', 'defeated']	['become', 'overwhelmed', 'defeated']	['becom', 'overwhelm', 'defeat']	['become', 'overwhelmed', 'defeated']

4.1.3 Feature Extraction

There are two approaches for feature extractions-

1. Count Occurrence

2. TF-IDF

Count Occurrence:

It simply helps to determine the frequency of a word in a document. It is helpful when we have multiple texts, and we wish to convert each word in each text into vectors. Let's consider three documents from the dataset:

Doc 1: I didn't feel humiliated.

Doc 2: I'm grabbing a minute to post I feel greedy wrong.

Doc 3: I have been feeling the need to be creative.

Firstly, we will clean the documents using preprocessing such as removing punctuation, removing stopwords, stemming, and lemmatizing. Next, we will tokenize the words with frequency. Table 4.2 shows the count occurrence for the three documents.

Table 4.2: Count Occurrence for Documents

	creative	feel	grab	greedy	humiliate	minute	need	post	wrong
Doc 1	0	1	0	0	1	0	0	0	0
Doc 2	0	1	1	1	0	1	0	1	1
Doc 3	1	1	0	0	0	0	1	0	0

TF-IDF:

It stands for Term Frequency - Inverse Document Frequency. It can be defined as the calculation of how relevant a word in a series or corpus is to a text. We will take previous example to determine TF-IDF.

Step-1: Find TF

We will determine the values (table 4.3) of TF according to the following formula -

$$\text{TF}(t,d) = \frac{\text{Number of times term } t \text{ appears in } d}{\text{Total number of terms in } d} \tag{4.1}$$

Table 4.3: Term Frequency of Documents

Words/ Documents	**Doc 1**	**Doc 2**	**Doc 3**
creative	0	0	0.33
feel	0.5	0.16	0.33
grab	0	0.16	0
greedy	0	0.16	0
humiliate	0.5	0	0
minute	0	0.16	0
need	0	0	0.33
post	0	0.16	0
wrong	0	0.16	0

Step-2: Find IDF

We need the IDF values because computing just the TF alone is not sufficient to understand the importance of words. We will determine the IDF values (table 4.4) according to the following formula -

$$\text{IDF}(t) = \log(\frac{\text{Total number of documents}}{\text{Number of documents with term } t}) \tag{4.2}$$

Table 4.4: Inverse Document Frequency of Documents

Words/ Documents	**IDF Values**
creative	$\log_e(3/1)$
feel	$\log_e(3/3)$
grab	$\log_e(3/1)$
greedy	$\log_e(3/1)$
humiliate	$\log_e(3/1)$
minute	$\log_e(3/1)$
need	$\log_e(3/1)$
post	$\log_e(3/1)$
wrong	$\log_e(3/1)$

Step-3: Find TF-IDF

We can determine the values (table 4.5) of TF-IDF of every word in the documents by using the formula-

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) * \text{IDF}(t) \tag{4.3}$$

Table 4.5: Term Frequency-Inverse Document Frequency of Documents

	creative	feel	grab	greedy	humiliate	minute	need	post	wrong
Doc 1	0	0	0	0	0.55	0	0	0	0
Doc 2	0	0	0.18	0.18	0	0.18	0	0.18	0.18
Doc 3	0.36	0	0	0	0	0	0.36	0	0

4.1.4 Data Splitting

The dataset contains total 25,000 sentences. Here, training data consists 70% and test data consists 30% of the actual dataset.

4.1.5 Fitting Algorithm

After splitting the dataset, we feed the training data to seven machine learning models. Between Count Occurrence & TF-IDF technique, we found out that the TF-IDF performs better than the Count Occurrence technique. So later, we proceeded with the models using the TF-IDF technique.

4.1.6 Predictions and Results

We used the 30% of the actual dataset for predictions and got the results from the seven machine learning models.

Figure 4.1 shows a chart of actual data, right predictions & wrong predictions of the SVM model.

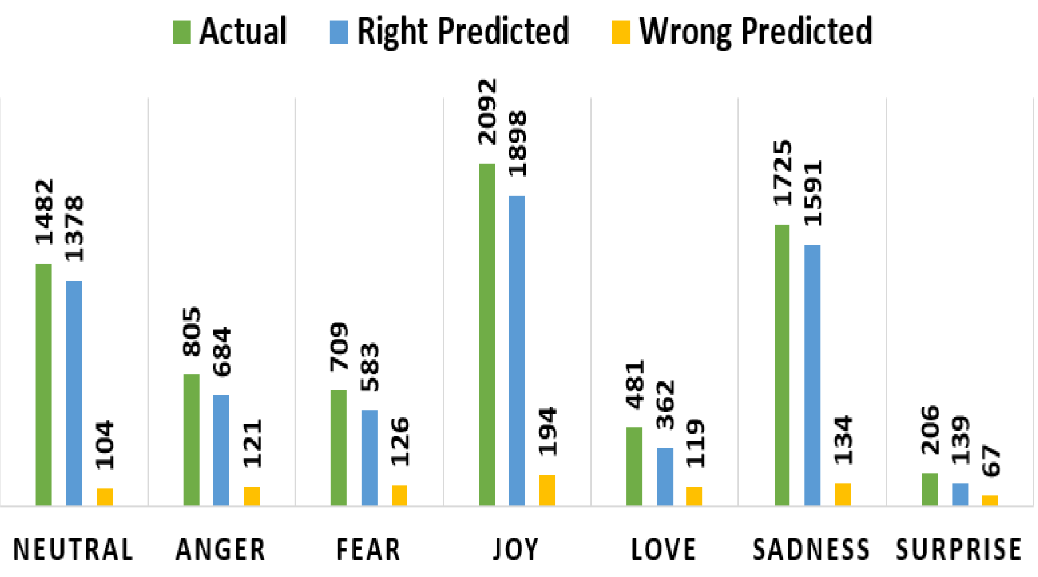


Figure 4.1: Emotion Classification of Support Vector Machine

Figure 4.2 shows a chart of actual data, right predictions & wrong predictions of the Logistic Regression model.

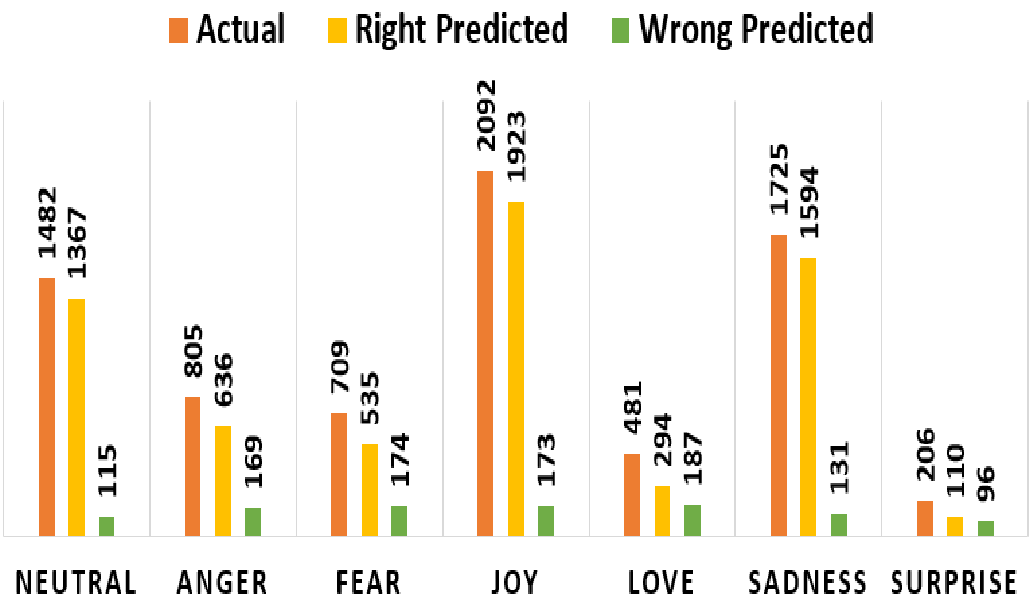


Figure 4.2: Emotion Classification of Logistic Regression

Figure 4.3 shows a chart of actual data, right predictions & wrong predictions of the Random Forest model. We observed that Random Forest gives poor performance among all the implemented models.

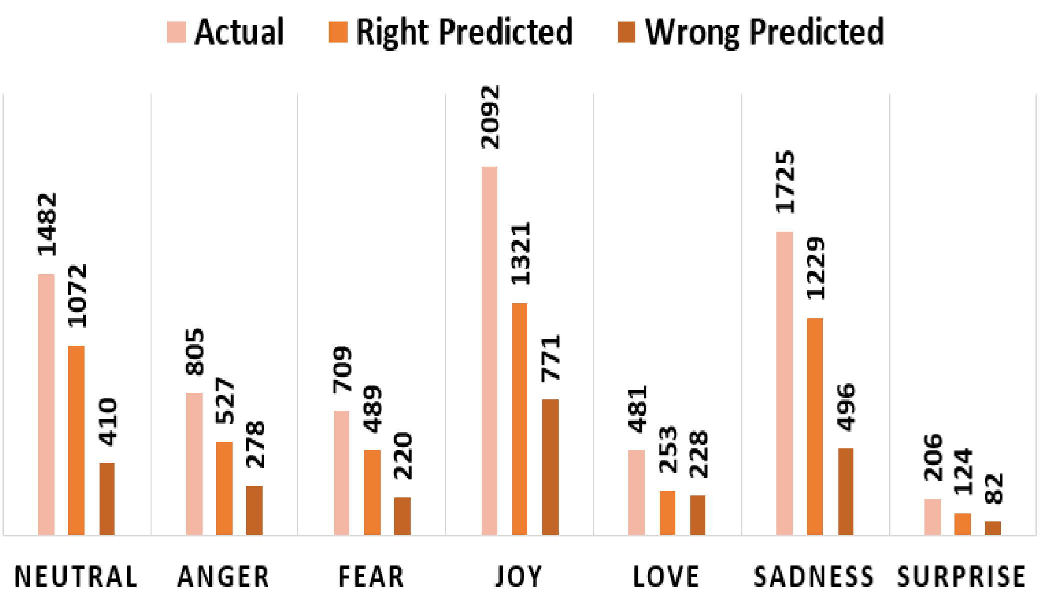


Figure 4.3: Emotion Classification of Random Forest

Figure 4.4 shows a chart of actual data, right predictions & wrong predictions of the Multinomial Naïve Bayes model. After using Multinomial Naïve Bayes, we observed that this model gives a poor accuracy compared to the other models but better than the Random Forest model.

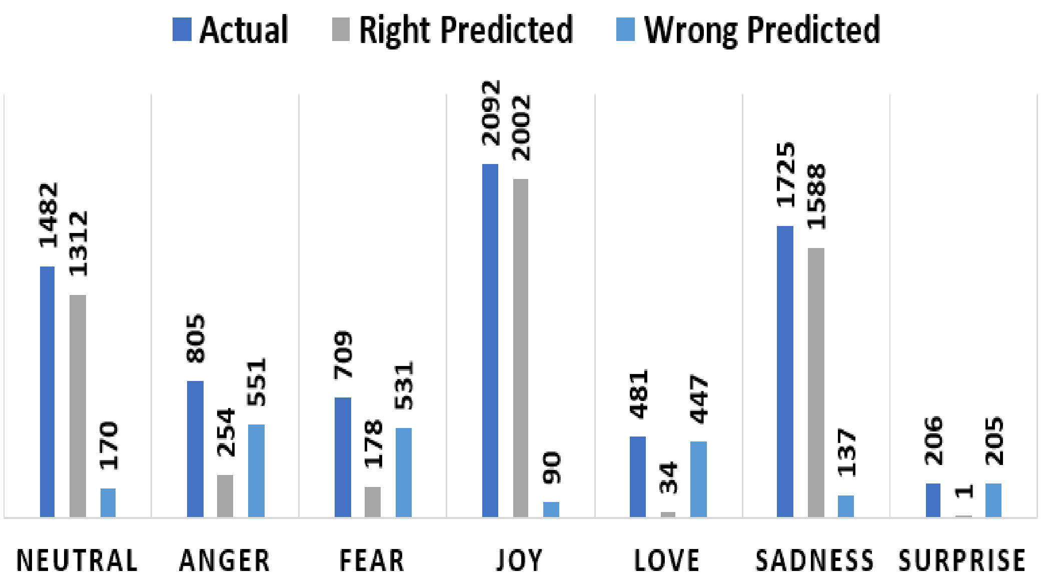


Figure 4.4: Emotion Classification of Multinomial Naïve Bayes

Figure 4.5 shows a chart of actual data, right predictions & wrong predictions of the Random Forest model. We observed that XGBoost Gives the highest result among all the models.

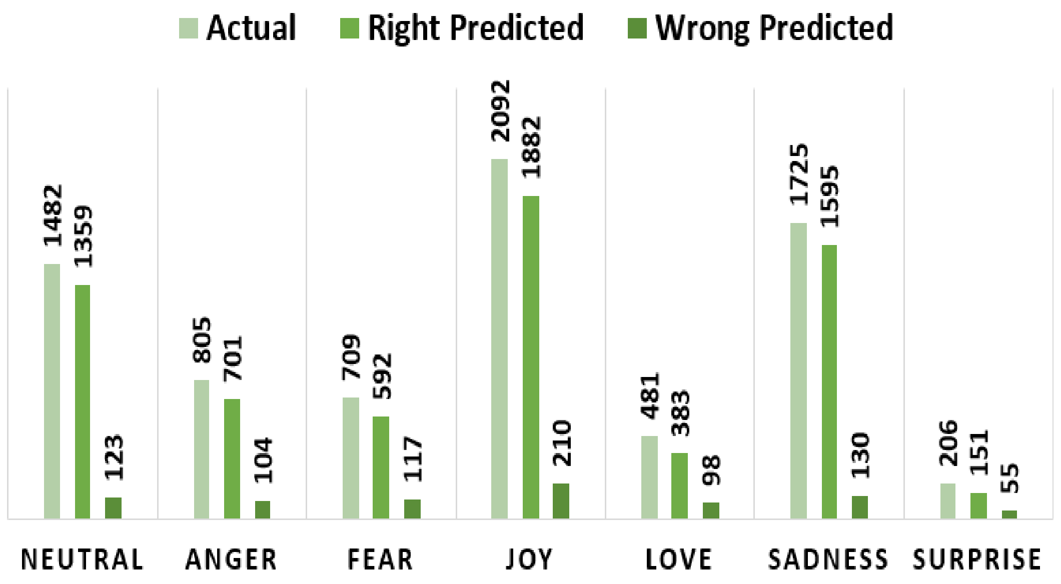


Figure 4.5: Emotion Classification of XGBoost

Figure 4.6 shows a chart of actual data, right predictions & wrong predictions of the Decision Tree model. We observed that the Decision Tree performs well than the Random Forest and the Naive Bayes models.

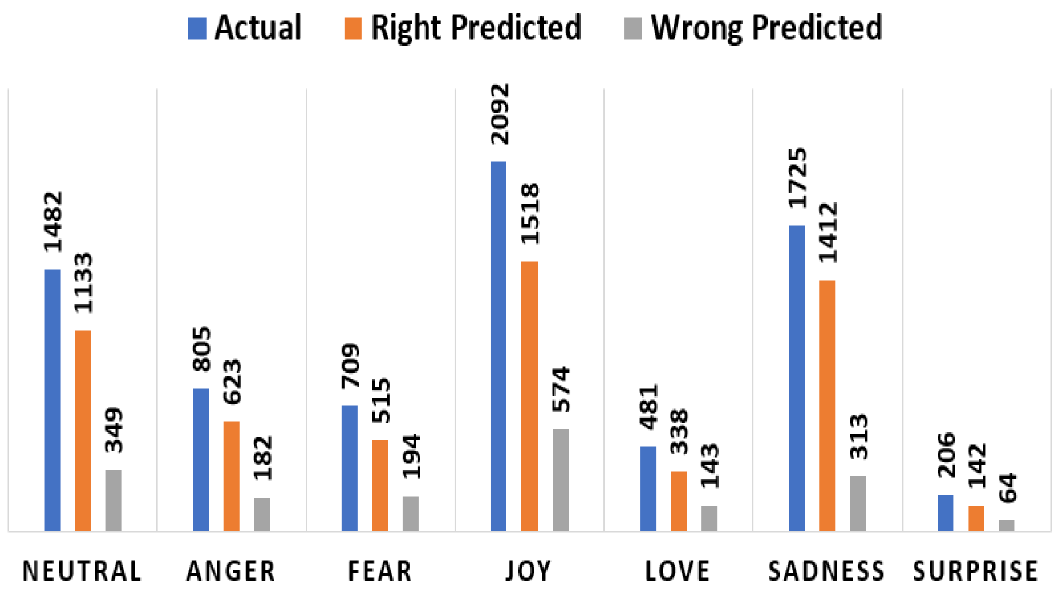


Figure 4.6: Emotion Classification of Decision Tree

Figure 4.7 shows a chart of actual data, right predictions & wrong predictions of the Multilayer Perceptron(MLP) model. We observed that MultiLayer Perceptron(MLP) performs a lot better than all models except the SVM and the XGBoost model.

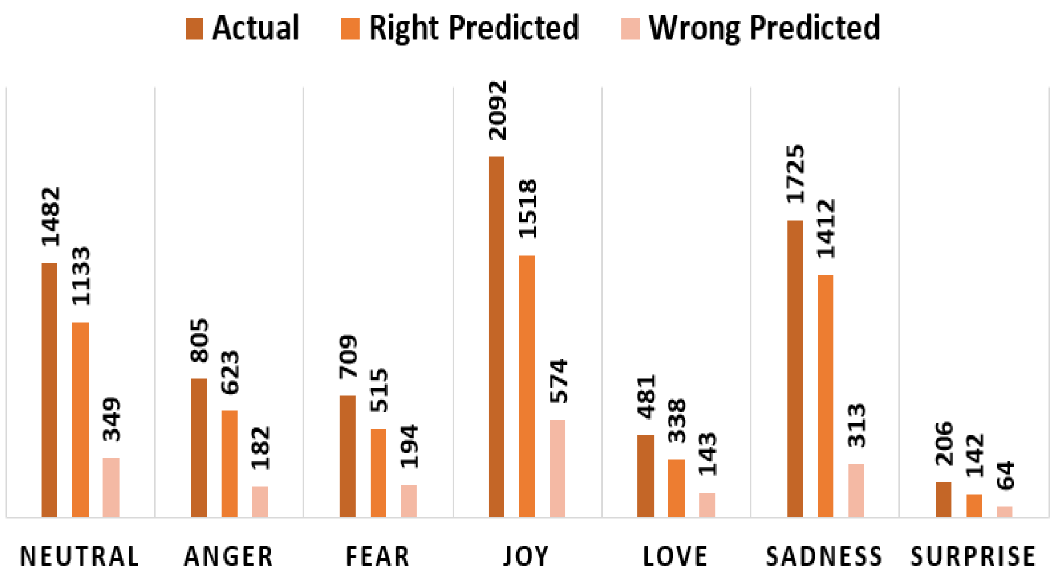


Figure 4.7: Emotion Classification of Multilayer Perceptron (MLP)

4.1.7 Classification Report

It's a performance evaluation metric in machine learning. It is used to show the precision, recall, F1 Score, and accuracy of the trained classification models.

Accuracy

Accuracy is defined as the percentage of correct predictions for the test data. Here Accuracy of seven models is shown. So that we can easily understand which model is performing better. From figure 4.8 we can easily observe the accuracy of all the models.

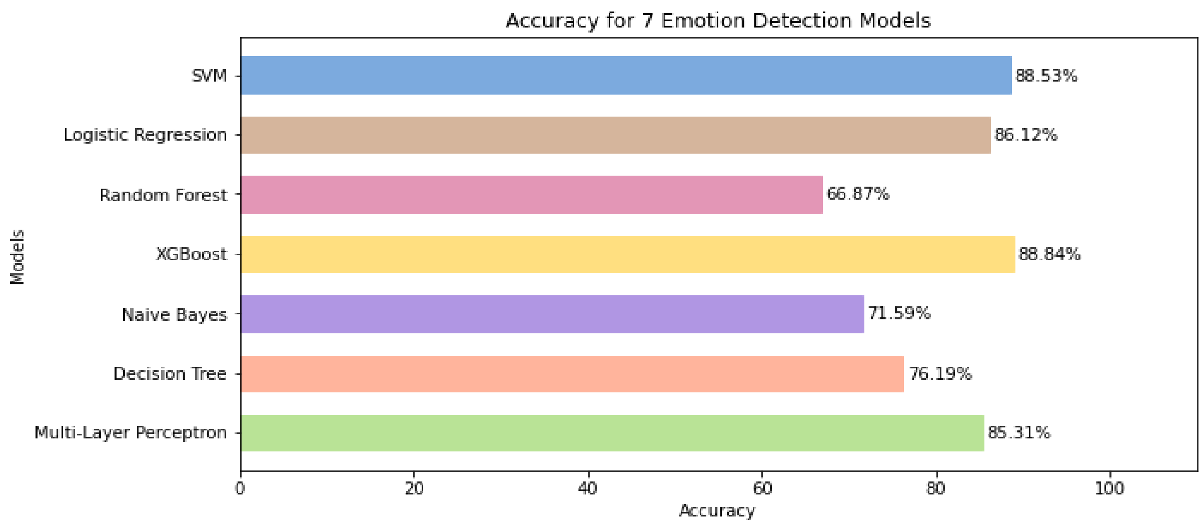


Figure 4.8: Accuracy of All Models

Precision of all Models:

Precision is the ability of a classification model to identify only the relevant data points. Mathematically, precision is the number of true positives divided by the number of true positives plus the number of false positives. Table 4.6 shows the precision of all models.

Table 4.6: Precision of All Models for Every Emotions

	SVM	Logistic Regression	Random Forest	XGBoost	Naive Bayes	Decision Tree	MLP
Neutral	0.895	0.866	0.639	0.904	0.870	0.680	0.855
Anger	0.891	0.892	0.600	0.883	0.901	0.797	0.821
Fear	0.857	0.878	0.681	0.858	0.918	0.734	0.838
Joy	0.884	0.830	0.744	0.897	0.635	0.789	0.848
Love	0.831	0.831	0.560	0.785	0.971	0.736	0.751
Sadness	0.905	0.882	0.692	0.929	0.682	0.831	0.885
Surprise	0.845	0.902	0.559	0.726	1.000	0.638	0.757

Recall of all Models:

Recall is the ability of a model to find all the relevant cases within a data set. Mathematically, we define recall as the number of true positives divided by the number of true positives plus the number of false negatives. Table 4.7 shows the recall of all models.

Table 4.7: Recall of All Models for Every Emotions

	SVM	Logistic Regression	Random Forest	XGBoost	Naive Bayes	Decision Tree	MLP
Neutral	0.930	0.922	0.723	0.917	0.885	0.773	0.904
Anger	0.851	0.790	0.655	0.871	0.316	0.762	0.825
Fear	0.818	0.755	0.690	0.835	0.251	0.729	0.738
Joy	0.912	0.919	0.632	0.900	0.957	0.731	0.871
Love	0.748	0.611	0.526	0.796	0.071	0.724	0.682
Sadness	0.924	0.924	0.713	0.925	0.921	0.824	0.889
Surprise	0.660	0.534	0.602	0.733	0.005	0.685	0.665

4.1.8 Confusion Matrix

Confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm. Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class or vice versa.

Figure 4.9 and 4.10 shows the confusion Matrix of SVM and XGBoost models respectively:

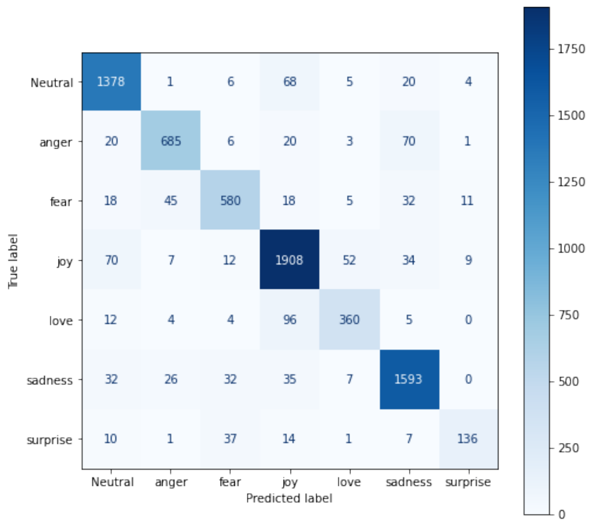


Figure 4.9: Confusion Matrix of SVM

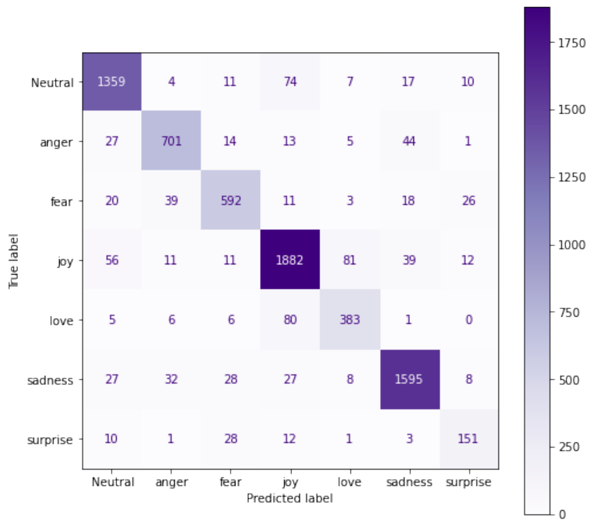


Figure 4.10: Confusion Matrix of XGBoost

Figure 4.11 and 4.12 shows the confusion Matrix of Random Forest and Logistic Regression models respectively:

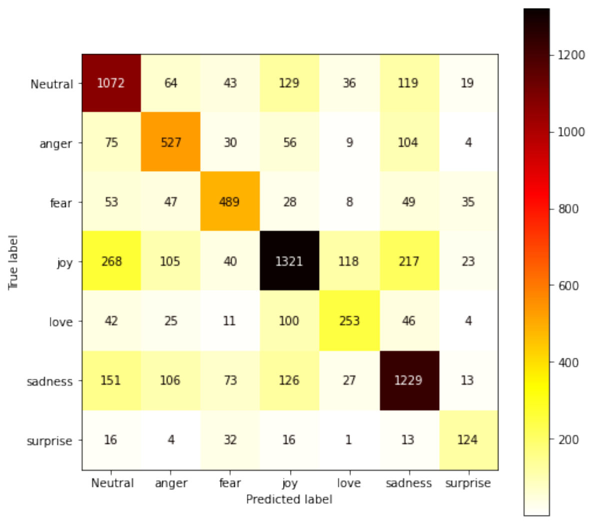


Figure 4.11: Confusion Matrix of Random Forest

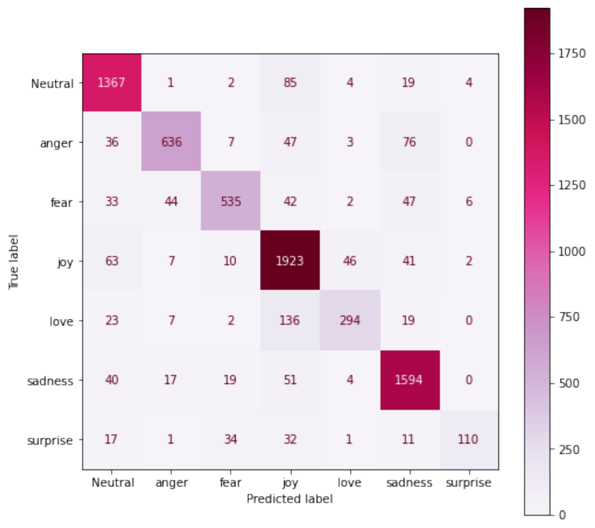


Figure 4.12: Confusion Matrix of Logistic Regression

Figure 4.13 and 4.14 shows the confusion Matrix of Multinomial Naive Bayes and Decision Tree models respectively:

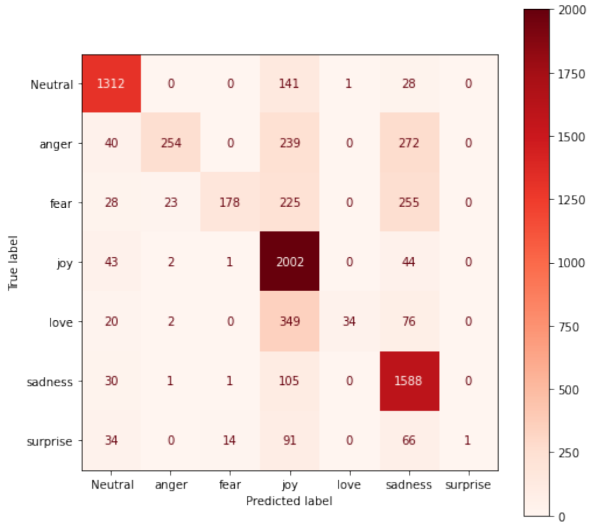


Figure 4.13: Confusion Matrix of Multinomial Naive Bayes

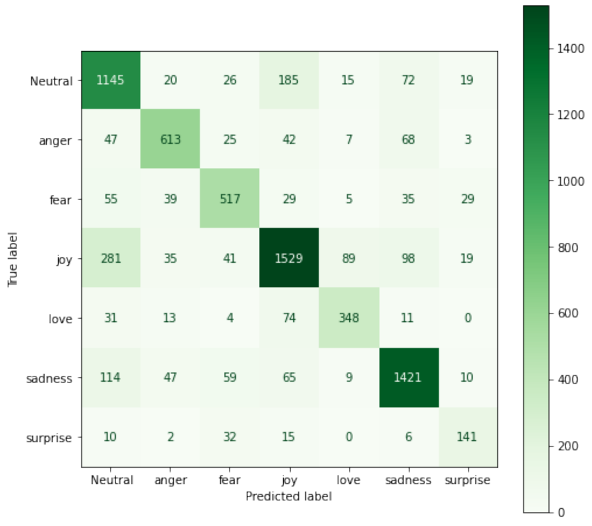


Figure 4.14: Confusion Matrix of Decision Tree

Figure 4.15 shows the confusion Matrix of Multi Layer Perceptron (MLP) model:

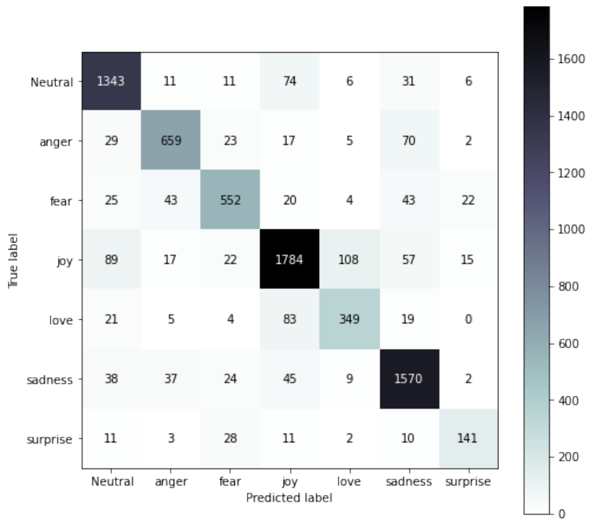


Figure 4.15: Multi Layer Perceptron (MLP)

4.1.9 Summary of Models

Table 4.8 shows the summary of the emotion detection models such as Accuracy, Precision, Recall, and F1-Score.

We observed that XGBoost Classifier gives the most accuracy, recall, and f1-score which is **88.84%** and **0.854** and **0.854** respectively. SVM gives the most precision which is **0.870**.

Table 4.8: Summary of the Emotion Detection Models

Models	Accuracy	Precision	Recall	F1-Score
Support Vector Machine (SVM)	88.47%	**0.870**	0.837	0.852
Logistic Regression	86.12%	0.869	0.779	0.813
Random Forest	66.87%	0.639	0.649	0.642
XGBoost Classifier	**88.84%**	0.854	**0.854**	**0.854**
Naive Bayes	71.59%	0.853	0.486	0.490
Decision Tree	75.75%	0.743	0.743	0.742
Multilayer Perceptron (MLP)	85.31%	0.824	0.811	0.817

4.2 Conversational Chatbot

A chatbot converses with users. Firstly, the chatbot needs to get the chat intent from the user's text. Based on that intent, the chatbot will give a reply as well. We used ensemble learning for getting the intent and cosine distance for getting the reply. Before the intent selection, first, we've to clean the data [Reference: 6]. Then we have to do the feature extraction [Reference: 3.1.2]. After that, we will fit the processed data in our models. An Ensemble Learning algorithm is used to get the best intent among all of the intent. In our chatbot model, seven classifiers are used. Then ensemble learning algorithm is applied to have the best intent.

4.2.1 Text Cleaning

For cleaning text, we approached these methods - replace Emoji with text, remove punctuation, lemmatizing. The description of these methods was previously stated in the implemented model chapter [Reference: Data Cleaning 6].

We have also approached the contraction replacement method for cleaning our text.

Contraction Replacement

Contraction is the shortened form of words. We are replacing the contractions with their full forms.

For Example: we've = we have, I've been = I have been

4.2.2 Data Splitting

The dataset contains total 2064 sentences. Here, training data consists 75% and test data consists 25% of the actual dataset.

4.2.3 Best Intent Selection:

Here, our chatbot model has a total of twenty-seven intents. Our models will predict chat intents and then we will choose the best chat intent among them through votings. Our models are weighted according to the accuracy of the models for voting purposes. We normalized the weights and used them for the voting. So, the best intent will be selected using the weighted votes of the models. Then the chatbot will retrieve the appropriate reply based on the best intent.

For example:
Input text: I like playing football.
Intent using SVM: 'Hobby'
Intent using Logistic Regression: 'Hobby'
Intent using Decision Tree: 'Hobby'
Intent using Naive Bayes: 'Sad'
Intent using XGBoost: 'Surprise_Amazed'
Intent using Random Forest: 'Hobby'
Intent using Multi Layer Perceptron: 'Hobby'

We already know ensemble learning works on the principle that a weak learner predicts poorly when alone. But when combined with other weak learners, they create a strong learner and that strong learner performs a lot better than before.

Here, from our example, ensemble learning chooses the best intent.

('Hobby', 0.84)
('Sad', 0.14)
('Surprise_Amazed', 0.16)
....
....
('Greeting', 0.0)
('Insult', 0.0)
('Info', 0.01)
....
....
('Time', 0.0)

From all **twenty-seven** intents along with its probability we can conclude that, **'Hobby'** is the best intent with the highest probability.

Best Intent: 'Hobby'

Score: 0.84

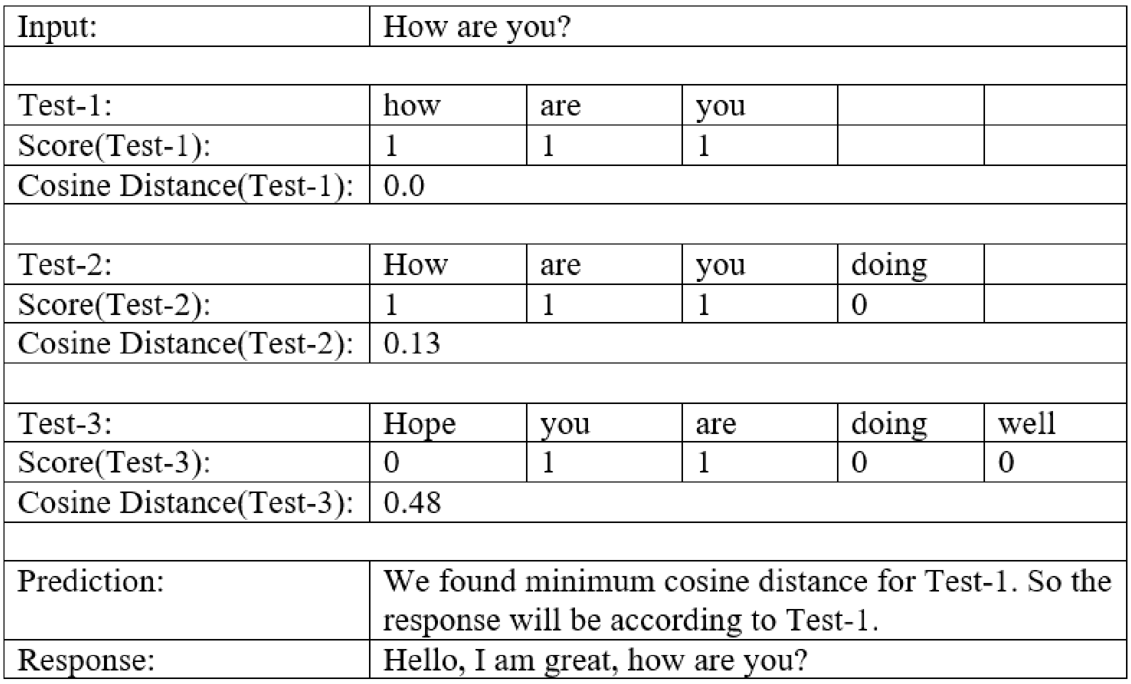
After getting the best intent we've to find the best reply. Cosine distance is used for getting the reply.

4.2.4 Cosine Distance :

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction or not. It is often used to measure document similarity in text analysis. After getting best intent in order to give the best reply cosine distance is used to get the most suitable reply for that particular intent. In case multiple same cosine similarity random function is used to give any one of the replies with same cosine similarity value.

For example:
Here, the input is "How are you?". We are running three test cases here.
For Test-1, we found the cosine distance 0.0
For Test-2, we found the cosine distance 0.13
For Test-3, we found the cosine distance 0.48
As we know that the less the value of cosine distance, the more suitable it is to get the response according to that test case. So, here we will get the response according to Test-1. Table 4.9 shows the detailed calculation.

Table 4.9: Calculation of Minimum Cosine Distance



Input:	How are you?				
Test-1:	how	are	you		
Score(Test-1):	1	1	1		
Cosine Distance(Test-1):	0.0				
Test-2:	How	are	you	doing	
Score(Test-2):	1	1	1	0	
Cosine Distance(Test-2):	0.13				
Test-3:	Hope	you	are	doing	well
Score(Test-3):	0	1	1	0	0
Cosine Distance(Test-3):	0.48				
Prediction:	We found minimum cosine distance for Test-1. So the response will be according to Test-1.				
Response:	Hello, I am great, how are you?				

4.2.5 Prediction and Results

After training our model we got some result according to our dataset. In the following section we can visualize the results from confusion matrix and accuracy chart.

Accuracy

The accuracy of seven models for our chatbot reply is shown in figure 4.16 and we can easily understand which model is performing better from observing the accuracy.

Here, the SVM model gives the best accuracy that is **71.9%** among all the other models.

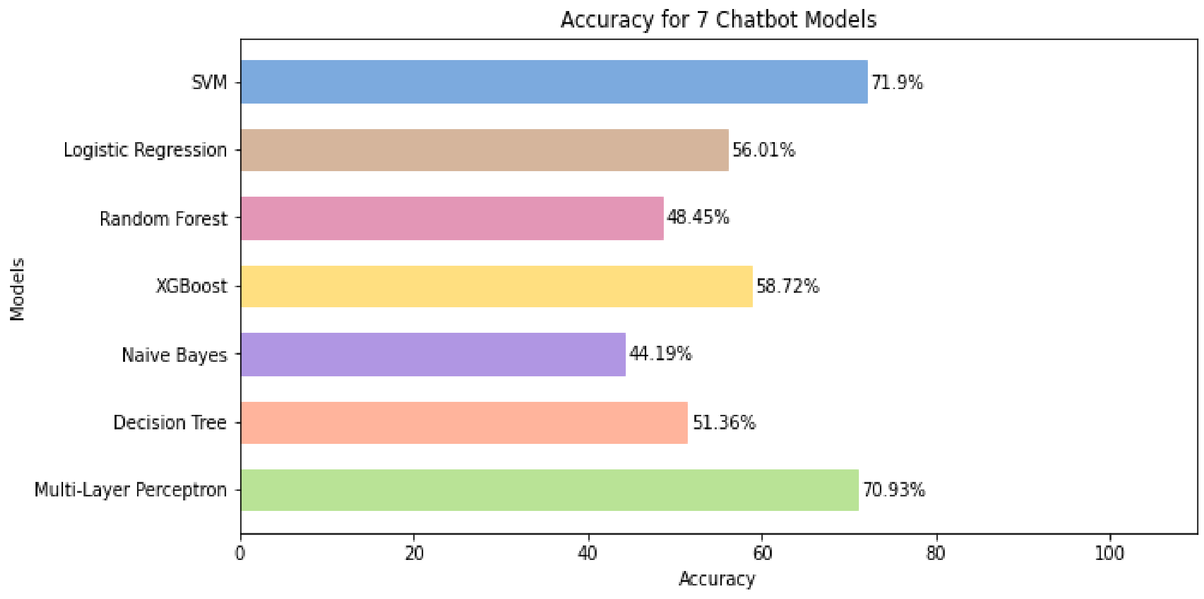


Figure 4.16: Accuracy of all models

Summary of Models

Table 4.10 shows the summary of the chatbot models such as Accuracy, Precision, Recall, and F1-Score.

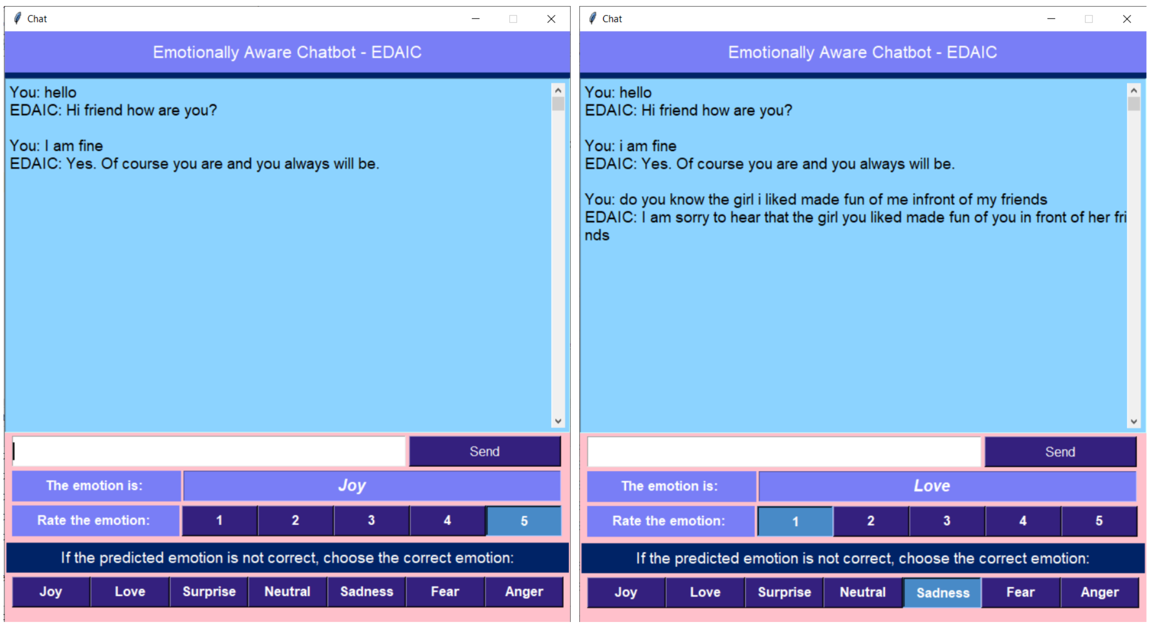
We observed that SVM gives the most accuracy, recall, and f1-score which is **71.71%** and **0.633** and **0.615** respectively. Multilayer Perceptron gives the most precision which is **0.658**.

Table 4.10: Summary of the Chatbot Models

Models	Accuracy	Precision	Recall	F1-Score
Support Vector Machine (SVM)	**71.71%**	0.652	**0.633**	**0.615**
Logistic Regression	56.01%	0.437	0.289	0.314
Random Forest	48.45%	0.442	0.408	0.384
XGBoost Classifier	58.72%	0.485	0.499	0.459
Naive Bayes	44.19%	0.107	0.093	0.076
Decision Tree	53.68%	0.542	0.476	0.485
Multilayer Perceptron (MLP)	70.93%	**0.658**	0.619	0.612

4.3 Emotion Feedback

We implemented a simple application for our chatbot. So that we can detect the emotions of chats while doing chat with EDAIC. Here, we will see how we're going to take emotional feedback from the users.


Chat
Emotionally Aware Chatbot - EDAIC
Send
The emotion is:
Joy
Rate the emotion:
1
2
3
4
5
If the predicted emotion is not correct, choose the correct emotion:
Joy
Love
Surprise
Neutral
Sadness
Fear
Anger
Chat
Emotionally Aware Chatbot - EDAIC
Send
The emotion is:
Love
Rate the emotion:
1
2
3
4
5
If the predicted emotion is not correct, choose the correct emotion:
Joy
Love
Surprise
Neutral
Sadness
Fear
Anger

You: hello
EDAIC: Hi friend how are you?

You: I am fine
EDAIC: Yes. Of course you are and you always will be.

You: hello
EDAIC: Hi friend how are you?

You: i am fine
EDAIC: Yes. Of course you are and you always will be.

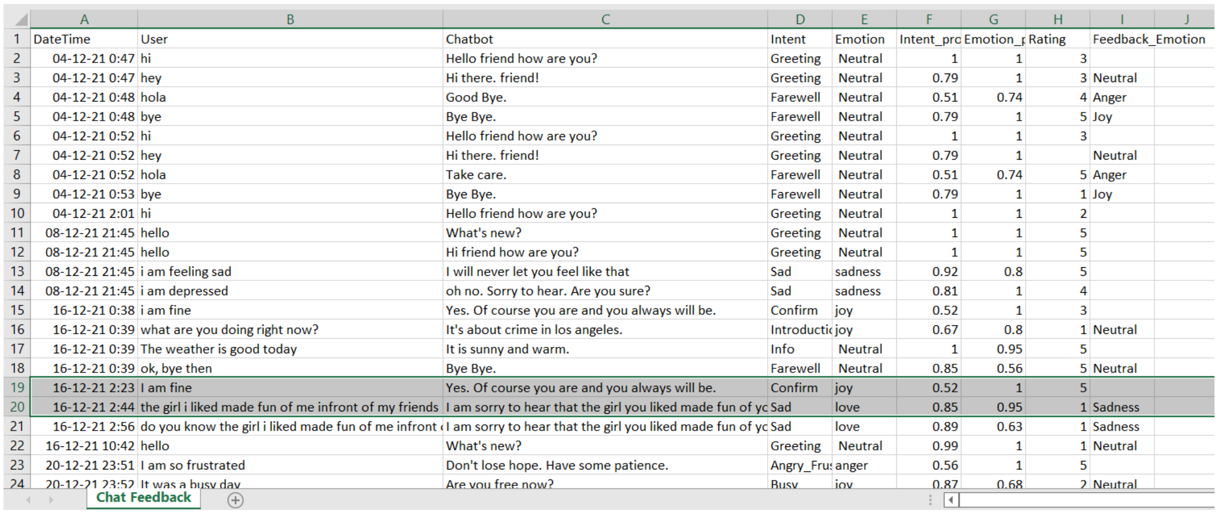
You: do you know the girl i liked made fun of me infront of my friends
EDAIC: I am sorry to hear that the girl you liked made fun of you in front of her fri nds

Figure 4.17: Emotion feedback application

On the left side, the user gave a chat. The predicted emotion shows that it is **Joy**. Now the user can rate the emotion. Here, the user rated it 5 that is very good.

On the right side, the user gave another chat. The predicted emotion shows that it is **Love**. But the predicted emotion was not correct according to the user. The user rated it 1 that is very bad. So, if the predicted emotion was not right and the rating was not good, the user can also choose the correct emotion and give feedback. Hence, the user selected the right emotion that is **Sadness**.

This feedback will be used in the future for the self-learning purpose of our chatbot. Here, the figure4.18 shows a sample of how our feedback data is being stored.



	A	B	C	D	E	F	G	H	I	J
1	DateTime	User	Chatbot	Intent	Emotion	Intent_pro	Emotion_p	Rating	Feedback_Emotion	
2	04-12-21 0:47	hi	Hello friend how are you?	Greeting	Neutral	1	1	3		
3	04-12-21 0:47	hey	Hi there. friend!	Greeting	Neutral	0.79	1	3	Neutral	
4	04-12-21 0:48	hola	Good Bye.	Farewell	Neutral	0.51	0.74	4	Anger	
5	04-12-21 0:48	bye	Bye Bye.	Farewell	Neutral	0.79	1	5	Joy	
6	04-12-21 0:52	hi	Hello friend how are you?	Greeting	Neutral	1	1	3		
7	04-12-21 0:52	hey	Hi there. friend!	Greeting	Neutral	0.79	1		Neutral	
8	04-12-21 0:52	hola	Take care.	Farewell	Neutral	0.51	0.74	5	Anger	
9	04-12-21 0:53	bye	Bye Bye.	Farewell	Neutral	0.79	1	1	Joy	
10	04-12-21 2:01	hi	Hello friend how are you?	Greeting	Neutral	1	1	2		
11	08-12-21 21:45	hello	What's new?	Greeting	Neutral	1	1	5		
12	08-12-21 21:45	hello	Hi friend how are you?	Greeting	Neutral	1	1	5		
13	08-12-21 21:45	i am feeling sad	I will never let you feel like that	Sad	sadness	0.92	0.8	5		
14	08-12-21 21:45	i am depressed	oh no. Sorry to hear. Are you sure?	Sad	sadness	0.81	1	4		
15	16-12-21 0:38	i am fine	Yes. Of course you are and you always will be.	Confirm	joy	0.52	1	3		
16	16-12-21 0:39	what are you doing right now?	It's about crime in los angeles.	Introducti	joy	0.67	0.8	1	Neutral	
17	16-12-21 0:39	The weather is good today	It is sunny and warm.	Info	Neutral	1	0.95	5		
18	16-12-21 0:39	ok, bye then	Bye Bye.	Farewell	Neutral	0.85	0.56	5	Neutral	
19	16-12-21 2:23	I am fine	Yes. Of course you are and you always will be.	Confirm	joy	0.52	1	5		
20	16-12-21 2:44	the girl i liked made fun of me infront of my friends	I am sorry to hear that the girl you liked made fun of yc	Sad	love	0.85	0.95	1	Sadness	
21	16-12-21 2:56	do you know the girl i liked made fun of me infront	I am sorry to hear that the girl you liked made fun of yc	Sad	love	0.89	0.63	1	Sadness	
22	16-12-21 10:42	hello	What's new?	Greeting	Neutral	0.99	1	1	Neutral	
23	20-12-21 23:51	I am so frustrated	Don't lose hope. Have some patience.	Angry_Fru	anger	0.56	1	5		
24	20-12-21 23:52	It was a busy day	Are you free now?	Busy	joy	0.87	0.68	2	Neutral	

Figure 4.18: A sample of the emotion feed-backs

Here, we can see that the selected feedbacks in the CSV are stored according to the user's feedback that is previously mentioned above. Along with the emotion rating and correct emotion, we are storing other data also such as the user chat and EDAIC's reply, and so on.

Chapter 5

Conclusion

In our model, we have developed an application that has created an atmosphere where our chatbot (EDAIC) and a human can make a conversation. We have applied seven machine learning models to our chatbot. They are SVM, Logistic Regression, Random Forest, XG-Boost, Naive Bayes, Decision Tree, and Multilayer Perceptron. We have generated the reply based on the best intent and cosine distance. At first, We have used ensemble learning to choose the best intent. After finding the best intent, we have used cosine distance to receive relatively suitable responses. Among all the models, SVM has given the most accuracy for chats that is **71.9%**.

Detecting emotions from text is relatively complex. Because, in some cases, it is not easy to recognize appropriate emotions from the text. Again, we have used seven machine learning models for our emotion detector, and XGBoost gave the most accuracy that is **88.84%**.

We have implemented an application combining two models - chatbot and emotion detector. Here, users can chat with EDAIC, and they can also see the emotions of their texts. There is a feature for rating emotions. So, one can easily give a rating to the emotion that has been predicted by the chatbot. Also, a user can choose the correct emotion if the predicted emotion seems incorrect to the user and gives a bad rating. If the predicted emotion is rated by a user, this feedback will be stored in a file. So, after collecting a certain amount of feedback we can re-train our model for better performance.

5.1 Challenges

At the time of our research, we have faced some challenges. We have tried our best to deal with these challenges.

- Data set according to our research was not appropriate enough.

- At the time of implementation, we found that it was tough to get the correct words from the text after lemmatizing.
- Improving accuracy from training data was hard too.
- Sometimes, we got two different emotions from one text which made it difficult to predict the real emotion.
- Deployment of the chatbot application as PyInstaller doesn't support sklearn libraries.

5.2 Future Work

In the future, we will work to overcome the constraints that now exist in our work. We'll concentrate on:

- Improvement of the current accuracy of these implemented models.
- As we will collect the user's feed back, more relevant responses according to user input will be considered after analyzing the feedback.
- Converting voice messages to text and detecting emotions from it will be implemented.
- We will implement Self learning chatbot according to updated data and user feedback.

References

[1] "Count vectorizer description from scikit-learn." https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html. Accessed: 2021-12-23.

[2] "Tkinter description." https://python-textbok.readthedocs.io/en/1.0/Introduction_to_GUI_Programming.html. Accessed: 2021-12-23.

[3] "Scikit-learn description." https://www.tutorialspoint.com/scikit_learn/scikit_learn_introduction.htm. Accessed: 2021-12-23.

[4] "Ensemble learning." http://www.scholarpedia.org/article/Ensemble_learning. Accessed: 2021-12-23.

[5] "Support vector machine." https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/. Accessed: 2021-12-23.

[6] "Support vector machine – simplified." https://www.analyticsvidhya.com/blog/2014/10/support-vector-machine-simplified/?utm_source=blog&utm_medium=understandingsupportvectormachinearticle. Accessed: 2021-12-23.

[7] "Logistic regression description." https://en.wikipedia.org/wiki/Logistic_regression. Accessed: 2021-12-23.

[8] "Random forest classifier description." https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/. Accessed: 2021-12-23.

[9] "Xgboost classifier description." https://analyticsindiamag.com/xgboost-internal-working-to-make-decision-trees-and-deduce-predictions/. Accessed: 2021-12-23.

[10] A. M. Kibriya, E. Frank, B. Pfahringer, and G. Holmes, “Multinomial naive bayes for text categorization revisited,” in *Australasian Joint Conference on Artificial Intelligence*, pp. 488–499, Springer, 2004.

[11] “Decision tree classifier.” `https://www.analyticssteps.com/blogs/introduction-decision-tree-algorithm-machine-learning`. Accessed: 2021-12-23.

[12] V. Ramalingam, A. Pandian, A. Jaiswal, and N. Bhatia, “Emotion detection from text,” in *Journal of Physics: Conference Series*, vol. 1000, p. 012027, IOP Publishing, 2018.

[13] “NRC-emotion-lexicon.” `https://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm`. Accessed: 2021-12-23.

[14] “Emotions in text.” `https://www.kaggle.com/ishantjuyal/emotions-in-text`. Accessed: 2021-12-23.

[15] “Tf-idf technique.” https://medium.com/analytics-vidhya/tf-idf-term-frequency-technique-easiest-explanation-for-text-classification-in-nlp-with-code-8ca3912e58c3. Accessed: 2021-12-23.

[16] “chatbots-intent-recognition-dataset.” `https://www.kaggle.com/elvinagammed/chatbots-intent-recognition-dataset`. Accessed: 2021-12-23.

[17] “context-free-dataset.” `https://github.com/lukalabs/replika-research/tree/master/context-free-dataset`. Accessed: 2021-12-23.

Appendix A

Codes

A.1 Chatbot

A.1.1 Necessary Libraries

```python
import pandas as pd
import numpy as np

# Visualization libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Text Libraries
import nltk
import string
import re

# Feature Extraction Libraries
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import train_test_split

# Classifier Model libraries
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
import xgboost as xgb
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn import tree
from sklearn.neural_network import MLPClassifier
```

```python

# Performance Matrix libraries
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import ConfusionMatrixDisplay

# other
import pickle
import os
import warnings
warnings.filterwarnings("ignore")
```

A.1.2 Dataset

```python
df = pd.read_csv('Chatbot Dataset_v12.11.csv',encoding='ISO-8859-1')
df = df.dropna(axis=0)

print('Dataset size:',df.shape)
print('Columns are:',df.columns)

# visualize the Intent
sns.countplot(x = 'Intent', data = df)

# printing the Intents
print('Number of Intents:', len(df['Intent'].unique()))
print(df['Intent'].unique())

# printing the distribution of Intent
import collections
intent_counter = collections.Counter(df['Intent'])
print(intent_counter)
```

A.1.3 Text Preprocessing

- Replace emojis & contractions
- Remove Punctuation
- Tokenization
- Lemmatization

```python
# Removing Punctuation and replacing emojis and replacing contraction
emojis = pd.read_csv('emojis.txt',sep=',',header=None)
emojis_dict = {i:j for i,j in zip(emojis[0],emojis[1])}
pattern = '|'.join(sorted(re.escape(k) for k in emojis_dict))

def replace_emojis(text):
    text = re.sub(pattern,lambda m: emojis_dict.get(m.group(0)), text, flags=re
    .IGNORECASE)
    return text

def remove_punct(text):
    text = replace_emojis(text)
    text  = "".join([char for char in text if char not in string.punctuation])
    text = re.sub('[0-9]+', '', text)
    return text

file = open('contraction.txt',"r")
contraction_dict = {}
for line in file:
    row = []
    x1,x2 = line.split(':')
    if (x2[-1] == '\n'):
        x2 = x2[:-1]
    contraction_dict[x1] = x2
file.close()
def _get_contractions(contraction_dict):
    contraction_re = re.compile('(%s)' % '|'.join(contraction_dict.keys()))
    return contraction_dict, contraction_re
contractions, contractions_re = _get_contractions(contraction_dict)



def replace_contractions(text):
    def replace(match):
        return contractions[match.group(0)]
    return contractions_re.sub(replace, text)

# Tokenization
def tokenization(text):
    text = text.lower()
    text = re.split('\W+', text)
    return text

# Lemmatization
nltk.download('wordnet')
wn = nltk.WordNetLemmatizer()

```

```python
def lemmatizer(text):
    text = [wn.lemmatize(word) for word in text]
    return text

def clean_text(text):
    text = replace_contractions(text)
    text = remove_punct(text)
    text = tokenization(text)
    text = lemmatizer(text)
    return text
```

A.1.4 Feature Extraction

```python
# Train Test Split — 75%–25%
X_train, X_test, y_train, y_test = train_test_split(df['User'], df['Intent'],
    test_size=0.25, random_state = 32)

countVectorizer1 = CountVectorizer(analyzer=clean_text)
countVector1 = countVectorizer1.fit_transform(X_train)

countVector2 = countVectorizer1.transform(X_test)

tfidf_transformer_xtrain = TfidfTransformer()
x_train = tfidf_transformer_xtrain.fit_transform(countVector1)

tfidf_transformer_xtest = TfidfTransformer()
x_test = tfidf_transformer_xtest.fit_transform(countVector2)
```

A.1.5 Models

1. Support Vector Machine (SVM)
2. Logistic Regression
3. Random Forest Classifier
4. XGBoost Classifier
5. Multinomial Naive Bayes
6. Decision Tree
7. Multilayer Perceptron (MLP)

Support Vector Machine (SVM)

```python
# Support Vector Machine (SVM)

svm = SGDClassifier()
svm.fit(x_train, y_train)

y_pred = svm.predict(x_test)

svm_acc = accuracy_score(y_pred, y_test)
svm_prec = precision_score(y_test, y_pred, average='macro')
svm_recal = recall_score(y_test, y_pred, average='macro')
svm_cm = confusion_matrix(y_test,y_pred)
svm_f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', '{0:.3f}'.format(svm_acc*100))
print('Precision:', '{0:.3f}'.format(svm_prec*100))
print('Recall:', '{0:.3f}'.format(svm_recal*100))
print('F1-score:', '{0:.3f}'.format(svm_f1*100))
print(classification_report(y_test,y_pred))

# Confusion Matrix
cm_display_svm = ConfusionMatrixDisplay(svm_cm, display_labels=svm.classes_)
fig, ax = plt.subplots(figsize=(10,10)) # adjust the size
ax.set_xticklabels(svm.classes_, rotation='vertical', fontsize=10)
cm_display_svm.plot(ax=ax,cmap='Blues')
```

Logistic Regression

```python
# Logistic Regression

logisticRegr = LogisticRegression()

logisticRegr.fit(x_train, y_train)

y_pred = logisticRegr.predict(x_test)

lr_acc = accuracy_score(y_pred, y_test)
lr_prec = precision_score(y_test, y_pred, average='macro')
lr_recal = recall_score(y_test, y_pred, average='macro')
lr_cm = confusion_matrix(y_test,y_pred)
lr_f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', '{0:.3f}'.format(lr_acc*100))
print('Precision:', '{0:.3f}'.format(lr_prec*100))
print('Recall:', '{0:.3f}'.format(lr_recal*100))
print('F1-score:', '{0:.3f}'.format(lr_f1*100))
print(classification_report(y_test,y_pred))

# Confusion Matrix
cm_display_lr = ConfusionMatrixDisplay(lr_cm, display_labels=logisticRegr.
    classes_)
fig, ax = plt.subplots(figsize=(10,10)) # adjust the size
cm_display_lr.plot(ax=ax,cmap='PuRd')
```

Random Forest Classifier

```
# Random Forest Classifier

rfc = RandomForestClassifier(n_estimators=1, random_state=0)

rfc.fit(x_train, y_train)

y_pred = rfc.predict(x_test)

rfc_acc = accuracy_score(y_pred, y_test)
rfc_prec = precision_score(y_test, y_pred, average='macro')
rfc_recal = recall_score(y_test, y_pred, average='macro')
rfc_cm = confusion_matrix(y_test,y_pred)
rfc_f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', '{0:.3f}'.format(rfc_acc*100))
print('Precision:', '{0:.3f}'.format(rfc_prec*100))
print('Recall:', '{0:.3f}'.format(rfc_recal*100))
print('F1-score:', '{0:.3f}'.format(rfc_f1*100))
print(classification_report(y_test,y_pred))

# Confusion Matrix
cm_display_rfc = ConfusionMatrixDisplay(rfc_cm, display_labels=rfc.classes_)
fig, ax = plt.subplots(figsize=(10,10)) # adjust the size
cm_display_rfc.plot(ax=ax,cmap='hot_r')
```

XGBoost Classifier

```
# XGBoost Classifier

xgbc = XGBClassifier(max_depth=16, n_estimators=1000,nthread = 6)
xgbc.fit(x_train,y_train)
y_pred = xgbc.predict(x_test)

xgbc_acc = accuracy_score(y_pred, y_test)
xgbc_prec = precision_score(y_test, y_pred, average='macro')
xgbc_recal = recall_score(y_test, y_pred, average='macro')
xgbc_cm = confusion_matrix(y_test,y_pred)
```

```
xgbc_f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', '{0:.3f}'.format(xgbc_acc*100))
print('Precision:', '{0:.3f}'.format(xgbc_prec*100))
print('Recall:', '{0:.3f}'.format(xgbc_recal*100))
print('F1-score:', '{0:.3f}'.format(xgbc_f1*100))
print(classification_report(y_test,y_pred))

# Confusion Matrix
cm_display_xgbc = ConfusionMatrixDisplay(xgbc_cm, display_labels=xgbc.classes_)
_, ax = plt.subplots(figsize=(10,10)) # adjust the size
cm_display_xgbc.plot(ax=ax,cmap='Purples')
```

Multinomial Naive Bayes

```
# Multinomial Naive Bayes

mnb = MultinomialNB()
mnb.fit(x_train, y_train)

y_pred = mnb.predict(x_test)

mnb_acc = accuracy_score(y_pred, y_test)
mnb_prec = precision_score(y_test, y_pred, average='macro')
mnb_recal = recall_score(y_test, y_pred, average='macro')
mnb_cm = confusion_matrix(y_test,y_pred)
mnb_f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', '{0:.3f}'.format(mnb_acc*100))
print('Precision:', '{0:.3f}'.format(mnb_prec*100))
print('Recall:', '{0:.3f}'.format(mnb_recal*100))
print('F1-score:', '{0:.3f}'.format(mnb_f1*100))
print(classification_report(y_test,y_pred))

# Confusion Matrix
cm_display_mnb = ConfusionMatrixDisplay(mnb_cm, display_labels=mnb.classes_)
fig, ax = plt.subplots(figsize=(10,10)) # adjust the size
cm_display_mnb.plot(ax=ax,cmap='Reds')
```

Decision Tree

```
# Decision Tree

dt = tree.DecisionTreeClassifier()
dt.fit(x_train, y_train)
y_pred = dt.predict(x_test)

```

```
dt_acc = accuracy_score(y_pred, y_test)
dt_prec = precision_score(y_test, y_pred, average='macro')
dt_recal = recall_score(y_test, y_pred, average='macro')
dt_cm = confusion_matrix(y_test,y_pred)
dt_f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', '{0:.3f}'.format(dt_acc*100))
print('Precision:', '{0:.3f}'.format(dt_prec*100))
print('Recall:', '{0:.3f}'.format(dt_recal*100))
print('F1-score:', '{0:.3f}'.format(dt_f1*100))
print(classification_report(y_test,y_pred))

# Confusion Matrix
cm_display_dt = ConfusionMatrixDisplay(dt_cm, display_labels=dt.classes_)
fig, ax = plt.subplots(figsize=(10,10)) # adjust the size
cm_display_dt.plot(ax=ax,cmap='Greens')
```

Multilayer Perceptron (MLP)

```
# Multilayer Perceptron (MLP)

mlp = MLPClassifier(random_state=5, max_iter=300)

mlp.fit(x_train, y_train)

y_pred = mlp.predict(x_test)

mlp_acc = accuracy_score(y_pred, y_test)
mlp_prec = precision_score(y_test, y_pred, average='macro')
mlp_recal = recall_score(y_test, y_pred, average='macro')
mlp_cm = confusion_matrix(y_test,y_pred)
mlp_f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', '{0:.3f}'.format(mlp_acc*100))
print('Precision:', '{0:.3f}'.format(mlp_prec*100))
print('Recall:', '{0:.3f}'.format(mlp_recal*100))
print('F1-score:', '{0:.3f}'.format(mlp_f1*100))
print(classification_report(y_test,y_pred))

# Confusion Matrix
cm_display_mlp = ConfusionMatrixDisplay(mlp_cm, display_labels=mlp.classes_)
fig, ax = plt.subplots(figsize=(10,10)) # adjust the size
cm_display_mlp.plot(ax=ax,cmap='bone_r')
```

A.1.6 Prediction & Reply Generation

```
# Setting model weights according to accuracies and normalize the weights
```

```python
accuracies = np.array([svm_acc, lr_acc, rfc_acc, xgbc_acc, mnb_acc, dt_acc,
    mlp_acc])
norm_accuracy = accuracies - min(accuracies)
model_weight = norm_accuracy / sum(norm_accuracy)  #SVM, Logistic Regression,
    RF, XGB, NB, DT, MLP
Intents = df['Intent'].unique()

def extract_best_intent(list_intent_pred):
    intent_scores = {}
    for intent in Intents:
        intent_scores[intent] = 0.0
    for i in range(len(list_intent_pred)):
        intent_scores[list_intent_pred[i]] += model_weight[i]
    si = sorted(intent_scores.items(), key = lambda pair:pair[1],reverse=True)
    [:7]
    return si[0][0],si

import random
def response_generate(text, intent_name):
    reply = respond(text, intent_name)
    return reply


def respond(text, intent_name):
    maximum = float('-inf')
    response = ""
    closest = ""
    replies = {}
    list_sim, list_replies = [],[]
    dataset = df[df['Intent']==intent_name]
    for i in dataset.iterrows():
        sim = cosine_distance_countvectorizer_method(text, i[1]['User'])
        list_sim.append(sim)
        list_replies.append(i[1]['Chatbot'])

    for i in range(len(list_sim)):
        if list_sim[i] in replies:
            replies[list_sim[i]].append(list_replies[i])
        else:
            replies[list_sim[i]] = list()
            replies[list_sim[i]].append(list_replies[i])
    d1 = sorted(replies.items(), key = lambda pair:pair[0],reverse=True)
    return d1[0][1][random.randint(0,len(d1[0][1])-1)]

def cosine_distance_countvectorizer_method(s1, s2):

    # sentences to list
    allsentences = [s1 , s2]

    from scipy.spatial import distance

    # text to vector
    vectorizer = CountVectorizer()
    all_sentences_to_vector = vectorizer.fit_transform(allsentences)

    text_to_vector_v1 = all_sentences_to_vector.toarray()[0].tolist()
    text_to_vector_v2 = all_sentences_to_vector.toarray()[1].tolist()

    # distance of similarity
    cosine = distance.cosine(text_to_vector_v1, text_to_vector_v2)
    return round((1-cosine),2)

while True:
    input_str = input("What's in your mind: ")
    if input_str == 'nothing':
        break

    processed_text = tfidf_transformer_xtest.fit_transform(countVectorizer1.transform([input_str]))
    print('Intent using SVM: ',end = '')
    svm_intent = svm.predict(processed_text)[0]
    lr_intent = logisticRegr.predict(processed_text)[0]
    dt_intent = dt.predict(processed_text)[0]
    mnb_intent = mnb.predict(processed_text)[0]
    xgbc_intent = xgbc.predict(processed_text)[0]
    rfc_intent = rfc.predict(processed_text)[0]
    mlp_intent = mlp.predict(processed_text)[0]
    print(svm_intent)

    print('Intent using Logistic Regression: ',end = '')
    print(logisticRegr.predict(processed_text))
    print('Intent using Decision Tree: ',end = '')
    print(dt.predict(processed_text))
    print('Intent using Naive Bayes: ',end = '')
    print(mnb.predict(processed_text))
    print('Intent using XGBoost: ',end = '')
    print(xgbc.predict(processed_text))
    print('Intent using Random Forest: ',end = '')
    print(rfc.predict(processed_text))
    print('Intent using Multi Layer Perceptron: ',end = '')
    print(mlp.predict(processed_text))


    # generating reply
    list_intent = [svm_intent, lr_intent, rfc_intent, xgbc_intent, mnb_intent, 
    dt_intent, mlp_intent]
    best_intent, si = extract_best_intent(list_intent)
    print(si)
    print('Best Intent:',best_intent)
    print('EDAIC: ',response_generate(input_str, best_intent))
    print()
```

A.2 Emotion Detection

A.2.1 Dataset

```
df_emotion = pd.read_csv('text_emotions_neutral.csv')

print('Dataset size:',df_emotion.shape)
print('Columns are:',df_emotion.columns)

# Visualize the Sentiments
sns.countplot(x = 'sentiment', data = df_emotion)

# printing the frequency of Sentiment
import collections
emotion_counter = collections.Counter(df_emotion['sentiment'])
emotion_counter
```

A.2.2 Text Preprocessing

- Remove Punctuation & Replace emojis
- Tokenization
- Remove Stopwords
- Lemmatization

```
# Remove Punctuation, Replace emojis
emojis = pd.read_csv('emojis.txt',sep=',',header=None)
emojis_dict = {i:j for i,j in zip(emojis[0],emojis[1])}
pattern = '|'.join(sorted(re.escape(k) for k in emojis_dict))

def replace_emojis(text):
    text = re.sub(pattern,lambda m: emojis_dict.get(m.group(0)), text, flags=re
    .IGNORECASE)
    return text

```

```
def remove_punct(text):
    text = replace_emojis(text)
    text  = "".join([char for char in text if char not in string.punctuation])
    text = re.sub('[0-9]+', '', text)
    return text

# Tokenization
def tokenization(text):
    text = text.lower()
    text = re.split('\W+', text)

    return text

# Remove Stopwords
nltk.download('stopwords')
stopword = nltk.corpus.stopwords.words('english')
stopword.extend(['yr', 'year', 'woman', 'man', 'girl','boy','one', 'two', 'sixteen', 'yearold', 'fu', 'weeks', 'week',
                'treatment', 'associated', 'patients', 'may','day', 'case','old', 'u','n','didnt','ive','ate','feel','keep'
                 ,'brother','dad','basic','im',''])

def remove_stopwords(text):
    text = [word for word in text if word not in stopword]
    return text

# Lemmatization
nltk.download('wordnet')
wn = nltk.WordNetLemmatizer()

def lemmatizer(text):
    text = [wn.lemmatize(word) for word in text]
    return text

def clean_text(text):
    text = remove_punct(text)
    text = tokenization(text)
    text = remove_stopwords(text)
    text = lemmatizer(text)
    return text
```

A.2.3 Feature extraction

```
# Dataset Splitting in 70%-30%
X_train, X_test, y_train, y_test = train_test_split(df_emotion['content'], df_emotion['sentiment'],test_size=0.3, random_state = 116)
```

```

countVectorizer1 = CountVectorizer(analyzer=clean_text)
countVector1 = countVectorizer1.fit_transform(X_train)

countVector2 = countVectorizer1.transform(X_test)

tfidf_transformer_xtrain = TfidfTransformer()
x_train = tfidf_transformer_xtrain.fit_transform(countVector1)

tfidf_transformer_xtest = TfidfTransformer()
x_test = tfidf_transformer_xtest.fit_transform(countVector2)
```

A.2.4 Models

1. Support Vector Machine (SVM)
2. Logistic Regression
3. Random Forest Classifier
4. XGBoost Classifier
5. Multinomial Naive Bayes
6. Decision Tree
7. Multilayer Perceptron (MLP)

Support Vector Machine (SVM)

```
# Support Vector Machine (SVM)

svm = SGDClassifier()
svm.fit(x_train, y_train)

y_pred = svm.predict(x_test)

svm_acc = accuracy_score(y_pred, y_test)
svm_prec = precision_score(y_test, y_pred, average='macro')
svm_recal = recall_score(y_test, y_pred, average='macro')
svm_cm = confusion_matrix(y_test,y_pred)
svm_f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', '{0:.3f}'.format(svm_acc*100))
print('Precision:', '{0:.3f}'.format(svm_prec*100))
print('Recall:', '{0:.3f}'.format(svm_recal*100))
print('F1-score:', '{0:.3f}'.format(svm_f1*100))
print(classification_report(y_test,y_pred))

# Confusion Matrix
cm_display_svm = ConfusionMatrixDisplay(svm_cm, display_labels=svm.classes_)
fig, ax = plt.subplots(figsize=(8,8)) # adjust the size
cm_display_svm.plot(ax=ax,cmap='Blues',values_format='')
```

Logistic Regression

```
# Logistic Regression

logisticRegr = LogisticRegression()

logisticRegr.fit(x_train, y_train)

y_pred = logisticRegr.predict(x_test)

lr_acc = accuracy_score(y_pred, y_test)
lr_prec = precision_score(y_test, y_pred, average='macro')
lr_recal = recall_score(y_test, y_pred, average='macro')
lr_cm = confusion_matrix(y_test,y_pred)
lr_f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', '{0:.3f}'.format(lr_acc*100))
print('Precision:', '{0:.3f}'.format(lr_prec*100))
print('Recall:', '{0:.3f}'.format(lr_recal*100))
print('F1-score:', '{0:.3f}'.format(lr_f1*100))
print(classification_report(y_test,y_pred))

# Confusion Matrix
cm_display_lr = ConfusionMatrixDisplay(lr_cm, display_labels=logisticRegr.
    classes_)
fig, ax = plt.subplots(figsize=(8,8)) # adjust the size
cm_display_lr.plot(ax=ax,cmap='PuRd',values_format='')
```

Random Forest Classifier

```
# Random Forest Classifier

rfc = RandomForestClassifier(n_estimators=1, random_state=0)

rfc.fit(x_train, y_train)

y_pred = rfc.predict(x_test)

rfc_acc = accuracy_score(y_pred, y_test)
rfc_prec = precision_score(y_test, y_pred, average='macro')
```

```
rfc_recal = recall_score(y_test, y_pred, average='macro')
rfc_cm = confusion_matrix(y_test,y_pred)
rfc_f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', '{0:.3f}'.format(rfc_acc*100))
print('Precision:', '{0:.3f}'.format(rfc_prec*100))
print('Recall:', '{0:.3f}'.format(rfc_recal*100))
print('F1-score:', '{0:.3f}'.format(rfc_f1*100))
print(classification_report(y_test,y_pred))

# Confusion Matrix
cm_display_rfc = ConfusionMatrixDisplay(rfc_cm, display_labels=rfc.classes_)
fig, ax = plt.subplots(figsize=(8,8)) # adjust the size
cm_display_rfc.plot(ax=ax,cmap='hot_r',values_format='')
```

XGBoost Classifier

```
# XGBoost Classifier

xgbc = XGBClassifier(max_depth=16, n_estimators=1000,nthread = 6)
xgbc.fit(x_train,y_train)
y_pred = xgbc.predict(x_test)

xgbc_acc = accuracy_score(y_pred, y_test)
xgbc_prec = precision_score(y_test, y_pred, average='macro')
xgbc_recal = recall_score(y_test, y_pred, average='macro')
xgbc_cm = confusion_matrix(y_test,y_pred)
xgbc_f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', '{0:.3f}'.format(xgbc_acc*100))
print('Precision:', '{0:.3f}'.format(xgbc_prec*100))
print('Recall:', '{0:.3f}'.format(xgbc_recal*100))
print('F1-score:', '{0:.3f}'.format(xgbc_f1*100))
print(classification_report(y_test,y_pred))

# Confusion Matrix
cm_display_xgbc = ConfusionMatrixDisplay(xgbc_cm, display_labels=xgbc.classes_)
_, ax = plt.subplots(figsize=(8,8)) # adjust the size
cm_display_xgbc.plot(ax=ax,cmap='Purples',values_format='')
```

Multinomial Naive Bayes

```
# Multinomial Naive Bayes

mnb = MultinomialNB()
mnb.fit(x_train, y_train)

```

```
y_pred = mnb.predict(x_test)

mnb_acc = accuracy_score(y_pred, y_test)
mnb_prec = precision_score(y_test, y_pred, average='macro')
mnb_recal = recall_score(y_test, y_pred, average='macro')
mnb_cm = confusion_matrix(y_test,y_pred)
mnb_f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', '{0:.3f}'.format(mnb_acc*100))
print('Precision:', '{0:.3f}'.format(mnb_prec*100))
print('Recall:', '{0:.3f}'.format(mnb_recal*100))
print('F1-score:', '{0:.3f}'.format(mnb_f1*100))
print(classification_report(y_test,y_pred))

# Confusion Matrix
cm_display_mnb = ConfusionMatrixDisplay(mnb_cm, display_labels=mnb.classes_)
fig, ax = plt.subplots(figsize=(8,8)) # adjust the size
cm_display_mnb.plot(ax=ax,cmap='Reds',values_format='')
```

Decision Tree

```
# Decision Tree

dt = tree.DecisionTreeClassifier()
dt.fit(x_train, y_train)
y_pred = dt.predict(x_test)


dt_acc = accuracy_score(y_pred, y_test)
dt_prec = precision_score(y_test, y_pred, average='macro')
dt_recal = recall_score(y_test, y_pred, average='macro')
dt_cm = confusion_matrix(y_test,y_pred)
dt_f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', '{0:.3f}'.format(dt_acc*100))
print('Precision:', '{0:.3f}'.format(dt_prec*100))
print('Recall:', '{0:.3f}'.format(dt_recal*100))
print('F1-score:', '{0:.3f}'.format(dt_f1*100))
print(classification_report(y_test,y_pred))

# Confusion Matrix
cm_display_dt = ConfusionMatrixDisplay(dt_cm, display_labels=dt.classes_)
fig, ax = plt.subplots(figsize=(8,8)) # adjust the size
cm_display_dt.plot(ax=ax,cmap='Greens',values_format='')
```

Multilayer Perceptron (MLP)

```python
# Multilayer Perceptron (MLP)

mlp = MLPClassifier(random_state=5, max_iter=300)

mlp.fit(x_train, y_train)

y_pred = mlp.predict(x_test)

mlp_acc = accuracy_score(y_pred, y_test)
mlp_prec = precision_score(y_test, y_pred, average='macro')
mlp_recal = recall_score(y_test, y_pred, average='macro')
mlp_cm = confusion_matrix(y_test,y_pred)
mlp_f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', '{0:.3f}'.format(mlp_acc*100))
print('Precision:', '{0:.3f}'.format(mlp_prec*100))
print('Recall:', '{0:.3f}'.format(mlp_recal*100))
print('F1-score:', '{0:.3f}'.format(mlp_f1*100))
print(classification_report(y_test,y_pred))

# Confusion Matrix
cm_display_mlp = ConfusionMatrixDisplay(mlp_cm, display_labels=mlp.classes_)
fig, ax = plt.subplots(figsize=(8,8)) # adjust the size
cm_display_mlp.plot(ax=ax,cmap='bone_r',values_format='')
```

A.2.5 Emotion Prediction

```python
# Setting model weights according to accuracies and normalize the weights
accuracies = np.array([svm_acc, lr_acc, rfc_acc, xgbc_acc, mnb_acc, dt_acc,
    mlp_acc])
norm_accuracy = accuracies - min(accuracies)
model_weight = norm_accuracy / sum(norm_accuracy)  #SVM, Logistic Regression,
    RF, XGB, NB, DT, MLP

def extract_best_emotion(list_emotion_pred):
    emotion_scores = {}
    for emotions in Emotions:
        emotion_scores[emotions] = 0.0
    for i in range(len(list_emotion_pred)):
        emotion_scores[list_emotion_pred[i]] += emotion_model_weight[i]
    se = sorted(emotion_scores.items(), key = lambda pair:pair[1],reverse=True)
    return se[0][0], round(se[0][1],2)

while True:
    input_str = input("What's in your mind: ")
    if input_str == 'nothing':
        break

    processed_text = tfidf_transformer_xtest.fit_transform(countVectorizer1.transform([input_str]))

    svm_emotion = svm.predict(processed_text)[0]
    lr_emotion = logisticRegr.predict(processed_text)[0]
    dt_emotion = dt.predict(processed_text)[0]
    mnb_emotion = mnb.predict(processed_text)[0]
    xgbc_emotion = xgbc.predict(processed_text)[0]
    rfc_emotion = rfc.predict(processed_text)[0]

    list_emotion_pred = [svm_emotion, lr_emotion, rfc_emotion, xgbc_emotion, mnb_emotion, dt_emotion]
    best_emotion, prob = extract_best_emotion(list_emotion_pred)
    print('Best Emotion:',best_emotion,':',prob)

    print('Emotion using SVM: ',end = '')
    print(svm.predict(processed_text))
    print('Emotion using Logistic Regression: ',end = '')
    print(logisticRegr.predict(processed_text))
    print('Emotion using Decision Tree: ',end = '')
    print(dt.predict(processed_text))
    print('Emotion using Naive Bayes: ',end = '')
    print(mnb.predict(processed_text))
    print('Emotion using XGBoost: ',end = '')
    print(xgbc.predict(processed_text))
    print('Emotion using Random Forest: ',end = '')
    print(rfc.predict(processed_text))
    print()
```

Generated using Undegraduate Thesis LaTeX Template, Version 1.4. Department of Computer Science and Engineering, Ahsanullah University of Science and Technology, Dhaka, Bangladesh.

This thesis was generated on Sunday 9th January, 2022 at 5:37pm.