Islamic University of Technology (IUT)
Organization of Islamic Cooperation (OIC)
Department of Computer Science and Engineering (CSE)

Bachelor of Science in Engineering (B.Sc.Engg)

Optimized Human-Emotion Detection in Written-Text using Hybrid Machine Learning Classification Algorithm

Authored By

Mr.*Fopa Yuffon Amadou Olabi*
Student-ID: **154446**
Dept. of CSE, IUT
fopayuffon@iut-dhaka.edu

Mr.*Mohamadou Moctar*
Student-ID: **160041082**
Dept. of CSE, IUT
mohamadoumoctar@iut-dhaka.edu

Mr.*Mikayilou Namba*
Student-ID: **160041085**
Dept. of CSE, IUT
mikayilou@iut-dhaka.edu

Supervisor
Abu Raihan Mostafa Kamal, PhD
Professor & Head of Dept.
Dept. of CSE, IUT

Co-Supervisor
Md. Hamjajul Ashmafee
Lecturer
Dept. of CSE, IUT

A thesis submitted in partial fulfilment of the requirements
for the degree of B. Sc. Engineering in Computer Science and Engineering

Academic Year: 2019-2020

March, 2021.

Thesis Dissertation

Optimized Human-Emotion Detection in Written-Text using Hybrid Machine Learning Classification Algorithm

Submitted by,

Mr. *Fopa Yuffon Amadou Olabi*, ***154446***,
fopayuffon@iut-dhaka.edu

Mr. *Mohamadou Moctar*, ***160041082***,

mohamadoumoctar@iut-dhaka.edu

Mr. *Mikayilou Namba*, ***160041085***,

mikayilou@iut-dhaka.edu

Department of *Computer Science and Engineering*, CSE

Islamic University of Technology, IUT
Dhaka, Bangladesh

In Fulfillment of our ***Bachelor of Science in Engineering***,
Supervised by *Md. Hamjajul Ashmafee*, **Lecturer**, CSE,
and *Prof. Dr. Abu Raihan Mostafa Kamal*, **Head of Department**, CSE,
under the Thesis Group, Networking and Data Analysis (NDAG),

Wednesday 17th March, 2021

Declaration of Authorship

This is to certify that the work presented in this thesis dissertation is the outcome of the analysis and experiments carried out by F.Y. Amadou Olabi, Mohamadou Moctar, and Mikayilou Namba, Engineering Students of the CSE Department, under the supervision and mentorship of Md. Hamjajul Ashmafee, Lecturer of the Department of Computer Science and Engineering (CSE),and Abu Raihan Mostafa Kamal, Professor and Head of the Department of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Dhaka, Bangladesh. It is also declared that neither of this thesis nor any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Authors:

Fopa Yuffon Amadou Olabi [illegible]

Student ID - 154446

Mohamadou Moctar [illegible]

Student ID - 160041082

Mikayilou Namba [illegible]

Student ID - 160041085

Approved By:

Supervisors:

[illegible]

Md. Hamjajul Ashmafee
Lecturer
Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT), OIC

[illegible]

Abu Raihan Mostafa Kamal, PhD
Professor & Head of Dept.
Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT), OIC

Acknowledgement

All the praises and thanks be to Allah, the Lord of the Alamin (mankind, jinn, and all that exists). Qur'an (v.1:2).

First and foremost, I thank the Almighty Allah (SWT) for blessing us with this beautiful thesis work and for giving us the knowledge, strength, determination, spirit of team work, and for accompanying us throughout the successful completion of this thesis.

My Special thanks goes to **Md. Hamjajul Ashmafee**, *Lecturer of CSE*, and **Abu Raihan Mostafa Kamal**, *PhD, Professor & Head of CSE Dept*, for supervising this research work by supporting and directing us throughout the successful completion of this thesis.

Dedication

This thesis work is dedicated to our respective families, namely, the **Nganju Njifenjou** & **Eyabi** Families, the **Mohamadou** Family, and the **Namba** Family, who were a constant source of support and encouragement for us during the challenges of our undergraduate school and life. We are truly thankful for having them in our life. This work is also dedicated to our friends, comrades, fellow students, and the IUT faculty members.

Contents

List of Figures

List of Tables

Abstract

No part of our psychological life is more essential to the quality and significance of our reality than emotions. In psychology, Emotion is often defined as a complex state of feeling that results in physical and psychological changes that influence thought and behavior. Emotionality is associated with a range of psychological phenomena, including temperament, personality, mood, and motivation. In 1972, psychologist Paul Eckman suggested that six basic emotions are universal throughout human cultures: fear, disgust, anger, surprise, happiness, and sadness. Emotion Recognition is an important area of work to improve the interaction between humans and machines. Emotion Detection will play a promising role in the field of Artificial Intelligence, especially in the case of Human-Machine Interface Development, Human-Computer Interaction (HCI), User-Experience (UX), and Designs.
In our study case, we went through the vast area of Emotion Recognition and Detection from an AI and ML perspective, in which different parameters were taken into consideration. In this work, through our research, we developed a Human-Emotion Detection methodology based on Written-Text using a preprocessing technique based on **meaningless stop words removal** and a **Hybrid-ML Algorithm**, which is made of a Naïve-Bayes Classifier (**NBC**) and a Convolutional Neural Network (**CNN**) for a better accuracy alongside with an Optimized Text-Analysis method for Preprocessing.
The preprocessing is built up around many different techniques that help the data to be reliable, standardized, and clean. It all started with the stop word removal which is one of the key parts of our work, then the **standardization** of the data and the following part was **tokenization**, followed by the **TF-IDF Vectorization** which was applied and we finished by a **vocabulary construction**.

Keywords — Daily Dialog Dataset, OEDHML Framework, Term Frequency Inverse Document Frequency (TF-IDF), Text Vectorization, Text-Analysis, Text Classification, Emotion Detection, Naive-Bayes Classification (NBC), Convolutional Neural Networks (CNN), Stop-words Removal, Standardization, Text Tokenization, Vocabulary Construction, Padded Sequences, TensorFlow and TensorFlow-Text.

Abbreviations

- **OEHML** or **OEDHML** : Optimized Human-Emotion Detection in Written-Text using Hybrid Machine Learning Classification Algorithm.
- **NBC** : Naive Bayes Classifier
- **CNN** : Convolutional Neural Networks
- **TF** : Term Frequency
- **IDF** : Inverse Document Frequency

1 Introduction

1.1 Overview

Emotion is a psychological state or a process that connects goals with events in the real world. Emotion can establish a relationship between the author and the readers, and it can show a person's response to something [11]. The development of internet users makes the delivery of information rapid and broader. The number of internet users in 2012 was 63 million, and in 2017 it was 143.6 million [10]. The most accessed service was social media with 87.13% of Internet users. This number was greater than the usage of the internet to access search engines, which was only 74.84% of internet users in 2017 [10].
Emotion detection has proven to be one of the most challenging tasks in natural language processing, not only because humans are very complex but also because there is a lot of possibilities and outcomes. The emotion detection sector has received a lot of interest from academia, businesses, and also advertisement giants (Google, Twitter, MacDonald, etc.) in recent years. That interest has resulted in multiple numbers of tough and advanced researches with a wide variety of applications in real-life scenarios. Emotions can be found on news snippets, short messages and tweets, Facebook comments, and other social media.
Applications of emotion detection and recognition using supervised, unsupervised, or hybrid learning methodology to classify emotions in different categories. Supervised learning provides powerful tools to classify and process data using machine language. With supervised learning you use labeled data, which is a data set that has been classified, to infer a learning algorithm [8]. Unsupervised Learning is a machine learning technique in which the users do not need to supervise the model. Instead, it allows the model to work on its own to discover patterns and information that was previously undetected [4]. It mainly deals with the unlabeled data [4]. Compared to the supervised approach, unsupervised learning only operates on input data without outputs or target variables. As such, unsupervised learning does not require a master to correct the model, as in the case of supervised learning. The lines between unsupervised and supervised learning are blurry, and many hybrid approaches draw from each field of study.
In this study, our approach is to improve the existing emotion classification techniques in two aspects. Firstly, the improvement will be done on the preprocessing aspect by filtering words that can change the sentence's sense or impact its meaning in one way or another. Secondly, we will be using a supervised learning approach that implements a hybrid technique by merging Naïve Bayes and Convolutional Neural Network for better accuracy and performance.

1.2 Problem Statement

Many industries and even academics are closing their gate days by days, Businesses are getting attacked, social media are being boycotted, which can result not only in social chaos but also in an economical crisis for many firms. These attacks or boycotts are direct actions resulting from people's state of mind, in other words, their emotions. People are gladly expressing their emotions all over the internet, it can be observed in the form of tweets, comments, reviews, online diaries, blogs, but also during a daily conversation. What if all this information were analyzed and process? from somebody's comments or tweet, a shop could know why their newly released product is not successful, a teaching center will know what they are doing wrong and correct it, the police can even predict if somebody can get murdered if a shop can get attacked of the stolen only base on the way that people express their self on the social media. Media is the center of emotional expression, it's the best place for data collection. Data which will help to predict malicious actions based on angry comments, or unsatisfied customers from negative reviews. Although being a new study area, many scientists have published a multitude of research papers wherein the preprocessing part some were processing meaningless information and others removed meaningful words. This could play a greater role in the text understanding by the machine which also could better the model accuracy.

1.3 Objectives

To achieving the goal of our proposed methodology, the objectives below should be satisfied.

- To Identify people's state of mind based on their textual information across the internet, and bring out the emotions expressed in their daily social media life.
- To Classify these emotions based on emotional standards such as anger, disgust, fear, happiness, sadness, surprise, or no emotion at all.
- To Improve existing preprocessing techniques to fit the model with meaningful data for better predictions.
- To build a hybrid system that can classify emotions based on different comments or text through social media or even written dialogue that could score higher than existing works.

1.4 Challenges

Here are some key parts where we faced a lot of difficulties.

- **Data Acquisition/Collection**: There was indeed a real challenge in finding proper datasets since we intended to work with a set of at least 10 emotions while processing sentiments.
- **Data Preprocessing**: The real difficulty was to preprocess the sample data without removing crucial and deterministic keywords from the text while applying Text Analysis Methodology.
- **Machine Learning Models to be chosen for better accuracy**: Knowing the performance of ML Algorithms related to Classification problems. We intend to use a Hybrid Model to tackle the given thematic which could better the accuracy of such model. The actual challenge faced here is, to choose wisely a combination of models which could perform homogeneously.
- **Finding Recent Research Papers related to this thesis work**.

1.5 Significance of Study

Human is a very sensitive being which could be emotionally affected by the decision. The study will have an ethical impact in a way that will help people decide by considering other's points of view. Sometimes higher authorities or people in a higher position take a decision that hurts their employees or even their firm without them knowing it. Thanks to emotion detection they can prevent this kind of scenario and avoid losses by detecting dissatisfaction from their employees or customers. The study will improve communication amount boss and employee, customers and firms and can also help elucidate crime or prevent it.

2 Literature Review

Emotion classification or even detection is a new concept in which people a getting interesting day by day. People express their feelings in our daily life from numerous ways. It can be by body gesture, by speech, or by what concern us "Text". Many researchers are interested in this study and that interest led to multiple works and publications.

2.1 Emotion Intensity Detection for Social Media Data [14]

As with any order emotion detection paper, this one also uses different techniques to extract emotion using Liza Wikarsa and Sherly Novianti Thahir's approach but focuses more on the extraction of the emotional intensity. Here The feature vector along with the intensity is given as input to the prediction algorithm (Linear regression). This learned model can be then used for predicting intensity levels of unseen tweets.

2.2 A Survey of Textual Emotion Detection [3]

This paper is a survey which outlines the limitation and gaps of the recent works and directs the possible future research to fulfill these gaps in this increasingly developed field. For emotion detection, we have to keep in mind that emotion is from many types (discrete/dimensional), and depending on its basis the number of emotions can vary from 2 to even 14. It also says that the approach for emotion detection can be divided into four approaches. It also shows the performance metrics of the different techniques, the different datasets that can be used, and the preprocessing techniques.

2.3 Social Signal Processing for Evaluating Conversations using Emotion Analysis and Sentiment Detection [19]

This paper presents a multi-modality framework to analyze customer satisfaction levels, especially in determining dissatisfied customers, using an image, speech, and text analysis. It employs a two-level synthesis: Emotion analysis of speech signals and Sentiment detection from text data, which is converted from the same speech, and facial emotion analysis from image data. Their approach helps in better understanding of customers and identification of their exact perspective concerning services provided by the industries. As classifiers, they used Convolutional Neural Networks (CNN) for object recognition which performs detection of the non-verbal sentiments on images, and SVM classifiers with VGGImageNet as the pre-trained model. The disadvantage of SVM is that the SVM algorithm is not suitable for large data sets and does not perform very well when the data set has more noise.

2.4 Tweet Analysis Based on Distinct Opinion of Social Media Users [6]

This paper predicts emotion by considering the text context associated with the emoticons and punctuation instead of using the particular word and emotion alone. The proposed system calls Future Prediction Architecture Based on Efficient Classification (FPAEC) is designed with many different classification algorithms such as Fisher's Linear Discriminant Classifier (FLDC), Support Vector Machine (SVM), Naïve Bayes Classifier (NBC), and Artificial Neural Network (ANN) Algorithm along with the BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) clustering algorithm. The Twitter data are classified into three classes negative, positive and neutral.

2.5 Emotion Detection from Text Using Skip-thought Vectors [9]

In this research, they apply a recently proposed deep learning model named skip-thought, an approach to learning fixed-length representations of sentences, to face the problem of emotion detection from text. They propose a new framework that takes advantage of the pre-trained model and pre-trained word vectors. They found that skip-thought vectors are well suited for the emotion detection task.

2.6 EmoDet2: Emotion Detection in English Textual Dialogue using BERT and BiLSTM Models [2]

The proposed system in this paper uses deep learning approaches to detect emotion. The system is combining a fully connected neural network and BiLSTM neural network. The performance obtains using this proposed system gives an improvement of F1-Score 0.748. The system can determine the emotion in English textual dialogue and classify it into four categories (Happy, Sad, Angry, and Other). Features are extracted using a combination of GloVe Word Embedding, BERT Embedding, and a set of psycho-linguistic features (e.g. from AffectiveTweets Weka-package).
Disadvantage: They did not use the normal pre-processing techniques such as stemming, token, removing stop-words. Instead they applied EKphrasis to achieve high performance in spelling corrections on the data which did not better the accuracy metrics.

2.7 Tweets Emotion Prediction by Using Fuzzy Logic System [21]

This paper uses Fuzzy logic to classify each tweet to emotion with different degrees of intensity. They develop two classification systems one inspects the text it is referred to as (TCFL) and the second inspects emoji associated with each tweet it is referred to as (ECFL). This paper also extends the number of emotions they can be extracted up to eight instead of four in the more general case which are joy, sadness, anger, disgust, trust, fear, surprise, and associate. each of emotion with a degree of intensity which is Extremely High, Very High, High, Medium, Low, Very Low, and Extremely Low. Comparing the developed two systems with human-based classification, TCFL outperformed ECFL with a 48.96% match as compared to a 32.54% match for ECFL.
Disadvantage: More emotion class can lead to misclassification and poor performance.

2.8 Implementation of Text Mining Model to Emotions Detection on Social Media Comments Using Particle Swarm Optimization and Naive Bayes Classifier [12]

This research paper presents a method to classify the dataset into four emotional categories anger, fear, joy, and sadness. The method used is a combination of Naïve Bayes classifier and swarm optimization, testing Naïve Bayes without optimization gives an accuracy of 65%, and adding optimization to Naïve Bayes augmented the accuracy to 66.54%. But the computation process is longer than simple Naïve Bayes.
Advantages: the advantages of the word-based approach such as its simplicity and ease of implementation.
Drawbacks: The 66.54% accuracy is low and need to be improved the accuracy. Here are some of the key pitfalls of this paper which negatively impacted the entire experiment proposed.

- Its major drawbacks include the ambiguity of the word selection found on double meaning words or the words that change their meaning according to the context,
- The existed subjectivity of determining the emotion lexicon contents,
- The existence of sentences with no direct emotional keywords
- Its inability to be applicable in a wide range of application.
- The Datasets were not really insightful to work efficiently with their proposed algorithms.
- The preprocessing removes meaningful words that can be helpful in classification phase while training the model.

3 Proposed Methodology

In this section, we aim to propose a suitable solution that can result in a high-efficiency rate than the previously mentioned research works and implementations in section 2.

3.1 Proposed Framework Overview

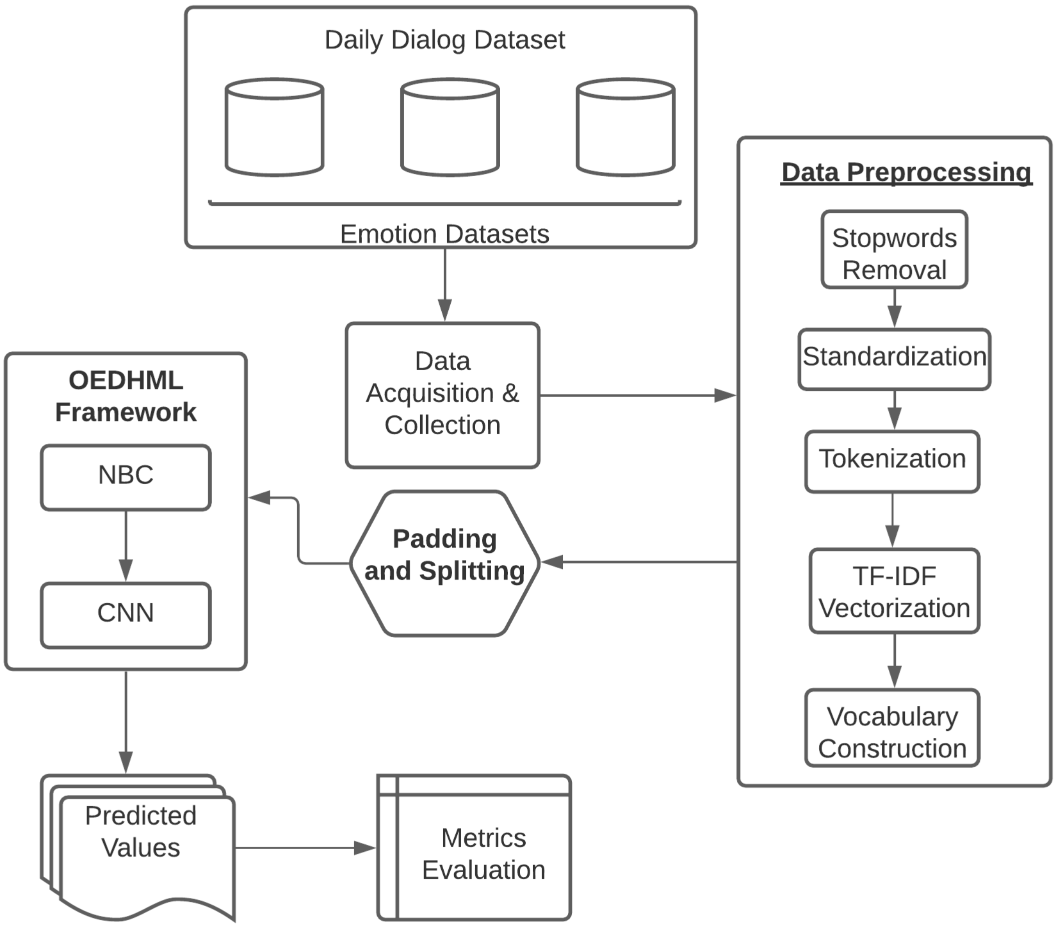
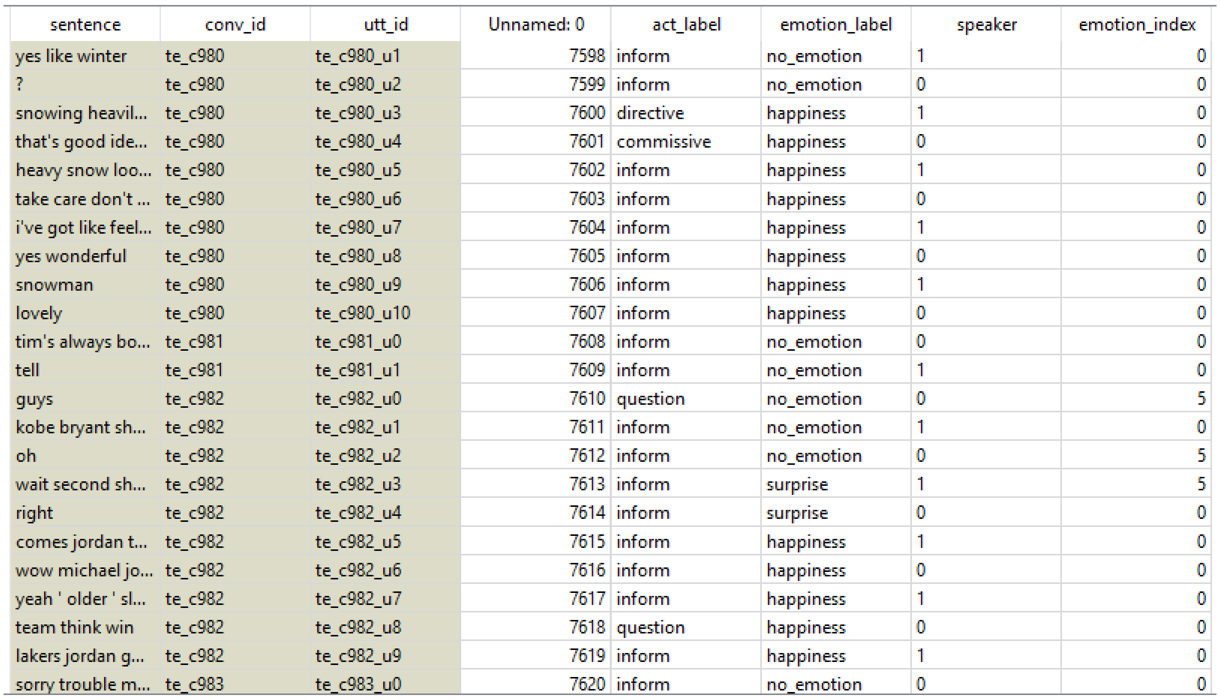


Figure 1: Overview of Framework Architecture

3.2 Data Collection and Feature Analysis

3.2.1 Data Collection

For this research purpose, we used the Daily-Dialog is a high-quality multi-turn open-domain English dialog dataset [13]. It contains 102,978 dialogues split into a training set with 87,532 dialogues and validation and test sets with 15,447 dialogues each. On average there are around 8 speaker turns per dialogue with around 15 tokens per turn. The language is human-written and less noisy. The dialogues in the dataset illustrate how we interact daily and cover a wide range of topics. The dataset is marked with details about communication intent and emotion [13]. This dataset is built from 03 JSON files combined namely, “train.json”, “valid.json”, and “test.json”, with shapes (87532, 8), (9268, 8), and (6179, 8) respectfully.



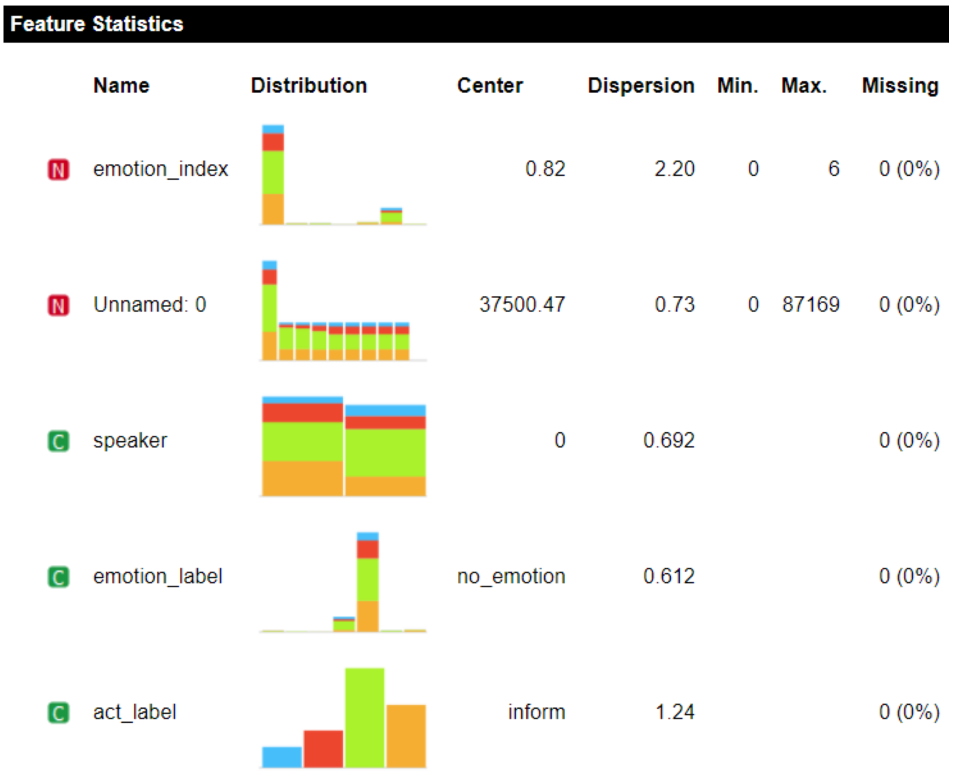
sentence	conv_id	utt_id	Unnamed: 0	act_label	emotion_label	speaker	emotion_index
yes like winter	te_c980	te_c980_u1	7598	inform	no_emotion	1	0
?	te_c980	te_c980_u2	7599	inform	no_emotion	0	0
snowing heavil...	te_c980	te_c980_u3	7600	directive	happiness	1	0
that's good ide...	te_c980	te_c980_u4	7601	commissive	happiness	0	0
heavy snow loo...	te_c980	te_c980_u5	7602	inform	happiness	1	0
take care don't ...	te_c980	te_c980_u6	7603	inform	happiness	0	0
i've got like feel...	te_c980	te_c980_u7	7604	inform	happiness	1	0
yes wonderful	te_c980	te_c980_u8	7605	inform	happiness	0	0
snowman	te_c980	te_c980_u9	7606	inform	happiness	1	0
lovely	te_c980	te_c980_u10	7607	inform	happiness	0	0
tim's always bo...	te_c981	te_c981_u0	7608	inform	no_emotion	0	0
tell	te_c981	te_c981_u1	7609	inform	no_emotion	1	0
guys	te_c982	te_c982_u0	7610	question	no_emotion	0	5
kobe bryant sh...	te_c982	te_c982_u1	7611	inform	no_emotion	1	0
oh	te_c982	te_c982_u2	7612	inform	no_emotion	0	5
wait second sh...	te_c982	te_c982_u3	7613	inform	surprise	1	5
right	te_c982	te_c982_u4	7614	inform	surprise	0	0
comes jordan t...	te_c982	te_c982_u5	7615	inform	happiness	1	0
wow michael jo...	te_c982	te_c982_u6	7616	inform	happiness	0	0
yeah ' older ' sl...	te_c982	te_c982_u7	7617	inform	happiness	1	0
team think win	te_c982	te_c982_u8	7618	question	happiness	0	0
lakers jordan g...	te_c982	te_c982_u9	7619	inform	happiness	1	0
sorry trouble m...	te_c983	te_c983_u0	7620	inform	no_emotion	0	0

Figure 2: Table View (Snapshot) of the Daily Dialogue Dataset

3.2.2 Feature Analysis

From the compiled samples data obtained above, we formed our experimental dataset based on 08 features which are:

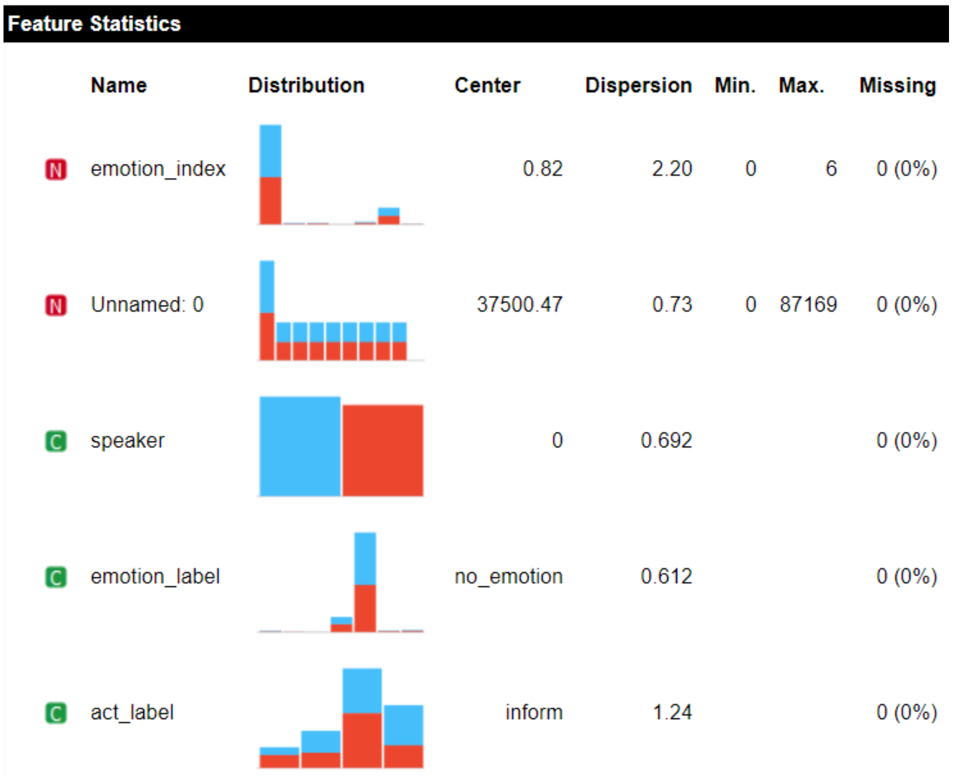
- ‘**ID**’: this column contains the representative identifiers of each written conversational exchange in dialogue.
- ‘**sentence**’: it contains the written conversational exchange in dialogue.
- ‘**act_label**’: column which represents the act or role of speaker in dialogue, denoting whether a speaker’s role is either commissive, directive, inform, or question.
- ‘**emotion_label**’: column which represents the labels of each written conversational exchange, denoting whether an exchange is either no emotion, anger, disgust, fear, happiness, sadness, or surprise.
- ‘**speaker**’: column represents the nature of the speaker which could be either moderating or contributing to a dialogue.
- ‘**conv_id**’: column which represents the identifier of the conversation.
- ‘**emotion_index**’: column which represents the indexed labels of each written conversational exchange, denoting whether the emotion of the sentence column is either no emotion, anger, disgust, fear, happiness, sadness or surprise, which is indexed in a range of [0 — 6] respectfully.



Feature Statistics

	Name	Distribution	Center	Dispersion	Min.	Max.	Missing
N	emotion_index		0.82	2.20	0	6	0 (0%)
N	Unnamed: 0		37500.47	0.73	0	87169	0 (0%)
C	speaker		0	0.692			0 (0%)
C	emotion_label		no_emotion	0.612			0 (0%)
C	act_label		inform	1.24			0 (0%)

Figure 3: Feature Analysis related to Speaker's Act Role



Feature Statistics

	Name	Distribution	Center	Dispersion	Min.	Max.	Missing
N	emotion_index		0.82	2.20	0	6	0 (0%)
N	Unnamed: 0		37500.47	0.73	0	87169	0 (0%)
C	speaker		0	0.692			0 (0%)
C	emotion_label		no_emotion	0.612			0 (0%)
C	act_label		inform	1.24			0 (0%)

Figure 4: Feature Analysis related to Speaker's Type

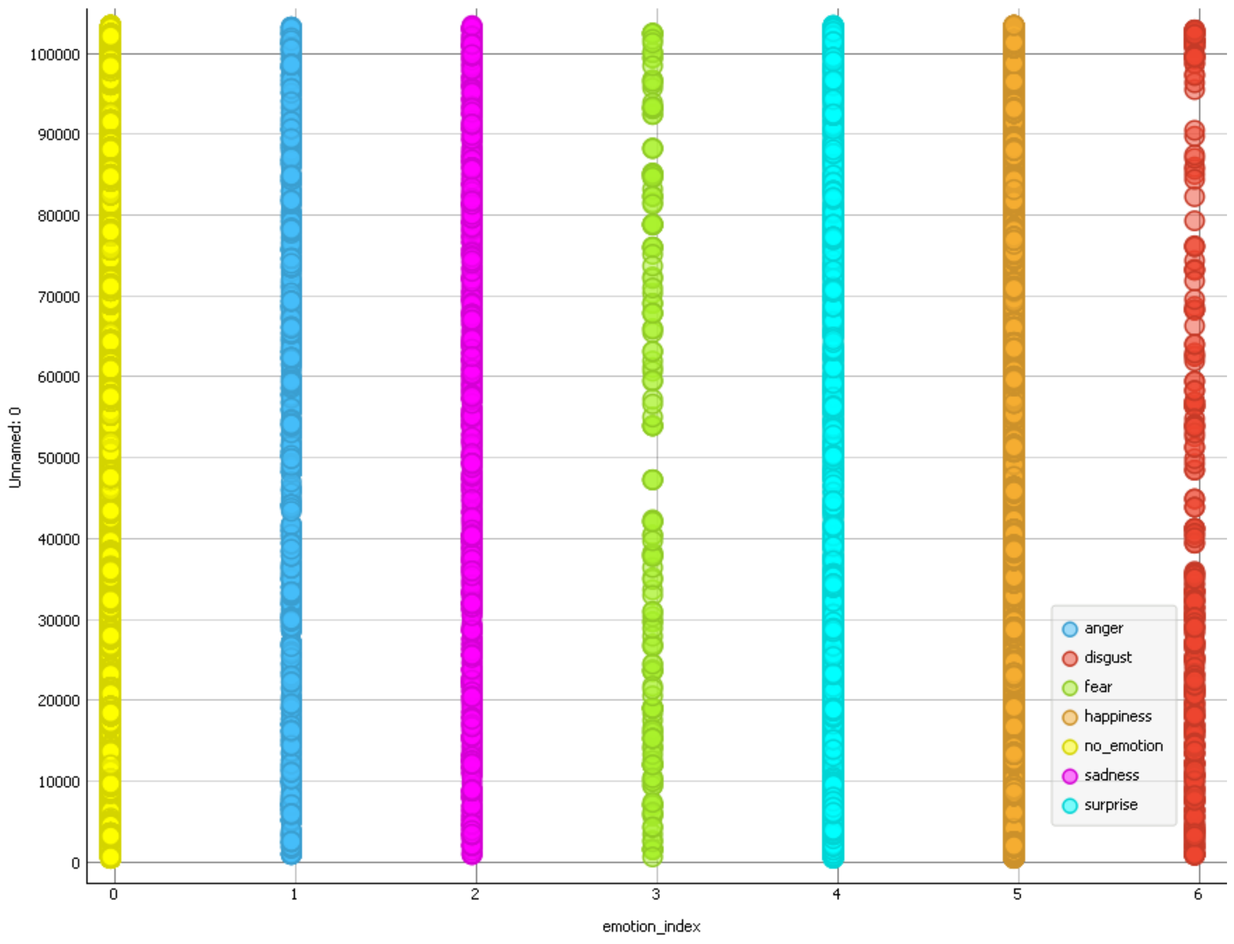


Figure 5: Scatter Plot Representation of the Daily Dialogue Dataset

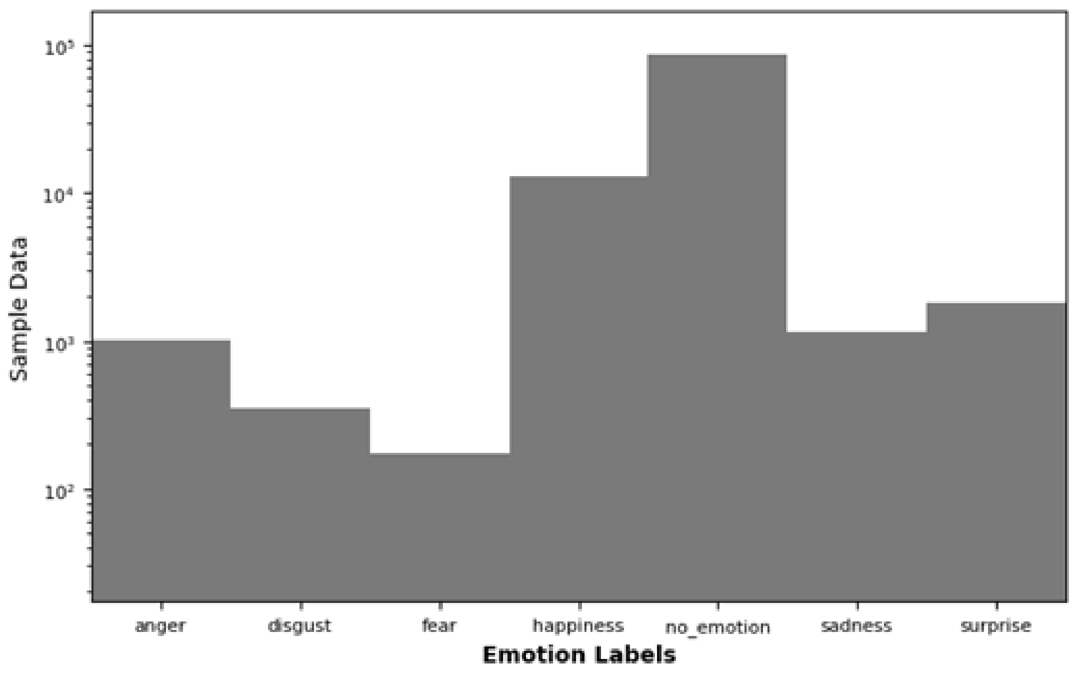


Figure 6: Emotion Labels from the the Daily Dialogue Dataset

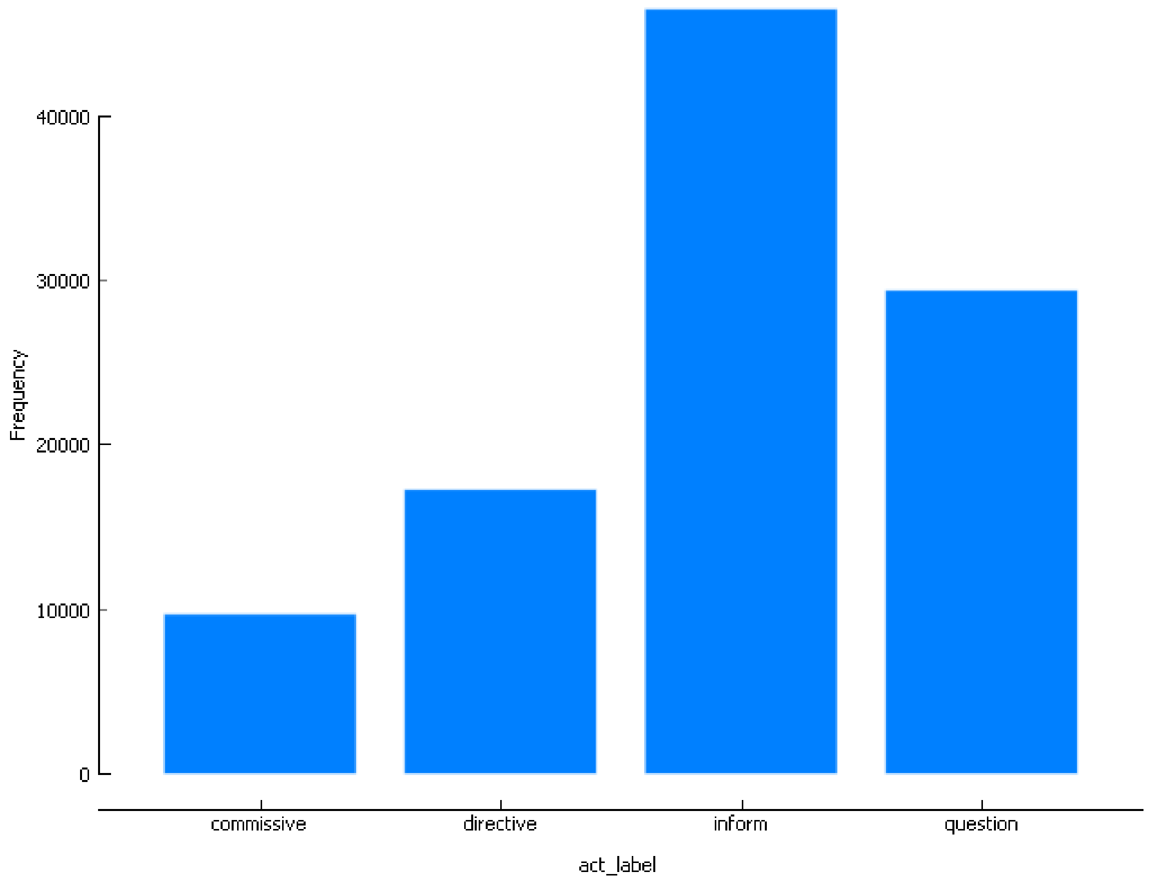


Figure 7: Act Labels from the Daily Dialogue Dataset

3.3 Preprocessing

In Machine Learning, Data-Preprocessing is a very essential step that helps enhance the quality of meaningful data for better feature extraction. It refers to the technique of preparing (cleaning, formatting, and organizing) the raw data to make it suitable for building and training Machine Learning models. In simple words, data preprocessing in Machine Learning is a data mining technique that transforms raw data into an understandable and readable format. Typically, real-world data is incomplete, inconsistent, inaccurate (contains errors or outliers), and often lacks specific attribute values/trends. For this study, the proposed preprocessing technique uses the lower-level utilities of the "**TensorFlow.python.data.experimental.op**s" package to load the dataset from the CSV file, and the "**TensorFlow.text**" package to preprocess the data for finer-grain control, meaning that each data item has its access control policy. This type of access control is typically used in cloud computing, where often a large quantity of data types and data sources may be stored together but each data item must be accessed based on different criteria [5].

3.3.1 Loading the dataset

For this study, we used the dataset "***dailydialog.csv***" as mentioned in section 3.2. The dataset was loaded using **CsvDataset** from TensorFlow library. It's all be automatically optimized for large data accessibility and parallelly provides efficient computation of data. The loaded dataset is accessed through a tuple of elements of 08 columns whose types are Tensors data [22].

3.3.2 Preparing Dataset for Training

In our study, we used the following steps to prepare the sample data.

Stop Words Removal

After loading the dataset, we notice that the text data were containing some garbage data which was altering the quality of the information that we were supposed to extract. Therefore, we decided to clean the dataset before using it by removing meaningless stop words and punctuation. During building the stop words list, we notice that in general English, some of the **stop words** can be **meaningful** therefore **necessary**. After this insightful discovery, we decided to keep those in the data while removing unnecessary stop words.
Below is the list of stop words to keep:

Above	Before	But	didn't	haven't	isn't	wouldn't
against	Below	can't	doesn't	hasn't	mustn't	shan't
any	between	cannot	don't	hadn't	no	
aren't	both	couldn't	down	during	nor	
not	won't	weren't	wasn't	under	shouldn't	

Table 1: Meaningful Stop-words to be left while preprocessing.

Standardization

The process of translating data into a common format so that users can process and interpret is known as **Data Standardization**. For many purposes, data standardization is important. First and foremost, it aids in the establishment of clearly specified elements and attributes, resulting in a detailed catalog of your results. Whatever insights you're searching for or issues you're trying to solve; a good understanding of your data is a must-have first step. To get there, you'll need to translate the data into a standard format with logical and consistent definitions. These meanings can be used to create metadata, which are labels that define what, how, why, who, when, and where your data is stored. Your **Data Standardization** process is based on this base. We used the **TensorFlow-Text** libraries to standardize the data.

Tokenization, Vectorization and Vocabulary Construction

Afterward, instead of using the TensorFlow **Text-Vectorization** layer to preprocess the text data of our compiled dataset, we used the **TensorFlow-Text** libraries to tokenize these data, which later help to build a vocabulary table using the **Static-Vocabulary-Table** from TensorFlow. This will be mapped with each token of the vocabulary according to the number of occurrences in the dataset to be fitted to the proposed model, with a range of *[2,* ***VOCAB_SIZE*** *+ 2]*. This vocabulary table is built by sorting tokens by frequency and keeping the top ***VOCAB_SIZE*** tokens. As with the **Text-Vectorization** layer, **0** is reserved to denote padding, and **1** is reserved to denote an **out-of-vocabulary** (**OOV**) token.
While TensorFlow-Text provides various tokenizers, we will use the **UnicodeScriptTokenizer** to tokenize with the converted lower-case of our dataset. This will then be mapped to each row of text for proper tokenization of the dataset.
Finally, we transformed the entire dataset into encoded vectors of weighted sums with aid of the vocabulary table to map each text of the dataset with vectors of numbers representing the occurrences of a word.

Splitting and Padding the dataset

We split the dataset into the train, valid, and test sets with a ratio of 85%, 9%, and 6% respectfully representing 87532 rows, 9268 rows, and 6179 rows. These sets were having different dimensions per row, which couldn't be fitted to the model. This is the reason why we padded these sets (train, valid, test) of data with a batch size of 64 to obtain a shape evenly distributed across the entire dataset.

Padded Sequences — tensorflow.keras.preprocessing.sequence.pad_sequences

This function transforms a list (of length **num_samples**) of sequences (lists of integers) into a 2D Numpy array of shape (**num_samples**, **num_timesteps**). **num_timesteps** is either the maxlen argument if provided, or the length of the longest sequence in the list. Sequences that are shorter than **num_timesteps** are padded with value until they are **num_timesteps** long. Sequences longer than **num_timesteps** are truncated so that they fit the desired length. The position where padding or truncation happens is determined by the arguments padding and truncating, respectively. Pre-padding or removing values from the beginning of the sequence is the default.

3.4 Proposed Model (OEDHML Framework)

For this study, we used a supervised learning approach with two different Machine Learning Algorithms, namely, Naïve Bayes Classifier (NBC) and Convolutional Neural Network (CNN), to evaluate the goodness of our proposed preprocessing technique.

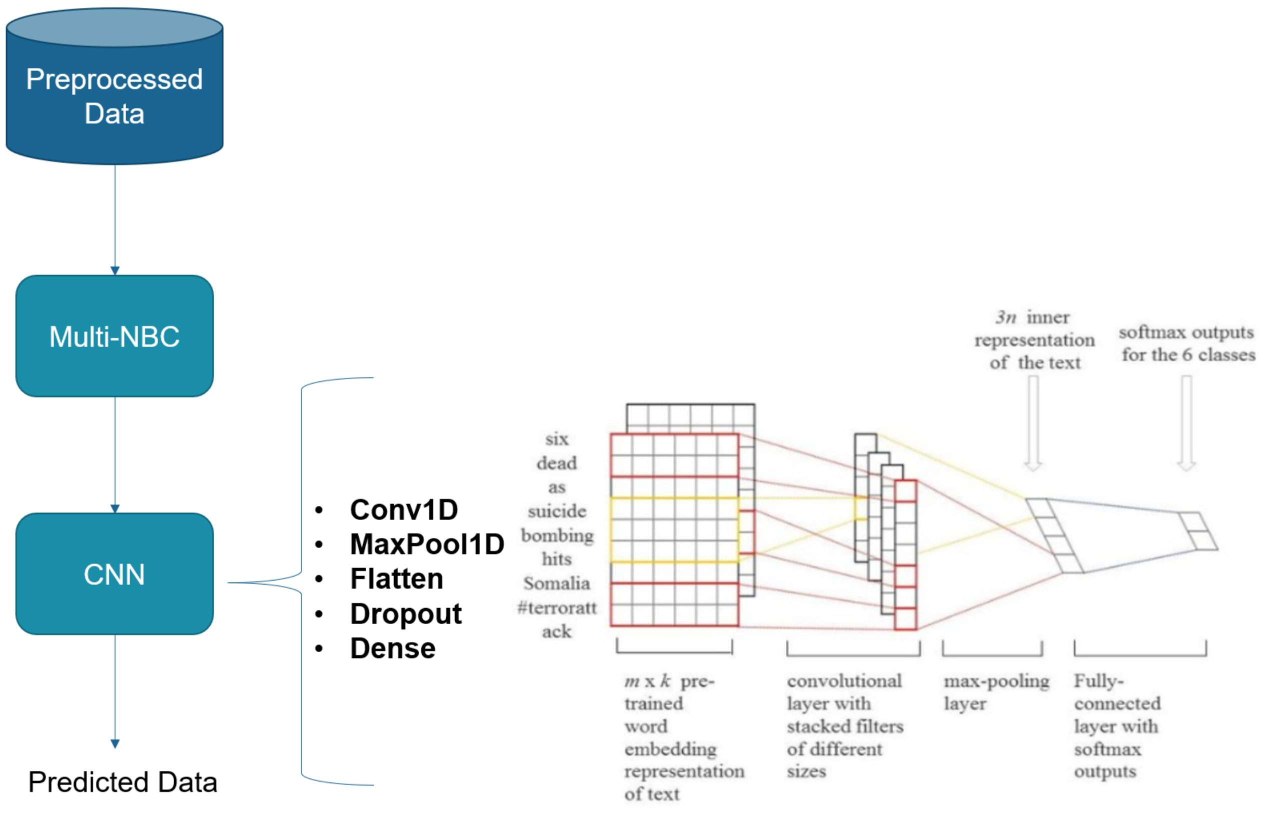

Preprocessed Data
Multi-NBC
CNN
Predicted Data
Conv1D
MaxPool1D
Flatten
Dropout
Dense
six dead as suicide bombing hits Somalia #terroratt ack
3n inner representation of the text
softmax outputs for the 6 classes
m x k pre-trained word embedding representation of text
convolutional layer with stacked filters of different sizes
max-pooling layer
Fully-connected layer with softmax outputs

Figure 8: OEDHML FRAMEWORK

3.4.1 Naïve Bayes Classifier

Naive Bayes is a straightforward method for creating classifiers, which are models that assign class labels to problem instances represented as vectors of feature values, with the class labels drawn from a finite set. Regardless of any possible correlations between the active position, severity, and type of speakers features, a Naive Bayes Classifier (NBC) considers each of these features to contribute independently to the probability that a written-text can output emotions like anger or happiness. In a supervised learning environment, naive Bayes classifiers can be trained very efficiently for some types of probability models. In several practical applications, the method of maximum likelihood is used to estimate parameters for naive Bayes models; in other words, the naive Bayes model can be used without endorsing Bayesian probability or using any Bayesian methods.

Mathematical Understanding

The naive Bayes classification method is based on Bayes' theorem. It's called 'Naive' because it assumes that every pair of features in the data is independent. Let $(x_1, x_2, ..., x_n)$ be a feature vector, and y be the class label that corresponds to it.
Using Bayes' theorem as a guide,

$$P(y|x_1, ..., x_n) = \frac{P(y) * P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)} \tag{1}$$

Since, $(x_1, x_2, ..., x_n)$ are independent of each other,

$$P(y|x_1, ..., x_n) = \frac{P(y) * \prod_{i=1}^{n}(P(x_i|y))}{P(x_1, ..., x_n)} \tag{2}$$

Inserting proportionality by removing $P(x_1, ..., x_n)$ (because it's a constant).

$$P(y|x_1, ..., x_n) = P(y) * \prod_{i=1}^{n}(P(x_i|y)) \tag{3}$$

Therefore, the class label is decided by,

$$y = \arg\max_y [P(y) * \prod_{i=1}^{n}(P(x_i|y))] \tag{4}$$

$P(y)$ is the relative frequency of class label y in the training dataset. In the case of the Gaussian Naive Bayes classifier, $P(x_i|y)$ is calculated as,

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp{(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2})} \tag{5}$$

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a **Gaussian Distribution**. A Gaussian distribution is also called Normal distribution. When plotted, it gives a bell-shaped curve that is symmetric about the mean of the feature values as shown below:

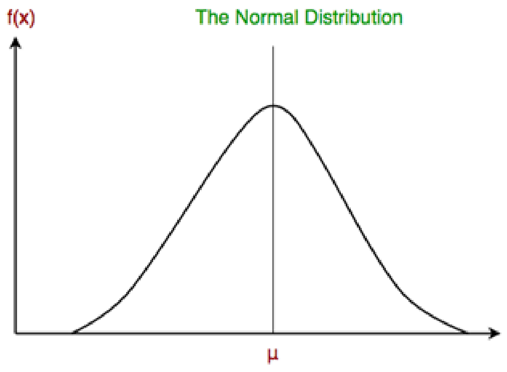


Figure 9: Naive Bayes: Normal/Gaussian Distribution Curve

3.4.2 Convolution Neural Network

A Convolutional Neural Network, also known as CNN or ConvNet, is a class of neural networks that have proven very effective in areas of Image Classification. In this proposed research work, we decided to use this advantage in the area of Text Classification. The CNN model is built by stacking the following layers, namely:

- Convolutions for 1-Dimension (Conv1D) Layer
- Max-Pooling (MaxPool1D) Layer
- Flatten Layer
- Dropout Layer
- Dense Layer

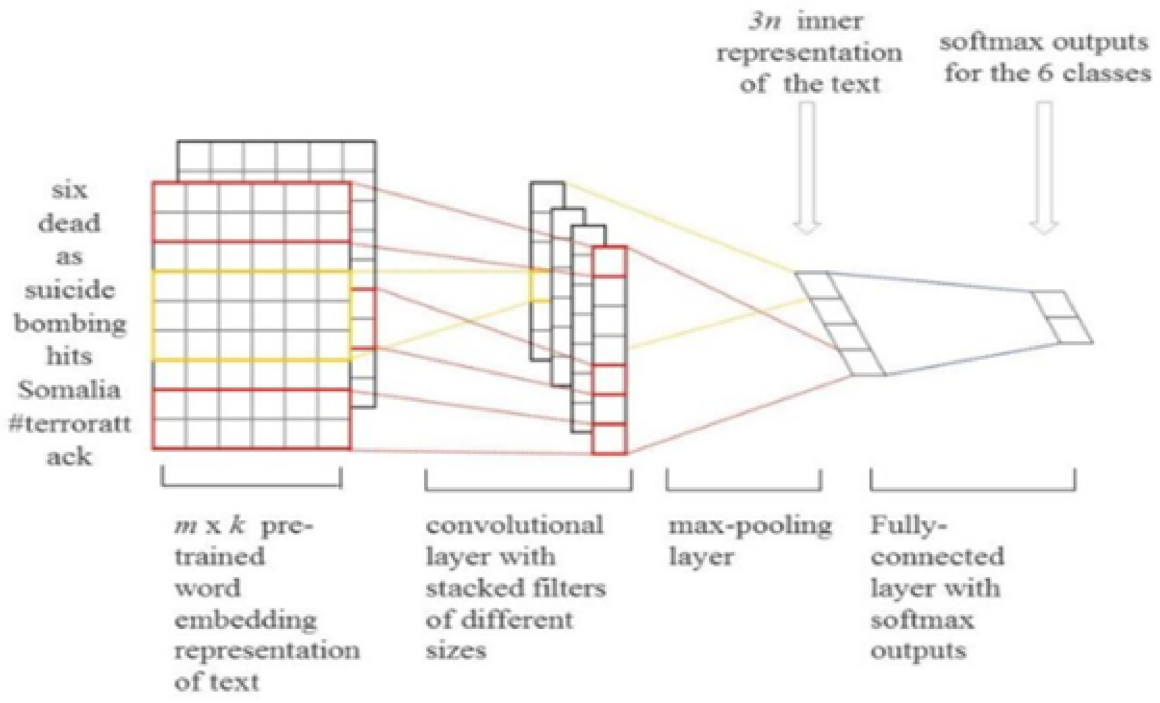

3n inner representation of the text
softmax outputs for the 6 classes
six dead as suicide bombing hits Somalia #terroratt ack
m x k pre-trained word embedding representation of text
convolutional layer with stacked filters of different sizes
max-pooling layer
Fully-connected layer with softmax outputs

Figure 10: Illustration of Convolutional Neural Networks for Text-Analysis

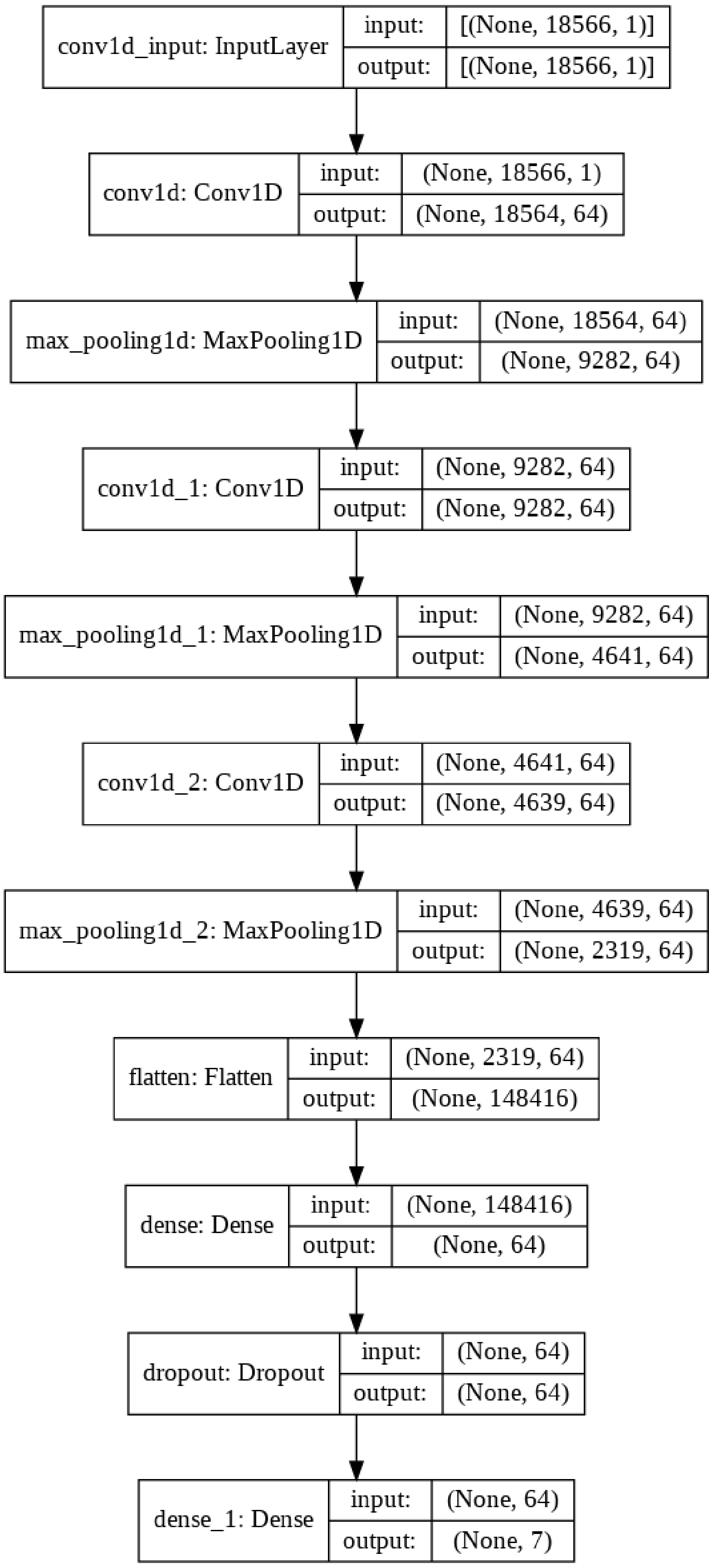


Figure 11: Proposed CNN Model Configuration

Convolutions for 1-Dimension Text Layer

This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs. If **use_bias** is True, a bias vector is created and added to the outputs. Finally, if activation is None, it is applied to the outputs as well.
When using this layer as the first layer in a model, provide an **input_shape** argument which can take a tuple of integers or None, e.g., (10,128) for sequences of 10 vectors of 128-dimensional vectors, or (None,128) for variable-length sequences of 128-dimensional vectors. In our case, we used an *input_shape* of $(None, 18566, 1)$ for sequences of **n** vectors of **18566**-dimensional vectors.
We focus on the task of text classification. We consider the common architecture in which each word in a document is represented as an embedding vector, a single convolutional layer with m filters is applied, producing an m-dimensional vector for each document n-gram. The vectors are combined using max-pooling followed by a ReLU activation. The result is then passed to a linear layer for the final classification.

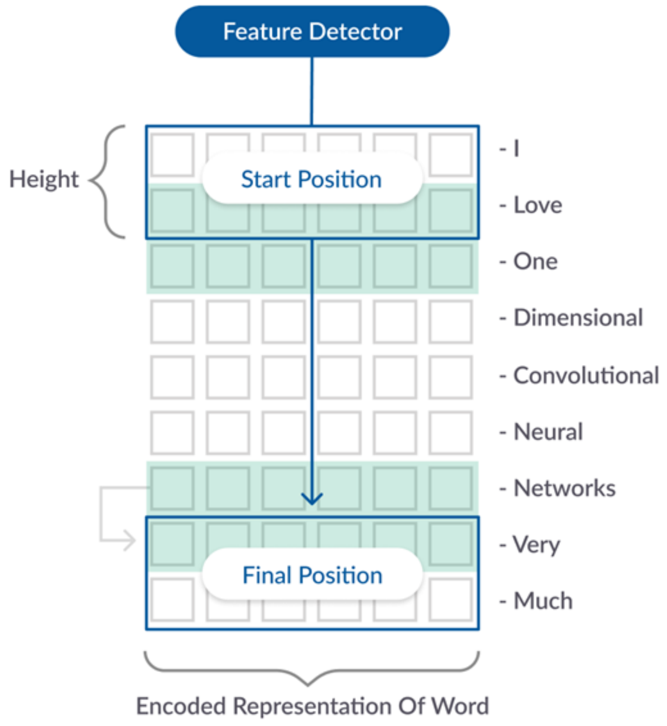


Figure 12: Illustration of 1-Dimensional Convolutions

For an n-words input text $(w_1, ..., w_n)$ we embed each symbol as d dimensional vector, resulting in word vectors $(w_1, ..., w_n) \in \mathbb{R}^d$. The resulting $d * n$ matrix is then fed into a convolutional layer where we pass a sliding window over the text. For each l-words n-gram:

$$u_i = [w_i, ..., w_{i+l-1}] \in \mathbb{R}^{d*l}; 0 \leq i \leq n - l \tag{6}$$

And for each filter $f_j \in R^{d*l}$ we calculate $< u_i, f_j >$. The convolution results in matrix $F \in R^{n\ddot{O}m}$. Applying max-pooling across the n-gram dimension results in $p \in R^m$ which is fed into ReLU non-linearity. Finally, a linear fully connected layer $W \in R^{c*m}$ produces the distribution over classification classes from which the strongest class is outputted.
Formally:

$$u_i = [w_i, ..., w_{i+l-1}] F_{ij} =< u_i, f_j > p_j = ReLU(\max_i F_{ij}) o = softmax(W_p) \tag{7}$$

In practice, we use multiple windows sizes $l \in L, L \subseteq N$ by using multiple convolution layers in parallel and concatenating the resulting p^l vectors. We note that the methods in this work are applicable for dilated convolutions as well.

Max-Pooling Layer

Pooling is a feature commonly imbibed into Convolutional Neural Network (CNN) architectures. The main idea behind a pooling layer is to “accumulate” features from maps generated by convolving a filter over an image. Formally, its function is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network. The most common form of pooling is max pooling.[1]

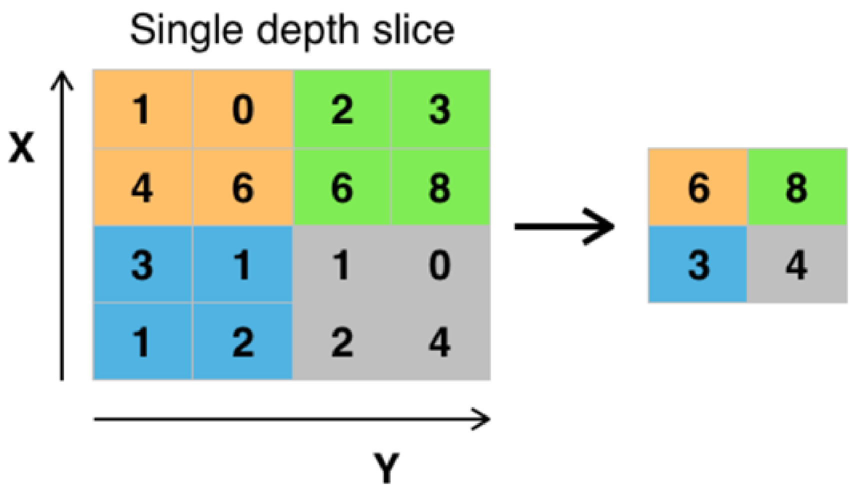


Figure 13: Max-Pooling Illustration

Advantages — Max pooling is done to help over-fitting by providing an abstracted form of the representation. As well, it reduces the computational cost by reducing the number of parameters to learn and provides basic translation invariance to the internal representation. Max pooling is done by applying a max filter to (usually) non-overlapping sub-regions of the initial representation. The other forms of pooling are average, general.

Flatten Layer

In between the convolutional layer and the fully connected layer, there is a ‘Flatten’ layer. Flattening transforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier.[15]

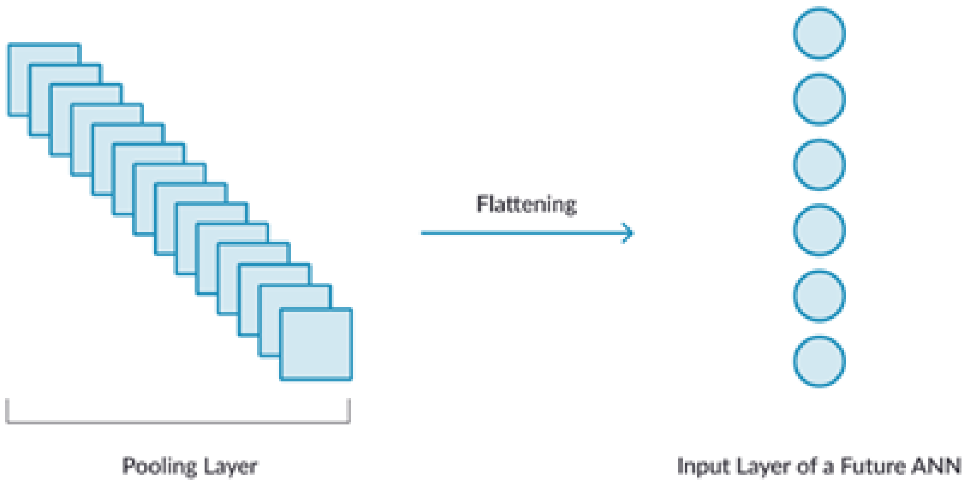


Figure 14: Flatten Layer Illustration

Dropout Layer

Dropout [20] is a technique to reduce over-fitting. Its central idea is to take an over-fitting model and train sub-models derived from it by randomly removing units for each training batch. A conceptual view of the standard dense network vs. the network structure after applying dropout is shown in the figure below.

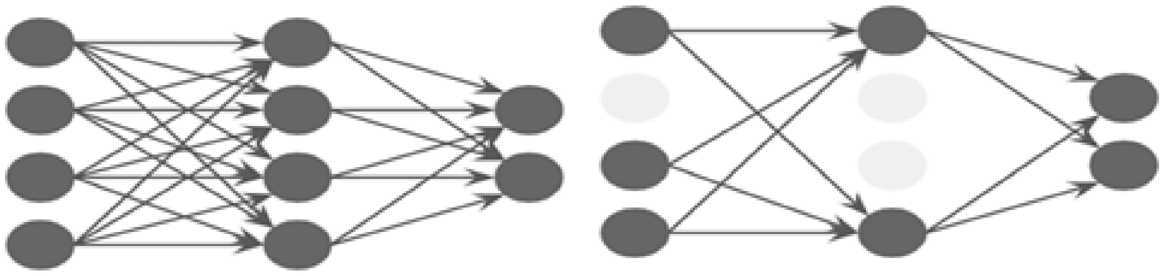


Figure 15: A conceptual view of dropout. Instead of using a fixed network structure, dropout randomly removes units from a fully connected network (left) to create a sub-model (right)

Dropout simulates a sparse activation from a given layer, which interestingly, in turn, encourages the network to learn a sparse representation as a side-effect. As such, it may be used as an alternative to activity regularization for encouraging sparse representations in auto-encoder models [20]. Because the outputs of a layer under dropout are randomly sub-sampled, it has the effect of reducing the capacity or thinning the network during training. As such, a wider network, e.g. more nodes, may be required when using dropout [20].

Dense Layer

The Dense Layer [17] suggests that layers are fully connected (dense) by the neurons in a network layer. Each neuron in a layer receives input from all the neurons present in the previous layer. In other words, the dense layer is a fully connected layer or a densely connected layer, meaning all the neurons in a layer are connected to those in the next layer.

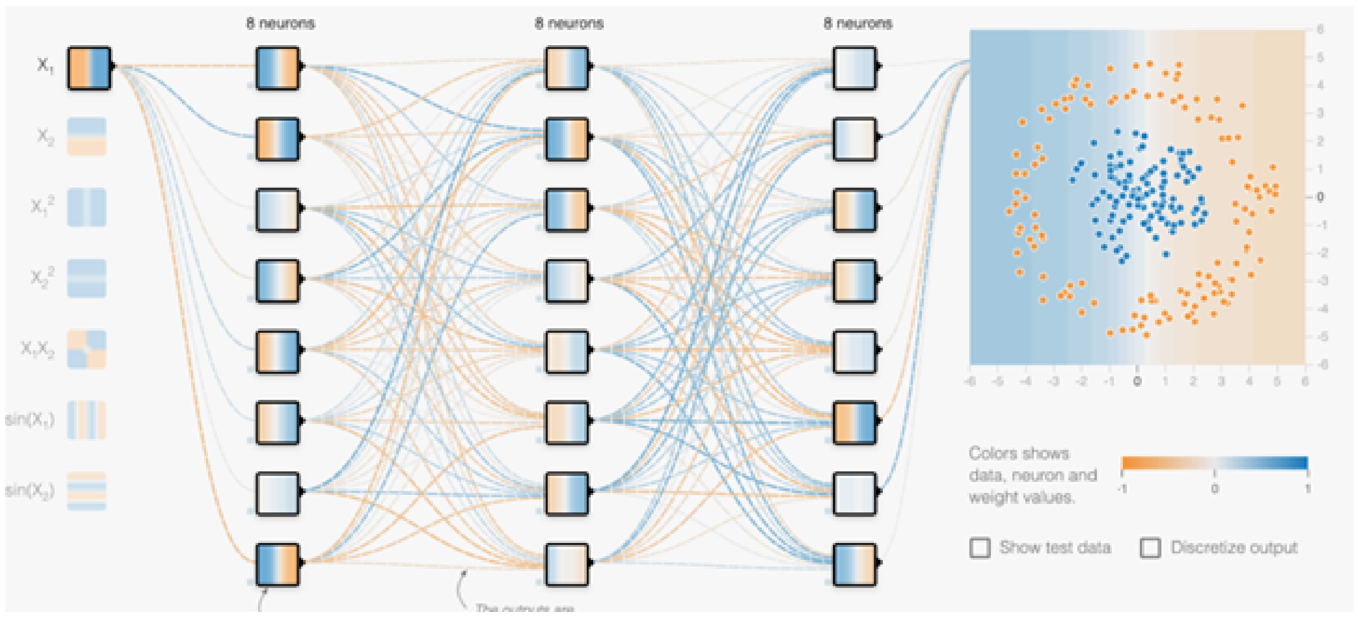


Figure 16: Dense Neural Network Representation on TensorFlow Playground

A densely connected layer provides learning features from all the combinations of the features of the previous layer, whereas a convolutional layer relies on consistent features with a small repetitive field.

4 Implementation

4.1 Prerequisites

TensorFlow Architecture works in three parts:

- Preprocessing the data
- Build the model
- Train and estimate the model

It is called TensorFlow because it takes input as a multi-dimensional array, also known as **tensors**. You can construct a sort of **flowchart** of operations (called a Graph) that you want to perform on that input. The input goes in at one end, and then it flows through this system of multiple operations and comes out at the other end as output. This is why it is called TensorFlow because the tensor goes in it flows through a list of operations, and then it comes out the other side.

What is Keras?

Keras is an Open-Source Neural Network library written in Python that runs on top of Theano or Tensorflow. It is designed to be modular, fast, and easy to use. It was developed by François Chollet, a Google engineer. Keras doesn't handle low-level computation. Instead, it uses another library to do it, called the "Backend".
Keras is a high-level API wrapper for the low-level API, capable of running on top of TensorFlow, CNTK, or Theano. Keras High-Level API handles the way we make models, defining layers, or set up multiple input-output models. In this level, Keras also compiles our model with loss and optimizer functions, training process with fit function. Keras in Python doesn't handle Low-Level API such as making the computational graph, making tensors, or other variables because it has been handled by the "backend" engine.

Sklearn for Feature Extraction such as TF-IDF Vectorizer and Naïve Bayes Models

Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms. It's built upon some of the technology you might already be familiar with, like NumPy, pandas, and Matplotlib. The functionality that scikit-learn provides include:

- Regression, including Linear and Logistic Regression
- Classification, including K-Nearest Neighbors
- Clustering, including K-Means and K-Means++
- Model selection
- Preprocessing, including Min-Max Normalization

As you move through Codecademy's Machine Learning content, you will become familiar with many of these terms. You will also see scikit-learn (in Python, sklearn) modules being used. For example: *sklearn.linear_model.LinearRegression*() is a Linear Regression model inside the $linear_model$ module of sklearn. The power of scikit-learn will greatly aid your creation of robust Machine Learning programs.

Pandas for Data-Frame Representation.

Pandas is an open-source, BSD-licensed library providing high-performance, easy-to-use data structures, and data analysis tools for the Python programming language. The Data Structures provided by Pandas are of two distinct types:

- Pandas Data-Frame, and
- Pandas Series

Pandas DataFrame is nothing but an in-memory representation of an excel sheet via Python programming language. Just like Excel, Pandas DataFrame provides various functionalities to analyze, change, and extract valuable information from the given dataset. In the real world, a Panda DataFrame will be created by loading the datasets from persistent storage, including but not limited to Excel, CSV, and MySQL databases.

Numpy for Matrix and Vectors Representations

Data is collected in many different formats from numbers to images, from categories to sound waves. However, we need the data represented with numbers to be able to analyze it on computers. Machine learning and deep learning models are data-hungry. The performance of them is highly dependent on the amount of data. Thus, we tend to collect as much data as possible to build a robust and accurate model. As the number of data increases, the operations done with scalars start to be inefficient. We need vectorizing or matrix operations to make computations efficiently. That's where **linear algebra** comes into play.
Linear algebra is one of the most important topics in the data science domain. In this post, we will cover some basic terms in linear algebra and go through examples using NumPy, a scientific computing library for Python.
There are different types of objects (or structures) in linear algebra:

- Scalar: Single number
- Vector: Array of numbers
- Matrix: a 2-dimensional array of numbers
- Tensor: N-dimensional array of numbers where $n > 2$

Scipy for Sparse Matrix Representation

Matrices that contain mostly zero values are called sparse, distinct from matrices where most of the values are non-zero, called dense. Large sparse matrices are common in general and especially in applied machine learning, such as in data that contains counts, data encodings that map categories to counts, and even in whole subfields of machine learning such as natural language processing.
It is computationally expensive to represent and work with sparse matrices as though they are dense, and much performance improvement can be achieved by using representations and operations that specifically handle the matrix sparsity.

Environment Configuration

To run our proposed implementation, we needed a PC with the following requirements:

- **System** — Windows 8/10 or Linux Distributions or latest versions.

- **PC Configuration** — 7th Gen. Core i7 CPU, RAM 16Gb, Storage (at least 10Gb), GPU Nvidia GTX 1070.
- **PC Architecture** — 64-bit OS, x64-based processor.
- **Tools** — Python 3.8.x, PyCharm IDE, Orange 3, Glue, Anaconda IDE, CuDNN and CUDA.
- **Main Libraries** — TensorFlow and TensorFlow-Text (both v.2.4.1), Keras, Sklearn and SciKit-Learn, Tensorboard, Seaborn, NLTK, Pandas, MatPlot.

4.2 Implementation Workflow

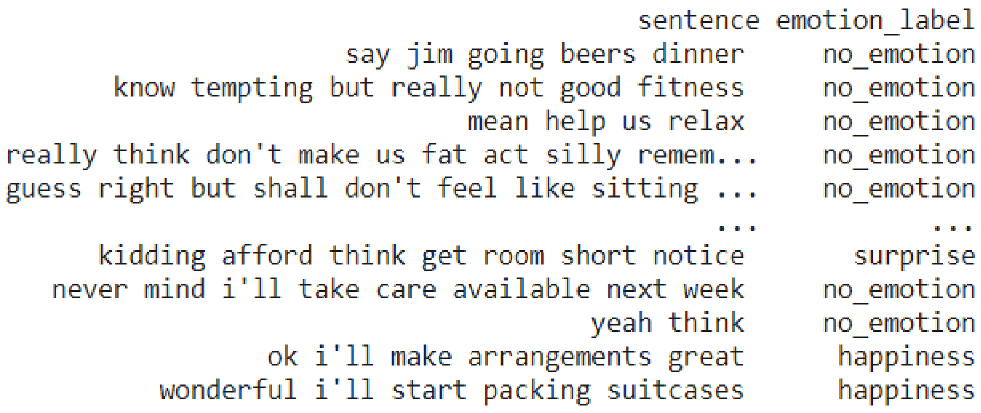
1. Building Dataset from JSON files of Daily Dialogue Data
2. Indexing Emotion Labels with Numbers
3. Vectorizing Dataset Columns [‘sentence’]
4. Padding and Splitting Vectorized Dataset in Train, Valid and Test sets
5. Building NB Classifier for training.
6. Building and Configuring CNN Model
7. Converting Train, Test, and Valid sets into a Dense Matrix
8. Categorizing Labeled data from y-train, y-test, and y-valid.
9. Fitting CNN Model with the converted train and valid sets.
10. Predicting and Evaluating the CNN Model with the test data.
11. Results Visualization

4.3 Feature Engineering and Preprocessing Mechanism

4.3.1 Feature Engineering

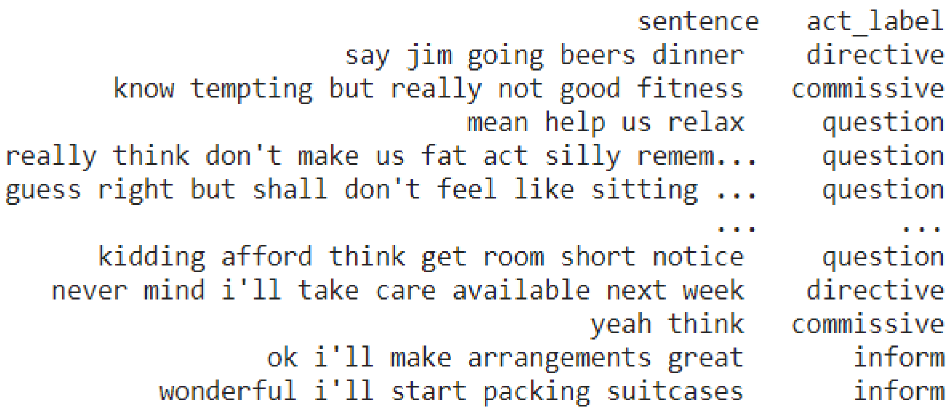
Feature engineering is the process of using domain knowledge to extract features from raw data via data mining techniques. These features can be used to improve the performance of machine learning algorithms. Feature engineering can be considered as applied machine learning itself.

Feature Extraction — It uses data reduction which allows the elimination of less important features. The collection of words obtained through tokenization is sorted to find the unique words. The unique words and the count will be stored separately for further processing.
For this research project, we selected the features *’sentence’*, *’act_label’* and *’emotion_label’* from our daily-dialog dataset shown in figures below.



```
                                        sentence emotion_label
                      say jim going beers dinner    no_emotion
         know tempting but really not good fitness    no_emotion
                              mean help us relax    no_emotion
really think don't make us fat act silly remem...    no_emotion
guess right but shall don't feel like sitting ...    no_emotion
                                             ...           ...
        kidding afford think get room short notice      surprise
      never mind i'll take care available next week    no_emotion
                                      yeah think    no_emotion
                  ok i'll make arrangements great     happiness
          wonderful i'll start packing suitcases     happiness
```

Figure 17: Dataset with selected features '**sentence**' and '**emotion_label**'



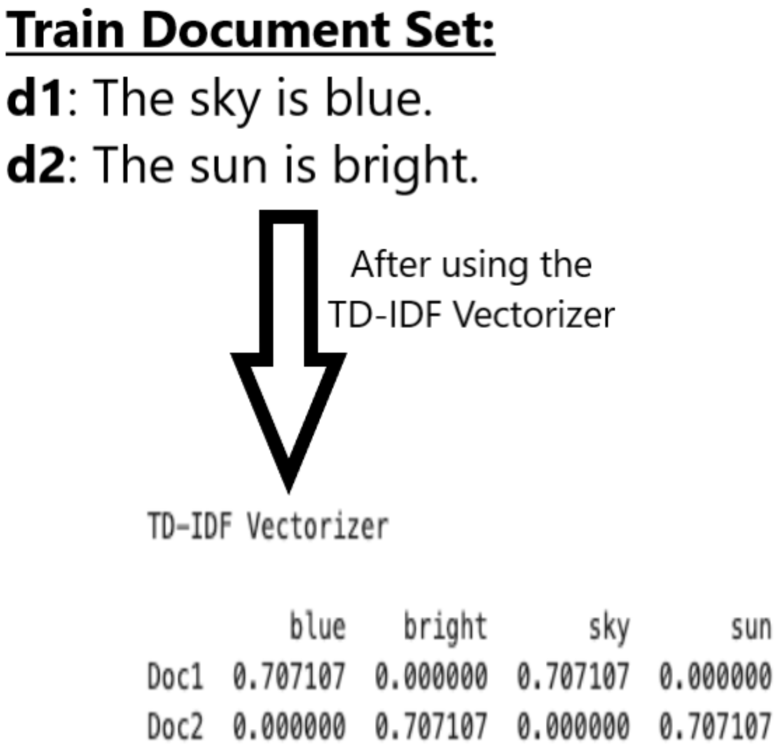
```
                                        sentence   act_label
                      say jim going beers dinner   directive
         know tempting but really not good fitness  commissive
                              mean help us relax    question
really think don't make us fat act silly remem...    question
guess right but shall don't feel like sitting ...    question
                                             ...         ...
        kidding afford think get room short notice    question
      never mind i'll take care available next week   directive
                                      yeah think  commissive
                  ok i'll make arrangements great      inform
          wonderful i'll start packing suitcases      inform
```

Figure 18: Dataset with selected features '**sentence**' and '**act_label**'

4.3.2 Preprocessing Mechanism

What is a TF-IDF Vectorizer?

TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency. This is very common algorithm to transform text into a meaningful representation of numbers which is used to fit machine algorithm for prediction. Let's take sample example and explore two different spicy sparse matrix before go into deep explanation.



Train Document Set:

d1: The sky is blue.

d2: The sun is bright.

TD-IDF Vectorizer

```
          blue    bright       sky       sun
Doc1  0.707107  0.000000  0.707107  0.000000
Doc2  0.000000  0.707107  0.000000  0.707107
```

Figure 19: Text Document converted to Spicy Sparse Matrix of TF-IDF Vectorizer.

Here, we can see clearly that TF-IDF Vectorizer consider overall documents of weight of words. A vocabulary is a dictionary that converts each token (word) to a feature index in the matrix, each unique token gets a feature index.
The TF-IDF Vectorizer converts a collection of raw documents into a matrix of TF-IDF features.

- **TF (Term Frequency)**: The number of times a word appears in a document is its Term Frequency. A higher value means a term appears more often than others, and so, the document is a good match when the term is part of the search terms.
- **IDF (Inverse Document Frequency)**: Words that occur many times in a document, but also occur many times in many others, maybe irrelevant. IDF is a measure of how significant a term is in the entire corpus.

Mathematical Understanding of TF-IDF|| TF-IDF is a measure of originality of a word by comparing the number of times a word appears in a document with the number of documents the word appears in.

$$TF\ IDF = TF(t,d) * IDF(t) \tag{8}$$

where $TF(t,d)$ is the number of times term t appears in a document d as shown:

$$TF(t,d) = \sum_{x \epsilon d} fr(x,t) \tag{9}$$

where $fr(x,t)$ is a simple function defined as:

$$fr(x,t) = \begin{cases} 1, & if \rightarrow x = t \\ 0, & otherwise \end{cases} \tag{10}$$

and **IDF**(t) is the Inverse Document Frequency which is computed as follow:

$$IDF(t) = \log(\frac{1+n}{1+DF(d,t)}) + 1 \tag{11}$$

where n is the number of documents and **DF**(d,t) is the Document Frequency of the term t.

Remark — In **TfidfVectorizer** we consider **overall document weightage** of a word. It helps us in dealing with the most frequent words. Using it we can penalize them. TfidfVectorizer weights the word counts by a measure of how often they appear in the documents.

5 Results Analysis

5.1 Comparative Analysis

In training, our proposed framework had an accuracy score of more than 84,63%.

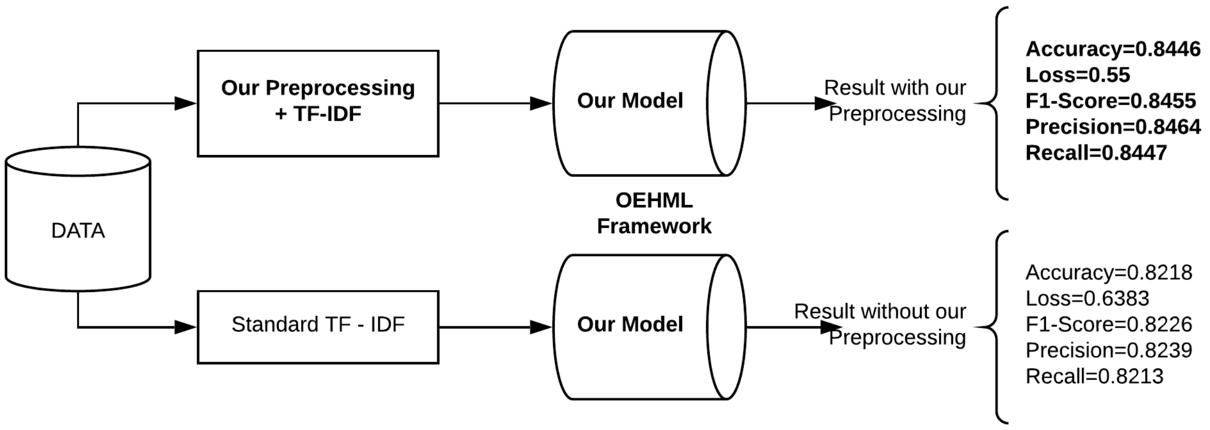


Figure 20: OEHML/OEDHML Framework Comparison with Standard Preprocessing

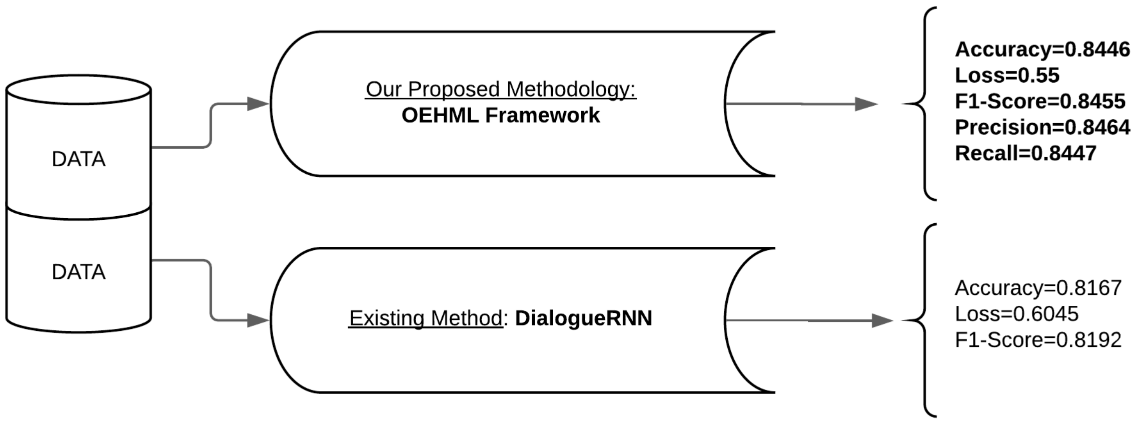
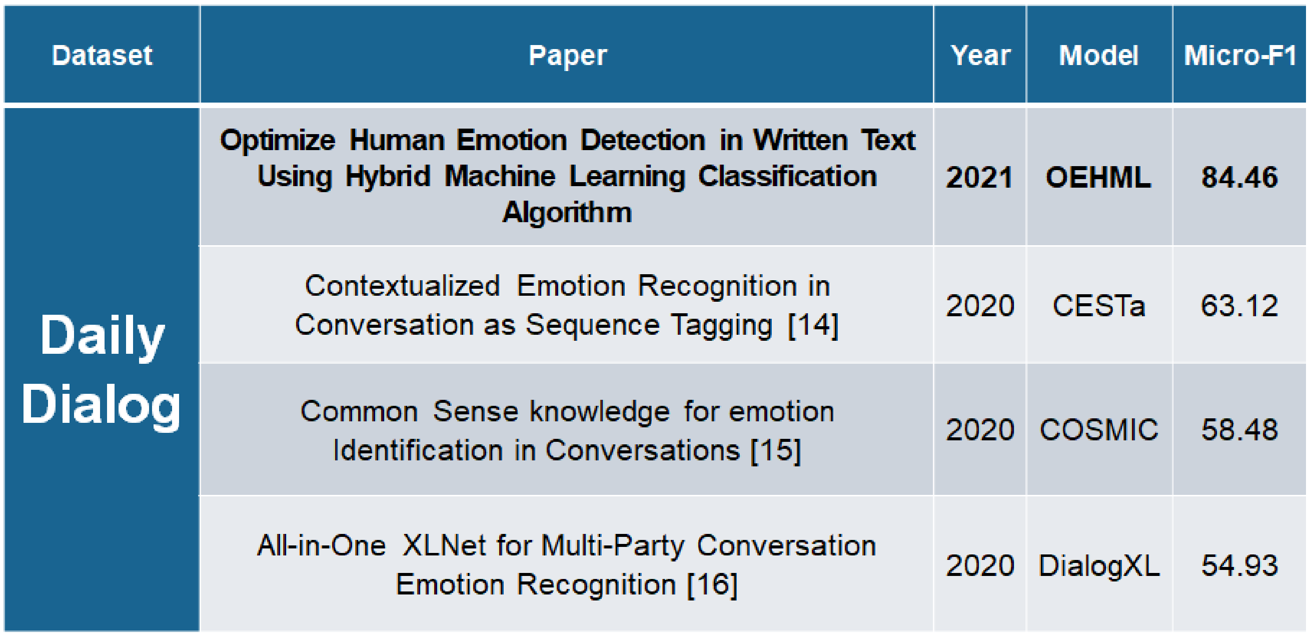


Figure 21: OEHML Framework Comparison with Existing Method (DialogueRNN)



Dataset	Paper	Year	Model	Micro-F1
Daily Dialog	**Optimize Human Emotion Detection in Written Text Using Hybrid Machine Learning Classification Algorithm**	**2021**	**OEHML**	**84.46**
	Contextualized Emotion Recognition in Conversation as Sequence Tagging [14]	2020	CESTa	63.12
	Common Sense knowledge for emotion Identification in Conversations [15]	2020	COSMIC	58.48
	All-in-One XLNet for Multi-Party Conversation Emotion Recognition [16]	2020	DialogXL	54.93

Table 2: Results: OEHML Framework vs. Other Research Papers

5.2 Metrics Evaluation

To evaluate the performance of our proposed framework for emotion detection problems in written-text, various evaluation metrics have been used. In this subsection, we review the most widely used metrics for Emotion Detection. Most existing approaches consider the emotion detection problem as a multi-classification problem that predicts whether a given text, comment, or tweet, is outputting an emotion like anger, disgust, fear, happiness, sadness, or surprise, or in some cases, no emotion.

- True Positive (**TP**)
- True Negative (**TN**)
- False Negative (**FN**
- False Positive (**FP**)

By formulating this as a multi-classification problem, we can define the following metrics,

$$Precision = \frac{|TP|}{|TP| + |FP|} \tag{12}$$

$$Recall = \frac{|TP|}{|TP| + |FN|} \tag{13}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{14}$$

$$Accuracy = \frac{|TP| + |TN|}{|TP| + |TN| + |FP| + |FN|} \tag{15}$$

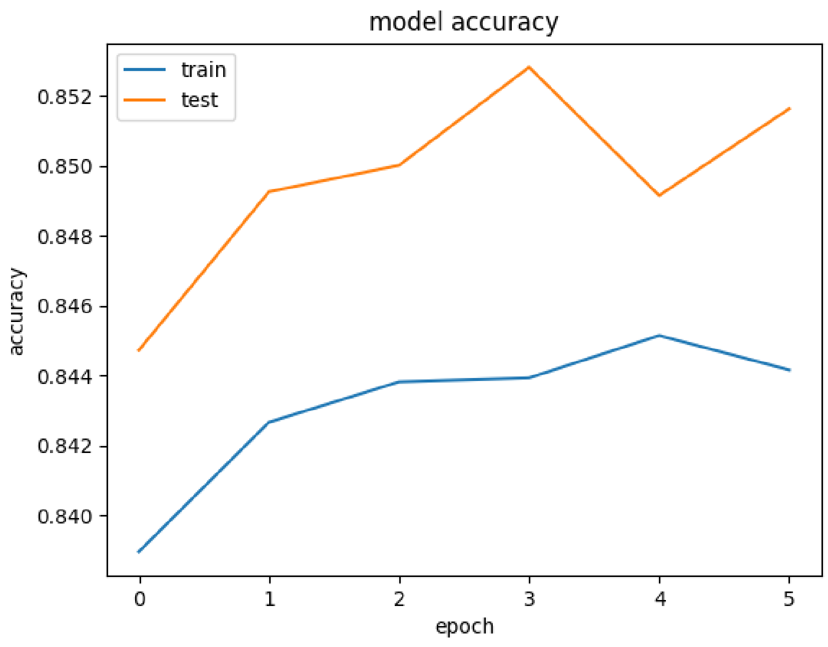


Figure 22: Model Accuracy Curve

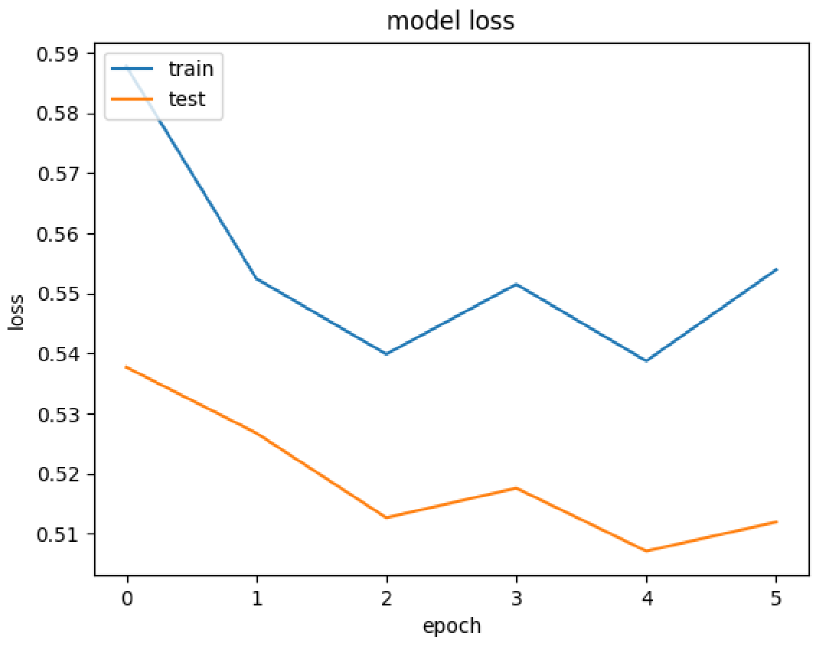


Figure 23: Model Loss Curve

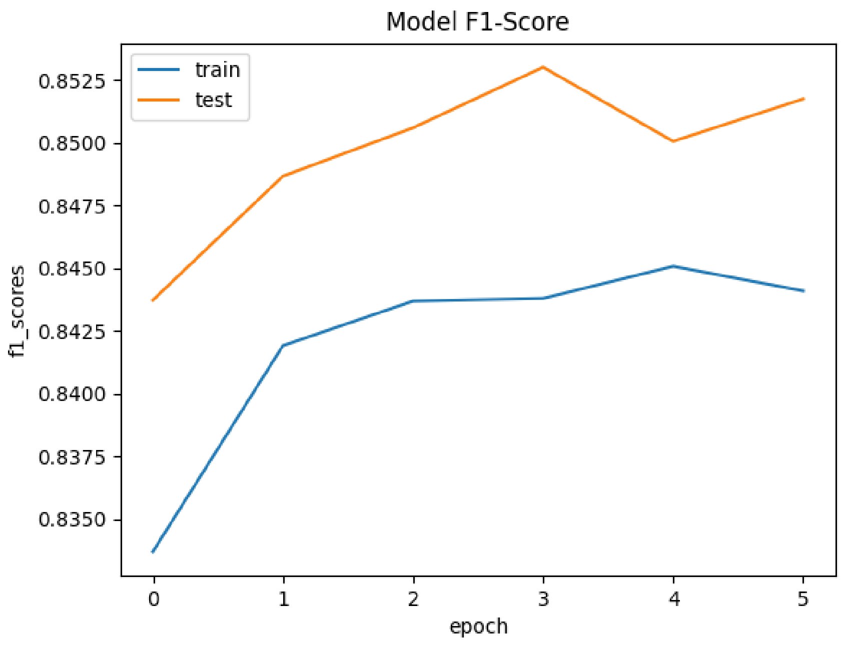


Figure 24: Model F1-Score Curve

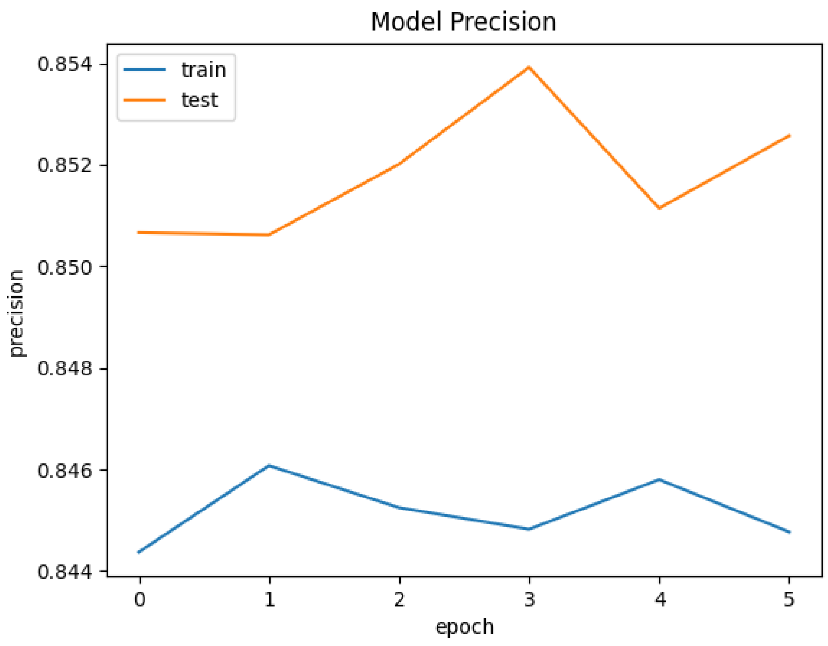


Figure 25: Model Precision Curve

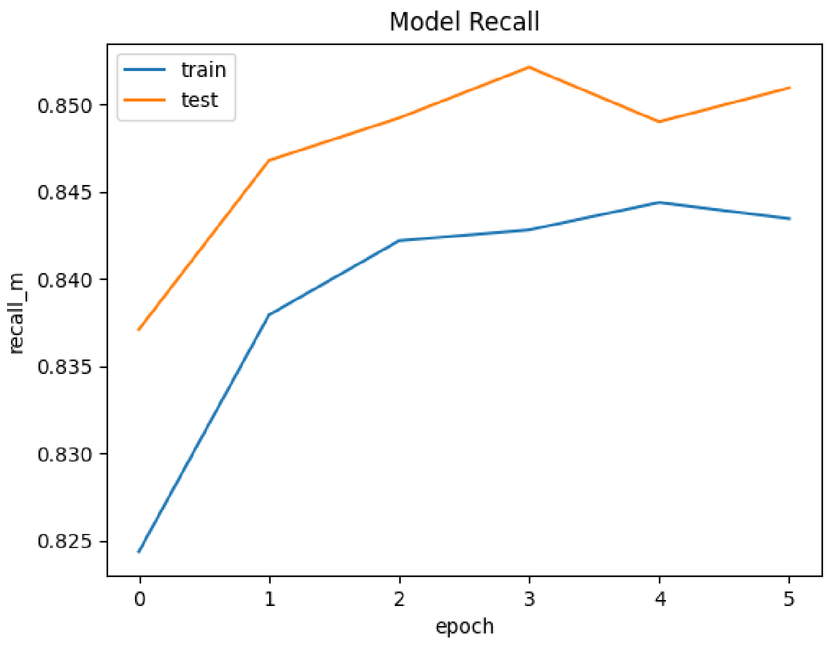


Figure 26: Model Recall Curve

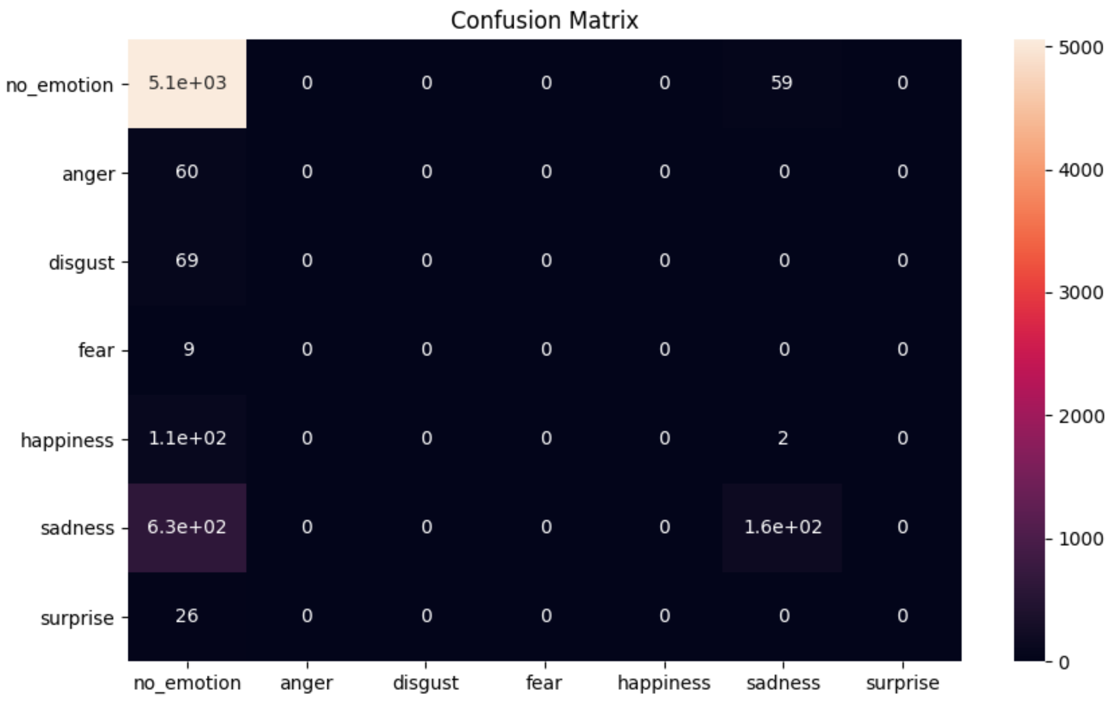


Figure 27: Resulted Confusion Matrix

.

6 Discussion

The evaluation process in this thesis research was conducted with several conditions. The accuracy obtained in the first test, which used only our model and did not use our preprocessing technique, was 82%. With our preprocessing technique applied to our model, we were able to achieve an accuracy of 84%. These findings suggest that our proposed methodology works, but we need to do further testing to be certain.

7 Conclusion & Future Work

7.1 Conclusion

Our thesis study's main goal was to develop an optimized sentiment detection system that extrapolates emotion from a given text. This research project has examined the role of social media networks in determining people's mood based on textual information they post. We developed a new preprocessing technique and a hybrid model (NB, CNN) to do the work. Our evaluation results led us to conclude, the proposed methodology has a significant improvement in the accuracy result.
The results show that by utilizing social media platforms e.g. twitter and daily dialogue, a tool can actually analyze people's mood towards a subject and inform the relevant authorities in real-time to know what the people's general mood towards a particular subject is. Making use of open python libraries and training data sets we were able to classify the general mood of people based on their emotional status.
Furthermore, this tool can be adopted by businesses, organizations and politicians who would like to have an edge and an insight into people's opinions over a giver topic.

7.2 Future Work

Through our research and our work, we face a great number of challenges that we overcame at the end of our path. We achieve our objectives by building a new preprocessing approach base on the stop words removal and we combine two different unsupervised learning algorithms into a hybrid one. All that led to obtaining an accuracy of 84.46 which is a success and better than the result obtained by the papers: Contextualized Emotion Recognition in Conversation as Sequence Tagging [23], Common Sense knowledge for emotion Identification in Conversations [7] and All-in-One XLNet for Multi-Party Conversation Emotion Recognition [18] which registered respectively an accuracy of $63.3, 58.48 and 54.93$. These results were obtained after processing only a total of 6 epochs due to the huge processing time. For our future works, we intend to compare these different works after running a minimum of 100 epochs which will consolidate, approve and certify our outstanding result. For other future works, we also intend to apply our methodology to many different data sets to see how it will act and perform.

References

[1] Deep AI. *What is Max-Pooling.* www.DeepAI.org, 2020.

[2] Hani Al-Omari, Malak A Abdullah, and Samira Shaikh. Emodet2: Emotion detection in english textual dialogue using bert and bilstm models. In *2020 11th International Conference on Information and Communication Systems (ICICS)*, pages 226–232. IEEE, 2020.

[3] Samar Al-Saqqa, Heba Abdel-Nabi, and Arafat Awajan. A survey of textual emotion detection. In *2018 8th International Conference on Computer Science and Information Technology (CSIT)*, pages 136–142. IEEE, 2018.

[4] Guru99 from www.Guru99.com. Unsupervised machine learning: What is, algorithms, example, 2021.

[5] GCA. *Fine-Grained Access Control.* GCA Headquarters, 2020.

[6] S Geetha and Kaliappan Vishnu Kumar. Tweet analysis based on distinct opinion of social media users'. In *Advances in Big Data and Cloud Computing*, pages 251–261. Springer, 2019.

[7] Deepanway Ghosal, Navonil Majumder, Alexander Gelbukh, Rada Mihalcea, and Soujanya Poria. Cosmic: Commonsense knowledge for emotion identification in conversations, 2020.

[8] Jiawei Han, Micheline Kamber, and Jian Pei. 1 - introduction. In Jiawei Han, Micheline Kamber, and Jian Pei, editors, *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 1–38. Morgan Kaufmann, Boston, third edition edition, 2012.

[9] Maruf Hassan, Md Sakib Bin Alam, and Tanveer Ahsan. Emotion detection from text using skip-thought vectors. In *2018 International Conference on Innovations in Science, Engineering and Technology (ICISET)*, pages 501–506. IEEE, 2018.

[10] J. Izza. *Infografis Penetrasi dan Perilaku Pengguna Internet.* Indonesia Tahun, 2017.

[11] K. Oatley J. M. Jenkins and D. Keltner. *Understanding Emotions.* John Wiley Sons, 2013.

[12] Erfian Junianto and Rizal Rachman. Implementation of text mining model to emotions detection on social media comments using particle swarm optimization and naive bayes classifier. In *2019 7th International Conference on Cyber and IT Service Management (CITSM)*, volume 7, pages 1–6. IEEE, 2019.

[13] Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. DailyDialog: A manually labelled multi-turn dialogue dataset. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 986–995, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing.

[14] Sonia Xylina Mashal and Kavita Asnani. Emotion intensity detection for social media data. In *2017 International Conference on Computing Methodologies and Communication (ICCMC)*, pages 155–158. IEEE, 2017.

[15] Machine Learning Mastery. Using the keras flatten operation in cnn models with code examples, 2020.

[16] Fopa Yuffon Amadou Olabi and al. *OEHML Framework For Emotion Detection.* @IUT-Thesis-Cmr_Team_237, 2021.

[17] Mohammed Rampurawala. *Classification with TensorFlow and Dense Neural Networks.* HEARTBEAT.FRITZ.AI, February 8, 2019.

[18] Weizhou Shen, Junqing Chen, Xiaojun Quan, and Zhixian Xie. Dialogxl: All-in-one xlnet for multi-party conversation emotion recognition, 2020.

[19] Praveen Singh, Neeharika Pisipati, Pisipati Radha Krishna, and Munaga VNK Prasad. Social signal processing for evaluating conversations using emotion analysis and sentiment detection. In *2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP)*, pages 1–5. IEEE, 2019.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[21] Yahya M Tashtoush and Dana Abed Al Aziz Orabi. Tweets emotion prediction by using fuzzy logic system. In *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 83–90. IEEE, 2019.

[22] Knowledge Transfer. *How to read CSV file data using TensorFlow DataSet API.* www.AndroidKT.com, February 1, 2020.

[23] Yan Wang, Jiayu Zhang, Jun Ma, Shaojun Wang, and Jing Xiao. Contextualized emotion recognition in conversation as sequence tagging. In *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 186–195, 1st virtual meeting, July 2020. Association for Computational Linguistics.

8 Appendix: Code Snippets and Result Reports

In the following pages, a snapshot of the entire source code and its relevant results obtained while running this research project, which can be found under the GitHub Repository entitled ***OEHML Framework For Emotion Detection***[16]. It contains a detailed implementation of the proposed solution that tackles the stated problem illustrated in the above sections. It was written in Python using **PyCharm IDE**,and **JupyterLab IDE**. The Data Analysis part was done using **Glue** and **Orange 3** from **Anaconda IDE**.
GitHub Repository Link —
`https://github.com/IUT-Thesis-Group-Cmr/OEHML-Framework-For-Emotion-Detection.git`

8.1 Main Framework

File - D:\amado\ProjectsHub\pythonProject\main.py

```python
import pandas
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from tensorflow.python.keras.utils.np_utils import to_categorical
from methodology.models.cnn import create_cnn_model
from methodology.models.nbc import create_nbc_model
from methodology.preprocessing.StopWordsFilter import StopWordsFilter
from methodology.preprocessing.TrainTestSplit import TrainTestSplit
from methodology.preprocessing.vectorization import TextVectorizer
from tensorflow.python.keras.utils.vis_utils import plot_model


if __name__ == '__main__':
    df = pandas.read_csv('methodology/data/dailydialog/dialog-with-columnNames.csv')
    # df = pandas.read_csv('methodology/data/dailydialog/dailydialog.csv')
    # print(df[['sentence']])
    print('Dataset', df.shape, df.columns.values, sep='\n',
          end='\n========================================================\n')

    swf = StopWordsFilter()
    print(len(swf.get_list()))
    swf.G_PATH = 'methodology/preprocessing/'
    swf.load_filter()
    swf.filter()

    # tv = TextVectorizer()
    tv = TextVectorizer(stop_words=swf.get_list())
    tv.vectorization(df['sentence'].values.astype('U'))
    print('Vectorized Dataset', type(tv.tfidf_x_train
), tv.tfidf_x_train.shape, sep='\n',
          end='\n========================================================\n')
    print(len(tv.tfidf_vectorizer.get_feature_names()))
    print(tv.tfidf_vectorizer.get_feature_names())
    print(tv.tfidf_x_train.shape)
    print(tv.tfidf_x_train)
    """
        Vectorization Completed without StopWords.
    """

    tts_train = TrainTestSplit(tv.tfidf_x_train, df['emotion_index'],
                               test_set_ratio=0.15)

    x_train, x_valid_test, y_train, y_valid_test = tts_train.split()

    x_valid, x_test, y_valid, y_test = TrainTestSplit(x_valid_test, y_valid_test, test_set_ratio=0.4).split()

    print('Training', x_train.shape, y_train.shape, sep='\n',
          end='\n========================================================\n')
    print('Validation', x_valid.shape, y_valid.shape, sep='\n',
          end='\n========================================================\n')
    print('Testing', x_test.shape, y_test.shape, sep='\n',
          end='\n========================================================\n')

    print('\t\tNAIVE BAYES CLASSIFIER')

    nbc = create_nbc_model(type_='multi')
    nbc.fit(x_train, y_train)
    nbc.fit(x_valid, y_valid)
    y_predict = nbc.predict(x_test)
    # conf_mat = confusion_matrix(y_predict, y_test)
    conf_mat = confusion_matrix(y_test, y_predict)
    print(conf_mat)
    print('accuracy::', round(accuracy_score(y_test
, y_predict) * 100, 2), '%')
    # print(classification_report(y_test, y_predict
, labels=list(set(df['emotion_label']))))
    print(classification_report(y_test, y_predict,
labels=[0, 1, 2, 3, 4, 5, 6]))
    #
    print(
'======================================================
=====')
    print('\t\tCONVOLUTION NEURAL NETWORK')
    # cnn = create_nn_model(input_dim=x_train.shape[
1])

    cnn = create_cnn_model(input_shape=(x_train.
shape[1], 1))

    plot_model(cnn, to_file='model_plot.png',
show_shapes=True, show_layer_names=True)

    x_train = x_train.todense()
    x_valid = x_valid.todense()
    x_test = x_test.todense()

    x_train = np.array(x_train)
    x_valid = np.array(x_valid)
    x_test = np.array(x_test)

    x_train = x_train.reshape((x_train.shape[0],
x_train.shape[1], 1))
    x_valid = x_valid.reshape((x_valid.shape[0],
x_valid.shape[1], 1))
    x_test = x_test.reshape((x_test.shape[0], x_test
.shape[1], 1))

    y_train = to_categorical(y_train, 7)
    y_valid = to_categorical(y_valid, 7)
    y_test = to_categorical(y_test, 7)

```

```python
    print(x_train.shape, x_train.ndim)
    print(x_valid.shape, x_valid.ndim)
    print(x_test.shape, x_test.ndim)

    history = cnn.fit(x_train, y_train, epochs=16, validation_data=(x_valid, y_valid))

    for key in history.history.keys():
        print(key, '->', history.history[key])

    cnn.save('methodology/models/pretrained_neural_network/cnn_trained_model.h5')

    # cnn.fit(x_train, y_train, epochs=20, verbose=True, batch_size=1,
    #          validation_data=(x_valid, y_valid))
    y_predict_prob = cnn.predict(x_test)

    test_eval = cnn.evaluate(x_test, y_test, verbose=0)
    print('Evaluation Metrics ->', test_eval)

    # Visualization of confusion-matrix with a heat-map
    conf_mat = confusion_matrix(y_test.argmax(axis=1), y_predict_prob.argmax(axis=1))
    print(conf_mat)
    fig = plt.figure(figsize=(10, 6))
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')

    sns.heatmap(conf_mat, annot=True,
                xticklabels=['no_emotion', 'anger', 'disgust', 'fear', 'happiness', 'sadness', 'surprise'],
                yticklabels=['no_emotion', 'anger', 'disgust', 'fear', 'happiness', 'sadness', 'surprise'])
    plt.show()

    print(classification_report(y_test.argmax(axis=1), y_predict_prob.argmax(axis=1),
                                labels=[0, 1, 2, 3,
4, 5, 6]))

    print('\n\nPredicted Values\n', y_predict_prob)

    print(history.history.keys())
    # summarize history for accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='upper left')
    plt.show()

    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='upper left')
    plt.show()

    # summarize history for f1-score
    plt.plot(history.history['f1_m'])
    plt.plot(history.history['val_f1_m'])
    plt.title('Model F1-Score')
    plt.ylabel('f1_scores')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='upper left')
    plt.show()

    # summarize history for precision
    plt.plot(history.history['precision_m'])
    plt.plot(history.history['val_precision_m'])
    plt.title('Model Precision')
    plt.ylabel('precision')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='upper left')
    plt.show()

    # summarize history for recall
    plt.plot(history.history['recall_m'])
    plt.plot(history.history['val_recall_m'])
    plt.title('Model Recall')
    plt.ylabel('recall_m')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='upper left')
    plt.show()

```

8.2 Testing Framework

File - D:\amado\ProjectsHub\pythonProject\test_env.py

```python
import pandas
import matplotlib.pyplot as plt
import tensorflow_text as tf_text

from methodology.preprocessing.vectorization import *
from methodology.models.metrics import *

df = pandas.read_csv('methodology/data/dailydialog/dailydialog.csv')
print('Dataset', df.shape, df.columns.values, sep='\n',
      end='\n========================================================\n')

columns_names = df.columns.values
features_names = columns_names[:-1]
label_name = columns_names[-1]

df = tf.data.experimental.CsvDataset('methodology/data/dailydialog/dialog-without-columnNames.csv', columns_names)
# df = tf.keras.utils.get_file(fname='methodology/data/dailydialog/dailydialog.csv')
#
# df = tf.contrib.data.make_csv_dataset(df,batch_size,column_names=columns_names,label_name=label_name,num_epochs=1)

print(type(df))
print(df)
# for row in df.as_numpy_iterator():
#     print(row[-1])
tok_df = df.map(tokenize)
tok_df = configure_dataset(tok_df)
vocab_dict = collections.defaultdict(lambda: 0)
# print(vocab_dict)
print('Loading Vocabulary...')
for item in tok_df.as_numpy_iterator():
    # print(item[1])
    for token in item[1]:
        vocab_dict[token] += 1
print('Vocabulary loaded...')

```

Page 1 of 7

```python
vocab = sorted(vocab_dict.items(), key=lambda x: x[1], reverse=True)
vocab = [token for token, count in vocab]
vocab_size = len(vocab)
print("Vocab size: ", vocab_size)
print("First five vocab entries:", vocab[:10])

keys = vocab
values = range(2, len(vocab) + 2)  # reserve 0 for padding, 1 for OOV

init = tf.lookup.KeyValueTensorInitializer(
    keys, values, key_dtype=tf.string, value_dtype=tf.int64)

num_oov_buckets = 1
vocab_table = tf.lookup.StaticVocabularyTable(init, num_oov_buckets)
print(vocab_table)

# index, sentences, act_label, emotion_label, speaker, conv_id, utt_id, labels = next(iter(df))
_, sentences, _, _, _, _, _, labels = next(df.as_numpy_iterator())
print(sentences)
print(labels)

vxt, em = preprocess_text(sentences, vocab_table, labels)
print(vxt.numpy(), em)


def preprocess_punctuation(x):
    for punctuation in '"\'!&?.,}-/<>#$%\()*+:;=?@[\\]^_`|\~':
        x = x.replace(punctuation, ' ')

    x = ' '.join(x.split())
    x = x.lower()

    return x


def preprocess_text_test(num, text, act_label,
emotion_label, speaker, conv_id, utt_id,
emotion_index):
    tokenizer = tf_text.UnicodeScriptTokenizer()
    standardized = tf_text.case_fold_utf8(text)
    tokenized = tokenizer.tokenize(standardized)
    vectorized = vocab_table.lookup(tokenized)
    emotion_index = tf.strings.to_number(
emotion_index, out_type=tf.int64)
    return vectorized, emotion_index


encoded_df = df.map(preprocess_text_test)
# count = 0
# for row in encoded_df.as_numpy_iterator():
#     count += 1
# print(count)

# for row in encoded_df.take(10):
#     print(row[1])

"""Splitting Dataset into train and test"""
VALID_SIZE = 9268 + 6179
BATCH_SIZE = 32
TRAIN_SIZE = 87532

train_data = encoded_df.skip(VALID_SIZE)
valid_data = encoded_df.take(VALID_SIZE)

valid_data = valid_data.skip(6179)
test_data = valid_data.take(6179)
print('Dataset Split Successfully')

vocab_size += 2
max_len = 0
for row in encoded_df.as_numpy_iterator():
    if max_len < len(row[0]):
        max_len = len(row[0])
print(max_len)
print('Got MAX-LEN....')


def padding_data(inputs, label):
    pad_size = abs(inputs.get_shape()[1] - max_len)
    inputs = inputs.padded_batch(pad_size)
```

```python

    # print(inputs)
    # print(inputs.get_shape(), type(inputs))
    # inputs = inputs.eval(session=tf.compat.v1.Session())
    # # inputs = tf.constant(inputs.numpy())
    # # inputs = tf.make_tensor_proto(inputs)
    # inputs = tf.make_ndarray(inputs)
    # if len(inputs) < max_len:
    #     arr = numpy.zeros(max_len - len(inputs), dtype=int)
    #     inputs = numpy.append(inputs, arr)
    return inputs, label


print(train_data)
print(valid_data)
print(test_data)

print('Padding Data...')
# train_data = train_data.map(padding_data)
# valid_data = valid_data.map(padding_data)
# test_data = test_data.map(padding_data)

# train_data = train_data.padded_batch(1, padded_shapes=max_len)
# valid_data = valid_data.padded_batch(1, padded_shapes=max_len)
# test_data = test_data.padded_batch(1, padded_shapes=max_len)

train_data = train_data.padded_batch(BATCH_SIZE)
valid_data = valid_data.padded_batch(BATCH_SIZE)
test_data = test_data.padded_batch(BATCH_SIZE)

train_data = configure_dataset(train_data)
valid_data = configure_dataset(valid_data)
test_data = configure_dataset(test_data)

count = 0
for row in train_data.as_numpy_iterator():
    # print(row[0].shape)
    count += 1
print(count)
```

```python
count = 0
for row in valid_data.as_numpy_iterator():
    count += 1
print(count)
count = 0
for row in test_data.as_numpy_iterator():
    count += 1
print(count)


def create_model(vb_size, num_labels):
    model_ = tf.keras.Sequential([
        tf.keras.layers.Embedding(vb_size, 64, mask_zero=True),
        tf.keras.layers.Conv1D(64, 5, padding='valid', activation='relu', strides=2),
        tf.keras.layers.GlobalMaxPool1D(),
        tf.keras.layers.Dense(num_labels)
    ])
    return model_


model = create_model(vocab_size, 7)
model.summary()
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy', f1_m, precision_m, recall_m])

print(type(train_data))
print(train_data)

# train_data = to_categorical(train_data, 7)
history = model.fit(train_data, validation_data=valid_data, epochs=10)
y_predict_prob = model.predict(test_data)
# y_predict_classes = model.predict_classes(test_data)
# loss, acc = model.evaluate(test_data)
mtr = model.evaluate(test_data)
# print(y_predict[0])
print(y_predict_prob)
```

```python
# print(y_predict_classes)
print("\nLoss: ", mtr[0])
print("Accuracy: {:2.2%}".format(mtr[1]))
print(mtr)

print(history.history['accuracy'])
# print(history.history['accuracy'])
for row, y in zip(test_data, y_predict_prob):
    print(row[-1], round(max(y)), sep='\t-> ')

# score_f1_micro = f1_score(test_data[1],
y_predict_prob, average='micro')
# score_f1_macro = f1_score(test_data[1],
y_predict_prob, average='macro')
#
# print(score_f1_micro, score_f1_macro)
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['f1_m'])
plt.plot(history.history['val_f1_m'])
plt.title('Model F1-Score')
plt.ylabel('f1_scores')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

```

8.3 Daily Dialogue Dataset API

File - D:\amado\ProjectsHub\pythonProject\methodology\data\dailydialog.py

```python
import pandas
import pandas as pd
import os
import tensorflow as tf

from methodology.preprocessing.StopWordsFilter import StopWordsFilter

swf = StopWordsFilter()


def preprocess_text(x):
    for punctuation in '"!&?.,}-/<>#$%\()*+:;=?@[\\]^_`|\~':
        x = x.replace(punctuation, ' ')

    x = ' '.join(x.split())
    x = x.lower()

    return remove_stopwords(x)
    # return x


def remove_stopwords(x):
    swf.G_PATH = '../preprocessing/'
    swf.load_filter()
    swf.filter()
    x = swf.clean_text(x)
    return x


def create_utterances(filename, split):
    sentences, act_labels, emotion_labels, speakers, conv_id, utt_id = [], [], [], [], [], []

    # lengths = []
    with open(filename, 'r') as f:
        for c_id, line in enumerate(f):
            s = eval(line)
            for u_id, item in enumerate(s['dialogue']):
                sentences.append(item['text'])
                act_labels.append(item['act'])
                emotion_labels.append(item['emotion'
```

Page 1 of 4

```python
])
                conv_id.append(split[:2] + '_c' + str(c_id))
                utt_id.append(split[:2] + '_c' + str(c_id) + '_u' + str(u_id))
                speakers.append(str(u_id % 2))
                # u_id += 1
    data = pd.DataFrame(sentences, columns=['sentence'])
    data['sentence'] = data['sentence'].apply(lambda x: preprocess_text(x))
    data['act_label'] = act_labels
    data['emotion_label'] = emotion_labels
    data['speaker'] = speakers
    data['conv_id'] = conv_id
    data['utt_id'] = utt_id
    return data


if __name__ == '__main__':
    path = 'dailydialog'
    print(os.listdir(path))
    train = create_utterances(path + '/train.json', 'train')
    valid = create_utterances(path + '/valid.json', 'valid')
    test = create_utterances(path + '/test.json', 'test')
    print(train.shape, train.columns.values)
    print(valid.shape, valid.columns.values)
    print(test.shape, test.columns.values)

    print(set(train['emotion_label']))
    print(set(valid['emotion_label']))
    print(set(test['emotion_label']))

    ext = '.csv'
    print(os.listdir(path))
    print(train.columns.values)
    print(train.shape)
    print(valid.shape)
    print(test.shape)

    # df = pandas.DataFrame(columns=train.columns.
values)
    df = pandas.concat([train, valid, test])
    print(df.shape)
    print(set(df['emotion_label']))
    # print(set(df_train['emotion_label']))
    emot_index = pandas.DataFrame(columns=['emotion_index'])
    count = 0
    for label in df.itertuples():
        # print(getattr(label, 'emotion_label'))
        if getattr(label, 'emotion_label') == 'no_emotion':
            emot_index.loc[count] = tf.dtypes.cast([0], dtype=tf.int64)
        elif getattr(label, 'emotion_label') == 'anger':
            emot_index.loc[count] = tf.dtypes.cast([1], dtype=tf.int64)
        elif getattr(label, 'emotion_label') == 'sadness':
            emot_index.loc[count] = tf.dtypes.cast([2], dtype=tf.int64)
        elif getattr(label, 'emotion_label') == 'fear':
            emot_index.loc[count] = tf.dtypes.cast([3], dtype=tf.int64)
        elif getattr(label, 'emotion_label') == 'surprise':
            emot_index.loc[count] = tf.dtypes.cast([4], dtype=tf.int64)
        elif getattr(label, 'emotion_label') == 'happiness':
            emot_index.loc[count] = tf.dtypes.cast([5], dtype=tf.int64)
        elif getattr(label, 'emotion_label') == 'disgust':
            emot_index.loc[count] = tf.dtypes.cast([6], dtype=tf.int64)
        count += 1

    df['emotion_index'] = emot_index['emotion_index']

    df.to_csv(path + '/' + 'dailydialog' + ext)
```

8.4 Preprocessing Frameworks: Stop Words Removal

File - D:\amado\ProjectsHub\pythonProject\methodology\preprocessing\StopWordsFilter.py

```python
from typing import Set

import nltk
from nltk.corpus import stopwords


class StopWordsFilter:
    stop_words_list: Set[str]
    G_PATH: str

    def __init__(self, lang='english'):
        self.exclude_words = list()
        nltk.download('stopwords')
        self.stop_words_list = set(stopwords.words(lang))
        self.G_PATH = ''
        pass

    def load_stopwords(self):
        with open(self.G_PATH + 'stop_words_english.txt', "r", encoding='utf-8') as f:
            lines = f.readlines()
        lines = [line.replace('\n', '') for line in lines]
        self.stop_words_list = set(lines)
        pass

    def load_filter(self):
        with open(self.G_PATH + 'sw_important.txt', "r", encoding='utf-8') as f:
            tokens = f.readlines()
            tokens = [tok.replace('\n', '') for tok in tokens]
        self.exclude_words.clear()
        for token in tokens:
            self.exclude_words.append(token)
        pass

    def filter(self):
        for word in self.exclude_words:
            self.stop_words_list.discard(word)
        pass

    def filter_from_tokens(self, remove_list,
add_list):
        if remove_list is None:
            pass
        else:
            if type(remove_list) is list:
                for elt in remove_list:
                    self.stop_words_list.remove(elt)
            elif type(remove_list) is str:
                self.stop_words_list.remove(
remove_list)
        if add_list is None:
            pass
        else:
            if type(add_list) is list:
                for elt in add_list:
                    self.stop_words_list.add(elt)
            elif type(add_list) is str:
                self.stop_words_list.add(add_list)

    def get_list(self):
        return list(self.stop_words_list)

    def clean_text(self, text):
        text = text.split()
        sentence = ''
        for token in text:
            # print(token)
            if token in self.get_list():
                pass
            else:
                sentence += token
                sentence += ' '
        return sentence


if __name__ == '__main__':
    tv = StopWordsFilter()
    # tv.load_stopwords()
    tv.load_filter()
    tv.filter()
    print(len(tv.get_list()))
    print(tv.get_list())
    print("i'm" in tv.get_list())
    print(tv.clean_text("i'm in the mall"))
```

8.5 Preprocessing Frameworks: Text Vectorization

File - D:\amado\ProjectsHub\pythonProject\methodology\preprocessing\vectorization.py

```python
import collections
from time import time

# from pandas import DataFrame
# import pandas
import tensorflow as tf
import tensorflow_text as tf_text
from sklearn.feature_extraction.text import TfidfVectorizer


class TextVectorizer:
    def __init__(self, stop_words=None):
        if stop_words is None:
            self.tfidf_vectorizer = TfidfVectorizer(stop_words='english')
            # self.tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)
        else:
            # self.tfidf_vectorizer = TfidfVectorizer(stop_words, max_df=0.7)
            self.tfidf_vectorizer = TfidfVectorizer(stop_words)
        self.df = None
        self.tokens = None
        self.tfidf_x_train = None
        # self.tfidf_x_test = None
        # self.tfidf_y_test = None
        # self.tfidf_y_train = None
        self.start = 0.0
        self.end = 0.0
        pass

    def vectorization(self, data):
        # if x_test is None:
        #     x_test = pandas.DataFrame(x_train)
        # if x_train is None or x_test is None:  # or not isinstance(x_train, DataFrame) or not isinstance(x_test,
        #     # DataFrame):
        #     raise Exception('Only uses DataFrame type')
        self.start = time()
        # Vectorization
```

Page 1 of 4

```python
        self.tfidf_x_train = self.tfidf_vectorizer.fit_transform(data)
        self.end = time()
        pass

    def runtime_cost(self):
        return self.end - self.start

    def get_vocab_dict(self):
        vocab_dict = collections.defaultdict(lambda: 0)
        for tokens in self.tfidf_x_train.as_numpy_iterator():
            for tok in tokens:
                vocab_dict[tok] += 1

        vocab = sorted(vocab_dict.items(), key=lambda x: x[1], reverse=True)
        vocab = [token for token, count in vocab]
        vocab = vocab[:len(self.tfidf_vectorizer.get_feature_names())]
        vocab_size = len(vocab)
        print("Vocab size: ", vocab_size)
        print("First five vocab entries:", vocab[:5])
        return vocab

    def get_vocab_table(self):
        vocab = self.get_vocab_dict()
        keys = vocab
        values = range(2, len(vocab) + 2)  # reserve 0 for padding, 1 for OOV

        init = tf.lookup.KeyValueTensorInitializer(
            keys, values, key_dtype=tf.string, value_dtype=tf.int64)

        num_oov_buckets = 1
        vocab_table = tf.lookup.StaticVocabularyTable(init, num_oov_buckets)
        return vocab_table

    def preprocess_text(self, text):
        tokenizer = tf_text.UnicodeScriptTokenizer()
        vocab_table = self.get_vocab_table()
        standardized = tf_text.case_fold_utf8(text)
        tokenized = tokenizer.tokenize(standardized)
        vectorized = vocab_table.lookup(tokenized)
        return vectorized

    def tokenization(self, input_data):
        self.df = input_data
        tokenizer = tf_text.UnicodeScriptTokenizer()
        # self.tokens = MapDataset
        for row in self.df:
            row = str(row)
            row = tf_text.case_fold_utf8(row)
            token = tokenizer.tokenize(row)

        pass


def tokenize(num, text, act_label, emotion_label, speaker, conv_id, utt_id, emotion_index):
    text = tf_text.case_fold_utf8(text)
    tokenizer = tf_text.UnicodeScriptTokenizer()
    return num, tokenizer.tokenize(text), act_label, emotion_label, speaker, conv_id, utt_id, emotion_index


def configure_dataset(dataset):
    """
    To optimize the accessibility of the dataset
    by storing it in the cache memory.
    :param dataset: Tensor Dataset -> Dataset to be accessed
    :return: Optimized Dataset
    """
    AUTOTUNE = tf.data.AUTOTUNE
    return dataset.cache().prefetch(buffer_size=AUTOTUNE)


def preprocess_text(text, vocab_table, label):
    tokenizer = tf_text.UnicodeScriptTokenizer()
    standardized = tf_text.case_fold_utf8(text)
    tokenized = tokenizer.tokenize(standardized)
    vectorized = vocab_table.lookup(tokenized)
    return vectorized, label

```

8.6 Preprocessing Frameworks: Splitting Dataset

File - D:\amado\ProjectsHub\pythonProject\methodology\preprocessing\TrainTestSplit.py

```python
from sklearn.model_selection import train_test_split


class TrainTestSplit:
    def __init__(self, dataset, labels_, test_set_ratio):
        self.x_train = None
        self.y_train = None
        self.x_test = None
        self.y_test = None
        self.ratio = test_set_ratio
        self.df = dataset
        self.labels = labels_
        pass

    def split(self, test_set_ratio=None):
        if test_set_ratio is None:
            self.x_train, self.x_test, self.y_train, self.y_test = train_test_split(self.df, self.labels,
                                                                                    test_size=self.ratio,
                                                                                    random_state=7)
        else:
            self.x_train, self.x_test, self.y_train, self.y_test = train_test_split(self.df, self.labels,
                                                                                    test_size=test_set_ratio,
                                                                                    random_state=7)
        return self.x_train, self.x_test, self.y_train, self.y_test


if __name__ == "__main__":
    # how to use the module
    tts = TrainTestSplit(None, None, None)
    tts.split()
    pass

```

Page 1 of 1

8.7 Proposed Model Frameworks: Naive Bayes Classifier

File - D:\amado\ProjectsHub\pythonProject\methodology\models\nbc.py

```python
from sklearn.naive_bayes import GaussianNB,
MultinomialNB


def create_nbc_model(type_='gaussian'):
    model = None
    if type_ == 'gaussian':
        model = GaussianNB()
    elif type_ == 'multinomial' or type_ == 'multi':
        model = MultinomialNB()

    return model


if __name__ == '__main__':
    import numpy as np

    rng = np.random.RandomState(1)
    X = rng.randint(5, size=(6, 100))
    y = np.array([1, 2, 3, 4, 5, 6])
    clf = create_nbc_model(type_='multinomial')
    clf.fit(X, y)
    print(clf.predict(X[2:3]))

```

Page 1 of 1

Proposed Model Frameworks: Convolutional Neural Networks

File - D:\amado\ProjectsHub\pythonProject\methodology\models\cnn.py

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv1D, MaxPool1D

from methodology.models.metrics import f1_m, precision_m, recall_m


def create_cnn_model(input_shape=None, input_dim=None, optimizer_='RMSprop', extra_model=None):
    # input_shape = (48, 48, 3)

    model_ = Sequential()
    if extra_model is not None:
        model_.add(extra_model)
    if input_shape is not None:
        # model_.add(Conv2D(6, (5, 5), input_shape=input_shape, padding='same', activation='relu'))
        model_.add(Conv1D(64, 3, input_shape=input_shape, activation="relu"))
    elif input_dim is not None:
        # model_.add(Conv2D(100, (3, 3), input_dim=input_dim, activation='relu'))
        model_.add(Conv1D(64, 3, input_dim=input_dim, activation="relu"))

    model_.add(MaxPool1D(pool_size=2))
    # model_.add(MaxPooling2D(pool_size=(2, 2)))

    # model_.add(Conv2D(16, (5, 5), padding='same', activation='relu'))
    model_.add(Conv1D(64, 5, padding='same', activation='relu'))

    # model_.add(Activation('relu'))

    # model_.add(MaxPooling2D(pool_size=(2, 2)))
    model_.add(MaxPool1D(pool_size=2))

    # model_.add(Conv2D(64, (3, 3), activation='relu'))
    model_.add(Conv1D(64, 3, activation='relu'))

    # model_.add(MaxPooling2D(pool_size=(2, 2)))
```

Page 1 of 2

```python
    model_.add(MaxPool1D(pool_size=2))

    model_.add(Flatten())
    model_.add(Dense(64, activation='relu'))
    model_.add(Dropout(0.5))
    model_.add(Dense(7, activation='softmax'))

    if optimizer_ == 'adam':
        model_.compile(loss='categorical_crossentropy', metrics=['accuracy', f1_m, precision_m, recall_m], optimizer='adam')
    elif optimizer_ == 'RMSprop':
        model_.compile(loss='categorical_crossentropy', metrics=['accuracy', f1_m, precision_m, recall_m], optimizer='RMSprop')
    model_.summary()
    return model_


def create_nn_model(input_dim=None, input_shape=None):
    model_ = Sequential()
    if input_dim is not None:
        model_.add(Dense(10, input_dim=input_dim, activation='relu'))
    elif input_shape is not None:
        model_.add(Dense(10, input_shape=input_shape, activation='relu'))
    model_.add(Dense(1, activation='sigmoid'))

    model_.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    model_.summary()
    return model_


if __name__ == '__main__':
    from methodology.models.nbc import create_nbc_model

    create_cnn_model((48, 48, 3), create_nbc_model())
    # create_nn_model(input_dim=7000)
    # create_nn_model(input_shape=(7000, 11055))
    pass
```

8.8 Metrics Implementation

File - D:\amado\ProjectsHub\pythonProject\methodology\models\metrics.py

```python
from tensorflow.keras import backend as K


def recall_m(y_true, y_predict):
    true_positives = K.sum(K.round(K.clip(y_true * y_predict, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall


def precision_m(y_true, y_predict):
    true_positives = K.sum(K.round(K.clip(y_true * y_predict, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_predict, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision


def f1_m(y_true, y_predict):
    precision = precision_m(y_true, y_predict)
    recall = recall_m(y_true, y_predict)
    return 2 * ((precision * recall) / (precision + recall + K.epsilon()))

```

Page 1 of 1

8.9 Results while running the OEHML Framework

File - main

```
C:\Users\IUT-Cmr-Thesis-CSE\Desktop\pythonProject\venv\
Scripts\python.exe C:/Users/IUT-Cmr-Thesis-CSE/Desktop/
pythonProject/main.py
2021-03-02 16:07:15.241874: I tensorflow/stream_executor/
platform/default/dso_loader.cc:49] Successfully opened
dynamic library cudart64_110.dll
Dataset
(102979, 8)
['Unnamed: 0' 'sentence' 'act_label' 'emotion_label' '
speaker' 'conv_id'
 'utt_id' 'emotion_index']
=======================================================
[nltk_data] Downloading package stopwords to C:\Users\IUT-
Cmr-Thesis-
[nltk_data]     CSE\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
C:\Users\IUT-Cmr-Thesis-CSE\Desktop\pythonProject\venv\lib
\site-packages\sklearn\utils\validation.py:70:
FutureWarning: Pass input=['is', 'for', 'what', 'have', '
yours', 'with', 'all', 'up', 'ours', 't', 'once', "it's
", 'm', "needn't", 'itself', 'while', 'my', 'that', 'mustn
', 'i', 'through', 'so', 'off', 'were', 'been', "you'll
", 'after', 'themselves', 'which', 'hers', 'if', 'why', '
to', 'won', 'wouldn', 'yourself', 'of', 'too', 'further
', 'it', "that'll", 'should', 'your', 'other', 'has', '
weren', "you've", 'there', 'only', 'few', 'couldn', 'doing
', 'how', 'then', 'where', 'you', 'me', 'haven', 'its', 's
', 'on', 'theirs', 'd', 'him', 'he', 'does', 'or', '
herself', 'each', 'shan', 'here', 'be', 'yourselves', "you
're", 'they', 'than', 'll', 'his', "should've", 'we', '
hasn', 'can', 'very', 'some', 've', 'out', 'her', 'whom
', 'from', 'just', 'don', 'this', 'into', 're', 'again', '
will', 'own', 'their', 'was', "you'd", 'over', 'isn', '
about', 'being', 'now', 'o', 'a', 'as', 'aren', 'wasn', '
ain', 'y', 'by', 'because', 'those', 'am', 'an', 'at', "
mightn't", 'had', 'the', 'in', 'are', 'didn', 'such', '
myself', 'doesn', 'shouldn', 'mightn', 'until', 'having
', 'these', 'himself', 'hadn', 'do', 'ourselves', "she's
", 'and', 'she', 'same', 'ma', 'needn', 'who', 'more', '
when', 'our', 'most', 'did', 'them'] as keyword args. From
 version 1.0 (renaming of 0.25) passing these as
positional arguments will result in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From
version "
179
Vectorized Dataset
<class 'scipy.sparse.csr.csr_matrix'>
```

Page 1 of 86

```
(102979, 18566)
=======================================================
18566
['00', '000', '007', '0085', '01', '010', '01088256798', '
011', '01563', '01705', '0234', '04', '0411', '05', '
050920', '053', '060', '06230221', '075', '08', '09', '
09112223', '099', '0h', '0k', '0kay', '0n', '10', '100', '
1000', '10000', '1005', '100715', '1008', '10086', '100m
', '100ml', '100rmb', '100th', '101', '1010', '1019', '102
', '1021', '103', '104', '105', '1050', '1050ft', '106', '
108', '1088', '109', '10am', '10kg', '10s', '10th', '11
', '110', '1100', '1106', '110cm', '112', '1127', '113', '
115', '1150', '117', '1177', '1199866', '1199886', '11am
', '11th', '12', '120', '1200', '12000', '1201', '1202', '
1205', '120cm', '120mm', '1212', '1218', '123', '12300', '
1234', '124', '125', '1256', '1267', '129', '12c', '12th
', '13', '130', '1305', '1308', '1314', '132', '1336', '
134986', '135', '13661306917', '137', '138', '13811658', '
1388', '139', '13924774026', '1394', '139xxxxx345', '13th
', '14', '140', '1400rmb', '1405', '1408', '1419', '1425
', '1433', '1446', '1492', '14th', '15', '150', '1500', '
15066688866', '151', '15273478841', '1552', '1564', '
15699324873', '1588', '158cm', '15mins', '15minutes', '
15th', '16', '160', '1600', '1616', '162', '16211469', '
1639', '164', '1644', '165', '166', '167', '168', '16th
', '17', '1700', '1739', '175', '176', '1780', '17th', '18
', '180', '1805', '1808', '180c', '1818', '1825', '1830
', '184796', '185', '1862', '1867', '1879', '1886', '1889
', '189', '1890', '1892', '1893', '18cm', '18k', '18th', '
19', '190', '1910', '1911', '1916', '1917', '1920', '1920s
', '1930', '1932', '1940', '1944', '1949', '195', '1955
', '1960', '1963', '1965', '1969', '1970', '1976', '1977
', '1978', '198', '1980', '1980s', '1982', '1984', '1985
', '1986', '1987', '1988', '1989', '199', '1991', '1992
', '1993', '1994', '1995', '1996', '1997', '1998', '1999
', '19th', '1e', '1poj7403', '1st', '1y', '20', '200', '
2000', '200085', '2001', '2002', '2003', '2004', '2005', '
2006', '2007', '2008', '2009', '200yuan', '201', '2010', '
2012', '202', '2022', '203', '20310', '204', '205', '2065
', '207', '208', '209', '20am', '20kg', '20s', '20th', '
20tha', '20x', '21', '210', '2101', '212', '2123456', '213
', '2135', '2135367', '214', '215', '216', '218', '21st
', '22', '220', '220150', '221', '2210', '2213', '222', '
2244', '225', '228', '2289', '22nd', '23', '230', '2300
', '231', '232', '2323', '233', '2331', '2345', '235', '
2356', '236', '2367', '2368', '238', '23rd', '23th', '24
', '240', '241', '2424', '2453', '2456', '2477', '249', '
', 'yoga', 'yoghurt', 'yogurt', 'yokohama', 'yolanda', 'yolk', 'yon', 'yong', 'yongmei', 'yor', 'york', 'yos', 'you', 'youknow', 'young', 'younger', 'youngest', 'youngsters', 'yourfingers', 'yourorder', 'yous', 'youse', 'youth', 'youths', 'yt', 'yuan', 'yuanmingyuan', 'yuanxiao', 'yuanyang', 'yucatan', 'yuck', 'yue', 'yummy', 'yunnan', 'yup', 'yuppie', 'yuri', 'yuu', 'yvonne', 'yw132', 'yy', 'z3264356', 'zach', 'zat', 'ze', 'zealand', 'zealander', 'zero', 'zeros', 'zeta', 'zhang', 'zhaopin', 'zheg', 'zhejiang', 'zhengjun', 'zhenjiang', 'zhilian', 'zhongguancun', 'zhongshan', 'zhouzhuang', 'zhuang', 'zhuhai', 'zhumulangma', 'zina', 'zinc', 'zip', 'zipper', 'zipping', 'zither', 'zizhuyuan', 'zn741', 'zodiac', 'zoe', 'zombie', 'zombies', 'zone', 'zones', 'zongzi', 'zoo', 'zoom', 'zoos', 'zu', 'zucchini', 'zurich', 'zw203', 'zzz']
(102979, 18566)
  (0, 5218) 0.4092451893084657
  (0, 2109) 0.6098096520176245
  (0, 7544) 0.2968291315798868
  (0, 9220) 0.49807524710307427
  (0, 14407)0.3527946686144274
  (1, 6824) 0.5505481065005402
  (1, 7557) 0.2524933795467948
  (1, 11403)0.2515560360981949
  (1, 13406)0.26959749784204967
  (1, 2865) 0.22690320571428813
  (1, 16547)0.6161243048934756
  (1, 9499) 0.25707275571263655
  (2, 13667)0.6524469753949736
  (2, 17540)0.43567100825794725
  (2, 8123) 0.3963443429262471
  (2, 10516)0.47688036098356545
  (3, 16809)0.21738990063172572
  (3, 9638) 0.2678893131885065
  (3, 13710)0.3132017762156408
  (3, 14999)0.4087202876600346
  (3, 965)  0.44251362673554545
  (3, 6608) 0.400987740065293
  (3, 10259)0.2430781418363468
  (3, 5481) 0.21300380184000514
  (3, 16685)0.20091052840226814
  : :
  (102974, 11415)   0.45107130640082943
  (102974, 1108)0.45768958051066366
  (102974, 7451)0.24552276193271674
  (102974, 14136)   0.31975321737196877
  (102974, 16685)   0.23995883998054948
  (102975, 1803)0.4251066067527789
  (102975, 10755)   0.36841282562098965
  (102975, 3075)0.4022419604098753
  (102975, 9988)0.2785200433106108
  (102975, 11271)   0.35845722777329003
  (102975, 16348)   0.2804458411888824
  (102975, 18026)   0.3542036515353818
  (102975, 11294)   0.334039926998 1387
  (102976, 18447)   0.792522626060015 7
  (102976, 16685)   0.609842510147445
  (102977, 1598)0.6962124304188702
  (102977, 11576)   0.34830629800301
  (102977, 9988)0.3378840184756713 4
  (102977, 7687)0.3773526802842974 7
  (102977, 10259)   0.3706889791222249
  (102978, 16070)   0.5968975766414103
  (102978, 11896)   0.5146221268753249
  (102978, 15695)   0.37086543059526206
  (102978, 9988)0.2697199357767614
  (102978, 18272)   0.41059388471322755
Training
(87532, 18566)
(87532,)
=======================================================
Validation
(9268, 18566)
(9268,)
=======================================================
Testing
(6179, 18566)
(6179,)
=======================================================
        NAIVE BAYES CLASSIFIER
[[5116    0    0    0    0    1    0]
 [  60    0    0    0    0    0    0]
 [  69    0    0    0    0    0    0]
 [   9    0    0    0    0    0    0]
 [ 110    0    0    0    0    0    0]
 [ 786    0    0    0    0    2    0]
 [  26    0    0    0    0    0    0]]
accuracy:: 82.83 %
=======================================================
        CONVOLUTION NEURAL NETWORK
2021-03-02 16:07:21.289327: I tensorflow/compiler/jit/
xla_cpu_device.cc:41] Not creating XLA devices,
tf_xla_enable_xla_devices not set
```

```
2021-03-02 16:07:21.290596: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library nvcuda.dll
2021-03-02 16:07:21.311837: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1720] Found device 0 with properties:
pciBusID: 0000:01:00.0 name: GeForce GTX 1070 computeCapability: 6.1
coreClock: 1.683GHz coreCount: 15 deviceMemorySize: 8.00GiB deviceMemoryBandwidth: 238.66GiB/s
2021-03-02 16:07:21.312156: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cudart64_110.dll
2021-03-02 16:07:21.399320: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cublas64_11.dll
2021-03-02 16:07:21.399494: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cublasLt64_11.dll
2021-03-02 16:07:21.433733: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cufft64_10.dll
2021-03-02 16:07:21.440885: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library curand64_10.dll
2021-03-02 16:07:21.486791: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cusolver64_10.dll
2021-03-02 16:07:21.515543: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cusparse64_11.dll
2021-03-02 16:07:21.518360: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cudnn64_8.dll
2021-03-02 16:07:21.518913: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1862] Adding visible gpu devices: 0
2021-03-02 16:07:21.520055: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-03-02 16:07:21.521083: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1720] Found device 0 with properties:
```

```
pciBusID: 0000:01:00.0 name: GeForce GTX 1070 computeCapability: 6.1
coreClock: 1.683GHz coreCount: 15 deviceMemorySize: 8.00GiB deviceMemoryBandwidth: 238.66GiB/s
2021-03-02 16:07:21.521511: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cudart64_110.dll
2021-03-02 16:07:21.521714: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cublas64_11.dll
2021-03-02 16:07:21.521960: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cublasLt64_11.dll
2021-03-02 16:07:21.522399: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cufft64_10.dll
2021-03-02 16:07:21.522567: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library curand64_10.dll
2021-03-02 16:07:21.522699: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cusolver64_10.dll
2021-03-02 16:07:21.522833: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cusparse64_11.dll
2021-03-02 16:07:21.522971: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cudnn64_8.dll
2021-03-02 16:07:21.523156: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1862] Adding visible gpu devices: 0
2021-03-02 16:07:23.144793: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1261] Device interconnect StreamExecutor with strength 1 edge matrix:
2021-03-02 16:07:23.144974: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1267]      0
2021-03-02 16:07:23.145085: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1280] 0:   N
2021-03-02 16:07:23.147612: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1406] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 6301 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1070, pci bus id: 0000:01:00.0, compute capability: 6.1)
2021-03-02 16:07:23.150083: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not creating XLA devices, tf_xla_enable_xla_devices not set
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
conv1d (Conv1D)             (None, 18564, 64)         256
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 9282, 64)         0
_________________________________________________________________
conv1d_1 (Conv1D)           (None, 9282, 64)          20544
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 4641, 64)         0
_________________________________________________________________
conv1d_2 (Conv1D)           (None, 4639, 64)          12352
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 2319, 64)         0
_________________________________________________________________
flatten (Flatten)           (None, 148416)            0
_________________________________________________________________
dense (Dense)               (None, 64)                9498688
_________________________________________________________________
dropout (Dropout)           (None, 64)                0
_________________________________________________________________
dense_1 (Dense)             (None, 7)                 455
=================================================================
Total params: 9,532,295
Trainable params: 9,532,295
Non-trainable params: 0
_________________________________________________________________
(87532, 18566, 1) 3
(9268, 18566, 1) 3
(6179, 18566, 1) 3
2021-03-02 16:08:44.612041: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registered 2)
Epoch 1/6
2021-03-02 16:08:46.950712: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cublas64_11.dll
2021-03-02 16:08:48.510074: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cublasLt64_11.dll
2021-03-02 16:08:48.574379: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cudnn64_8.dll
2021-03-02 16:08:57.122418: I tensorflow/core/platform/windows/subprocess.cc:308] SubProcess ended with return code: 0

2021-03-02 16:08:57.169884: I tensorflow/core/platform/windows/subprocess.cc:308] SubProcess ended with return code: 0

2736/2736 [==============================] - 234s 80ms/step - loss: 0.6334 - accuracy: 0.8342 - f1_m: 0.8201 - precision_m: 0.8381 - recall_m: 0.8067 - val_loss: 0.5377 - val_accuracy: 0.8447 - val_f1_m: 0.8437 - val_precision_m: 0.8507 - val_recall_m: 0.8371
Epoch 2/6
2736/2736 [==============================] - 207s 76ms/step - loss: 0.5550 - accuracy: 0.8408 - f1_m: 0.8400 - precision_m: 0.8449 - recall_m: 0.8354 - val_loss: 0.5268 - val_accuracy: 0.8493 - val_f1_m: 0.8487 - val_precision_m: 0.8506 - val_recall_m: 0.8468
Epoch 3/6
2736/2736 [==============================] - 206s 75ms/step - loss: 0.5435 - accuracy: 0.8430 - f1_m: 0.8428 - precision_m: 0.8447 - recall_m: 0.8410 - val_loss: 0.5127 - val_accuracy: 0.8500 - val_f1_m: 0.8506 - val_precision_m: 0.8520 - val_recall_m: 0.8492
Epoch 4/6
```

```
2736/2736 [==============================] - 206s 75ms/step - loss: 0.5463 - accuracy: 0.8433 - f1_m: 0.8430 - precision_m: 0.8441 - recall_m: 0.8420 - val_loss: 0.5176 - val_accuracy: 0.8528 - val_f1_m: 0.8530 - val_precision_m: 0.8539 - val_recall_m: 0.8522
Epoch 5/6
2736/2736 [==============================] - 206s 75ms/step - loss: 0.5465 - accuracy: 0.8426 - f1_m: 0.8425 - precision_m: 0.8433 - recall_m: 0.8418 - val_loss: 0.5071 - val_accuracy: 0.8492 - val_f1_m: 0.8501 - val_precision_m: 0.8511 - val_recall_m: 0.8490
Epoch 6/6
2736/2736 [==============================] - 206s 75ms/step - loss: 0.5501 - accuracy: 0.8434 - f1_m: 0.8432 - precision_m: 0.8440 - recall_m: 0.8425 - val_loss: 0.5119 - val_accuracy: 0.8516 - val_f1_m: 0.8518 - val_precision_m: 0.8526 - val_recall_m: 0.8510
Evaluation Metrics -> [0.5272582769393921, 0.8462534546852112, 0.8472246527671814, 0.8476523160934448, 0.8468105792999268]


Predicted Values
 [[8.9660525e-01 1.0561549e-02 1.3125801e-02 ... 1.1793253e-02
  6.6634171e-02 1.1806774e-03]
 [1.9501290e-01 3.9942763e-05 1.2446606e-05 ... 1.4088744e-03
  8.0352265e-01 2.6760954e-06]
 [8.2745451e-01 2.1576179e-02 2.1374131e-02 ... 3.6477439e-02
  7.9636887e-02 9.9258469e-03]
 ...
 [9.1742301e-01 7.3389951e-03 9.2443721e-03 ... 1.0792229e-02
  5.4818597e-02 3.7138377e-04]
 [9.0705246e-01 7.5163343e-03 9.0127774e-03 ... 8.9763766e-03
  6.6854618e-02 5.5826688e-04]
 [8.7877345e-01 1.4296273e-02 1.6122404e-02 ... 1.6009919e-02
  7.1656808e-02 2.7259788e-03]]
dict_keys(['loss', 'accuracy', 'f1_m', 'precision_m', 'recall_m', 'val_loss', 'val_accuracy', 'val_f1_m', 'val_precision_m', 'val_recall_m'])

Process finished with exit code 0
```