CLASSIFYING 6 BASIC EMOTIONS USING DIFFERENT PRE-TRAINED TRANSFORMER MODELS

by

THIRI MAY THU @ ALICE
URN: 6635583

A dissertation submitted in partial fulfilment of the
requirements for the award of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

May 2024

Department of Computer Science
University of Surrey
Guildford GU2 7XH

Supervised by: Stella Kazamia

I declare that this dissertation is my own work and that the work of others is acknowledged and indicated by explicit references.

Thiri May Thu @ Alice
May 2024

Abstract

The process of extracting emotions out of a piece of text will be of a great importance to better identify and understand the users online in this digital world. As this has many benefits such as getting a sense of the general emotion that the crowd of users leaning towards, especially towards news and events through comments. However, creating machine learning algorithms to accurately classify the emotion of a text is a challenge that many are facing with any classification. Therefore, this dissertation's aim is to search for better models, mainly transformers models to compare and contrast, which model works the best. This will be in terms of accuracy, f1-score, confusion matrix and any evaluation functions. The main models that this project will focus on will be BERT based models and Large Language Models, mainly text generative models, as these proposed mechanisms will use multiple self-attention layers. Such that these models could reveal the relationships of each word towards each other and with the emotions present in them, which sparks my motivation to take on this project to further investigate and fine-tune different pre-trained transformers models.

Acknowledgements

I am greatly thankful for my supervisor, Stella Kazamia for their guidance for the topic of this project as well as throughout every stages of this project's development.

I also would like to thank Diptesh Kanojia for providing his knowledge in the area of Nature Language Process which greatly helps with this project's research and implementation.

Contents

Abbreviations

NLP	Natural Language Processing
GPU	Graphic Processing Unit
LLM	Large Language Model
BERT	Bidirectional Encoder Representations from Transformers
RoBERTa	Robustly (optimized BERT pretraining approach)

Chapter 1

Introduction

1.1 Chapter Overview

This chapter will mainly focus on the introduction of the project. It will discuss its background, its aims and objectives as well as its overview, and limitations that the technologies that are work with might have.

1.2 Project Background

"What is emotion?" According to the Oxford English Dictionary, it means a strong feeling deriving from one's circumstances, mood, or relationships with others (*akrasia, n.* n.d.). One's emotion can be portrayed through facial expressions, speaking and writing. Nowadays with the ever rising popularity of social media such as TikTok and X (previously named Twitter) provide a way to post their opinions, mood and emotions online, which in turn, could also express with contempts towards one another. Therefore, the task of identifying the emotions that the other party's feeling toward the contempt-filled post or both parties are necessary as emotions are a fundamental part of human life, influencing both physical and mental health(Ameer, Bölücü, Siddiqui, Can, Sidorov & Gelbukh 2023). Emotion classification is a context-based device so with visual and vocal inputs along with text, will be easier to classify. However, without other inputs such as facial expressions and tone of voice, it is hard to create a model that could detect emotions accurately through only text.

Sentiment Analysis is used to classify the overall sentiment of the text in terms of positive, negative and neutral, and is one of the fields in NLP which is a machine learning algorithms with statistical computation of human Language (computational linguistics) to generate text and speech (IBM 2024). Sentiment Analysis is used in many companies in their marketing and services online to see the trend and mood of their consumers as well as potential consumers, and it is proven to be very useful. Nevertheless, emotion classification goes deeper and aims to identify underlying emotion/(s) in a given sentence. This is a problem for multi-label classification as the given statement could have different dimensions and an instance could have subset of emotions or other labels(Ameer et al. 2023). Nevertheless, this project is working with single-label classification which means that it might not face this problem however, the accuracy that emotion that is output might not fully consider the underlying semantics. For example, a given statement could have more than one emotion, but it is reduced down to single emotion. The

basic emotions include six basic emotions such as sadness, joy, love, anger, surprise and fear, and many more. For this project, The dataset (Pandey 2021) found is labelled only with six basic emotions mentioned previously.

1.3 Project Overview

This project is to attempt to compare different pre-trained models of transformers to find which of them are the best model for classifying the emotion of the English twitter (now called 'X') messages within the bounds of six basic emotions (Pandey 2021). This project will begin with the literature reviews about the research papers similar to this project and different pre-trained transformer models that will be compared and implemented into this project. The general theory behind transformer architecture and how different the models that are implemented from their former architecture and with each other will be discussed. This project will then break down the problems for each of the models and their technical parts of the implementation. Finally, the results will be presented at the end. The gathered results will be analysed and compared to find the best single-label emotion classification models on the applicable dataset.

1.4 Project Aims & Objectives

The overall aims of the project is to compare and demonstrate relatively newer transformer models that will be implemented in this project will be better at detecting emotion in the given piece of text. The following list below is the list of objectives for this project:

- Explore different pre-trained transformers models that are both relatively new and old in the fields of NLP.
- Review the relevant or similar literatures for emotion classification and the usage of different transformer models.
- Discover suitable dataset for training and testing.
- Implement the two or more pre-trained models and use the dataset collected to train and test the models implemented.
- Provide a critical comparative analysis of the different models used to determine which give the best single-label emotion classification results.

1.5 Limitations

For the resources to carry out this project, Google Colab and Vscode will be two primary platforms for coding, training and testing. Google Colab have limited GPU runtime which hindered the progress of the implementation therefore local GPU is used to progress further. However, some models like Large Language Models (LLMs) and large datasets will use more GPU power as well as CPU and amount of RAM given which also slow down the training process.

Furthermore, this project and dataset unfortunately do not account for or identify sarcasm which could result in wrong emotion for the sarcastic statements.

Chapter 2

Literature Review

Emotion detection problem has a problem that has been tackling alongside with the developments of Deep Learning and NLP models in recent years. Numerous neural network models as well as various transformers models have been suggested to solve this problem, including the papers that are going to be discussed in the following as well as it is the goal of this dissertation.

2.1 What are basic emotions?

Emotions are essential components of a human life. They are expressed in various ways which could be influence by their culture, relations, environments and so on. With all those various emotions, in emotional psychology, they are divided into two groups: basic and complex.

Figure 2.1: Six basic emotions

For this project, we will be focussing on basic emotions. Basic emotions are emotions that are recognised them through facial expressions and tend to happen automatically (Uwa 2023). Charles Darwin is the first to proposed that the emotions that are expressed thorough facial expression are universal. Emotional psychologist, Paul Ekman identified six basic emotion: sadness, joy, fear, anger, surprise and disgust, shown in 2.1. However, the dataset that will be used will replace disgust with love which equalises the positive and negative emotions.

2.2 Why Emotion Classification?

Research into classifying emotions across various mediums has been ongoing and evolving alongside the rapid development of new Deep Learning and NLP models. Thus, many have been polishing text emotion classification with different models such as neural networks and transformer models. However, accurately detecting emotions from text is still a goal that many are trying to reach.

The application of emotion classification, especially in medical field, could help greatly for therapist. For example, detecting a person's emotion while therapy or consultation through a chat will be a great assist for any therapist and consultant. Its application can venture into customer services as well as education (feedback for modules and lecturers)(Amrullah 2023). Other uses may include enhancement of machine translation as understanding the emotions behind the text could provide more related translation in another languages which often a struggle(Amrullah 2023). Finally, emotion based content in social media and other online platforms(Amrullah 2023).

There are many other uses for emotion classification models and this project is to find the best suited transformer model for this task.

2.3 Transformers

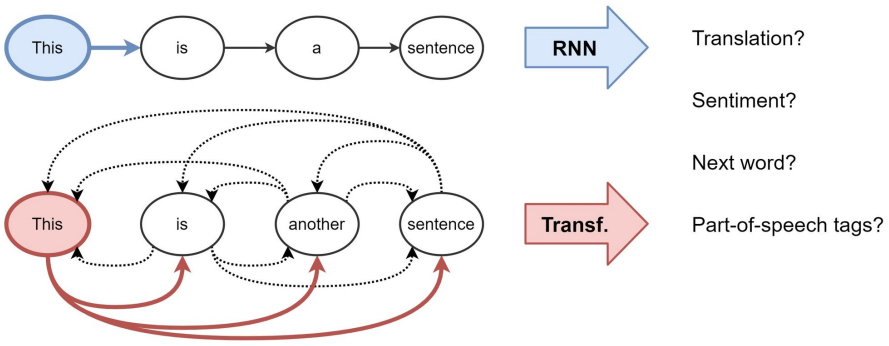

Figure 2.2: Comparison of RNN and Transformers methods

Transformer is a model architecture that changes the world of Deep Learning and NLP to what we have now with models such as BERT based models, generative models and so on. The difference between older neural networks such as recurrent neural networks and transformer models is that rather than depending on recurrence, the model depends entirely on an attention mechanism (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser & Polosukhin 2023). By using an attention mechanism, transformer build features of each word to figure out the importance of each word in the sentence as well as relation with each other in the given sentence like shown in Figure[2.2] (Vaswani et al. 2023).

There are many models both new and old all stem from Transformers. Such that some might only build with encoder such as BERT and BERT based models, some might only use decoder like GPT and Generative models and others might be used both encoder-decoders such as BART like the transformer architecture shown in Figure[2.3].

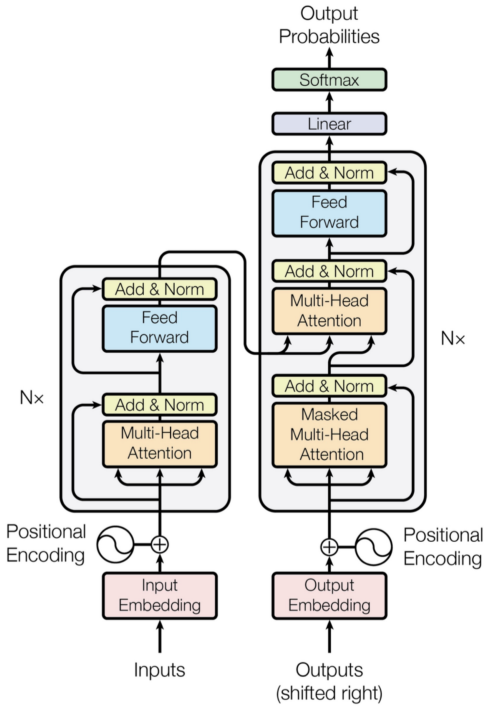

Figure 2.3: Architecture of Transformer

2.3.1 BERT based models

BERT or Bidirectional Encoder Representations from Transformers, is one of the first few variants of transformer models both in the field of deep learning and NLP. Before BERT, language models could only read input text sequentially, which means either left-to-right or right-to-left at a time (Hashemi-Pour & Lutkevich 2024). However, with BERT it can be done in both directions at once (Hashemi-Pour & Lutkevich 2024) as a result, more contexts can be brought out and work better for text classification.

With this model, there are many variations of BERT based models, including RoBERTa and miniLM which are in the scope of this project.

2.3.1.1 RoBERTa

RoBERTa (Robustly (optimized BERT pretraining approach)) is an improved version of BERT. It modifies key hyperparameters, removing the next-sentence pretraining objective and training with much mini-batches and learning rates (Sharma 2022). It trained in much larger datasets than BERT which is under-trained. Furthermore, it was trained with dynamic masking, large mini-batches, larger byte-level BPE (Byte-Pair Encoding), and full-sentences without NSP (Next Sentence Prediction) loss (Sharma 2022).

The study, "Multi-label emotion classification in texts using transfer learning" (Ameer et al. 2023), is about utilising different self-attention mechanisms and then-popular, transformer models to solve the multi-label emotion classification problem. However, because of the lack of multi-label emotion datasets, this project will be using single-label dataset. They used two different datasets with a different set of emotion labels as well as 2 different languages, one of which being in English and the other in Chinese. The transformer models they experimented with were XLNet, DistilBERT, and RoBERTa as well as multi-attention layers of each model. Their results shown that RoBERTa with multi-attention layers (RoBERTa-MA) is the best model for

multi-label emotion classification with 62.4% accuracy and f1-score being 74.2% and the second place with just RoBERTa which had an accuracy of 61.2% and 73.7% for the f1-score for English dataset.

2.3.1.2 MiniLM

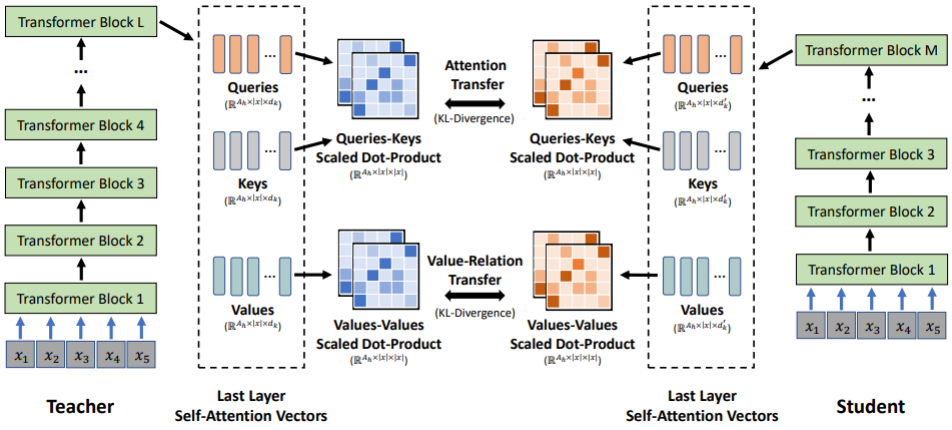

Figure 2.4: Overview of Deep Self-Attention Distillation

Another variant of pretrained BERT is miniLM: Deep Self-attention distillation for task-agnostic compression of pretrained transformer (Wang, Wei, Dong, Bao, Yang & Zhou 2020). It compresses large pretrained transformer based models to smaller model, otherwise called deep self-attention distillation. This means that "The small model (student) is trained by deeply mimicking the self-attention module, which play a crucial role in transformer networks, of the large model (teacher)" according to the paper (Wang et al. 2020), the overview shown in figure[2.4] It also has fewer parameters than its predecessor which made it easier to fine-tune and run the model with lesser cost.

In this paper (Wang et al. 2020), they may not do the same classification or the research as this project, but they fine-tuned and experimented with GLUE (General Language Understanding Evaluation) benchmark which consists of 9 sentence level classification tasks such as SQuAD2, MNLI-m, SST-2 and so-on. The average of all 8 tasks with 4 runs for each task is 80.4% accuracy which is slightly less than BERT with 81.5%. Nonetheless, it works better than expected for a small model.

2.3.2 GPT

GPT (Generative pretrained transformer) is a series of language models that is developed by OpenAI (Jorge 2023). GPT have 4 different versions: GPT, GPT-2, GPT-3.5 and GPT4. The latest two versions may not open-sourced however the public could still use GPT-1 and 2 to experiment or compare with the newly developments of LLMs. GPT shines mainly in text generation which produces coherent and contextually relevant sentences (Jorge 2023).

For the related research for this model, "Generative Pretrained Transformers for Emotion Detection in a Code-Switching Setting" (Nedilko n.d.), they used GPT models with the zero-shot or few-shot approaches to detect the human emotions. As they could not access GPT-4, they used ChatGPT for this experiment with few-shot-method and got 73.13% for the accuracy and 70.38% for the macro-f1. If there is access for GPT-4 as shown in the previous paper above, the

result of this experiment will be greater.

2.3.3 Llama2

Llama stands for Large Language Model Meta AI which is a subset of LLMs which is introduced by Meta AI. Llama models vary in size, ranging from 7 billions parameters to 70 billions. The model is an autoregressive language model and based on the transformer decoder architecture. It is also a generative text model, which processes a sequence of words as input and iteratively predicts the next token using a sliding window (Iraqi 2023).

In the research article "Sentiment Analysis in the Age of Generative AI" (Krugmann & Hartmann 2024), they did 3 experiments with different classifications. The first experiment is more related to this project which is binary and three-class sentiment classification. They did the zero-shot for llama2 and GPT models using different datasets. They also compared the results with models like BERT and RoBERTa which are fine-tuned. The average accuracies of llama2, GPT-4, BERT and RoBERTa for all 16 datasets are as follows 90.9%, 93.1%, 90.5% and 92.0% respectively. The best model being GPT-4 however it is not open-source. Nevertheless, for zero-shot testing, Llama 2 did better than expected.

On the other hand, for the paper "DialogueLLM: Context and Emotion Knowledge-Tuned Large Language Models for Emotion Recognition in Conversations" (Zhang, Wang, Wu, Tiwari, Li, Wang & Qin 2024), as the title suggested, it is about classifying emotion from dialogues. The datasets they used have instruction, video description, context and input like Figure[2.5] and the output will be single emotion. They used other models such as MTL, Llama2 to compare with their model DialogueLLM. The result they got for llama2 is significantly lower than the article above, average accuracy being 25.31% and f1-score being 21.91%. In which their best model being DialogueLLM with 61.4% and 60.52% respectively. This suggested that Llama2 is not suitable for extracting emotion from the dialogue based input.

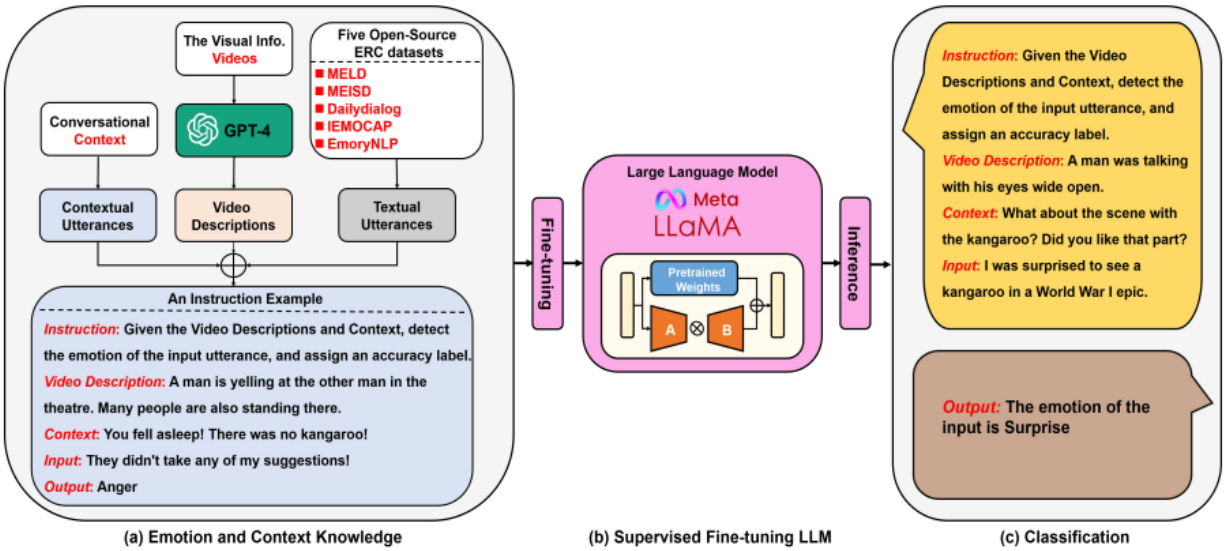

Figure 2.5: Overview of DialogueLLM fine-tuning and classification pipeline from the paper mentioned above

2.4 Overall

Most of the models, especially RoBERTa, Llama2 and GPT-2 shown that they might be suitable for downstream task such as emotion classification. Their accuracy and f1-score are mostly above 60%. Therefore, this project's purpose is to experiment with these models, fine-tuned them according to my dataset and, evaluate and test the model to find out which model is the best suited for the emotion detection task.

For miniLM, it may not have any paper regarding with emotion classification, the project's aim is to also find possible model for emotion classification. In which miniLM's methods and size and results of GLUE benchmark, it is included in this project's experimentation.

The following chapters will dive into the implementation of each model, the dataset, and how the results from the implementation will be compared and analysed as well as their methodologies and limitations will be explained.

Chapter 3

Technical Review

This chapter will go in depth about technical side of the project. It will start off with the dataset that being used. Then, it will discuss the resources that will be used and where the code and the models will be from. After then, will walk through each of the models that are implemented and the usage of libraries and methods.

3.1 Dataset

The dataset that is used to train and test is from *"Emotion Dataset for Emotion Recognition Tasks"* (Pandey 2021) from Kaggle which is based from *"CARER: Contextualized Affect Representations for Emotion Recognition"* (Saravia, Liu, Huang, Wu & Chen 2018) paper. It is an English Twitter message dataset, and it has 6 labels: sadness (0), joy (1), love (2), anger (3), fear (4) and surprise (5). It is for single-label emotion classification task, and it is already separated into train, test and validation dataset.

An example of 'train' dataset:

"label": 0,

"text": "im feeling quite sad and sorry for myself but ill snap out of it soon"

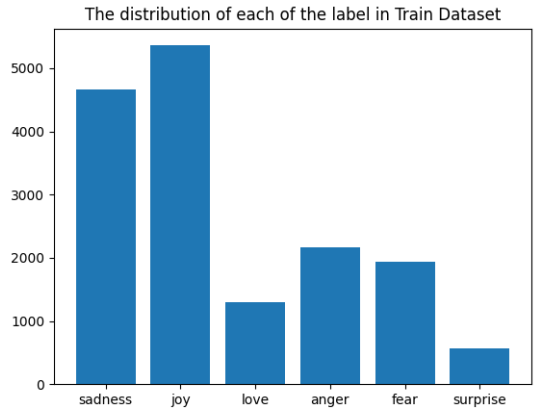

Figure 3.1: The graph for the distribution of each label in Train dataset

The distribution of each label for train dataset is not equal, the most being joy and the least being surprise. The Figure[3.1] will show the distribution of train dataset.

3.2 Methods

This section will dive deeper into each pre-trained model, and how it is implemented, what libraries have been used and why. The 4 pre-trained transformer models that are implemented for this research are as mentioned in literature review which are miniLM, Llama2, RoBERTa and GPT-2.

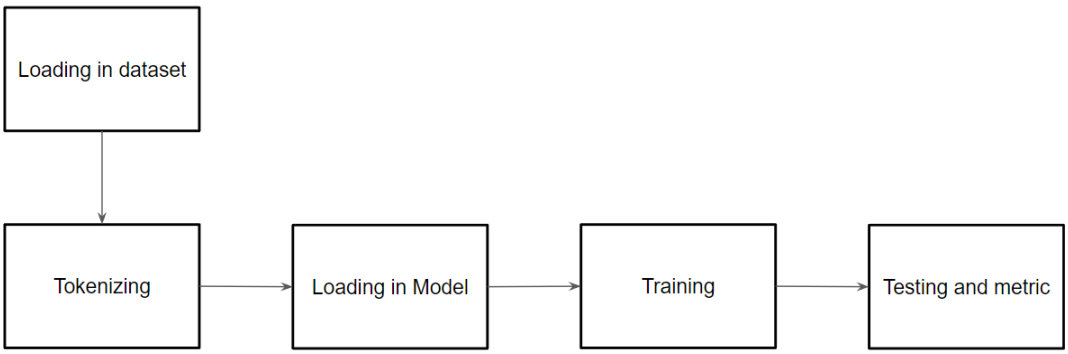

Figure 3.2: A pipeline of the overall flow of implementation

The Figure[3.2] shows the overall implementation of each model. The code for all the model will be followed roughly as the figure portrays and will be sectioned as such: Import libraries, Dataset, Model, Training and Testing/Evaluation. The implemented models will be run as Notebooks on either Google Colab or locally (in Vscode) depending on how big the model is.

The main benefit of using notebooks for the implementation is ability to break sections of code into blocks of code or 'cells' to make it easier to see and follow through. It also makes sectioned code easier to test and check for errors. For example, if the variables or code need to be checked to ensure their functionalities, or to check the features of datasets, cells can be added to verify and deleted after. Another benefit is that the cells can be 'markdown cells' which makes sectioning the code easier, can write titles and paragraphs overviewing each section, allowing anyone to understand the implementation quicker.

The pretrained models that I used for the implementation are all from Huggingface(HuggingFace n.d.*b*). Huggingface is a substantial platform for an AI community. It's also accommodating for any new keen learner with their work through videos and documents. It also provides many tools such as different models, datasets, accessibilities to any Ai applications made by the community, documents for specific libraries and methods, solutions and finally an ability to communicate or ask about any Ai related queries with other members in the community.

3.3 Main libraries

The following are the libraries used throughout all the models that are implemented, and the brief descriptions of their use and what they provide:

Torch: an essential python library which allows machine learning algorithms to run faster than if they were written in normal python. It also supports 'Cuda' to run models on the GPU.

Datasets: a library that provides easy access for loading in datasets from huggingface as well as your own datasets.

Transformers: give an ability to load in all the transformer models, provide tools such as Trainer, TrainingArguments and so on and pipeline from the huggingface.

Pandas: used for reformatting and organizing the datasets without actual changes to the datasets.

TQDM: displays progress of the model training and testing.

Numpy: a python library that supports and operates on large multidimensional arrays and matrices along with variety of mathematical functions.

3.4 Loading datasets

By using **"Datasets"** library, *load_ dataset, ClassLabel and Features* are imported. To load all the dataset from the folder, *Load_ dataset* is used.

However, the dataset come without the *ClassLabel*. It is useful to have in the dataset's features as the model can relate the label indices with the *ClassLabel*. Without it, the labels are just numbers and there are no correlation with the emotions or have any context for them. Therefore, the function for adding in the *ClassLabel* is implemented, and the code and its output can be seen in the listing(3.1).

```
    df = my_dataset["train"].to_pandas()
    labels = ['sadness','joy','love','anger', 'fear', 'surprise']
    ClassLabels = ClassLabel(num_classes=len(labels), names=labels)

    # Mapping Labels to IDs
    def map_label2id(example):
        example['label'] = ClassLabels.str2int(example['label'])
        return example

    my_dataset= my_dataset.map(map_label2id, batched=True)

    # Casting label column to ClassLabel Object
    my_dataset = my_dataset.cast_column('label', ClassLabels)


Output: {'text': Value(dtype='string', id=None), 'label': ClassLabel(names=['
    sadness', 'joy', 'love', 'anger', 'fear', 'surprise'], id=None)}
```

Listing 3.1: The code for adding *ClassLabel* and its results

3.5 Prompt and Prompt Engineering

A prompt is needed for training and testing the dataset as Llama2 model uses prompt engineering. Prompt engineering is a process of optimising prompts, queries and instructions for machine learning or AI models to output desirable results or predictions(Company 2024). Which means that a sentence such as "i didnt feel humiliated" is not a sufficient prompt for the model to train or test with. Thus, an additional text is added before every sample text in all the dataset. The text prompt added is *"Analyze the emotion of the texts in the square brackets, determine if it is*

sadness, joy, love, anger, fear, surprise, and return the answer as the corresponding sentiment label "sadness","joy","love","anger", "fear", "surprise"". The code for this can be seen below in the listing(3.2).

```
def generate_prompt(data_point):
    return f"""
            Analyze the emotion of the texts in the square brackets,
            determine if it is sadness, joy, love, anger, fear, surprise, and return the answer as
            the corresponding sentiment label "sadness","joy","love","anger", "fear", "surprise".

            [{data_point["text"]}] = {data_point["label"]}
            """.strip()

def generate_test_prompt(data_point):
    return f"""
            Analyze the emotion of the texts in the square brackets,
            determine if it is sadness, joy, love, anger, fear, surprise, and return the answer as
            the corresponding sentiment label 'sadness','joy','love','anger', 'fear', 'surprise'.

            [{data_point["text"]}] = """.strip()

X_train = pd.DataFrame(X_train.apply(generate_prompt, axis=1),
                       columns=["text"])
X_valid = pd.DataFrame(X_valid.apply(generate_prompt, axis=1),
                       columns=["text"])

y_true = X_test.label
X_test = pd.DataFrame(X_test.apply(generate_test_prompt, axis=1), columns=["text"])
```

Listing 3.2: The code for adding an additional prompt before the sentences

3.6 Preprocessing

Preprocessing the data before the diving into training and testing the models is necessity as stop words, upper or lowercase, punctuations and so on can hinder the model to learn or understand the input data. Fortunately, the dataset has already been preprocessed by removing punctuations, lowercasing all the words. The sample data can be seen in dataset section. Removing stop-word could affect the model performance as every word has to be contextualized and could make the model to perform better.

3.7 Pre-trained Models

A pre-trained model is a machine learning model that has undergone training on a vast amount of datasets and is adjustable for a particular downstream task. These models are frequently employed as an initial foundation for developing ML models, providing a starting set of weights and biases that can be fine-tuned for a particular task.

As more resources and time are needed for the full-training from scratch, this research opted to fine-tune the pre-trained transformer models for a specific downstream task like emotion classification. The following will discuss the implementation of each pre-trained models that the project used.

3.7.1 MiniLM

The English pre-trained model checkpoint that is loaded from huggingface is *'microsoft/MiniLM-L12-H384-uncased'*. It is the uncased 12-layer model with 384 hidden size distilled from an in-house pre-trained UniLM v2 model (it is not available for public) in BERT-base, 33 million parameters, and it is 2.7 times faster than BERT-Base (patrickvonplaten n.d.).

3.7.2 RoBERTa

This is another model from Meta, more specifically FacebookAI team. The pre-trained model used is *"FacebookAI/roberta-base"*, and the model is trained on five datasets such as BookCorpus, English Wikipedia, CC-News, OpenWebText and Stories(HuggingFace n.d.*a*).

3.7.3 GPT-2

GPT-2 has different sizes or versions such as GPT-Large, GPT-Medium, GPT-XL and GPT-2. The version that is used in this project is GPT-2 which is the smallest model with 124 million parameters. The model checkpoint is *"openai-community/gpt2"*(HuggingFace n.d.*e*).

3.7.4 Llama2

Llama 2 is a model from Meta, and it used to be the latest version of it when this project started however, LLama 3 is the newest addition to LLama family. LLama2 comprises a series of pre-trained and fine-tuned generative text models which ranges from 7 to 70 billion parameters(Meta 2023). In this project, we will be using the smallest model, 7 billion parameters model. The model checkpoint is *"meta-llama/Llama-2-7b-hf"*(Meta 2023).

3.8 Tokenisation

Tokenisation is a method where the input data are separated into small units called 'token'. These tokens can either be a letter, a character or a word. Different models have different ways of tokenisation as they are depends on the model's vocabulary, embedding and their understandings of the input. There are also special tokens which can either indicate the beginning and the end of the sentences, blank spaces and so on.

3.8.1 MiniLM

For MiniLM, all words in each sentence are separated into tokens in terms of white spaces. The special tokens are "[CLS]" and "[SEP]" which are the beginning of the sentence and the end of sentence respectively. Surprisingly, the tokenizer for MiniLM, can detect contraction of certain words such as "didnt" into "didn", "t" which rest models' tokenizer seem not be able to. **Truncation** is set to true as the max length of tokens that the model can have is 512. The sample list of tokens is as shown below and the code for initialising the tokenizer can be seen in listing(3.3).

Sample List of Tokens from Train dataset:

['[CLS]', 'i', 'didn', '##t', 'feel', 'humiliated', '[SEP]']

```
model_ckpt = "microsoft/MiniLM-L12-H384-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)

def tokenize_text(examples):
  return tokenizer(examples["text"], truncation=True, max_length=512)
```

Listing 3.3: The code for initialising tokenizer for MiniLM

3.8.2 RoBERTa

As for RoBERTa, the beginning and end tokens are "<s>" and "</s>" respectively and additional special token is "Ġ" which indicate white spaces before the word. "Ġ" is needed as some of the long words that are not in the vocabulary will be separated into smaller words or characters. The max token length is the same as MiniLM which is 512. The tokenizer code is the same as the listing(3.3) above apart from the model checkpoint. The sample list of tokens is shown below.

Sample List of Tokens from Train dataset:

['<s>', 'i', 'Ġdidnt', 'Ġfeel', 'Ġhumiliated', '</s>']

3.8.3 GPT-2

Tokenisation for GPT-2 is similar to RoBERTa without the special tokens for beginning and end of the sentence. It only has "Ġ" to indicate the white spaces. The max length for the tokens is 1024 which is more than both MiniLM and RoBERTa model. The GPT2Tokenizer is used to tokenise the input and the tokens are padded at the end of the sentence. The code for this is listed in the listing(3.4).

Sample List of Tokens from Train dataset:

['i', 'Ġdidnt', 'Ġfeel', 'Ġhumiliated']

```
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
tokenizer.pad_token = tokenizer.eos_token
model.config.pad_token_id = model.config.eos_token_id

def tokenize_text(examples):
    return tokenizer(examples["text"], truncation=True, max_length=512)
```

Listing 3.4: The code for initialising tokenizer for GPT-2

3.8.4 LLama2

The special tokens for the beginning of the sentence indication is the same as RoBERTa. The differences are instead of "Ġ" to indicate the white spaces, "_" is used. Another difference is the different vocabulary, the sample below shows that the word "humiliated" is split into 3 tokens which means that the word is not in the vocabulary of this model. The max length for this model is 1000000000000000019884624838656 and the padding for the tokens are added at the end of the sentences. Listing(3.5) shows that initialising of the tokenizer and the padding.

Sample List of Tokens from Train dataset:

['<s>', '_i', '_didnt', '_feel', '_hum', 'ili', 'ated']

```
    tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
    tokenizer.ad_token = tokenizer.eos_token
    tokenizer.padding_side = "right"
```

Listing 3.5: The code for initialising tokenizer for LLama2

3.9 Loading Models

Initialising the model for both MiniLM and RoBERTa are the same, using *AutoModelForSequenceClassification* from **Transformers** library. The code for it is listed below in listing(3.6). Adding in id2label and label2id is for the model to understand the relation between the string labels and index labels. As for GPT-2, *GPT2ForSequenceClassification* class is used and the parameters for it is the same as MiniLM and RoBERTa. The pad token of model for the model is needed to initialised as there is a mismatch in length from tokenizer and the model. The code can be seen in listing(3.7)

```
model = AutoModelForSequenceClassification.from_pretrained(model_ckpt,
                                                           num_labels=6,
                                                           id2label=id2label,
                                                           label2id=label2id)
```

Listing 3.6: The code for initialising model for both MiniLM and RoBERTa

```
model = GPT2ForSequenceClassification(configuration).from_pretrained("gpt2",
                                                           num_labels=6,
                                                           id2label=id2label,
                                                           label2id=label2id)
# set the pad token of the model's configuration
model.config.pad_token_id = model.config.eos_token_id
```

Listing 3.7: The code for initialising model for GPT-2

Finally, for Llama2, quantization is necessary for the model to be able to train and test as the model is massive. Quantization is a process of reducing the model size into smaller size by converting model weights from high-precision floating-point representation to low-precision floating-point like 16-bit or 8-bit(Deci 2024). This means that memory consumption is lessened while train LLMs with quantization. Therefore, **BitsandBytes** quantization method is used in this project to quantize the Llama2 model. 4-bit quantization is set be able to train with QLoRA which will be discussed in below section, saving memory without compromising the performance(Dettmers n.d.). The code demonstration is in the listing(3.8) below.

```
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=compute_dtype,
    bnb_4bit_use_double_quant=True,
)

model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    use_safetensors=True,
    quantization_config=bnb_config,
    trust_remote_code=True,
    device_map="auto"
)
```

Listing 3.8: The code for initialising model and BitsandBytes configuration for Llama2

3.10 Training Arguments and Trainer

TrainingArguments and Trainer from **Transformers** library is used to set the hyperparameters as well as for training the model easier. The TrainingArguments for RoBERTa and MiniLM are the same as well as the trainer. Learning rate is 0.00002 and the weight decay is 0.01, they need to be very low so that the checkpoint weights will not change very quickly and the model overfit. Therefore, lower learning rate and weight decay will allow the model to arrive to the optimal solutions smoothly. This is the same for both Llama2 and GPT-2 model. The loss function and optimier for all the models are the same as well which are *CrossEnthropyLoss* and *adamW_ torch*.

For Trainer, *WeightLossTrainer* class will be used from Trainer module. This is so that we could initialised our own class weights to make up for the imbalance between class labels in the train dataset, as seen in figure(3.1) in Dataset section above. This is done by increasing the class weights of the rare classes and lowering the other common classes, in the listing(3.9). This is then put in the *CrossEnthropyLoss* function and the code is listed in the listing(3.9).

```
class_weights = (1 - (dataset_df["label"].value_counts().sort_index() / len(
    dataset_df))).values
class_weights = torch.from_numpy(class_weights).float()

class WeightedLossTrainer(Trainer):
    def compute_loss(self, model, inputs, return_outputs=False):
      # Feed inputs to model and extract logits
      outputs = model(**inputs)
      logits = outputs.get("logits")
      # Extract labels
      labels = inputs.get("labels")
      #Define loss function with class weights
      loss_func=nn.CrossEntropyLoss(weight=class_weights)
      # Compute loss
      loss = loss_func(logits, labels)
      return (loss, outputs) if return_outputs else lossslstlisting
```

Listing 3.9: The code for WeightedLossTrainer in RoBERTa and MiniLM models

As for GPT-2, The *TrainingArguments* are the same with RoBERTa and MiniLM however, instead of *WeightedLossTrainer*, standard *Trainer* is implemented.

Finally, Llama2 is difference as it LLMs and the model is very huge to run normally. Thus, like it is discussed in the above section, *BitsandBytes*, as well as *PEFT* and *SFTTrainer* library to help train LLama2 efficiently(Huggingface n.d.*c*).

3.10.1 PEFT

PEFT or Parametric-Efficient Fine-tuning is a method which reduces down a huge amount of parameters into a smaller desirable size which makes the large models to be efficiently adapt to any downstream tasks(Huggingface n.d.*c*). This helps reducing the cost of storage and computation significantly(Huggingface n.d.*c*). *LoraConfig* from *PEFT* is used in this implementation. LoRA or Low-rank adaptation is a technique that allow the user to reduce significantly the number of original model's parameters to run faster and be efficient(HuggingFace n.d.*d*). *lora_ r* or LoRA rank is an inner dimension of the low-rank matrices; a higher rank means allowing more trainable parameters(HuggingFace n.d.*d*). the rank was set to 16 which is the middle ground as it is range from 4 to 64, so it will not take too much GPU consumptions(HuggingFace n.d.*d*). The figure(3.3) summarised the structure of LoRA. The code for the configuration is shown in the listing(3.10).

```
    lora_alpha = 32
    lora_dropout = 0.05
    lora_r = 16

    peft_config = LoraConfig(
        lora_alpha = lora_alpha,
        lora_dropout = lora_dropout,
        r = lora_r,
        bias="none",
        task_type="CAUSAL_LM"
    )
```

Listing 3.10: The code for LoRA configuration

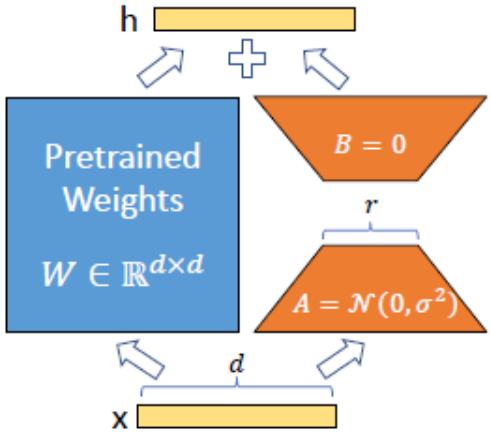

Figure 3.3: The diagram of LoRA technique to reduce down trainable parameters for efficiency

3.10.2 SFTTrainer

Supervised Fine-Tuning Trainer, in short for SFTTrainer, is a trainer optimised for fine-tuning pre-trained models with smaller datasets to train(Mudadla 2023). It is mainly for supervised learning tasks(Mudadla 2023). It also trains faster and efficiently use memory while training as it is compatible with *PEFT*(Mudadla 2023). These attributes are highly beneficial for this part of implementation as Llama2 will use significant amount of memory and will train slower than any models that is implemented in this project. The dataset that the project used is also small therefore using SFTTrainer instead of standard Trainer will be essential for training Llama2. The implementation of SFTTrainer is as shown in the listing(3.11).

```
    trainer = SFTTrainer(
        model=model,
        args=training_arguments,
        train_dataset=train_data,
        eval_dataset=val_data,
        peft_config=peft_config,
        dataset_text_field="text",
        tokenizer=tokenizer,
        max_seq_length=1024,
        packing=False,
        dataset_kwargs={
            "add_special_tokens": False,
            "append_concat_token": False,
        }
```

```
    )
```

Listing 3.11: The code for SFTTrainer

3.11 Testing Metrics

After training the model, testing is needed to be done to see how the models perform after the training. Test dataset is used to test the model. There are two methods used to test the models, *predict* and *evaluate* functions from *Trainer* class, and functions implemented from scratch. The *predict* functions from *Trainer* only provides the logits of the prediction outputs and metrics such as overall accuracy, f1-score and other metrics which are not of used for the analysis. For the *evaluate* function from *Trainer*, it outputs metrics only.

The *predict* funtion created from scratch will output index of the labels instead of the logits. The *evaluate* function will print out overall accuracy, accuracies for each labels, classification report which includes precision, recall, f1-score and support of each labels as well as the average, and confusion matrix. Those metrics are from *sklearn.metrics* library. The code for both functions are listed below as listing(3.12) and (3.13) respectively.

```
    def predict(model, tokenizer):
    y_pred = []

    for i in tqdm(range(len(X_test))):

        prompt = X_test.iloc[i]["text"]
        pipe = pipeline(task="text-classification",
                        model=model,
                        tokenizer=tokenizer,
                       )
        result = pipe(prompt)
        answer = result[0]['label'].split("=")[-1]
        if "sadness" in answer:
            y_pred.append(0)
        elif "joy" in answer:
            y_pred.append(1)
        elif "love" in answer:
            y_pred.append(2)
        elif "anger" in answer:
            y_pred.append(3)
        elif "fear" in answer:
            y_pred.append(4)
        else:
            y_pred.append(5)

    return y_pred
```

Listing 3.12: The code for predict function implemented

```
def evaluate(y_true, y_pred):

    # Calculate accuracy
    accuracy = accuracy_score(y_true=y_true, y_pred=y_pred)
    print(f'Accuracy: {accuracy:.3f}')

    # Generate accuracy for each labels
    unique_labels = set(y_true)  # Get unique labels output: {0,1,2,3,4,5}

    for label in unique_labels:
        # will output a list of the index of one emotion at a time
        label_indices = [i for i in range(len(y_true))
```

```
                            if y_true[i] == label]
        # will output the list of one emotion
        label_y_true = [y_true[i] for i in label_indices]
        # label_y_true = [label for i in range(len(y_true))]
        # will output list of the predicted emotion in the same order as
    label_y_true
        label_y_pred = [y_pred[i] for i in label_indices]
        accuracy = accuracy_score(label_y_true, label_y_pred)
        print(f'Accuracy for label {label}: {accuracy:.3f}')

    # Generate classification report
    class_report = classification_report(y_true=y_true, y_pred=y_pred)
    print('\nClassification Report:')
    print(class_report)

    # Generate confusion matrix
    conf_matrix = confusion_matrix(y_true=y_true, y_pred=y_pred, labels=['
    sadness', 'joy', 'love', 'anger', 'fear', 'surprise'])
    print('\nConfusion Matrix:')
    print(conf_matrix)
```

Listing 3.13: The code for evaluate function implemented

Chapter 4

Test Analysis

The analysis of testing and training will be discussed in this chapter. This includes graphs, tables, confusion matrix and so on to support any training and testing done for all the models in the implementation.

4.1 MiniLM

Before the training, I tried testing the model with the test dataset. The *predict* and *evaluate* function used are not from *Trainer* class. The result of zero-shot testing is rather surprising as it only predict "surprise" labels, making the accuracy to 3.33%. Other metrics are not noteworthy to mention as "surprise" label is the only one with the metrics. Therefore, the model is not suitable for zero-shot testing and further need to fine-tune and train the model for this project's downstream task.

The training hyperparameters are as shown in table(4.1) and the numbers of epochs trained are 6, 12 and 18. The hyperparameters are not changed for any of the epochs. The learning rate and the weight decay are set very low so that the weights from model checkpoint change so quickly. The higher the learning rate, the faster the convergence rate occurs and could cause overfitting.

Hyperparameters:

Hyperparameters	Values
Loss Func	CrossEnthropy (default)
Optimiser	adamW_torch (default)
Learning rate	2e-5
Batch size	64
Weight Decay	0.01

Table 4.1: Hyperparameters for MiniLM model

Training 6 epochs did not log any training loss during the training however, at the last epoch, it ended with 0.805 loss. Nonetheless, in epoch 12 and 18 trainings, the training losses and validation losses are logged and can be seen in figure(4.1) and (4.2). The validation loss started off with more than 1 however, in both figures, it was reduced down to less than 0.2. The losses between training and validation are similar which means that the model is neither over nor under fitted.

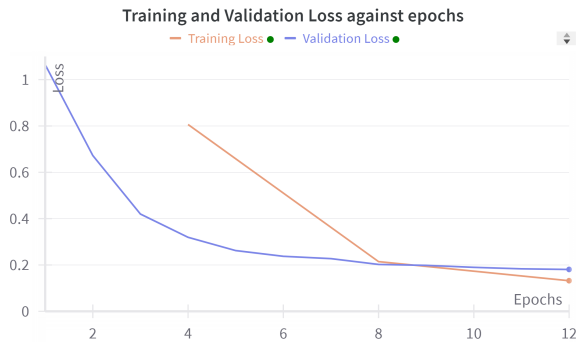

Figure 4.1: The graph of MiniLM's training and validation loss with 12 epochs

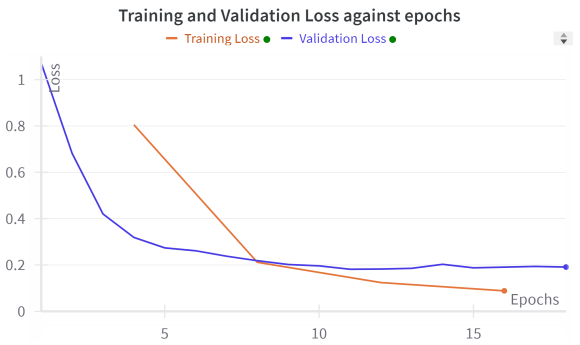

Figure 4.2: The graph of MiniLM's training and validation loss with 18 epochs

Furthermore, the accuracy and f1-score during training with 12 and 18 epochs are shown in the figure(4.3) and (4.4). In both figures, the highest accuracy score reach is 0.93 and the same for f1-score as well. However, there is a small dip for epoch 18, making it 0.928 for both accuracy and f1-score.

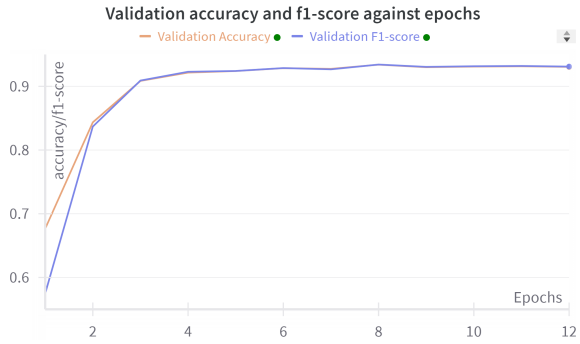

Figure 4.3: The graph of MiniLM's validation accuracy and f1-score during training with 12 epochs

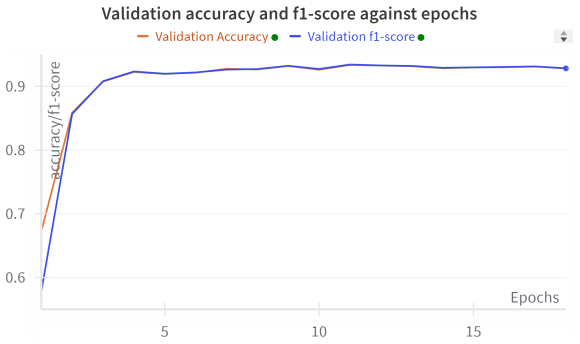

Figure 4.4: The graph of MiniLM's validation accuracy and f1-score during training with 18 epochs

The table(4.2) shows all the average of metrics for each epoch ran. It seems that the numbers of epochs do not affect the performance of the model as there is no big changes in any of the metrics. The best accuracy and f1-score being 0.93 and 0.89 respectively.

Epochs	Overall Precision	Overall Recall	Overall F1-score	Overall Accuracy
6	0.90	0.86	0.88	0.92
12	0.90	0.87	0.89	0.93
18	0.90	0.87	0.88	0.93

Table 4.2: The table for the overall values of every metrics in each epoch

The following diagrams are confusion matrices for each epoch, figure(4.5), (4.6) and (4.7). Overall, the model prediction for "sadness", "joy" and "fear" labels improve as the numbers of epochs increase. The most wrong prediction done for all across the epochs is when the model predicted

"joy" but the true label is "love". The prediction improved as the model was trained with more epochs reducing from 55 to 35 at 18 epochs. On the other hand, the increase of label "love" was predicted instead of "joy". It increased from 4 to 17 at epoch 18.

Nevertheless, the model did well in differentiating positive and negative emotions as "joy" and "love" labels were not predicted too much as "anger", "fear" or "sadness", and vice versa.

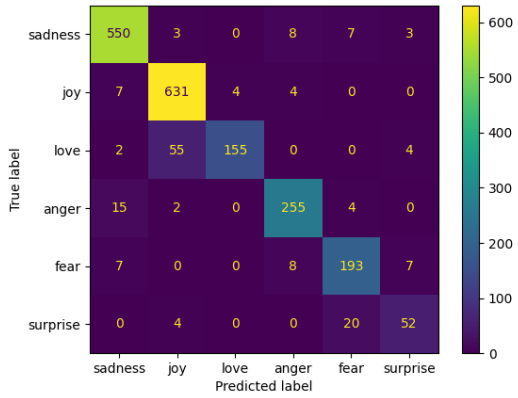

Figure 4.5: Confusion Matrix 6 epochs

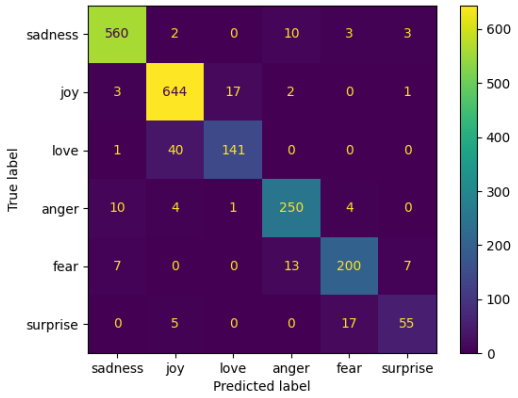

Figure 4.6: Confusion Matrix 12 epochs

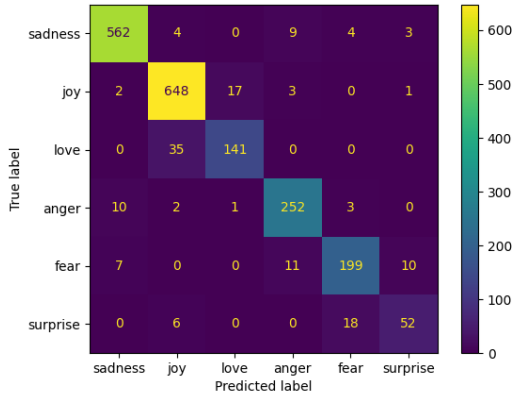

Figure 4.7: Confusion Matrix 18 epochs

4.2 RoBERTa

The zero-shot testing is done for RoBERTa as well and the accuracy was 32% making it 10 times better than MiniLM. Even so, the predictions the model made is still worse as it only predict either label "joy" or "surprise" making accuracy for "joy", 91% and 9.1% for "sadness" label's accuracy. The overall f1-score is around 9%. This means that RoBERTa is also a model which needs to be fine-tuned before testing the model.

The hyperparameters for training RoBERTa model is the same as MiniLM model. The table of hyperparameters is as listed in table(4.1).

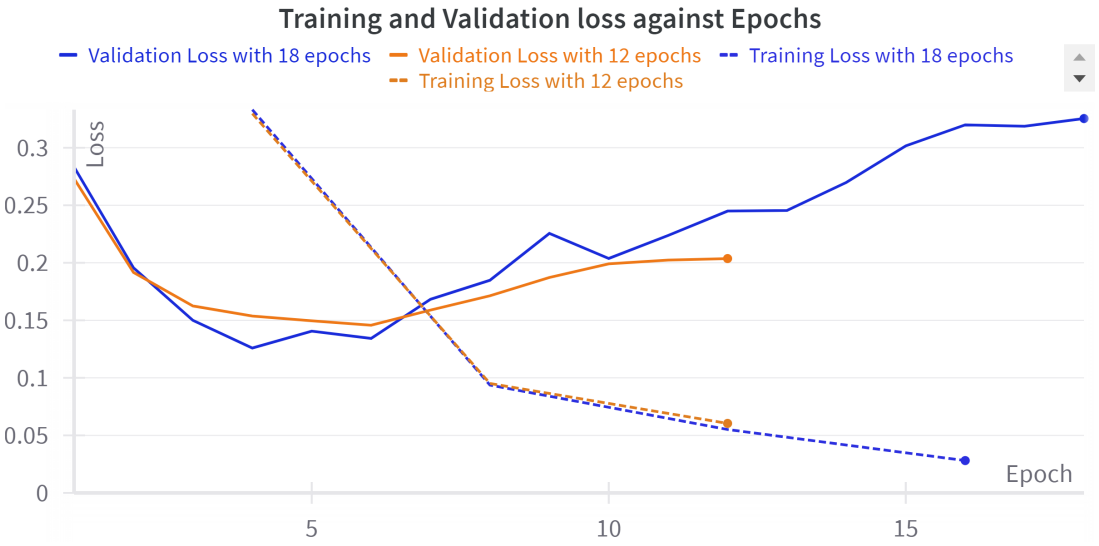

Figure 4.8: The graph for Training and Validation loss for both 12 and 18 epochs across epochs

The line graph in the figure(4.8) shows the training and validation losses during the training. The training losses are the similar for both epochs and reaching to the lowest of around 0.0281. On the other hand, the validation loss increases as the number of epochs increases and the highest being 0.3254 at 18 epochs. The training loss and validation loss converges as the number of epochs increases which means that the more the model trained, the more it will overfit.

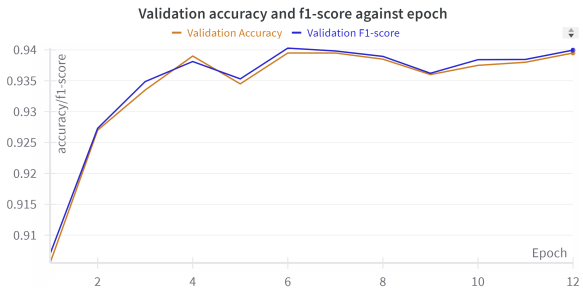

Figure 4.9: The graph of RoBERTa's validation accuracy and f1-score during training with 12 epochs

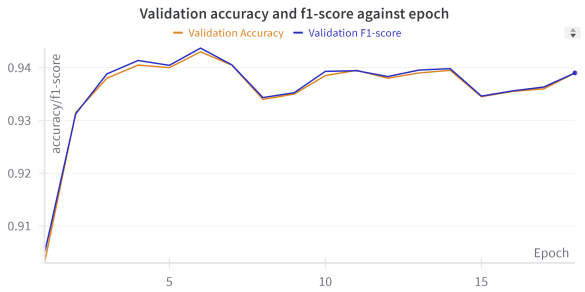

Figure 4.10: The graph of RoBERTa's validation accuracy and f1-score during training with 18 epochs

The Figure(4.9) and (4.10) show that accuracy and f1-score are similar across epochs. Both of the graphs also shows that the accuracy and f1-score could go as high as around 94%.

Epochs	Overall Precision	Overall Recall	Overall F1-score	Overall Accuracy
6	0.87	0.9	0.88	0.927
12	0.88	0.9	0.89	0.929
18	0.89	0.9	0.89	0.93

Table 4.3: The table for the overall values of every metrics in each epoch

The table(4.3) shows that the best epochs out of all three is epoch 18 as all overall metrics are higher than the other two.

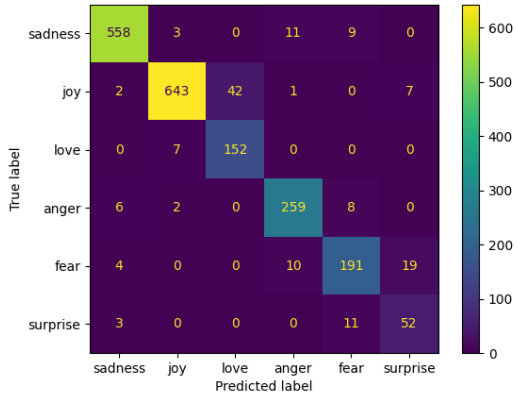

Figure 4.11: Confusion Matrix 6 epochs

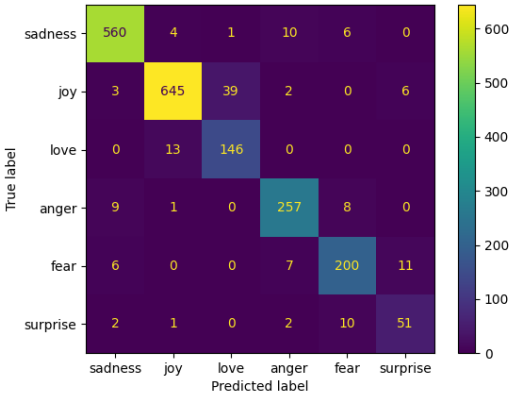

Figure 4.12: Confusion Matrix 12 epochs

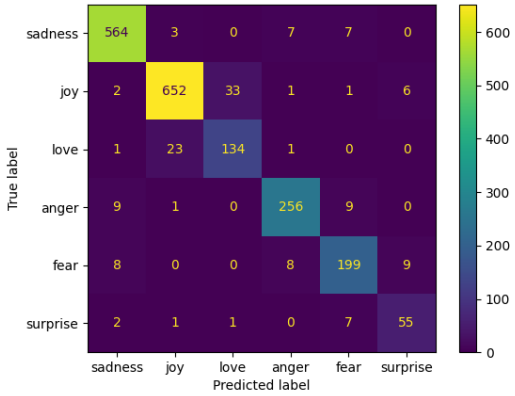

Figure 4.13: Confusion Matrix 18 epochs

The model is good at classifying the labels such as "sadness", "joy", "anger" and "surprise" as the number for each label correctly predicted increases. The diagrams of confusion matrix for each number of epochs can be seen in figure(4.11), (4.12) and (4.13). The label "joy" and "love" was mixed up, similar to MiniLM model however, the model predicted "joy" instead of "love" more as the number of epochs increased.

4.3 GPT-2

As for GPT-2, the prediction's accuracy is a bit better than the models above when it was tested before the training. The accuracy overall for it is 34.6% however, the label that is mainly predicted is "joy". The f1-score is 9%. Thus, GPT-2 is also the model which is not good for zero-shot testing.

Hyperparameters:

Hyperparameters	Values
Loss Func	CrossEnthropy (default)
Optimiser	adamW_torch (default)
Learning rate	2e-5
Batch size	1
Weight Decay	0.01

Table 4.4: Hyperparameters for GPT-2 model

The table(4.4) shows the hyperparameters for GPT-2's training arguments. The batch size is 1 as more than one uses up the memory that it make the model not be able to train. The model is also only train for 2 epochs as it will take more than 8 hours if it was trained on Google Colab which has the limited GPU memory therefore GPU ran out before it is fully trained. If it was locally trained it took nearly a day. Thus, training 2 epochs will be optimal option for GPT-2 model.

The model training loss started off with 5 and gradually lower in each logging steps within each epoch and at the end reaches to around 0.479 and as for validation loss, the model started off with around 0.298 when it was logged and ended with about 0.246. The model did not seem to overfit during the training with 2 epochs. However, it cannot be said if it trained for more than 2 epochs.

Epochs	Overall Precision	Overall Recall	Overall F1-score	Overall Accuracy
2	0.89	0.88	0.88	0.931

Table 4.5: The table for the overall values of every metrics in each epoch

Labels	Accuracy
surprise	0.727
fear	0.866
joy	0.967
anger	0.927
love	0.805
sadness	0.972

Table 4.6: Accuracy for each label

	Precision	Recall	F1-score
anger	0.93	0.93	0.96
fear	0.91	0.87	0.78
joy	0.94	0.97	0.79
love	0.88	0.81	0.83
sadness	0.96	0.97	0.83
surprise	0.72	0.73	0.83

Table 4.7: Precision, recall and f1-score for each label

Nonetheless, the model did well with emotion classification task as its overall accuracy is 93% and macro average f1-score of 88% with just 2 epochs of training. All the accuracies for each label is mostly above 80% with an exception to "surprise" label which has an accuracy of 72.7%. It was the same for f1-score of each labels as well. The result can be seen in the table(4.5), (4.6) and (4.7) above.

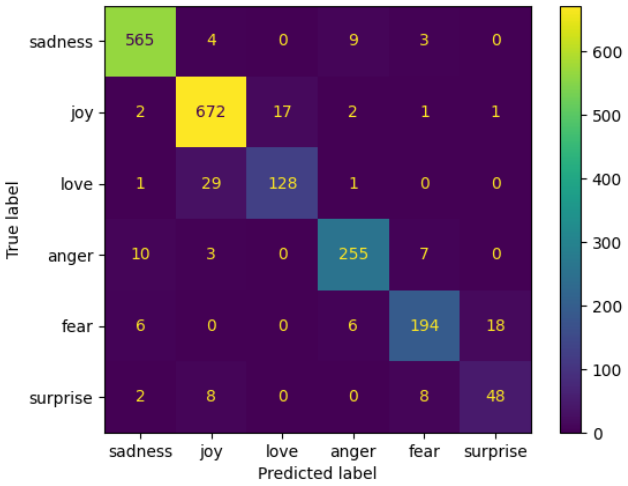

Figure 4.14: Confusion Matrix for GPT-2 model with 2 epochs

Furthermore, the confusion matrix can be seen in figure(4.14). The most mistake that the model make is the same as MiniLM model. The model was predicting "joy" label when the true label is "love" the same mistake is done in the opposite but a bit less for it. Another mistake that it make is predicting label "surprise" when it should be the label "fear".

4.4 Llama2

Llama2 is the biggest model that this project has done in terms of parameters and weights. Therefore, I was only able to do one training for Llama2. Firstly, the zero-shot testing was done like all the models that the project implemented. The overall accuracy is the best out of all the models, which is 49.4%. Which is also the same for overall f1-score with 33%, it can be seen in table(4.8). However, it predicted outside the range of the labels such as "angst", "disgust", "i guess feelings" and some numbers. The number of samples predicted such labels are 37.

Epochs	Overall Precision	Overall Recall	Overall F1-score	Overall Accuracy
zero-shot	0.43	0.34	0.33	0.494
3	0.05	0.17	0.08	0.290

Table 4.8: The table for the overall values of every metrics in zero-shot testing and 3 epochs

Labels	Accuracy for zero-shot	Accuracy for 3 epochs
surprise	0.167	1
fear	0.080	0
joy	0.167	0
anger	0.516	0
love	0.415	0
sadness	0.706	0

Table 4.9: Accuracy for each label both zero-shot and 3 epochs training

	Precision	Recall	F1-score
anger	0.44	0.52	0.47
fear	0.78	0.08	0.15
joy	0.74	0.49	0.59
love	0.26	0.42	0.32
sadness	0.48	0.71	0.57
surprise	0.31	0.17	0.22

Table 4.10: Precision, recall and f1-score for each label

Table(4.9) and (4.10) show that the accuracy, precision, recall and f1-score of each label. The best accuracy is the label "sadness" as well as the highest recall, and the best f1-score is the label "joy". The highest precision is the label "fear".

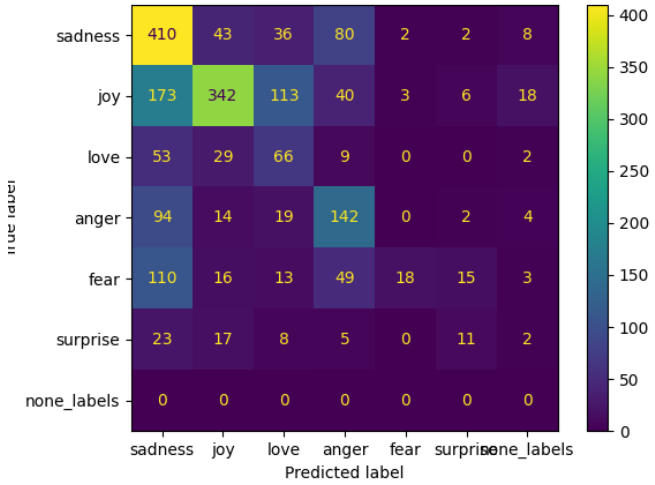

Figure 4.15: Confusion Matrix for Llama2 zero-shot

The none_labels in figure(4.15) shows that the labels which are not predicted within the bounds of the given labels. It can be seen that most of the labels should have predicted as "joy" but predicted some random words, number or beginning part of sentences. Apart from them, the model is still not the best at detecting most of the testing sample incorrectly. However, it seemed that it mostly did not predict negative emotions as positive and vice versa.

Chapter 5

Evaluation

5.1 Evaluation of Each Model

5.2 Overall Evaluation

5.3 Future Work

Chapter 6

Statement of Ethics

Chapter 7

Conclusion of this Project

Conclusion

Bibliography

akrasia, n. (n.d.).
URL: https://www.oed.com/view/Entry/240257?redirectedFrom=akrasia

Ameer, I., Bölücü, N., Siddiqui, M. H. F., Can, B., Sidorov, G. & Gelbukh, A. (2023), 'Multi-label emotion classification in texts using transfer learning', *Expert Systems with Applications* **213**, 118534.
URL: https://www.sciencedirect.com/science/article/pii/S0957417422016098

Amrullah, A. (2023), 'Classifying emotions in sentence text using neural networks'.
URL: https://www.analyticsvidhya.com/blog/2023/05/classifying-emotions-in-sentence-text-using-neural-networks/#:~:text=The%20main%20objective%20of%20doing,accuracy%20of%20machine%20translation%20systems.

Company, M. . (2024), 'What is prompt engineering?'.
URL: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-prompt-engineering

Deci (2024), 'Model quantization and quantization-aware training: Ultimate guide'.
URL: https://deci.ai/quantization-and-quantization-aware-training/#:~:text=What%20is%20Model%20Quantization%3F,%2Dbit%20or%208%2Dbit.

Dettmers, T. (n.d.), 'Bitsandbytes'.
URL: https://huggingface.co/docs/bitsandbytes/main/en/index

Hashemi-Pour, C. & Lutkevich, B. (2024), 'What is the bert language model?: Definition from techtarget.com'.
URL: https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model#:~:text=BERT%2C%20which%20stands%20for%20Bidirectional,calculated%20based%20upon%20their%20connection.

HuggingFace (n.d.*a*), 'Facebookai/roberta-base · hugging face'.
URL: https://huggingface.co/FacebookAI/roberta-base

HuggingFace (n.d.*b*), 'Huggingface homepage'.
URL: https://huggingface.co/

Huggingface (n.d.*c*), 'Huggingface/peft: State-of-the-art parameter-efficient fine-tuning.'.
URL: https://github.com/huggingface/peft

HuggingFace (n.d.*d*), 'Lora'.
URL: https://huggingface.co/docs/diffusers/main/en/training/lora

HuggingFace (n.d.*e*), 'openai-community/gpt2 · hugging face'.
URL: https://huggingface.co/openai-community/gpt2

IBM (2024), 'What is natural language processing?'.
URL: https://www.ibm.com/topics/natural-language-processing

Iraqi, M. (2023), 'Comparing the performance of llms: A deep dive into roberta, llama-2, and mistral-7b for disaster...'.
URL: https://medium.com/@mehdi.iraqui/comparing-the-performance-of-llms-a-deep-dive-into

Jorge, L. (2023), 'Roberta vs. gpt: A comprehensive comparison of state-of-the-art language models, with expert insights from cronj'.
URL: https://medium.com/@livajorge7/roberta-vs-86ee82a44969#:~:text=Pretraining%20Objectives%3A%20RoBERTa%20is%20pretrained,masked%20words%20in%20a%20sentence.

Krugmann, J. O. & Hartmann, J. (2024), 'Sentiment analysis in the age of generative ai - customer needs and solutions'.
URL: https://link.springer.com/article/10.1007/s40547-024-00143-4#Tab1

Meta (2023), 'Meta-llama/llama-2-7b-hf · hugging face'.
URL: https://huggingface.co/meta-llama/Llama-2-7b-hf

Mudadla, S. (2023), 'Difference between trainer class and sfttrainer (supervised fine tuning trainer) in hugging face?'.
URL: https://medium.com/@sujathamudadla1213/difference-between-trainer-class-and-sfttrai

Nedilko, A. (n.d.), 'Generative pretrained transformers for emotion detection in a code-switching setting'.
URL: https://aclanthology.org/2023.wassa-1.61/

Pandey, P. (2021), 'Emotion dataset for emotion recognition tasks'.
URL: https://www.kaggle.com/datasets/parulpandey/emotion-dataset/data

patrickvonplaten (n.d.), 'Microsoft/minilm-l12-h384-uncased · hugging face'.
URL: https://huggingface.co/microsoft/MiniLM-L12-H384-uncased

Saravia, E., Liu, H.-C. T., Huang, Y.-H., Wu, J. & Chen, Y.-S. (2018), CARER: Contextualized affect representations for emotion recognition, *in* 'Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing', Association for Computational Linguistics, Brussels, Belgium, pp. 3687–3697.
URL: https://www.aclweb.org/anthology/D18-1404

Sharma, D. (2022), 'A gentle introduction to roberta'.
URL: https://www.analyticsvidhya.com/blog/2022/10/a-gentle-introduction-to-roberta/

Uwa (2023), 'Science of emotion: The basics of emotional psychology: Uwa'.
URL: https://online.uwa.edu/news/emotional-psychology/

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2023), 'Attention is all you need'.
URL: https://arxiv.org/abs/1706.03762

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N. & Zhou, M. (2020), 'Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers'.
URL: https://arxiv.org/abs/2002.10957

Zhang, Y., Wang, M., Wu, Y., Tiwari, P., Li, Q., Wang, B. & Qin, J. (2024), 'Dialoguellm: Context and emotion knowledge-tuned large language models for emotion recognition in conversations'.