



UNIVERSITÉ
DE LORRAINE



Institut des
sciences du Digital
Management & Cognition

COMPOSANTE DE L'UNIVERSITÉ DE LORRAINE

UNIVERSITÉ DE LORRAINE

LINSTITUT DES SCIENCES DU DIGITAL MANAGEMENT
COGNITION

SUPERVISED PROJECT

Emoji-Based Hate Speech Detection on Social Media

Author:

Nalin SRUN
Shuwei LYU
Shu ZHANG

Supervisor:

Dr. Gaël GUIBON
Dr. Christophe CERISARA

*A final report of the Supervised Project
Academic year 2022-2023*

June 22, 2023

Abstract

This project focuses on the classification of emoji-based hate speech using both machine learning and deep learning approaches. The goal is to develop models that can accurately identify hate speech from text data specifically with emojis. The project includes data scraping from social media, data annotation, data preprocessing, feature extraction, model training for both binary and muticlass classification using various machine learning algorithms such as SVM, LightGBM, MLP classifier, and MultinomialNB, Deep learning models specifically CNN and Bi-LSTM, are also implemented and evaluation based on the quantitative and qualitative analyses are conducted to assess the performance and interpretability of the models, especially for qualitative analyses we try to use data from other domain (online open-source dataset) along with customized texts as inputs to analyze the result more deeply. For machine learning, the results indicate that SVM, LightGBM, and MLP classifiers perform well in binary classification, while MultinomialNB is effective for multiclass classification. For the deep learning model, we have made various comparisons between the same model by using multiple configurations for training, and compared to other models; we have also used FastText pre-trained embedding model comparing to the result of using only deep learning model embedding layer; additionally, we have conducted a comparison between the performance result of the model with various amount of training set; finally, we could find the best model for our task which is Bi-LSTM using FastText pre-trained embedding model which can achieve about 63% of F1-Score compared to CNN; moreover, we also try to use the pre-trained model from hugging face to apply to our multiclass classification. Problem and discussion are also provided such as overfitting, limited sample size, and feature selection in the conclusion section as well. Eventually, future work includes dealing the model with Emoji2vec and conducting multiple training runs for improved stability and performance.

Contents

Abstract	ii
Introduction	v
1 Dataset Creation	1
1.1 Data from Kirk	1
1.2 Data Scraped from Social Media	1
1.3 Illustration of the Scraped Dataset	2
1.3.1 Dataset Annotation	2
2 Methodology	7
2.1 Machine Learning	7
2.1.1 Dataset Preparation	7
2.1.2 Text Pre-processing	8
2.1.3 Feature Extraction	9
2.1.4 Proposed Machine Learning Models	9
2.2 Deep Learning	10
2.2.1 Data Preparation	10
2.2.2 Feature Extraction	11
2.2.3 Model Architecture of Binary Classification	11
Bidirectional Long Short-Term Memory (Bi-LSTM)	11
Convolutional Neural Network (CNN)	12
2.2.4 Model Architecture of Multiclass Classification	13
3 Experiment	14
3.1 Machine Learning	14
3.1.1 Binary Classification	15
3.1.2 Multiclass Classification	16
3.2 Deep Learning	19
3.2.1 Binary Classification	19
3.2.2 Multiclass Classification	19
3.2.3 Tuning Hyperparameter for Deep Learning Models	20
Binary Classification	20
Computation Time	23
Multiclass Classification	24
4 Qualitative Analysis	25
4.1 Machine Learning Binary Classification	25
4.2 Deep Learning Binary and Multiclass Classification	25

5 Conclusion	28
5.1 Problem and Discussion	28
5.2 Future Work	30
5.3 Publication	30
Bibliography	31

Introduction

The rise of Internet technology in the 21st century has progressively grown, and by profiting from such convenience and liberation in cyberspace, network users can express different opinions freely, including some vulgar or hateful languages (Mubarak, Hassan, and Chowdhury, 2022). This explained why the amount of harmful and abusive languages has increased steadily owing to the rapidly growing amount of online user-generated content (Wiegand and Ruppenhofer, 2021).

The spread of hateful language probably impacts the internet's well-being as well as individuals' psychological health (Althobaiti, 2022). Therefore, it is necessary to investigate potential solutions for detecting online hateful languages automatically and with high efficiency. In the past few years, several researchers in the fields of Natural Language Processing (NLP) and Artificial Intelligence (AI) have contributed to building different models to deal with the problem of hate speech (Waseem et al., 2017, Schmidt and Wiegand, 2017; Pereira-Kohatsu et al., 2019), and most of these studies focused on words and text-based messages. Although the situation of online hateful language usage seems to have improved to some extent, there are still a certain number of potentially harmful languages that cannot be detected. Since human beings are gifted at avoiding censorship, they can create some word plays to show the same meaning within different approaches and even utilise emojis to replace some harmful vocabulary. This kind of circumstance makes it hard for the existing hate speech models to detect harmful content.

When it comes to emojis, they were originally employed to demonstrate and reinforce the expression of emotion as attached to a text (Shardlow, Gerber, and Nawaz, 2022). Nevertheless, emojis are also adopted to convey some intended meanings, function as a semantic part in a sentence, and even be utilised to compose hate speech to avoid an automatic ban. Numerous social media platforms such as Twitter, Facebook, and Instagram are suffering from the problem of hate speech containing emojis, especially when emojis are used to substitute some sensitive words. For the basic models of automatic hate speech detection which concentrate on textual information, hateful content is diverse and complicated, and one certain challenging task is the usage of emojis for expressing hate (Kirk et al., 2022). (Kirk et al. (2022)) and other scholars have tested the existing content moderation models using the HATEMOJICHECK¹ dataset they built, and also proposed a new model by using HATEMOJIBUILD¹ dataset to obtain better performance for detecting the hateful content with the use of emojis. Since the hateful content is complex and keeps updating, the new variants of expressing hate with emojis also occur constantly. Therefore the main purpose of this research is to enhance hate speech detection dedicated to emoji-based hate speeches on social media. Stemmed on the dataset provided by (Kirk et al. (2022)), the related real social media messages can be obtained. Small additional data will also be scraped and labeled with at least two annotators, after that the Cohen Kappa Agreement Score is computed to see how well the annotation result. Then a new emoji-based hate speech detection model will be trained with the Kirk dataset and tested with a new dataset that is crawled from social media. Then this model will be qualitatively evaluated as well.

¹<https://github.com/HannahKirk/Hatemoji>

Chapter 1

Dataset Creation

The dataset used in this project combined two different domains: the first one is a recently dedicated dataset from Kirk et al. (2022), and the second part includes sets of social media messages utilizing emojis that have to be scraped from social media.

1.1 Data from Kirk

As mentioned above, there is a HATEMOJIBUILD dataset in the research of Kirk et al. (2022), which is created by three successive rounds of data generation and model retraining. It includes three sub-datasets: train, test, and validation, which were utilized by Kirk et al. (2022) in order to build a better model. This dataset has been adopted in the current project as well, as the first part of the dataset.

1.2 Data Scraped from Social Media

Due to the complexity and constant updating of the hateful contents, it is necessary to scrape more real world data to further improve the emoji-based hate speech detection model. Therefore, at the beginning of the project, scraping the comments from social media (i.e., TikTok¹ and YouTube²) became the first thing. The reason for choosing these two is that these two online video communities are popular in these days, and netizens are more willing to share their views towards the videos. Another reason is that there are few studies focusing on the emoji-based hate speech in the comments under the video, thus the current study would like to fill in this gap. The Selenium framework³ has been adopted in this process initially, since it has been planned that some automated scripts could be written to simulating human online behaviour and collecting the comments under the video. However, the video-based social network platforms such as TikTok and YouTube have a strict anti-crawling measures, and some pop-up windows are shown when refreshing the contents or going to the next step: they ask users to complete the puzzle or receive the verified code from the SMS. This might be solved if there is enough time, while considering that the project is time-limited and this step is only a basic part of the project, it is not worth to spend more time on dealing with the anti-crawling measures. Therefore, another method was used, which is to employ the existed API to obtain the data.

- **TikTok:** Tikhub⁴ (V 3.15)

The API called Tikhub (Version 3.15) has been adopted to crawling the comments under the Tiktok video. The general idea is to use the TikHub API to retrieve

¹<https://www.tiktok.com/>

²<https://www.youtube.com/>

³<https://www.selenium.dev/>

⁴<https://api.tikhub.io/docs>

objective, the nature of each message is considered.

Example 1: Its 🤔🤔 Tag: Hated, Type: Disgust.

Example 2: I 🥰 it Tag: Non-Hated, Type: Sweet.

By only one input utterance we will predict first whether it is “hate” or “non-hate” then as extra information, we will predict what type of that utterance is it “Disgust” comment or “Sweet” comment. Therefore, there are 9 types of negative speech which are sarcastic, criticism, threat, discriminatory, derogatory, rebellious, disgusting, mocking, and accusatory and for 10 types of positive speech which are excitement, support, acknowledgment, respect, admiration, humorous, amazement, amusement, concerning, and sweet. There are a total of 200 new scraped datasets, 100 for the hated group and 100 for the non-hated speech group and for each group, there are 50 for TikTok and 50 for YouTube as the reason is not to bias the prediction from the training model. From that, organize dataset we seem to have diverse testing reasons as well. As a result, the Cohen Kappa score between the two annotators is **0.7584**. Additionally, the Cohen Kappa agreement score of each label has been calculated as well in this [Figure 1.2](#). Noticeably, the most agreement labels between two annotators are “concerning” labels while “respect” labels are the lowest agreements between two annotators.

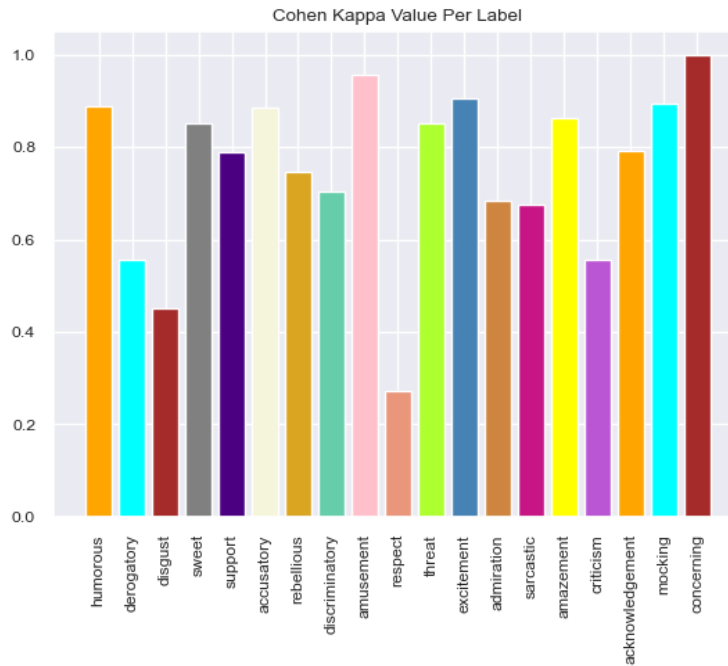


FIGURE 1.2: Cohen Kappa Value Per Label

We have checked the tagging dataset and we could find that the amount of sentences in label “respect” for the first annotator only 3 and the second annotator only 4 so it may easily lead to disagreement and get the Cohen Kappa score lower. If we check [Figure 1.3](#) we could see the most correlation label is “sweet” then we check the annotation guide to make sure about the definition of each label so the result we assume that some label may contain other label definition in it as well like the correlation of label “respect”, “sweet”, “support”(e.g. “i listened to this before it was popular 🤔”). Therefore, this kind separated meaning is not so much efficient when the labels have similar meaning or definition and especially the amount of data is

small. When we take a look at the “concerning” label it has around 7 sentences from both annotators, the annotate messages and the guideline definition mostly straightforward so both annotators can understand it more clearly. (e.g. “Giving me ideas 🤔 🤔” and “Yes sir I been waiting for this 🤔”)

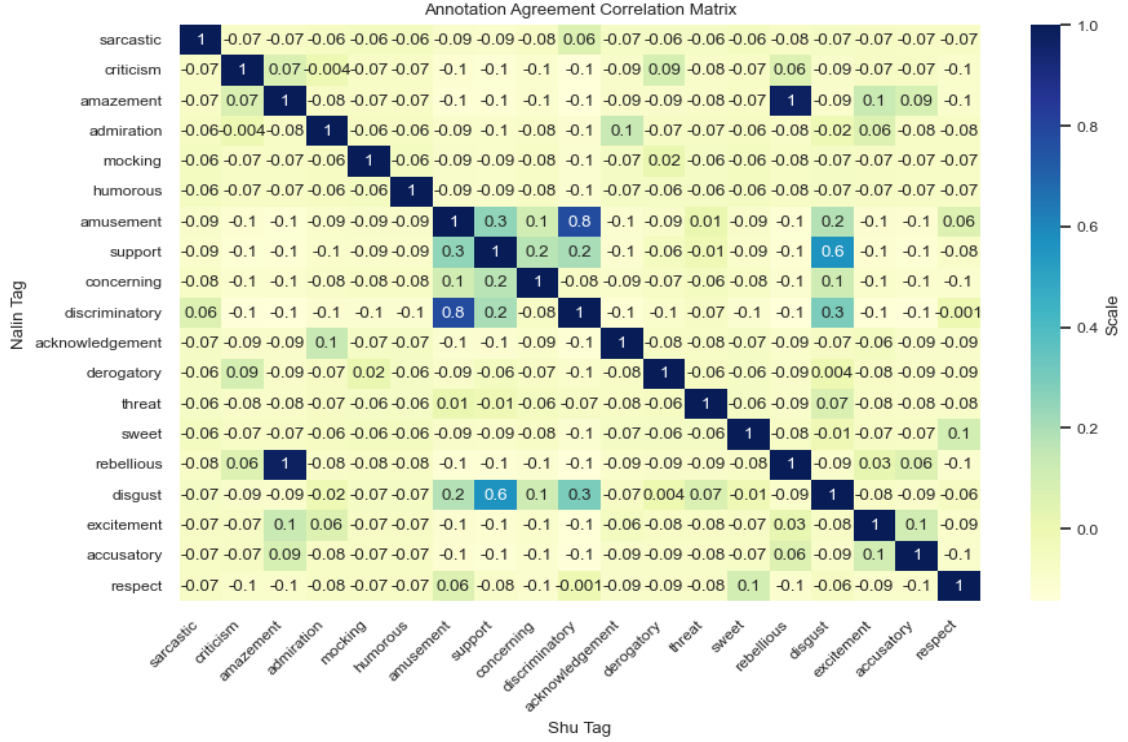


FIGURE 1.3: Annotation Agreement Correlation

The correlation annotation result is provided here [Figure 1.3](#). Interestingly, the first annotator focuses on the messages that are “rebellious” and the second annotator thinks it is “amazement”. If we analyze the meaning in the annotation guide⁶: “amazement” is a feeling of shock, astonished by someone or something while “rebellious” is a strong disagreement with government and mostly kind of emojis used is little bit similar to each other (e.g. “amazement” : 🤔, 😲, 🤯, 🤯, 🤯. “rebellious” : 🤔, 🤔, 🤔, 🤔, 🤔). As the sentences in our data do not contain many words, we concern more on the used emojis. By using that kind of similar emojis, the two annotators may confuse the semantic and annotate between that two labels. Additionally, labels “amusement” and “discriminatory” have a strong correlation of 0.8. “amusement” is a kind of positive message but people normally use some kind of these words (e.g. “The first one killed me 🤔 🤔”, “shi killed me that shi was funny asf 🤔 🤔”) to express their feeling of enjoy and entertainment. These kinds of messages are not “discriminatory” message because it only means that one person just entertains and interests with the contents of video but because of emojis used, so there is some disagreement between two annotators. Moreover, after checking the data “disgust” and “support” labels mostly about politics, country, etc., so emojis used also mostly about king and country as well. Therefore, only depending on the emojis used may be a little confusing between two annotators.

⁶<https://docs.google.com/document/d/1Y6dWHIBDYZqNvIDsJdCAj2ncEfmA0ND-MUI2CKA2XBpc/edit?usp=sharing>

Overall, there are some disagreements between annotators 1 and 2 and we could say that: when trying to separate each class, sometimes the meaning could be similar to each other; so when both annotators annotate in different labels, it does not mean that it is wrong. Additionally, because the annotation comments are not full sentences, it is just a utterance and the emojis used, it may be quite similar but perceived in two ways. If we want to be more accurate about the annotation result we have to check and analyze all the videos' contents to study the context. While this is not possible because it may took too long time and a little bit difficult for us to complete as we scraped it all and shuffle all the comments together.

Below [Figure 1.4](#) is the plot to show the count of different types of messages that have been annotated. We could see that the most occurring type of message is “sarcastic” with a percentage of 15% of 30 messages and the smallest amount of messages occurring in the whole dataset is “respect” which is only 2% and only 3 messages.

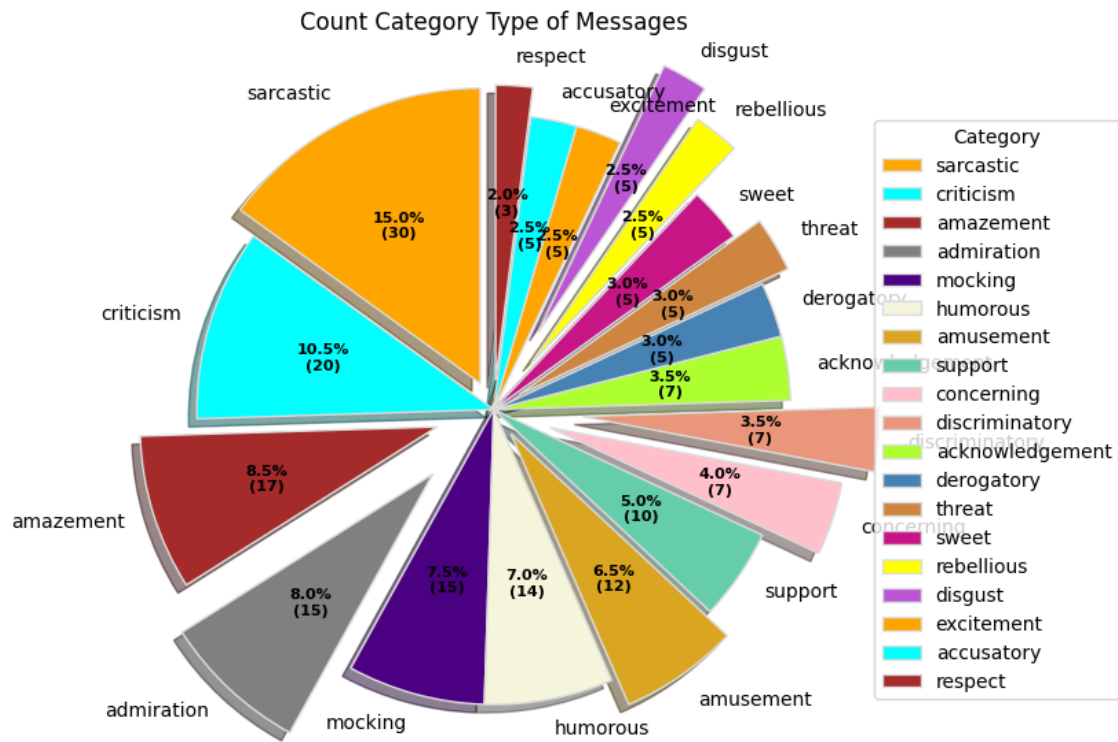


FIGURE 1.4: Messages Count Categories

Notice that in [Figure 1.4](#), each type of label we did not scrape on purpose: we just randomly selected whatever the messages with emojis. However, we just try to find the social media contents with most people who used emojis like video pranks, etc., to comment or react the video. Moreover, for the annotation, we just annotated based on the meaning in this guideline ⁷. Therefore, each label we may get a totally different amount of messages.

Notably, the 10 most used emojis in hate [Figure 1.5](#) and non-hated [Figure 1.6](#) speech are a little different from each other. We could see that 😂, 🤔, 🧠, and 😭

⁷<https://docs.google.com/document/d/1Y6dWHIBDYqNvIDsJdCAj2ncEfmA0ND-MUI2CKA2XBpc/edit?usp=sharing>

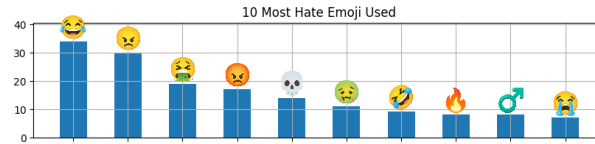


FIGURE 1.5: 10 Most Hated Emojis

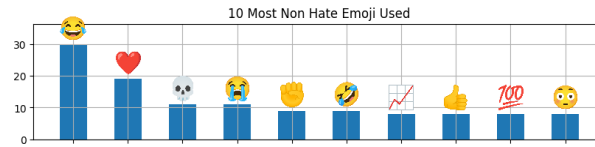


FIGURE 1.6: 10 Most Non-Hated Emojis

could be seen in both positive and negative context so we may need to understand the context surrounding it. Moreover, ❤️, 🙌, 🏆 mostly used in positive only while 😞, 🏳️, 😡, 🤔 mostly used in negative only. Therefore, from these kind of emojis we could think as it is positive or negative. However, this analysis only utilized 200 comments from social media, so it may require a larger dataset to ensure greater accuracy.

Chapter 2

Methodology

2.1 Machine Learning

2.1.1 Dataset Preparation

As mentioned above, the dataset of Kirk et al. (2022), there are three sub-datasets: train dataset includes 4728 instances, test dataset includes 593 instances and validation dataset includes 591 instances; in the dataset scraped from social media, there are 200 instances.

Two main tasks have been considered in the attempts for machine learning method: the first one was the binary classification to determine if one instance belongs to hate speech or not, the second one task was the multiclass classification to tell the type of instance (e.g., accusatory, admiration, sarcastic, etc.).

- For the binary classification task, the train and validation sub-datasets of Kirk et al. (2022) have been combined as the train dataset. There were two sub tasks to test the performance of the model, which means there would be two test datasets: the one was the test sub-dataset of Kirk et al. (2022), and the other one was the dataset scraped from social media.
- For the multiclass classification task, there were also two sub tasks: the first one used the same train dataset as the binary classification, and had a test on the test sub-dataset of Kirk et al. (2022); since the labels of type in Kirks dataset was different from those in our own dataset scraped from social media, this time for the second sub task, we only utilised our own scraped dataset to both train and test: 80% of the data were used for training, and the rest of 20% were used for testing.

To be clearer, for each of the task, the train dataset and test dataset has been arranged as follows (Table 2.1).

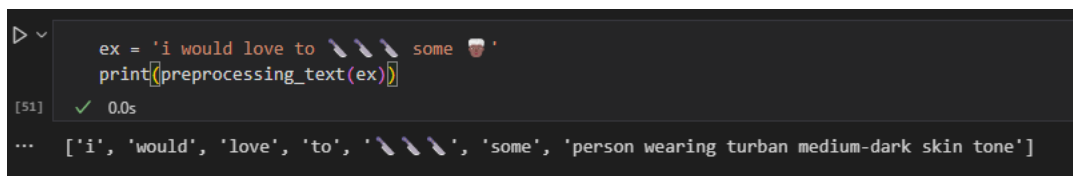
Task	Train Dataset		Test Dataset
Binary Classification	1	Kirk's train+Kirk's validation	Kirk's test
	2	Kirk's train+Kirk's validation	scraped data
Multi-Classification	1	Kirk's train+Kirk's validation	Kirk's test
	2	80% of scraped data	20% of scraped data

TABLE 2.1: Train and test dataset for each task in Machine Learning

2.1.2 Text Pre-processing

In NLP tasks, one of the significant issues is to pre-process the texts before other operations since it assists to clean and transform raw test data into a format which is more suitable for analysis, improving accuracy and efficiency in NLP tasks. After observing the raw data, it could be noticed that the capitalization did not carry lots of significant information, most of them only appeared at the beginning of the sentences. Therefore, the operation of lowercasing the letters has been considered in the pre-processing of text. As a part of text normalization, lowercasing all the letters helps to build a consistent and accurate vocabulary in the text classification task, and it also assists to decrease the number of unique features which means the dimensionality can be reduced. In terms of the punctuations, we noticed that in raw data, many sentences have been expressed with the combination of punctuations and emojis such as "👉 = 🐸", thus it was better to keep them. When it comes to stop words, since most of the sentences in raw data were short, we preferred to keep them, in case of losing some important information.

For pre-processing the emojis, there were some different opinions within our team. Some team members suggested that just keeping the original emojis was proper. However, others proposed that if the original emojis were kept, the vectorizer would only deal with them and create vectors according to their unicodes (e.g., 🙄 has the corresponding unicode U+1F620) directly, which means some related semantic meanings may be missed therefore, a new proposal was to substitute the emojis with their corresponding standard textual description. We could not immediately obtain a conclusion of which approach to dealing with emojis was more reasonable, so we had an experiment with both two approaches, and the results will be mentioned in the next part. In the process of substituting the emojis into textual description, the tokenization had to be executed to distinguish each word and emoji and then replace the emoji. At first the NLTK tokenizer¹ was employed, while we found there was a problem that the NLTK tokenizer could not separate the adjacent emojis (seen as Figure 2.1: the three 🐸 could not be separated, which would be a difficulty for the following substitution step). To deal with this problem, the tokenizer was changed



```

ex = 'i would love to 🐸🐸🐸 some 🙄'
print(preprocessing_text(ex))

[51] ✓ 0.0s

... ['i', 'would', 'love', 'to', '🐸🐸🐸', 'some', 'person wearing turban medium-dark skin tone']

```

FIGURE 2.1: Tokenized by NLTK

into spaCy², and this time the adjacent emojis were able to be divided successfully (seen as Figure 2.2). Hence, finally the spaCy was adopted to be utilised as the tokenizer.



```

ex = 'i would love to 🐸🐸🐸 some 🙄'
print(preprocessing_text(ex))

[58] ✓ 0.0s

... ['i', 'would', 'love', 'to', 'kitchen knife', 'kitchen knife', 'kitchen knife', 'some', 'person wearing turban', 'medium-dark skin tone']

```

FIGURE 2.2: Tokenized by spaCy

¹<https://www.nltk.org/api/nltk.tokenize.html>

²<https://spacy.io/>

2.1.3 Feature Extraction

For the feature extraction in the part of machine learning method, since we could not conclude easily that which feature extraction method would perform better, we would like to experiment on three kinds of feature extraction method, which were hashing vectorizer, Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer in Scikit-Learn³ (Pedregosa et al., 2011) as well as the word vectors obtained from self-trained FastText⁴ unsupervised model (using the training dataset to train this model). For each model, we applied these three vectorizing methods in order to observe if the performance of this model has been influenced. Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer as well as the word vectors obtained from self-trained FastText unsupervised model (using the training dataset to train this model). For each model, we applied these three vectorizing methods in order to observe if the performance of this model has been influenced.

In addition, before all the experiments, we have executed the categorical encoding for the target variables by using Label Encoder in Scikit-Learn. Since the class labels were texts, so they should be changed into the way that the algorithm could deal with.

2.1.4 Proposed Machine Learning Models

- **Binary Classification**

For the binary classification, some machine learning models such as Logistic Regression, SGD Classifier, SVM (Cortes and Vapnik, 1995), Decision Tree Classifier (Vega et al., 2009), Random Forest Classifier (Ho, 1995), LightGBM (Ke et al., 2017), BernoulliNB, and MLP Classifier have been employed in the two sub tasks (using the same training dataset while testing on two different test datasets). In addition, we not only used FastText (Bojanowski et al., 2017) to get word vectors (by using self-trained FastText unsupervised model), but also adopted the pre-trained supervised learning models directly provided by FastText which utilises word embedding and a shallow neural network for text classification.

- **Multiclass Classification**

For the two sub tasks in multiclass classification, the first one which was trained with Kirks train dataset and tested on Kirks test dataset, the machine learning models such as Logistic Regression, SVM, Decision Tree Classifier, Random Forest Classifier, LightGBM, MultinomialNB, MLP Classifier have been adopted as well.

Here we also met a problem: by observing the raw data, we found that the data was imbalanced (shown in Figure 2.3, the numbers of the instances in each category were various). Even though for some models we set the parameter about `class_weight` as balanced, the performances were not so ideal either. Therefore, we attempted to employ a data augmentation strategy to deal with the imbalance of the data, such as Synthetic Minority Over-sampling Technique (SMOTE (Chawla et al., 2002)) in Imbalanced-Learn⁵, we were not sure that if it would help, so it was like another experiment, and the results will be talked about in the next part as well. For the second one which was both trained and tested on our own scraped dataset, some machine learning models have been used as well, such as Logistic Regression, SVM, Decision Tree Classifier, Random Forest Classifier, LightGBM, MultinomialNB, MLP

³<https://scikit-learn.org/stable/>

⁴<https://fasttext.cc/>

⁵<https://imbalanced-learn.org/stable/index.html>

none	2662
derogation	1758
animosity	513
dehumanizinglanguage	205
threateninglanguage	181
Name: type_label, dtype: int64	

FIGURE 2.3: Imbalance of Data for multiclass Classification

Classifier. However, since the size of the dataset scraped from social media was small (200 instances), the SMOTE technique has not been utilised in this sub task. Due to the fact that this sub task was both trained and tested on one dataset (80% for training, and 20% for testing, and the dataset was randomly split by using the "train_test_split function" in Scikit-learn model selection after vectorizing), it was hard to prepare the text files for training and testing that were needed by FastText supervised model, so the pre-trained supervised learning models provided by Fasttext has not been applied in the second sub task of multiclass classification. Furthermore, for both two classification tasks, the grid search with cross-validation was employed to find the best hyperparameters for the models.

2.2 Deep Learning

2.2.1 Data Preparation

Domain	Dataset	Sentences
Kirk	Train	4728
	Validation	591
	Test	593
TikTok	Test	100
YouTube	Test	100

TABLE 2.2: Dataset Preparation

Below describing the dataset we organize to build deep learning model:

Phase 1

Domain	Dataset	Sentences
Kirk(Train+Validation)	Train	5319
TikTok+YouTube	Test	200

TABLE 2.3: Dataset Preparation Phase 1

Phase 2

Domain	Dataset	Sentences
Kirk(Train+Validation+Test)	Train	5912
TikTok+YouTube	Test	200

TABLE 2.4: Dataset Preparation Phase 2

2.2.2 Feature Extraction

There are many advantages of FastText⁶ compared to other feature numerical statistic representations (e.g. TF-IDF⁷, Hashing Vectorizer⁸, traditional bag-of-words, or word2vec⁹). FastText goes beyond individual words and takes into account sub-word information, such as character n-grams. This allows the model to capture out of vocabulary words. By representing words as a combination of character n-grams, FastText can handle unseen words more effectively. We can use pre-trained embeddings to save time and resources, especially when working with limited training data. Also, FastText is designed to capture semantic meaning in text and associate similar word representations for words that appear in similar contexts. In our work, we use FastText pre-trained embedding model for English which has size of 4G.

2.2.3 Model Architecture of Binary Classification

In the process of training each model, we set the number of epochs to 10, batch size 64. CrossEntropyLoss is used to calculate loss which is a commonly used loss function for classification tasks. It combines the softmax activation function and the negative log-likelihood loss to calculate the loss between the predicted and true labels. Then, we use the Adam optimizer Kingma and Ba, 2014, a popular optimization algorithm for training neural networks. It is initialized with a learning rate set to 0.005. During the fitting loop of the model, the early stop is applied, if the training model does not improve performance for the next 5 times (epochs) we shall stop the training process.

Bidirectional Long Short-Term Memory (Bi-LSTM)

Below Table 2.5, is the configuration of each first initial model for our task:

Bi-LSTM	Value
Bi-LSTM Embedding Layer(1st training) FastText Embedding(2nd training)	300
Bi-LSTM Layer	256
Activation Function	NA
Dropout	0.3
Dense Layer	256
Loss Function	cross entropy
Optimizer	Adam

TABLE 2.5: First Initial Hyper-parameters for Bi-LSTM model

The two different modified model settings were applied to train the model for both Bi-LSTM Hochreiter and Schmidhuber, 1997 and CNN LeCun, Bengio, and Hinton, 2015. First, Table 2.5 shows that the embedding layer takes input indices and maps them to dense vectors. It has an input size of (vocab size) and an output size of 300 (embedding size). The embedding layer is used to learn the distributed representation of the input words. Then, the next layer is the LSTM layer which takes an input size of 300 (embedding size) and a hidden size of 256. The LSTM is bidirectional

⁶<https://fasttext.cc/docs/en/crawl-vectors.html>

⁷https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

⁸https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.HashingVectorizer.html

⁹<https://radimrehurek.com/gensim/models/word2vec.html>

that process the input sequence in both forward and backward directions. Then, the dropout layer with a probability of 0.3 is used to prevent over-fitting by randomly settings elements of the input to zero during training. The fully connected linear layer maps the output of the LSTM to the output classes. It takes an input size of 256 (hidden size) and produces an output size of 2 possible classes. In other modified model architecture, additionally, we used a pre-trained embedding model which is FastText to build our word representation. So, the embed layer is initialized using a pre-trained embedding model.

Convolutional Neural Network (CNN)

CNN	Value
CNN Embedding Layer(1st training) FastText Embedding(2nd training)	300
Convolutional Layer 1(Kernel=2,Stride=1,ReLU)	128
Convolutional Layer 2(Kernel=2,Stride=1,ReLU)	128
Convolutional Layer 3(Kernel=2,Stride=1,ReLU)	128
Pooling Layer	max_pool1d
Dropout	0.3
Dense Layer	128
Loss Function	cross entropy
Optimizer	Adam

TABLE 2.6: First Initial Hyper-parameters for CNN model

The input is a vocabulary size, Table 2.6 describes each word is represented as a 300-dimensional embedding. The (embed) layer is responsible for mapping the input word indices to their corresponding embedding. There are 3 convolutional layers, conv1 performs a 1-dimensional convolution with a kernel size of 2 and stride of 1. During the feature mapping, we also apply ReLU to avoid negative value as well. The input size is 300 (embedding size), and the output size is 128 then conv2 and conv3 are the same as conv1. After that, the Dropout layer rate of 0.3 and is the technique to prevent over-fitting. The output of the last convolutional layer is passed through the linear layer (decision) to produce the final output. The final result if it is below 0 we consider it as negative class and if it is bigger 0 it is positive class.

The choice of hyperparameters is really important and can significantly impact the model training ability to learn meaningful representation from the input of data. First, we start to try with popular hyperparameters nowadays with all the big values of each hyperparameter then if the result is not suitable we try to tune it later in the tuning section because these choices are not fixed and can vary depending on the specific task, dataset characteristics, and experimentation. We choose to use 2 kinds of models which are CNN and Bi-LSTM. The reason why we choose Bi-LSTM is that as we mentioned the characteristics of our data in the illustration of data section we could see that it required the learning context surrounding more details and after we have done some overview of Bi-LSTM architecture it is really interesting to us as it learns the context surrounding more clearly especially it has the ability to move backward and forward so this model is kind of our motivation. Moreover, for CNN we see that the model could learn by the local pattern and we can custom how it learns to capture local patterns by moving with specific kernel and stride as our dataset is not such a long sentence and our task are to build the model to learn such a semantic

of the sentence so the model that learns more local deeply is also counted CNN too. CNN is successful with capturing pixels of images so the motivation is we want CNN to extract meaningful features from textual data.

2.2.4 Model Architecture of Multiclass Classification

After obtaining one suitable binary model to predict whether the message is hated or non-hated speech this time we start to build a multiclass classification model to predict which type of message (e.g. sarcastic, threat, criticism, support). As our dataset is not the same as the Kirk dataset because we have almost 20 categories of messages while Kirk only has 4 types of messages and only focuses on hate speech but for us, we also separate the type of positive messages as well. So it is hard for us to build a new model from scratch as the amount of data we got now only 200 messages. Therefore, we decided to tune the pre-trained model from Hugging Face. Therefore, we applied tuning in that model by adding more text and labels. The adding training set is 180 messages and adding testing set is 20 messages.¹⁰ and the model is called “bert-base-uncased”¹¹ and we can find the detail architecture of this training model in this paper (Devlin et al., 2019). Currently this model was trained with 110M parameters. Figure 2.4 provided by (Devlin et al., 2019) show how the “bert-base-uncased” was trained. This pre-trained model objective: first to predict masked words by randomly masking 15% word input as they want to train deep bidirectional representation and another objective is next sentence prediction by learning in sentence relationship. The advantage of this model is it was pre-trained with raw text only so it is suitable for our dataset. The model from Hugging Face uses Bidirectional Encoder Representation Transformers (BERT (Devlin et al., 2019)) model architecture¹², which is a transformer-based model for NLP tasks. It also uses BertTokenizer¹³ from the Hugging Face library, specifically for the “bert-base-uncased” variant. Then, for model instantiation it used BertForSequenceClassification, AdamW (Adam with weight decay (Loshchilov and Hutter, 2017)) is used as an optimizer algorithm for training neural networks. And the learning rate is set to 2e-5, and the loss function is CrossEntropyLoss. We also saved the best model during training based on the best validation loss, this allows us to save the model with the best performance on the validation set.

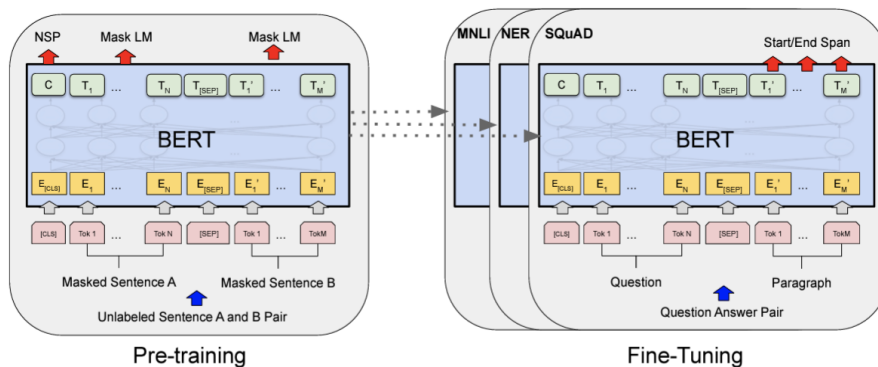


FIGURE 2.4: Pretrained and Fine Tuning procedures for BERT

¹⁰<https://huggingface.co/>

¹¹<https://huggingface.co/bert-base-uncased>

¹²https://huggingface.co/transformers/v3.0.2/model_doc/bert.html

¹³https://huggingface.co/transformers/v3.0.2/model_doc/bert.htmlberttokenizer

Chapter 3

Experiment

3.1 Machine Learning

As mentioned above, to check which approach to dealing with emojis was more reasonable (using them directly in vectorizer or replacing them with the textual description firstly) we have conducted an experiment: in the binary classification task which was tested on Kirks test dataset, we applied these two approaches respectively, and then utilised the logistic model with Hash vectorizer to observe if the way of treating emojis would influence the model performance. The results are presented as follow, since

	precision	recall	f1-score	support
0	0.4267	0.5825	0.4926	285
1	0.4167	0.2760	0.3320	308
accuracy			0.4233	593
macro avg	0.4217	0.4292	0.4123	593
weighted avg	0.4215	0.4233	0.4092	593
[[166 119]				
[223 85]]				

FIGURE 3.1: Using emojis directly

	precision	recall	f1-score	support
0	0.5488	0.6316	0.5873	285
1	0.6038	0.5195	0.5585	308
accuracy			0.5734	593
macro avg	0.5763	0.5755	0.5729	593
weighted avg	0.5773	0.5734	0.5723	593
[[180 105]				
[148 160]]				

FIGURE 3.2: Changing emojis with their textual representations

the samples were generally balanced, so we focused on macro average scores: Figure 3.1 above is the result that used emojis directly in vectorizer, with the accuracy of 0.4233, the precision of 0.4217, the recall of 0.4292 and the f1-score of 0.4123 on the test dataset; while Figure 3.2 below shows the result after replacing the emojis with the textual description, with 0.5763 for precision, 0.5755 recall and 0.5729 f1-score respectively on the same test dataset. It could be investigate that the performance of the model has been improved after substituting the original emojis to their corresponding textual descriptions. Therefore, to get models with better performance, in all the following tasks we will treat the emojis in the second approaches, which means replacing it with textual meaning.

Logistic Regression			SGD			SVM		
Hash	TF-IDF	FT	Hash	TF-IDF	FT	Hash	TF-IDF	FT
0.5830	0.6197	0.5337	0.5843	0.6182	0.5283	0.5768	0.6200	0.5351
Decision Tree			Random Forest			LightGBM		
Hash	TF-IDF	FT	Hash	TF-IDF	FT	Hash	TF-IDF	FT
0.5644	0.5841	0.5304	0.5832	0.6112	0.5514	0.5901	0.6027	0.5454
BernoulliNB			MLP					
Hash	TF-IDF	FT	Hash	TF-IDF	FT			
	0.6007		0.5634	0.6106	0.5336			

TABLE 3.1: Best accuracy score for each model in binary classification task

	Hashing		TF-IDF		FastText	
	Acc (↑)	F1 (↑)	Acc (↑)	F1 (↑)	Acc (↑)	F1 (↑)
Logistic Regression	0.5734	0.5729	0.5919	0.5916	0.5413	0.5276
SGD	0.5565	0.5536	0.5902	0.5901	0.5194	0.3418
SVM	0.5430	0.5430	0.5885	0.5885	0.5379	0.5103
Decision Tree	0.5582	0.5403	0.5750	0.5629	0.5126	0.4896
Random Forest	0.5194	0.5193	0.5919	0.5915	0.5548	0.5476
LightGBM	0.5514	0.5514	0.5919	0.5919	0.5379	0.5376
BernoulliNB			0.5868	0.5846		
MLP	0.5194	0.5174	0.5885	0.5884	0.5261	0.4947
FastText Model						0.5533

TABLE 3.2: Results of the 1st sub task of binary classification

3.1.1 Binary Classification

In the binary classification task, as mentioned before we attempted different machine learning models, and FastText pre-trained supervised learning models. The train dataset was the combination of train and validation sub-datasets of Kirk et al. (2022), three kinds of vectorizers and nine kinds of models, and the best accuracy score after grid searching, the test result on Kirks test dataset and the test result our own dataset scraped from social media are shown as follow respectively. Table 3.1 shows the best accuracy score after grid searching with cross validation, and presents the models performance on the training dataset. It can be discovered that the Support vector machines (SVM) with TF-IDF vectorizer had the highest accuracy score of 0.6200, with the best hyperparameters of "C=1, gamma='scale', kernel='linear'".

- **First sub task: test on Kirk's test dataset**

In the first sub task which is testing the model on Kirks test dataset, it can be explored in Table 3.2 that the LightGBM classifier with TF-IDF vectorizer seems to be the better one, with the hyperparameters of "boosting_type='gbdt', class_weight='balanced', learning_rate=0.05, max_depth=-1, min_child_samples=20, n_estimators=100, num_leaves=31".

- **Second sub task: test on the dataset scraped from social media**

	Hash		TF-IDF		FastText	
	Acc (\uparrow)	F1 (\uparrow)	Acc (\uparrow)	F1 (\uparrow)	Acc (\uparrow)	F1 (\uparrow)
Logistic Regression	0.5750	0.5671	0.5900	0.5867	0.5000	0.3333
SGD	0.5850	0.5748	0.5750	0.5711	0.5050	0.3443
SVM	0.5900	0.5850	0.5600	0.5524	0.5000	0.3333
Decision Tree	0.4950	0.3703	0.4800	0.3686	0.5000	0.3333
Random Forest	0.5800	0.5758	0.5550	0.5409	0.5850	0.5585
LightGBM	0.5900	0.5867	0.5250	0.4920	0.5100	0.5098
BernoulliNB			0.5350	0.5146		
MLP	0.6000	0.5967	0.5850	0.5686	0.5000	0.3333
FastText Model						0.5950

TABLE 3.3: Results of the 2nd sub task of binary classification

For the second sub task which is testing the model on the dataset scraped from social media, the result in Table 3.3 shows that the MLP Classifier with the vectors from Hash vectorizer performances better on this test dataset, with the hyperparameter of "activation='logistic', alpha=0.01, hidden_layer_sizes=(100,)".

If we consider all the results above as well as the confusion matrix, it can be observed that: for the Binary classification task the TF-IDF vectorizer performance better relatively, and the models such as SVM, LightGBM, MLP classifier with proper hyperparameters are more suitable for this task.

3.1.2 Multiclass Classification

When it comes to multiclass classification task, there were two sub-tasks as well: the first one used the combination of train and validation sub-datasets of Kirk et al. (2022) as the train dataset, and had a test on the Kirks test sub-dataset. The second sub task is utilising our own scraped dataset to both train and test: 80% of the data were used for training, and the rest of 20% were used for testing.

As discussed above, it was worthwhile to investigate how to deal with the imbalanced data. In spite of set the parameter of `class_weight` as balanced, we also wanted to know if we used some data augmentation strategies, whether the performance of the model would be impacts. Therefore, we employed the SMOTE technique to balance the train dataset by synthesizing new minority class samples, which assisted to prevent the model from being biased towards the majority class and improves its capability to recognize and classify minority class samples.

Best accuracy score: 0.3807050552062187				
	precision	recall	f1-score	support
0	0.1368	0.4727	0.2122	55
1	0.1440	0.6429	0.2353	28
2	0.4135	0.2709	0.3274	203
3	0.4595	0.1789	0.2576	285
4	0.3235	0.5000	0.3929	22
accuracy			0.2715	593
macro avg	0.2955	0.4131	0.2851	593
weighted avg	0.3939	0.2715	0.2812	593

FIGURE 3.3: Before using SMOTE

```

Best accuracy score: 0.7517655897821187
      precision    recall  f1-score   support

     0       0.1698       0.3273       0.2236         55
     1       0.2564       0.3571       0.2985         28
     2       0.3803       0.3990       0.3894        203
     3       0.4776       0.3368       0.3951        285
     4       0.2059       0.3182       0.2500         22

 accuracy                   0.3575         593
 macro avg       0.2980       0.3477       0.3113         593
 weighted avg    0.3952       0.3575       0.3673         593

```

FIGURE 3.4: After using SMOTE

Logistic Regression			SVM			Decision Tree		
Hash	TF-IDF	FT	Hash	TF-IDF	FT	Hash	TF-IDF	FT
0.3807	0.4028	0.2559	0.4427	0.4789	0.2400	0.3655	0.3886	0.3820
Random Forest			LightGBM			MultinomialNB		
Hash	TF-IDF	FT	Hash	TF-IDF	FT	Hash	TF-IDF	FT
0.4619	0.4606	0.4758	0.4553	0.4416	0.4570		0.5275	
MLP								
Hash	TF-IDF	FT						
0.4711	0.5039	0.5029						

TABLE 3.4: Best accuracy score for the 1st sub task of multiclass classification

We have had an experiment on the logistic regression with hash vectorizer: Figure 3.3 represents the result before using SMOTE, and Figure 3.4 is the result after applying SMOTE. It is noticeable that the best accuracy score on the train dataset after grid searching and cross validation has marked increased from 0.3807 to 0.7518; however when the model was applied on test dataset, it seems that the performance of the model only increased slightly. Since it could not improve the performance of the model a lot, and would cause the risk of over-fitting, so we did not adopt SMOTE in the following models in the first sub task.

- **First sub task: trained on Kirks train and validation sub-datasets, tested on Kirks test sub-datasets**

In terms of the models performance on the training dataset, Table 3.4 shows the best accuracy score after grid searching with cross validation. Since it is a multiclass classification task, and the samples are not so balanced, we focus on the weight average of F1-score this time. It can be noticed that the MultinomialNB (alpha=2.0) with TF-IDF vectorizer has the highest accuracy score. The performance of MLP classifier using TF-IDF vectorizer with the hyperparameter of "activation='logistic', alpha=0.01, hidden_layer_sizes=(100,)" is not so poor as well. When testing these models on Kirk's test dataset, the outcome will be exhibited as Table 3.5. This time, still the MultinomialNB using TF-IDF vectorizer and the MLP classifier using TF-IDF vectorizer with the same hyperparameters that were mentioned above have the better performance.

	Hash		TF-IDF		FastText	
	Acc (\uparrow)	F1 (\uparrow)	Acc (\uparrow)	F1 (\uparrow)	Acc (\uparrow)	F1 (\uparrow)
Logistic Regression	0.2715	0.2812	0.4165	0.4032	0.1012	0.0244
SVM	0.4030	0.4082	0.4890	0.4542	0.2766	0.2967
Decision Tree	0.3406	0.3445	0.3575	0.3611	0.3693	0.3692
Random Forest	0.3609	0.3357	0.3879	0.3736	0.4435	0.3800
LightGBM	0.4047	0.3887	0.3997	0.3930	0.4165	0.3800
MultinomialNB			0.5059	0.4430		
MLP	0.4722	0.4430	0.4992	0.4840	0.4840	0.3690
FastText Model						0.4636

TABLE 3.5: Results of the 1st sub task of multiclass classification

Logistic Regression			SVM			Decision Tree		
Hash	TF-IDF	FT	Hash	TF-IDF	FT	Hash	TF-IDF	FT
0.1375	0.1625	0.1375	0.1625	0.1563	0.1312	0.1688	0.1125	0.1188
Random Forest			LightGBM			MultinomialNB		
Hash	TF-IDF	FT	Hash	TF-IDF	FT	Hash	TF-IDF	FT
0.2188	0.1562	0.1688	0.1000	0.0750	0.1750		0.2188	
MLP								
Hash	TF-IDF	FT						
0.1813	0.2500	0.1938						

TABLE 3.6: Best accuracy score for the 2nd sub task of multiclass classification

- **Second sub task: trained and tested on our own dataset scraped from social media**

The train dataset in the second sub task of multiclass classification was really small, with the size of 160 instances. For some of the categories, there were less than 5 instances, thus we did not use SMOTE here. It can be found in Table 3.7 and Table 3.6 that the performances of the models are not ideal, the poor performance indicates that using only 160 instances for training is not an good decision.

	Hash		TF-IDF		FastText	
	Acc (\uparrow)	F1 (\uparrow)	Acc (\uparrow)	F1 (\uparrow)	Acc (\uparrow)	F1 (\uparrow)
Logistic Regression	0.2500	0.1996	0.2250	0.2427	0.1250	0.1417
SVM	0.2000	0.1304	0.1500	0.1330	0.2000	0.2171
Decision Tree	0.2000	0.2121	0.0750	0.0522	0.0500	0.0600
Random Forest	0.1500	0.1378	0.1500	0.1336	0.1250	0.1018
LightGBM	0.1250	0.0902	0.0250	0.0012	0.0750	0.0486
MultinomialNB			0.2500	0.2136		
MLP	0.1500	0.0650	0.1250	0.0924	0.100	0.0440

TABLE 3.7: Results of the 2nd sub task of multiclass classification

3.2 Deep Learning

3.2.1 Binary Classification

Model	Embedding Type	Testing Set F1-Score
Bi-LSTM (Phase 1 Table 2.3)	Bi-LSTM Embedding Layer	0.5344
	Pretrained FastText Embedding	0.5545
Bi-LSTM (Phase 2 Table 2.4)	Bi-LSTM Embedding Layer	0.5700
	Pretrained FastText Embedding	0.5779
CNN (Phase 1 Table 2.3)	Pretrained FastText Embedding	0.5800
CNN (Phase 2 Table 2.4)		0.5931

TABLE 3.8: Deep Learning Binary Classification Result

By seeing these results [Table 3.8](#), we could say that the CNN model works much better than the Bi-LSTM model as the best result in is from CNN which is **0.5931**. However, from [Figure 3.5](#) there is an over-fitting of the training model, like the more epoch the training loss increases as well.

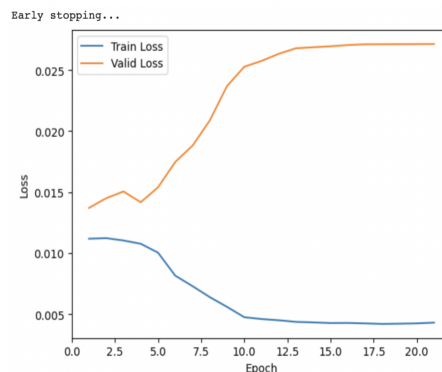


FIGURE 3.5: CNN Training Loss

3.2.2 Multiclass Classification

Below [Figure 3.6](#), is the result of tuning the “bert-based-cased” model with our new data social media data [Table 2.2](#) 20 types of messages.

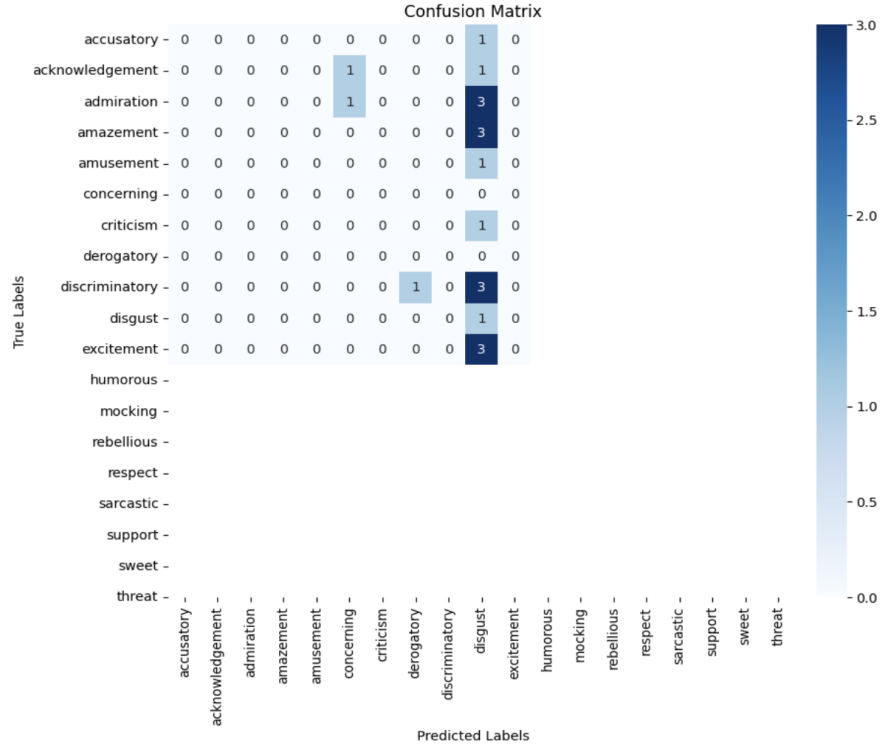


FIGURE 3.6: Multiclass Classification Result 1

We could see that the result is not so satisfactory because the test processing could not cover all the possible labels because we have got only 20 sentences as well as the result almost sometimes predict in the wrong class. We believe that this step is because of the small amount of data, if we have a big amount of data our multiclass classification tuning model could be improved a lot. However, in the tuning hyperparameter section, we will try to combine some labels that have similar meanings, especially group label that has a small number of sentences together.

3.2.3 Tuning Hyperparameter for Deep Learning Models

Binary Classification

From [Figure 3.5](#), we could see over-fitting of the training model, like the more epoch the training loss increases. Therefore, this time we try to fine-tune the model by lowering the hyper-parameter value, especially with different values of learning rate. And at this step, we only try to tune with the model training FastText embedding, as from previous results [Table 3.8](#), we could see the impact of FastText contributing to the model's learning.

Epoch, Learning Rate, and Batch Size

To understand the characteristic of model learning we try to use more epochs than before which are 50 epochs with the same batch size as the previous which is 64 batch size. For batch size, we also try with different values but still, this value is the best one for this task for other lower or higher values the accuracy of the model was lower than using 64 batch size. The suitable learning rate for each model right now is 0.002, we try both ways to increase and decrease the value of the learning rate from the previous one which is 0.005 and finally, we see this new value (lr=0.002) the

model could achieve the highest result compared to using other values.

Layers, Embedding Size, Hidden Size, F1-Score

Model	(Layer,Embed,Hidden)	3 Times Training F1-Score	Average F1-Score
CNN	(1,32,32)	1st=0.5901 2nd=0.6222 3rd=0.5993	0.6038
	(1,32,64)	1st=0.5985 2nd=0.6122 3rd=0.6249	0.6118
	(1,50,32)	1st=0.6228 2nd=0.6068 3rd=0.6090	0.6128
	(1,50,64)	1st=0.6290 2nd=0.5989 3rd=0.5945	0.6074
	(2,50,32)	1st=0.5893 2nd=0.5980 3rd=0.6142	0.6005
Bi-LSTM	(1,50,32)	1st=0.6005 2nd=0.6189 3rd=0.6161	0.6118
	(1,50,64)	1st=0.6238 2nd=0.6198 3rd=0.6121	0.6185
	(2,32,32)	1st=0.6145 2nd=0.6238 3rd=0.6142	0.6175
	(2,32,64)	1st=0.6317 2nd=0.6222 3rd=0.6329	0.6289

TABLE 3.9: Different Hyperparameter Testing Values F1-Score Result

After trying with various numbers of each hyper-parameter in [Table 3.9](#) and each training configuration, we train three times in order to prevent getting the best performance result by chance, and finally we could find the best CNN, which achieves **0.6128** of the F1-score on the validation set, and CNN is training with 1 layer, 50 FastText embedding sizes, and 32 hidden sizes. While the best Bi-LSTM, which achieves **0.6289** of the F1-score on the validation set, is training with 2 layers, 32 embedding sizes, and 64 hidden sizes.

This [Figure 3.7](#) could show that the models now are not over-fitting anymore and the training loss decrease when the epoch decrease as well. Also, we could find the best model at the epoch around epoch 3 for CNN and around 16 for Bi-LSTM.

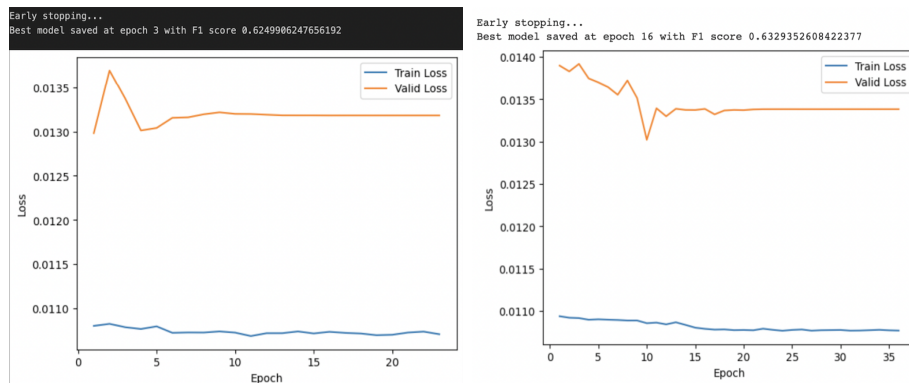


FIGURE 3.7: Final Best CNN and Bi-LSTM Training Loss

Even CNN could improve compared to the CNN first initial training pipeline result [Table 3.8](#), but if we compare the CNN with Bi-LSTM second training pipelines, we could see Bi-LSTM more robust to the simple model as it outperforms when small

configuration and hyper-parameters are used, while CNN could improve but cannot outperform Bi-LSTM as in the above result. Bi-LSTM could achieve it easily. Noticeably, when increasing the layer of CNN, it could increase the performance, but the computation time took longer than Bi-LSTM. The way CNN learns is much more complex as it learns to capture local patterns, so the process may take longer, and if it's a small task (small amount of dataset, etc.), it may lead to over-fitting easily.

Figure 3.8 and Figure 3.9 are the final architectures of each best model for our task:

Bi-LSTM

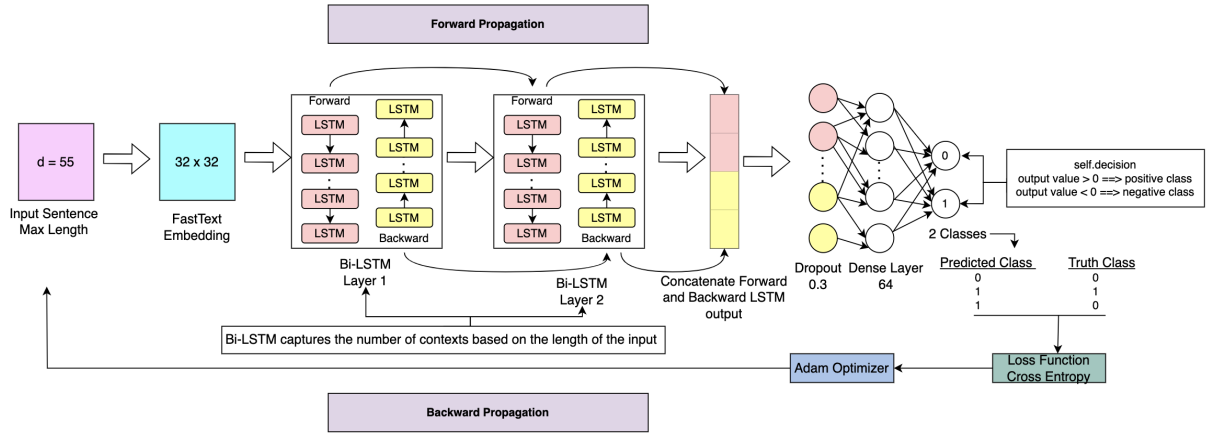


FIGURE 3.8: Final Best Bi-LSTM Architecture

Input Sentence: The input dimension has a size of 55, which is the maximum length of tokenized words in the sentence in the training set. Then we pad all the sentences that have tokenized words lower than 55 to all be the same size as 55 because we don't want to miss some information from the longest sentences. For the tokenized word task, we use NLTK ¹ libraries.

Embedding Layer: The model uses an embedding layer to convert input tokens into dense vector representations. The embedding layer is initialized with pre-trained word embedding and weights are not trainable, which means the pre-trained embeddings will not be updated during training.

Bi-LSTM Layer: The Bi-LSTM layer takes the embedded input sequence and processes it in both forward and backward directions. It has 64 memory units in each direction and there are 2 layers.

Dropout: After the Bi-LSTM Layer, a dropout layer with a dropout rate of 0.3 is applied. Dropout helps in regularizing the model by randomly setting a fraction of the output values to zero during training, which helps prevent over-fitting.

Dense Layer: The output of the dropout layer is fed into a linear layer (fully connected layer) with the 64 memory unit. This layer serves as the decision layer and maps the hidden state of the LSTM to the number of classes.

Loss Function: CrossEntropyLoss is a commonly used loss function for binary classification tasks. By penalizing deviations between the predicted probabilities and the true labels, the loss function encourages the model to assign higher probabilities to the correct classes and lower probabilities to the incorrect classes.

Optimizer: Adam is an optimization algorithm that combines the benefits of AdaGrad (Duchi, Hazan, and Singer, 2011) and RMSProp (Graves, 2013). It is widely

¹<https://www.nltk.org/api/nltk.tokenize.html>

used in deep learning for its adaptive learning rate capabilities. Adam dynamically adjusts the learning rate for each parameter during training. This adaptive learning rate scheme helps overcome the limitations of fixed learning rates and allows the optimizer to adapt the learning rate for each parameter independently.

Self.Decision: in this case, the output of the model is returned as it is, without any activation function being applied. This implied that the model assumes the final hidden states values will be used directly as the logits for classification. We set that if the logit is bigger than 0, it is in the positive class, while below 0 it is in the negative class. The reason for just using self.decision is because we don't want to make the model training more complex by using a lot of parameters and computation.

CNN

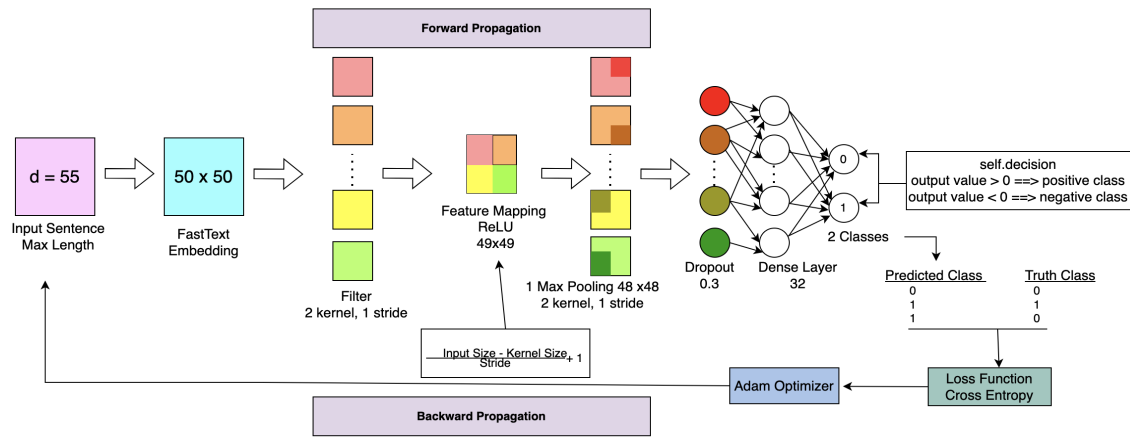


FIGURE 3.9: Final Best CNN Architecture

Convolutional Layers: The model uses a single convolutional layer with a kernel size of 2 and stride of 1. The conv layer convolves over the input embeddings to capture local patterns or features in the data.

Feature Mapping: the output of the convolutional layer is passed through a ReLU activation function to introduce non-linearity and allow the model to learn complex representations.

Pooling layer: the output of the feature mapping is then passed through a max pooling layer with a kernel size equal to 2 and stride of 1. This operation reduces the dimensionality of the output and captures the most salient features.

And every remaining hyperparameter describes the purpose the same as in Bi-LSTM.

Computation Time

Interestingly, when we try to reduce the dimension, the computation time is really fast and convenient, as training for 50 epochs takes about 2 minutes, and we can achieve around 63% of the F1-score on testing on the validation set using Bi-LSTM. This is very useful and practical, as before we took around 7 minutes to complete the task. The disadvantage of high dimensions is computation, and it is a very complex calculation, so it easily gets over-fitted like previous training losses [Figure 3.5](#). Moreover, while Bi-LSTMs offer advantages in capturing long-term dependencies and the context in sequential data, their sequential nature leads to slower computation times compared to CNN. The convolutional operations in CNN can be parallelized

effectively, making them computationally efficient. For our task, we could see that Bi-LSTM in average computation time longer than CNN is 5 minutes.

Multiclass Classification

Because some labels just contain 5 to 7 sentences so it is convenient to combine them as one class and apply them to the model. So, in this step, we combine the label, “respect” and “excitement” as “sweet” then “rebellious” and “accusatory” as “threat”, after that “derogatory” as “mocking”, and finally “disgust” as “Discrimination”.

Figure 3.10 is the result after combining some labels together and testing the model with a validation set of 20 sentences from TikTok and YouTube. After modifying the training dataset finally we got at epoch 10 the training loss is 0.7618 and the validation loss is 2.4345.

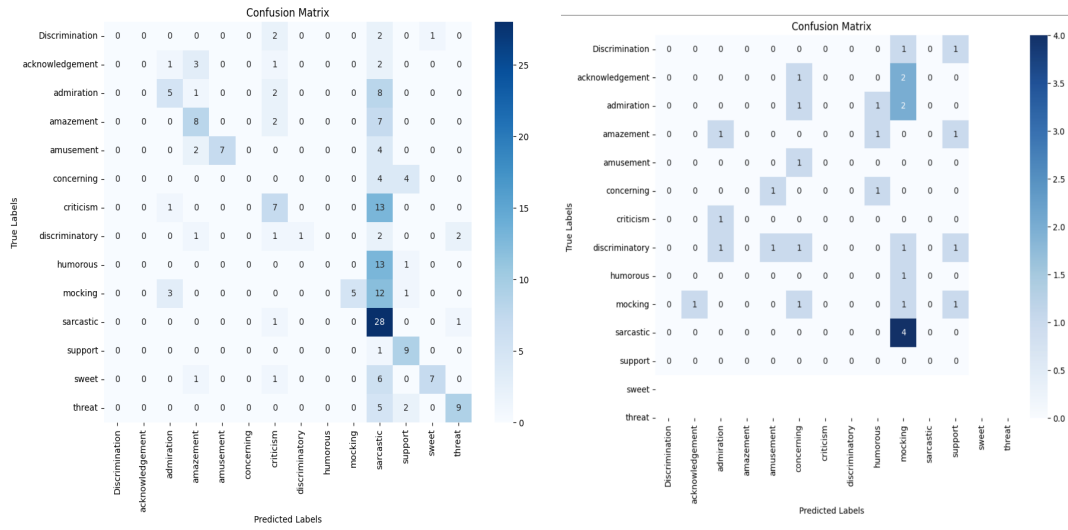


FIGURE 3.10: Tuning Multiclass Classification Result on Training and Testing Set

In Figure 3.10 the right figure, the result of predicting 20 sentences could not cover labels “sweet” and “threat” even we have tried to shuffle the data because there is not much testing data that contain that two labels and in the figure left is the result of testing on the training set so we can not make a prediction on that two labels and make conclusion exactly the result of multiclass classification. However, we could see the qualitative result in the next section testing on 8 sentences from open-source online messages.

Chapter 4

Qualitative Analysis

4.1 Machine Learning Binary Classification

In order to further test the performance of our models on some specific cases, we customized some examples that do not exist in the training or testing data. Our models have been employed to predict whether these examples belong to hate-speech or not. An interesting phenomenon has been investigated as follows:

Sentence 1: "MUGGLES NOT WELCOME"

=> SVM = Non-hate, LightGBM = Non-hate, MLP classifier = Non-hate

Sentence 2: "MUGGLES NOT WELCOME👊"

=> SVM = Hate, LightGBM = Hate, MLP classifier = Hate

To be not offensive, we used a concept "muggle" in J. K. Rowling's Harry Potter universe, which means a non-magical person. It can be noticed that, firstly if we only put a textual sentence "MUGGLES NOT WELCOME", all the predicted results from these three models are "Non-hate". If we literally analyse the ground truth of this sentence, it is vague to say the sentence is hateful or not, since it can be interpreted as "we not welcome muggles" or "muggles are not welcomed (in the magical world)". The first interpretation is a kind of hateful with discrimination, while the second is more like a statement. However, if an emoji of oncoming fist is added at the end of the sentence, the situation changes a lot: it will bring an aggressive expression, which is like someone wants to beat the muggles up. Indeed, the predicted results from all the three classifiers present that this sentence is hateful. From this example, it is able to be investigated that the emoji in a sentence does influence the its sensation - hateful or not; and it can also be found that the emoji takes more weight which is able to impact the predicted results of the binary classifiers.

4.2 Deep Learning Binary and Multiclass Classification

In this step, we also take a look into how the model will predict more specifically because we will take a few messages from an online resource ¹ that already has some datasets about hate and non-hate speech. So, we just took a few interesting messages to apply to our model to see how it predicts along with modifying that messages to see how the model will change the result. This way we will see the messages and predict them as what we think they should be and analyze how the model will predict

¹<https://www.kaggle.com/code/kerneler/starter-seven-nlp-tasks-with-twitter-fbb2ee39-0/input>

each token weight contribution. Therefore, we have got several sentences below.

Sentence 1: “😄 I need a bitch thats gon pay all my bills I dont like broke women 🙄👤🙄👤🙄” (Ground Truth= Hate)

==> CNN = Hate, Bi-LSTM = Hate, “humorous”

In this one we try to delete each token to see which token contributes to making the model predict it as hate messages. At first, we think that emojis are the most weight value that make the model predict them as hate messages but surprisingly, the emojis cannot make a change to the model prediction and we try to delete every token until this “I need a thats pay all my I like women”

After that, CNN still predicts it as hate, but Bi-LSTM predicts it as non-hate but if we add token “bills” Bi-LSTM predicts it as hate. So, it means tokens “bills” have the most weight value compared to others even the emojis. Then, we delete the token women and CNN predicts it as non-hate.

Sentence 2: “When you’re livin life on the edge but safety always comes first... 🙄” (Ground Truth= Non-Hate)

==> CNN = Non-Hate, Bi-LSTM = Non-Hate, “Sarcastic”

This one we may think that by adding this emoji 🙄 the model might think of it as a little hate message but its okay for both models with this type of message.

Sentence 3: “DEPORT RAPEFUGEE’S!!! 🙄🙄🙄🙄🙄🙄🙄🙄” (Ground Truth= Hate)

==> CNN = Non-Hate, Bi-LSTM = Non-Hate, “Sarcastic”

Sentence 4: “REFUGEES NOT WELCOME👊👊👊” (Ground Truth= Hate)

==> CNN = Non-Hate, Bi-LSTM = Non-Hate, “Sarcastic”

For Sentences 3 and 4, even if we replace these emojis with other hate emojis such as 🙄🙄🙄 but still both models predict it as non-hate speech, so maybe because of the written message itself the models have not deal with this capital letter yet.

Sentence 5: “@user stop controlling me!!!! 🙄🙄🙄🙄🙄” (Ground Truth= Non-Hate)

==> CNN = Non-Hate, Bi-LSTM = Non-Hate, “amusement”

When there are no emojis we would consider it a non-hate message but it is like an anger message when we put those emojis we can consider it as a joke and the model could understand the meaning of these emojis as well. Moreover, we try to understand the model emoji’s weight contributes to this prediction so we test it without adding the emojis like this “@user stop controlling me!!!!” and CNN still thinks it is non-hate while Bi-LSTM predicts it as hate. So this time we could see CNN predict better than Bi-LSTM.

Sentence 6: “ Im fckin dead @user 🐱💀 I fckin miss you hoe slut skank beetch 🥰💖👉 ” (Ground Truth= Non-Hate)

==> CNN = Hate, Bi-LSTM = Hate, “mocking”

Bi-LSTM is robust to “skank” and by this token, it predicts that this message it as hate but in this case we already delete “fckin, dead, and all emojis”. We also delete everything until this “@user I you” we could find out that because of “@user” that make CNN predict it as hate.

Sentence 7: “ ‘Replaceable’ best describes your life 😊 ” (Ground Truth= Non-Hate)

==> CNN = Non-Hate, Bi-LSTM = Hate, “criticism”

This one just the token “replaceable” could make Bi-LSTM think it is a hate message.

Sentence 8: “When they REALLY don’t want you to advertise 😂😂😂” (Ground Truth= Non-Hate)

==> CNN = Non-Hate, Bi-LSTM = Non-Hate, “mocking”

This one not even the emojis that contribute to non-hate speech but each token in this sentence is all positive.

Overall, we could conclude that the binary classification model totally is not working with this kind of sentence 3 and 4 as all text is on capital letter so the model does not understand what that word means. Moreover, the model does not always put higher weight value to reasonable token as in sentence 6 it puts the strong weight value weird token as supposed to have lower weight value. However, sometime these 2 models could provide reasonable answer as well in sentence 5 and sentence 2. But if we take a look at the multiclass classification result for sentence 3 and 4 even the sentences texts are all in capital but the model could predict it correctly as we seen in the method section describe the “bert-base-uncased” was trained with raw data so it can be deal with such these sentences. For sentence 7, as the word “replaceable” it is similar meaning to the “criticism” label so the model may be confused about that kind of word. For sentence 8 the meaning is a little bit as “mocking” because of the emojis used and as well as the context surrounds it.

Chapter 5

Conclusion

Finally, we could build the classification of emoji-based hate speech using both machine learning and deep learning approaches, especially we could get various models that can accurately identify hate speech from text data specifically with emojis. Along the project building, we could be involved in all of the NLP tasks including data scraping from social media, data annotation, data preprocessing, feature extraction, and model training for both binary and multi-class classification using various machine learning algorithms such as SVM, LightGBM, MLP classifier, and MultinomialNB, Deep learning models specifically CNN and Bi-LSTM, are also implemented and evaluation based on the quantitative and qualitative analyses are conducted to assess the performance and interpretability of the models, especially for qualitative analyses we try to use data from other domain (online open-source dataset) along with custom the text input to analyze the result more deeply. For machine learning, the results indicate that SVM, LightGBM, and MLP classifiers perform well in binary classification, while MultinomialNB is effective for multiclass classification. For the deep learning model, we made various comparison results between the same model by using multiple configurations for training and compared to other models as well such as using FastText pre-trained embedding model compared to the result of using only deep learning model embedding layer, additionally, we also made a comparison between the performance result of the model with various amount of training set and finally, we could find the best model for our task which is Bi-LSTM using FastText pre-trained embedding model which can achieve about 63% of F1-Score compared to CNN. Moreover, we also try to use the pre-trained model from hugging face to apply to our multi-class classification as well. Problem and discussion are also provided such as overfitting, limited sample size, and feature selection in the conclusion section as well. Finally, future work includes exploring Emoji2vec and conducting multiple training runs for improved stability and performance.

5.1 Problem and Discussion

For the machine learning model, according to the results above, we investigate that for the binary classification task the SVM, LightGBM, MLP classifier with proper hyperparameters has better performances among all the models we attempt; for the multiclass classification task, MultinomialNB with suitable hyperparameters also performs well. In terms of the feature extraction method, the models performances with the usage of Hash Vectorizer and TF-IDF Vectorizer is more ideal. Generally speaking, TF-IDF Vectorizer even performed better on machine learning part of our project. There are also some interesting phenomena, for instance, when we used BernoulliNB and MultinomialNB with these three feature extraction methods it showed that they did not accept the negative value. It was because that MultinomialNB assumes that

features have multinomial distribution that is a generalization of the binomial distribution, it explained why neither binomial nor multinomial distributions can contain negative values. Another phenomenon we noticed is that the best accuracy score on the train dataset after grid searching and cross validation is always higher than that on the test dataset, especially after using SMOTE in the first sub task of multiclass classification, it means that the model may have the problem of overfitting. For the second sub task of multiclass classification, the results are poor, the possible reason maybe the size of samples is too small to fully learn the features and patterns in the data, and the process of feature selection and tuning can become unstable and unreliable. Furthermore, there are many categories while a small number of instances under each category, so merging some of the similar categories may be another issue that can be improved.

For training the deep learning model, we tried several complex configurations to apply to the model, such as increasing the epoch, batch size, increasing more layers, and batch normalization, but the result is even worse than before. We could see from 3.5 the model kept increasing the loss when we tried to build a model with the complex configuration. However, training the complex does not always get a good result; the best thing is to find suitable hyper-parameters for our work. As we can see, our first training pipeline with complex hyper-parameters took too long to train with accuracy lower than the second tuning training pipeline, which was fast and a lot outperformed. The most important part of this task is word embedding. We try to keep the size of the embedding large and play around with all other hyper-parameters, but the final result is still at most 60%. When we just changed the size of the embedding to smaller, everything improved, and we could use the small value of hidden size, but the result was above the previous one. When we increase the number of layers in Bi-LSTM, it could get almost 63% compared to before; if we increase the size of the layer, it will achieve only around 40 percent. The embedding of a small size 32 may not include much more information, while 64 is not too much or too little to capture the information of the input sentences. We also tried with the 100 embedding sizes, but the result was even worse and it took too long to finish the training. Therefore, an embedding size of 64 is the best one. Moreover, we could see a struggle for both models to analyze the token in all capital letters, and they had never learned it before, so they completely generated the wrong result. Also, each token has its own weight, as in some sentences, even if we delete emojis, it could still predict correctly. And we could find out that CNN and Bi-LSTM quietly work well with each other, as some sentences naturally suppose that it is hate, but CNN predicts it as non-hate while Bi-LSTM predicts it as hate, and vice versa. Finally, we could see that in the quantitative analysis result, Bi-LSTM outperformed CNN, but in the qualitative analysis with non-seen messages, we could see that the performance of each model is quite similar to each other. However, the training time for CNN is a little bit higher than for Bi-LSTM.

Additionally, the way of using emojis really change the meaning and make the annotators really think different from each other and the risk getting value of Cohen Kappa annotation result would be lower even if we already have an annotation guide. So, when we have a small amount of data we should divide each class tag to at least around 20 sentences for each class. Therefore, it would be easier to analyze for example in our data some labels have only 3 or 5 comments so when it comes to analyzing we were not sure it is correct or not, especially when we want to make the comparison.

5.2 Future Work

Since the time is limited for this project, we would like to have more attempts such as using Emoji2vec¹ (Eisner et al., 2016) while the time does not allow, this will be done in the future. In addition, for the machine learning part, we will try to multiple training runs, obtain multiple sets of training results and observe how the model's performance varies across different runs, so that the stability of the model can be ensured. For the further study, the numbers of samples should be increased to obtain a better performed model, and we can also consider how the repeated times of emojis influence the extent of emotion in the sentences, for instance, assigning more weights to emojis can be a strategy to emphasize their importance in capturing the emotion. Accordingly, considering the amount of new dataset now, we would like to scrape more data from social media and reduce the annotation type to be only 14 types (as we combined in the multiclass classification training and testing section) to be the training set to see the performance of our final defining model again.

5.3 Publication

We also publish our code in GitHub², in this GitHub we provide documents and codes to scrape data from social media, how we check the statistic and analyze the data, as well as compute the annotate agreement score along with the annotation scheme, code to train machine learning and deep learning model, code to test model performance by customised inputs for models. For the dataset we obtained it from social media (TikTok and YouTube) in the user comment video we really care about data privacy so we will not share this stuff but we can provide with annotation guide, and public libraries we use to scrape that comment so if the other wants to obtain dataset just follow those steps the same as us. Additionally, we also provide the final best model (machine learning, deep learning for both binary and multiclass classification) to test how it will predict and see its performance as well.

¹<https://github.com/uclnlp/Emoji2vec>

²<https://github.com/SrunNalin/Emoji-Based-Hate-Speech-Detection-on-Social-Media>

Bibliography

- Althobaiti, Maha Jarallah (2022). “BERT-based Approach to Arabic Hate Speech and Offensive Language Detection in Twitter: Exploiting Emojis and Sentiment Analysis”. In: *International Journal of Advanced Computer Science and Applications* 13.5, pp. 972–980. DOI: [10.14569/IJACSA.2022.01305109](https://doi.org/10.14569/IJACSA.2022.01305109).
- Bojanowski, Piotr et al. (2017). *Enriching Word Vectors with Subword Information*. arXiv: [1607.04606](https://arxiv.org/abs/1607.04606) [cs.CL].
- Chawla, Nitesh V et al. (2002). “SMOTE: synthetic minority over-sampling technique”. In: *Journal of artificial intelligence research* 16, pp. 321–357.
- Cortes, Corinna and Vladimir Vapnik (1995). “Support-vector networks”. In: *Machine learning* 20.3, pp. 273–297.
- Devlin, Jacob et al. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv: [1810.04805](https://arxiv.org/abs/1810.04805) [cs.CL].
- Duchi, John, Elad Hazan, and Yoram Singer (2011). “Adaptive subgradient methods for online learning and stochastic optimization.” In: *Journal of machine learning research* 12.7.
- Eisner, Ben et al. (Nov. 2016). “emoji2vec: Learning Emoji Representations from their Description”. In: *Proceedings of the Fourth International Workshop on Natural Language Processing for Social Media*. Austin, TX, USA: Association for Computational Linguistics, pp. 48–54. DOI: [10.18653/v1/W16-6208](https://doi.org/10.18653/v1/W16-6208). URL: <https://aclanthology.org/W16-6208>.
- Graves, Alex (2013). “Generating sequences with recurrent neural networks”. In: *arXiv preprint arXiv:1308.0850*.
- Ho, Tin Kam (1995). “Random decision forests”. In: *Proceedings of 3rd international conference on document analysis and recognition*. Vol. 1. IEEE, pp. 278–282.
- Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long short-term memory”. In: *Neural computation* 9.8, pp. 1735–1780.
- Ke, Guolin et al. (2017). “Lightgbm: A highly efficient gradient boosting decision tree”. In: *Advances in neural information processing systems* 30, pp. 3146–3154.
- Kingma, Diederik P and Jimmy Ba (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980*.
- Kirk, Hannah et al. (July 2022). “Hatemoji: A Test Suite and Adversarially-Generated Dataset for Benchmarking and Detecting Emoji-Based Hate”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, United States: Association for Computational Linguistics, pp. 1352–1368. DOI: [10.18653/v1/2022.naacl-main.97](https://doi.org/10.18653/v1/2022.naacl-main.97). URL: <https://aclanthology.org/2022.naacl-main.97>.
- LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep learning”. In: *nature* 521.7553, pp. 436–444.
- Loshchilov, Ilya and Frank Hutter (2017). “Decoupled weight decay regularization”. In: *arXiv preprint arXiv:1711.05101*.
- Mubarak, Hamdy, Sabit Hassan, and Shammur Absar Chowdhury (2022). “Emojis as Anchors to Detect Arabic Offensive Language and Hate Speech”. In: *Natural Language Engineering*, pp. 1–21.

- Pedregosa, Fabian et al. (2011). “Scikit-learn: Machine learning in Python”. In: *the Journal of machine Learning research* 12, pp. 2825–2830.
- Pereira-Kohatsu, Juan Carlos et al. (2019). “Detecting and Monitoring Hate Speech in Twitter”. In: *Sensors (Basel, Switzerland)* 19.
- Schmidt, Anna and Michael Wiegand (2017). “A survey on hate speech detection using natural language processing”. In: *Proceedings of the fifth international workshop on natural language processing for social media*, pp. 1–10.
- Shardlow, Matthew, Luciano Gerber, and Raheel Nawaz (Mar. 2022). “One emoji, many meanings: A corpus for the prediction and disambiguation of emoji sense”. In: *Expert Systems with Applications* 198, pp. 116–862. DOI: [10.1016/j.eswa.2022.116862](https://doi.org/10.1016/j.eswa.2022.116862).
- Vega, FA et al. (2009). “Classification and regression trees (CARTs) for modelling the sorption and retention of heavy metals by soil”. In: *Journal of Hazardous Materials* 167.1-3, pp. 615–624.
- Waseem, Zeerak et al. (2017). “Understanding abuse: A typology of abusive language detection subtasks”. In: *arXiv preprint arXiv:1705.09899*.
- Wiegand, Michael and Josef Ruppenhofer (2021). “Exploiting emojis for abusive language detection”. In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 369–380.