Using machine learning to detect hate speech and offensive language through instant messaging

Final Report

DT282
BSc in Computer Science International

Daniel Krasovski

C18357323

Svetlana Hensman

School of Computer Science

Technological University, Dublin

08/04/2022

Abstract

Since the invention of the internet, one of the most used technologies is instant messaging. While this is not bad in anyway, it brings along a new way to spread offensive and hateful messages and statements. Especially in a medium such as Discord where there are many young users and no built-in moderation or detection of such messages. Users can be exposed to very hateful and offensive messages at an age where they should not. This is the problem I am attempting to solve in this final year project. I am going to be using Machine Learning and NLP to detect Hate speech and Offensive messages real time and use the built-in tools in discord to deal with the messages through various methods.

Disclaimer: Due to the nature of this project. This document has sentences that contain both hate speech and offensive language. Everything was written for the purpose of testing and not to offend any specific group or person.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Daniel Krasovski

04/04/2022

Acknowledgements

I would like to Acknowledge my project supervisor Svetlana Hensman for guiding me and making sure I am always on the right track. I would also like to thank my great friends that I made in this course Ire Adebari, Raphael Ofeimu, Kyle Butler and Nikolay Malyshev for all the help, advice, and encouragement I received during the previous 4 years.

Table of Contents

Table of Figures

1. Introduction

1.1. Project Background

Multiple research has shown that social media posts and other online messages can cause an increase in acts of violence. For example in Germany it was shown that hate speech on social media and messaging apps can lead to an uptick in hate crimes.(1)

With the amount of content being posted online being far too greater for any human to preview beforehand. This is also not considering the bias's they could have. So, to solve this problem I am going to attempt to solve this problem by using machine learning to train a model that can differentiate between hate speech, offensive language, and just normal messages.

There are also other issues apart from real-world violence with being exposed to hateful messages. Mental Health is also very affected by hateful messages. A study found that exposure to hateful messages leads to greater stress among college students.(2)

The goal of my project is to try to detect and deal with these kinds of messages.

Hate speech and offensive language can be very subjective. And there is no one label that can be put upon hate speech or offensive language. So, what one person could find offensive, another could not. In this project I will be attempting to deal with the most extreme of both.

1.2. Project Description

My project is going to be a machine learning model that when trained using appropriate data will be able to detect hate speech and offensive messages. To test and showcase the possibilities of what to do once the messages have been detected, I will be using the Discord API to make a discord bot. I will be using the discord.py wrapper which is written in python.(3)

The complexity of this project comes from the machine learning and real time detection. Every time a message is sent in a chat it will be scanned and then dealt with appropriately. Making sure the model is trained adequately and the differentiation between hate speech and offensive language will also be very import to get done correctly. The main data set which I will be using comes from the paper Automated Hate Speech Detection and the Problem of Offensive Language.(4) The data is a collection of tweets containing lexicon from hatebase.org. I will also attempt to implement a feature where the dataset could continuously get updated and retrained to improve accuracy and include more up to date lexicons.

Through my previous experience with machine learning algorithms and NLP, the difficult part of this project is going to be performance. Making sure the model can be trained in adequate time and that

the algorithm does not need to run for each time a prediction is required. Furthermore, connecting the backend machine learning and the front-end Discord API will also be a challenge and will have the possibility of running into issues that I could not foresee.

The approach for the project that I will take will be one of Feature-Driven Development (FDD). FDD is related to Scrum but instead of being delivery focused, it is feature focused.(5) I will work on the main feature of message detection and classification and once it is finished move onto new features and only move one once the feature is finished or is completely abandon features that are not feasible to implement.

The screenshots below show a possibility of a response once a valid detection is found

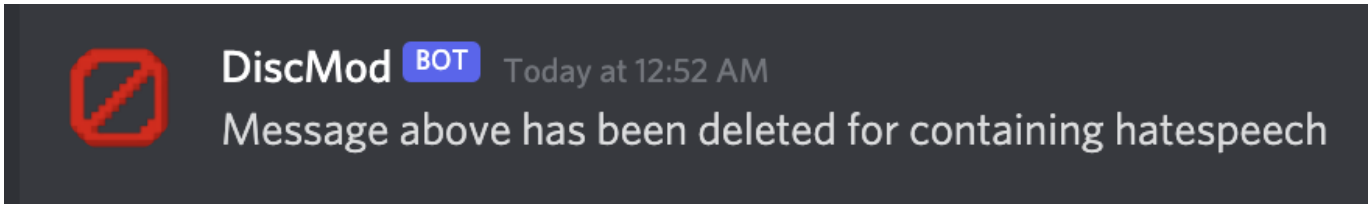


Figure 1 Example of Bot response

1.3. Project Aims and Objectives

The overall aim of this project is to provide a Discord moderation bot that can be put onto servers that have hundreds of messages sent per minute, where a group of moderators would not be able to handle spotting, analysing, and dealing with such messages. The features that I hope to have implement by the end of the project are:

Hate speech detection – scan every message sent and use a prediction model to get a probability if the message sent contains hate speech

Offensive language detection – same as hate speech except it will be for offensive language

And if I get these features finished, I hope to work on the following afterwards:

Data collection – let users report messages that do not get flagged and have moderators look over it and decide if it fits into any category

Increase languages supported, currently it will only work in English. Find a feasible way to add European languages such as German and French.

Find a way to implement a way to extract text from images and run that text through the classifier

1.4. Project Scope

This project will not end up solving the whole problem of offensive and inappropriate content being posted online. I will just be focusing on the instant messaging side of the problem. I will also be looking more at the technical aspect and how difficult each aspect would be to implement on a large scale. I will also not be making a standalone texting app to showcase this; I will instead be using the instant message app Discord.

1.5. Thesis Roadmap

1.5.1 Research

For research, this section will be about me researching similar projects to mine and what they managed to achieve. I will also be doing some research into the advantages and disadvantages of having a system like this. I will also be looking into the ethical and moral issues that could come up with such a project and find the best way to overcome these issues. It will also contain any other relevant research I will have completed for this project.

1.5.2 Design

For the design of the project, It will be about the choices I made for how it will be development and the reasoning for it. It will also contain the different datasets and methods I could have used to help with my development. Finding the correct machine learning platform and discussing the reasoning for which one I went will also be in this section.

1.5.3 Prototype Development

This section will go through all the different prototypes developed and the reasons for each prototype. There will be a discussion on the issues with each prototype and the steps taken to solve the issues

1.5.4 Project Development

This chapter will break down the entire development process, it will go into depth about any decisions made, any code I used and how they were changed and adapted. The challenges I encountered and how I overcame them will also be discussed in this chapter

1.5.5 Testing and Evaluation

This chapter will go into detail about how I tested and evaluated the project. From testing the model, to seeing how it all works together. I will also go into about how easy it would be to adapt, change and implement new features

1.5.6 Conclusions and Future Work

The final chapter will be an overview of the whole project, what was able to be completed and what other features could be added in the future.

2. Literature Review

2.1. Introduction

In this chapter I will be discussing the key areas of research I have done. I will go into detail about existing solutions, other similar projects that have been done and different technologies I have looked at and comparing the differences between each option I could have went with. I also done some research into Discord and the different API wrappers I could have used.

2.2. Alternative Existing Solutions to Your Problem

There were multiple research papers with code along it. They all used different datasets however the 2 that I found were most applicable to me were: hate-speech by PyAntony.(6) and Automated Hate Speech Detection and the Problem of Offensive Language by Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber.(4)

2.2.1 Hate speech by PyAntony

The dataset is from a white supremacist forum. There are 2392 observations with 1914 for training and 478 for testing. It uses the BERT modelling approach and has an API for prediction using post request. The project compares two approaches of the BERT model, Fine-tuning and embedding. With fine tuning appearing to be more effective with 82% accuracy and evaluation loss of 0.43

This project uses the BERT model for its natural language processing (NLP). It was created in google in 2018 and has been used in its search engine for nearly almost every engine language query. It is highly advanced and looks like to be the best option for classifying text. I will attempt to implement the BERT model for my finished model.

2.2.2 Hate Speech Detection and the Problem of Offensive Language

This paper uses a more simplistic approach to the model. It uses the sklearn library and logistic regression for its model. The interesting apart which would apply to my project would be the dataset. It is a sample of 25,000 tweets containing lexicon from hatebase.org. It was then manually coded by CrowdFlower workers whether the tweet contained hate speech, offensive language or neither. This dataset will be the one I will work with from the start and develop a protype with.

2.3. Technologies you've researched

2.3.1. Programming languages

From an early stage I knew that for this project I will be coding most of it, if not all, in python. Python is a high-level open-source language. It has a high number of libraries that allow for machine learning, data analysis and data visualisation. The language is also syntactically simple and there are extensive online resources for python and with over 60% of machine learning developers using python. It seemed like the perfect option to go with. (7)

The other option was R. R is a functional language that is primarily used for data analytics and for the visual of data. There are various resources and tutorials for it, however it does lack features compared to python and my lack of familiarity to it is the reason I did not go with it.

2.3.2. Machine learning

Machine Learning (ML) is a sub discipline of Artificial Intelligence and aims to understand and develop self-improving algorithms. There are three basic machine learning paradigms. These are supervised learning, unsupervised learning, and reinforcement learning.

Supervised Learning

Supervised learning is a way of developing and training a model using a set of pre-existing training instances. An issue is presented to the machine learning agents, and they must make a choice. This problem's right solution is already known. In the event of a loss, the model is updated after the decision has been determined. This is due to an error, which is the difference between the algorithm's option and the actual right solution. The goal of supervised learning is to generalize the training data such that the algorithm can finally make the correct decision for new issues with no prior instances. (8)

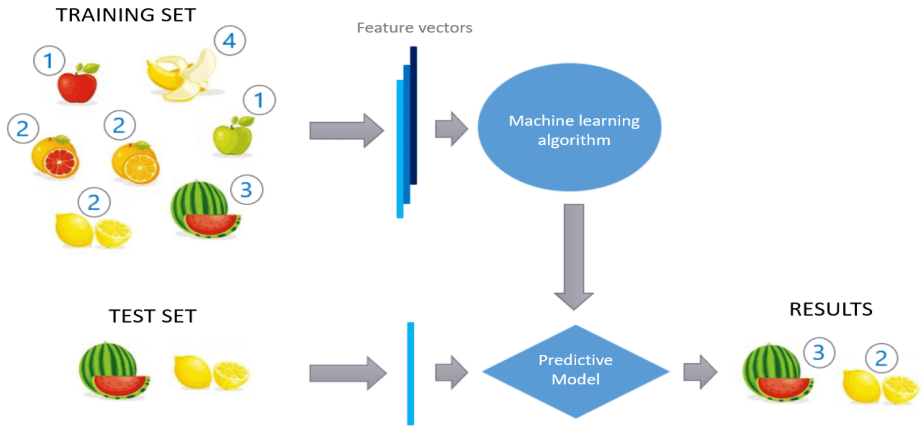


Figure 2 Supervised learning diagram(9)

Unsupervised Learning

In a similar fashion to supervised learning, Unsupervised learning utilises data sets to train machine learning algorithms. The difference lies in the goal, which is to enable the learning agent to recognise patterns in a data set, by determining how features are distributed, which it can further utilise to create clusters of similar data, extract the most important and crucial features of the data and generate new data based on the original. This training process can be applied to a neural network called auto encoder. It consists of two parts and attempts to map the output of a pair to the input by learning the identity function.(10)

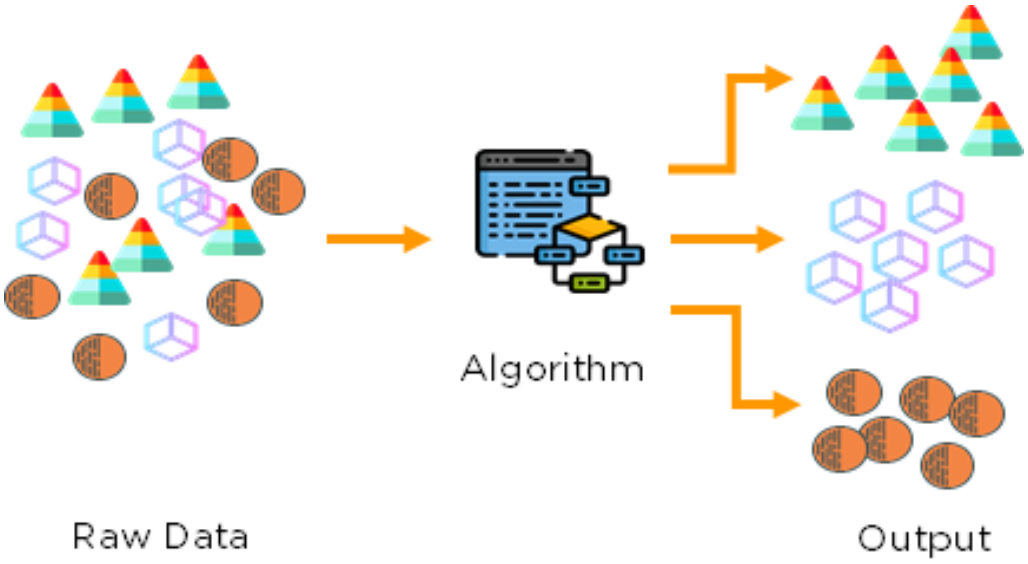


Figure 3 Unsupervised learning diagram(11)

Deep Learning

Deep learning, a branch of machine learning based on artificial neural networks, is another sub-discipline of machine learning. Deep learning makes use of neural networks to try to reproduce the way our brains work. Data is fed into artificial neural networks, which deep learning uses to train itself. The data is constantly analysed at each of the levels of the neural network, with each level focusing on different parts of the data and what might be used to differentiate it. The deeper the neural network goes, the more objects it will search for, such as eyes and noses, before attempting to determine if it comprises a face. (12)

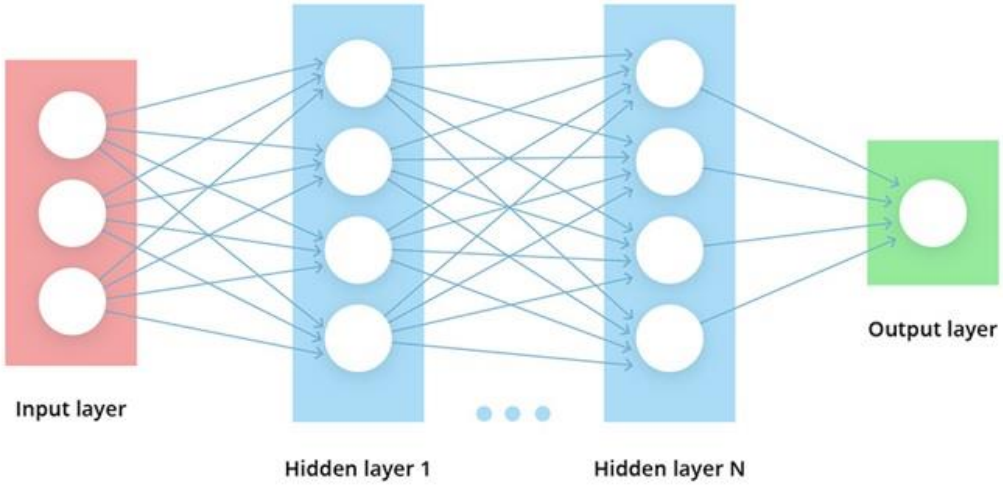


Figure 4 Deep learning neural network(13)

Natural Language Processing

To use machine learning with text it was essential to learn about Natural Language Processing (NLP). NLP is defined as the automatic manipulation of natural language using text. It has been around for over 50 years and grew with the rise of computers.(14) There are many uses of NLP anything from speech recognition, to natural language generation. The parts which I am going to attempt to implement in my project are:

- Text Classification – Text classification is a machine learning technique that categorizing text into organized groups. It can be used to understand if any text is negative or positive, what topic it is about and detect what language is being used.(15)
- Sentiment analysis – this is the process of computationally identifying and categorizing opinions from a text. It is used to determine whether the attitude used in the sentence Is positive, negative, or neutral.
- Text summarisation – This is the process in which large texts are summarised for quicker consumption(16)

2.3.3. Machine Learning Frameworks

There are lots of machine learning frameworks available for python from TensorFlow to Scikit-Learn.

TensorFlow

TensorFlow is an opensource end to end machine learning platform. It has a wide range of tools, libraries and community resources that enable users to develop machine learning algorithms easily and efficiently.(17) TensorFlow can create models for anything from image recognition to NLP. It works by allowing the developers to create structures that describe how data moves through a graph. Each node in the graph represents a mathematical operation with each edge between the nodes being a multidimensional array (Tensor). The main supported language is Python, however there are community-built APIs for other languages. The mathematical operations however are all computed through high-performance C++ binaries.

Scikit-Learn

Scikit learn is an opensource machine learning and data analysis library in python. It is considered by many to be the gold standard for machine learning in python. Scikit learn provides many algorithms for supervised and unsupervised learning and is built upon technology such as NumPy, pandas and matplotlib. Scikit learn is very easy to setup and learn to use. It is high level which allows you to define a model for prediction in a short amount of time and code.

2.3.4. Discord API wrapper

The Discord API wrapper that I will be using for this project is Discord.py. I went with discord.py because of my previous experience using it and it's easy to use, feature rich and async ready. One concerning issue at the start of the project was that the mainline development was ceased. However, by the time of writing this, development has restarted and is very active.

2.3.5. OpenCV and pytesseract

OpenCV is an open-source library that is used for real time computer vision. In this project I use it to process images for text extraction. Once the image is processed, I then use pytesseract to find all the text in the image and convert it to a string. This will then later be run against the model to check for hate speech and offensive language.

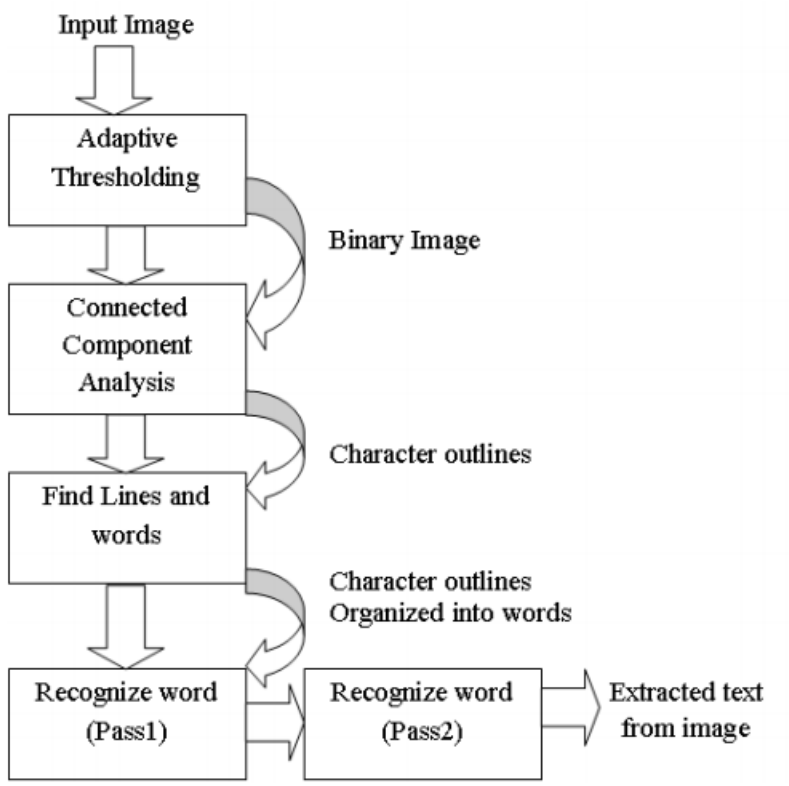


Fig 1: Tesseract flow

Figure 5 How Pytesseract and OpenCV work together(18)

2.3.6. BERT Language model

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based machine learning method for NLP developed by google in 2018. By late 2021 google has been using BERT in most of its English language queries. BERT will be used in this project to create the multi label text classification to detect both hate speech and offensive language. BERT works by taking a sentence and breaking it down into a sequence of words. It then uses a pretrained language model to create a representation of the sentence. The representation is then used to create a classification for each word in the sentence. The classification is then used to create a multi label classification. The multi label classification is then used to create a final classification for the sentence. Which is then returned

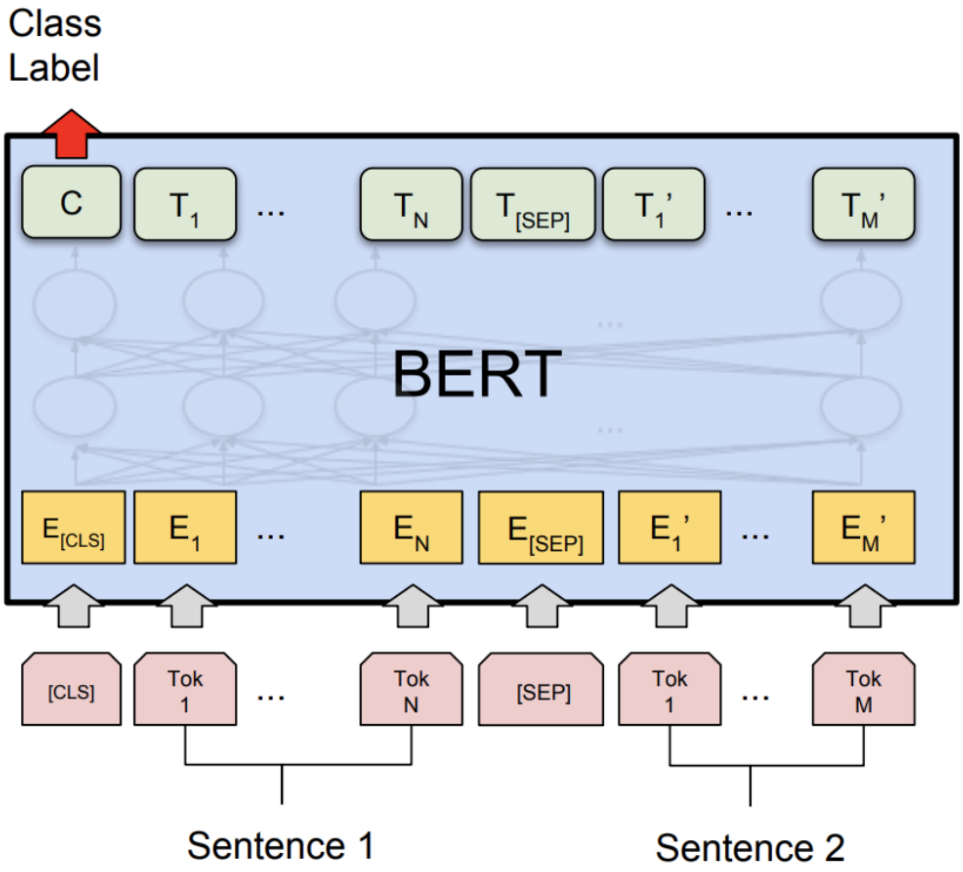


Figure 6 BERT Classification fine tuning(19)

2.4. Other Research you've done

Other research that I have done is looking into the side effects and consequences of hate speech and offensive content on the internet. Especially on young people, there can be long lasting negative effects.(20) Further more I have also looked into the issues of dealing with hate speech and offensive language and how to attempt at creating an unbiased model

2.5. Existing Final Year Projects

From my brief look at the previous final year projects, I was able to find one project that was similar to mine. It is titled Anti-Bullying with Machine Learning and it is by Shane O'Neill. The project uses various combinations of machine learning and text mining to build predictive models to detect cyber bullying. Mr O'Neill tested his models on 4 different datasets labelled for bullying or abusive content. He found that the data quality is very important and no one model can be used for all the datasets. He also created a demonstration interface where a user can enter any sentence and it would predict

if the sentence entered contains bullying. This final year project contains very useful information and is very insightful.(21)

2.6. Conclusions

From all this research, I feel very confident in the development of this project. The extensive technologies available and the vast amount of similar research papers into this topic gives me a very good platform to improve on all the previous work. Connecting to the discord API will also be very approachable due to the simple nature of the wrapper and with my previous experience using it.

Requirements Table

Name	Description	Priority
Hate speech	Accurately detect hate speech	HIGH
Offensive Language	Accurately detect offensive language	HIGH
Discord API	Connecting the model to the discord API	HIGH
Reporting system	Allow users to report messages that are not detected	MEDIUM
Spam detection	Detect when a user is spamming	MEDIUM
Dataset improvement	Improve the dataset for more accurate results	MEDIUM
Level of moderation	Make it possible to choose how “strict” the detections will be.	MEDIUM
Other languages	Look into supporting other languages	LOW
Text summarisation	Summarise messages that have been sent	LOW
Toxicity metric	Have a metric for each user, to see how offensive/ hateful they are	LOW
Text from image Extraction	Extract text from images and classify text to check if image contains hate speech or offensive content	LOW

3. Project Design

3.1 Introduction

In this chapter, I will be discussing how I will design and develop the prototype for this project. I will start with the methodology where I will discuss which method I chose and why. I will also compare other methods I could have gone with. Afterwards I will discuss the overview of the system. This will be about how the front end and the backend will interact and how they will work. Lastly, I will end with how I want to the prototype to look and function. I will also talk about the Development of the prototype in this section also as I done both steps at the same time. This is will also reduce the amount of information that will be repeated by having the development in its own section.

The GitHub link for the project is: https://github.com/Meapy/FYP---Realtime-hatespeech-detection-on-discord-messages

3.2. Software Methodology

3.2.1 Agile

The Agile development method Is used to reduce risk such as bugs, cost and changing requirements when adding new functionality. There are multiple forms of agile development including scrum, extreme programming (XP) and feature driven development. The main benefit of Agile development is that it allows for releases in iterations. This allows the developers find bugs and fix them earlier on and let the users to use the software earlier. One big downside, however, is that when working in big teams, it requires good communication and high time commitment to make it work efficiently.

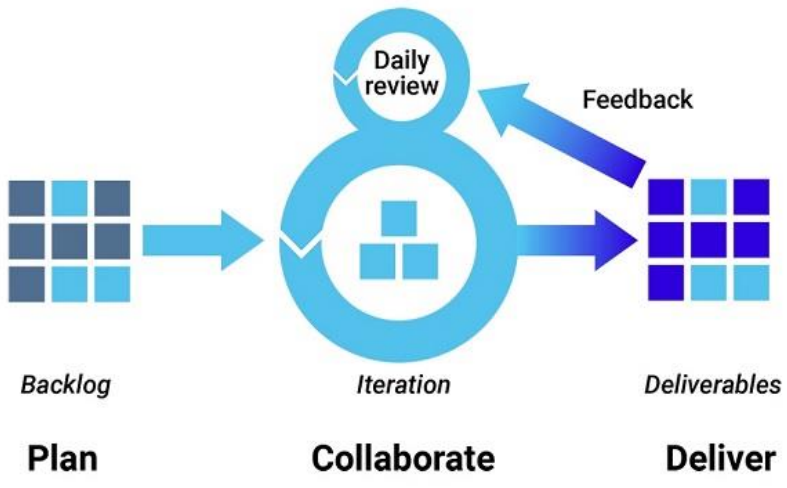


Figure 7 Agile development diagram (22)

Like I mentioned earlier in the paper, I will be using Feature driven development (FDD). FDD is iterative and incremental with the goal being delivering results as frequently as possible. It also encourages the use of proper documentation at all levels to improve progress and track the results. FDD works using five basic steps,

1. Build overall model
2. Create feature list
3. Plan each feature
4. Design the feature
5. Build the feature

The first two steps are done at the start while the final 3 steps are repeated for all the features in the project. Most of the time will be spent on designing and building the feature in FDD.(5)

3.2.2 Rapid Application Development

Rapid application development (RAD) is a shortened development approach that has the goal of delivering a high-quality system with a low cost of investment. It facilitates the ever-changing demands and requirements from the users and allows the developers to quickly adjust.

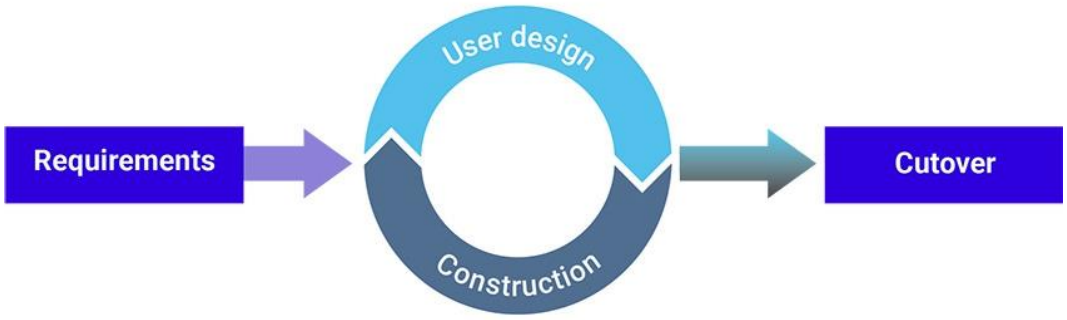


Figure 8 Rapid application Development (16)

The 4 stages of RAD are requirements, user design, construction, and cutover. The middle two phases are repeated until the product meets all the requirements. RAD is most effective for small projects that are time sensitive. (22)

3.3. Overview of System

With the FDD approach being used for this project. Each feature will be planned, implemented, and tested one at a time. Once one feature is finished, I will move onto the next one. I decided to go with this approach because I would like to implement as many features as possible without having some features not fully completed or implement.

The design and code will be developed in stages. Where each feature will be researched and designed before the implementation will being. This will make sure that the development of each

feature will be done with most efficient method and will reduce the risk for starting a feature that will not be feasible to implement. The approach will consist of:

1. Design and implementation of the twitter dataset prediction with the discord API and have the prediction working. accuracy will not matter too much as this will be the basis for being able to connect the front end with the backend.
2. Design other feature (such as proper prediction)
3. Implement the feature
4. Test the feature
5. Repeat steps 2-4 until no other features could be implemented. (Due to time or all features have been developed)

The next diagram will show the two layers of the program and how they will communication with each other.

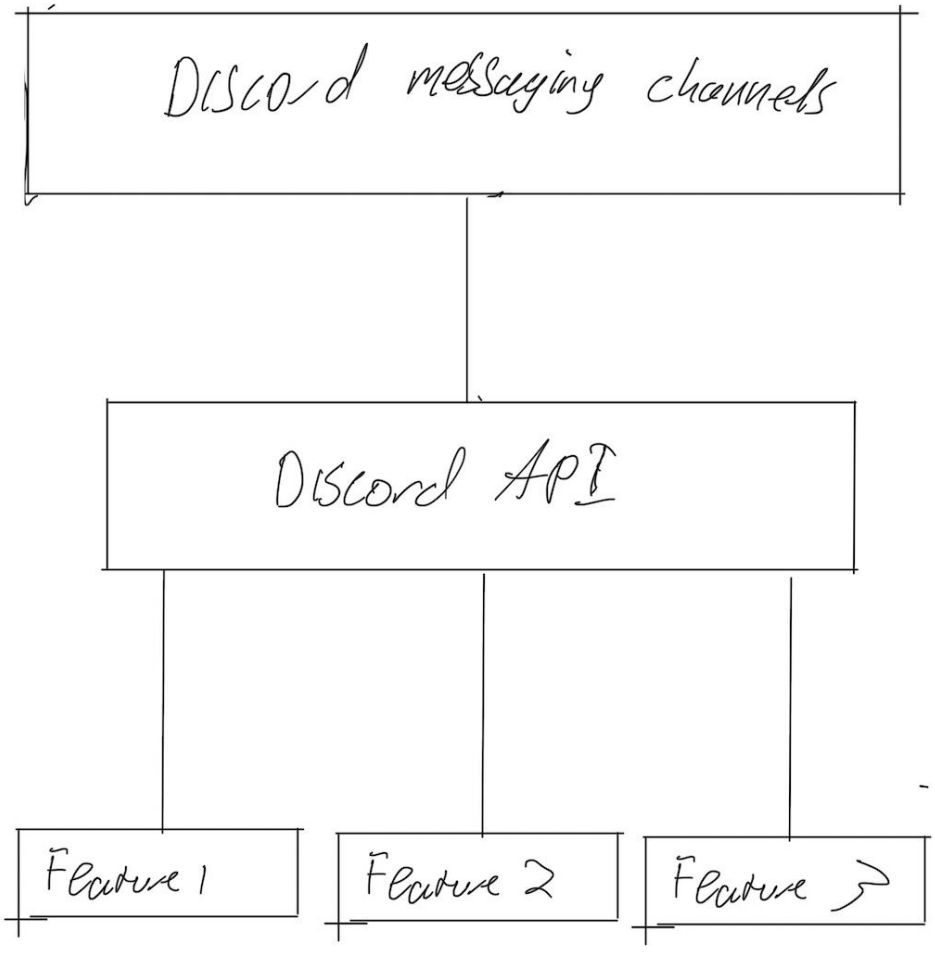


Figure 9 Overview of System

3.3. Overview of dataset

The dataset is a sample of 25,000 tweets from hatebase.org that contain vocabulary. CrowdFlower staff then manually coded whether the tweet contained hate speech, offensive language, or neither. The dataset is not uniformly distributed throughout the classes, which must be taken into account when training. The dataset is split into 5 different headings.

- Count – number of CrowdFlower employees that judged each tweet
- Hate_speech – how many CrowdFlower employees judged the tweet to contain hate speech
- Offensive_language - how many CrowdFlower employees judged the tweet to contain offensive language
- Neither - how many CrowdFlower employees judged the tweet to contain neither hate speech or offensive language
- Class – the classification of the tweet.
    - 0 - hate speech
    - 1 - offensive language
    - 2 - neither
- Tweet – the string of the tweet

	id	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	2	!!! RT MENTIONHERE: As a woman you shouldn't c...
1	1	3	0	3	0	1	!!!!! RT MENTIONHERE: boy dats cold...tyga dwn...
2	2	3	0	3	0	1	!!!!!!! RT MENTIONHERE Dawg!!!! RT MENTIONHERE...
3	3	3	0	2	1	1	!!!!!!!!! RT MENTIONHERE: MENTIONHERE she look...
4	4	6	0	6	0	1	!!!!!!!!!!!!! RT MENTIONHERE: The shit you hea...

Figure 10 First 5 columns in dataset

4. Prototype Development

4.1. Discord

The front-end messaging channels will be the discord group chats. To add the bot to a server, first you must have the correct permission and afterwards you just have to click on a link and then select the server you want to add it to. This is how this step looks:

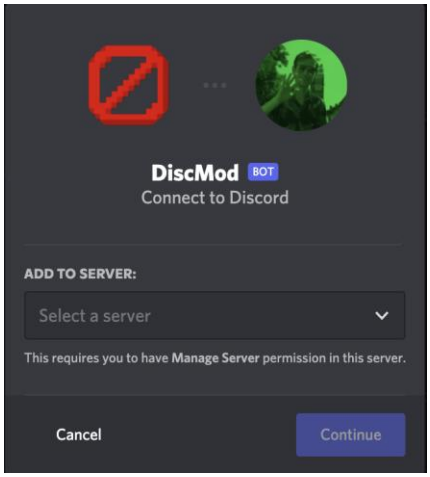


Figure 11 Add to server screenshot

Once the bot is in the server, it will be able to read the messages sent in every channel in the server. You can change the permissions of which channels the bot is able to see by changing its permissions. As seen in this image:

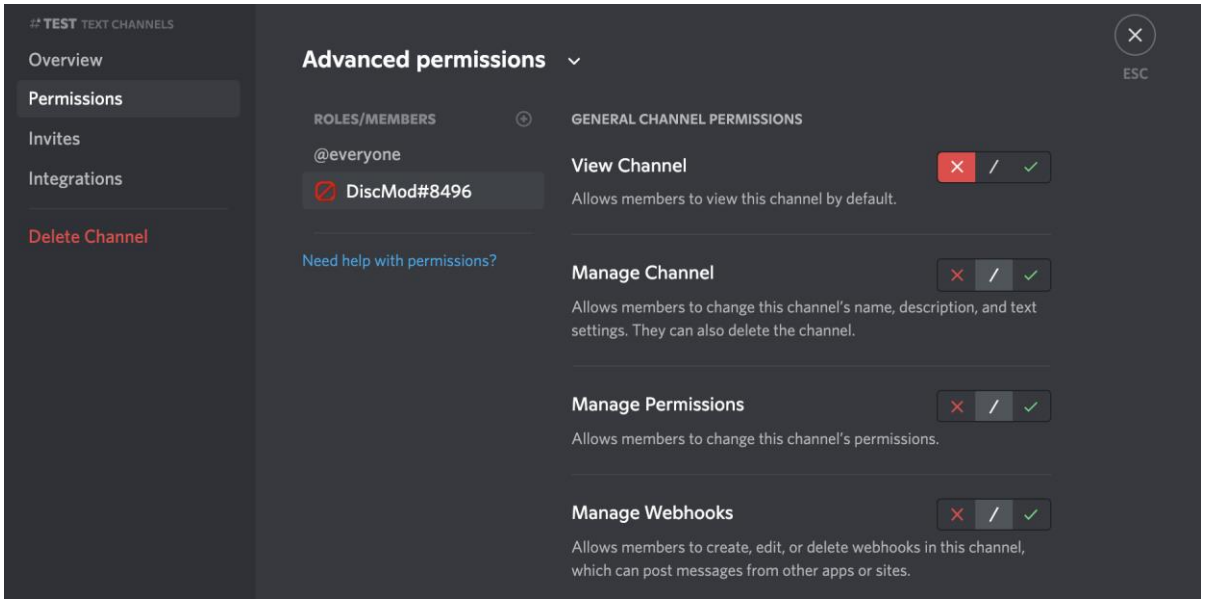


Figure 12 Edit permissions screenshot

Once the permissions are set up and you gave the bot to view a channel. You will be able to see the bot come up on the members list on the right-hand side. The bot will now be able to view the messages and interact with them. This is how the users will be able to interact with the bot.

4.2. Backend

4.2.1 Introduction

Throughout the development of this project, there were 3 prototype machine learning models for the hate speech and offensive language detection built. With the first two prototypes having major

issues that prevented me from using the model. The reasons why will be explored in this section along with the benefits and drawbacks of each model.

4.2.2 Prototype One

With the first prototype, I created a simple script that connect the bot to the servers it was added to. Afterwards I created a command that would allow a user to pass a message to the bot for prediction and the bot would respond with its prediction. Here is a code snippet of the discord bot using the discord.py API wrapper.

```
load_dotenv()
TOKEN = os.getenv("DISCORD_TOKEN")

client = commands.Bot(command_prefix='~')

@client.event
async def on_ready():
    guilds = list(client.guilds)
    print(f'{client.user} is connected to the following guilds:\n')
    for guild in guilds:
        print(f'{guild.name}(id: {guild.id})')

@client.command(pass_context=True)
async def test(ctx, *, message):
    channel = ctx.channel

    question = Classifier.predict_class(np.array([(str(message)), 0]))
    if(question == 0):
        response = "Message above has been deleted for containing hatespeech"
    elif (question == 1):
        response = "Message above has been deleted for containing offensive language"
    else:
        response = "No action taken"
    if not ctx.author.bot:
        await channel.send(f'{response}')

client.run(TOKEN)
```

Figure 13 Discord API wrapper prototype code

For the Classification I used the Automated hate speech detection and problem of offensive language project as the basis for which I made the classifier. However, it would not work without making changes. The changes I had to make to make the classifier were:

1. Update the print statements
2. Update Unicode to str
3. Save the tweet tags to a file (this was done to speed up the training process as obtaining the tweet tags each time took a considerable amount of time)
4. Update the pipeline
5. Create the prediction function

Here is a code snippet of the prediction function:

```
def predict_class(text):
    feats = get_feature_array(text)
    text_df = pd.DataFrame(text)
    tfidf = vectorizer.fit_transform(text).toarray()
    tweet_tags = []
    for t in text:
        tokens = basic_tokenize(preprocess(t))
        tags = nltk.pos_tag(tokens)
        tag_list = [x[1] for x in tags]
        tag_str = " ".join(tag_list)
        tweet_tags.append(tag_str)
    pos = pos_vectorizer.fit_transform(pd.Series(tweet_tags)).toarray()
    M = np.concatenate([tfidf, pos, feats], axis=1)
    final = pd.DataFrame(M)
    # check which columns are expected by the model, but not exist in the inference dataframe
    not_existing_cols = [c for c in X.columns.tolist() if c not in final]
    # add this columns to the data frame
    final = final.reindex(final.columns.tolist() + not_existing_cols, axis=1)
    # new columns dont have values, replace null by 0
    final.fillna(0, inplace=True)
    # use the original X structure as mask for the new inference dataframe
    final = final[X.columns.tolist()]
    final.dropna(inplace=True)
    # predict the class of the new dataframe
    preds = model.predict(final)
    # if the probably of the prediction is higher than 0.7 and 0.4 respectavility, return the prediction
    if preds[0] == 1 and model.predict_proba(final)[0][1] > 0.7:
        return preds[0]
        pass
    elif preds[0] == 0 and model.predict_proba(final)[0][0] > 0.4:
        return preds[0]
        pass
    else:
        pass
```

Figure 14 prediction prototype function

With the Both of them connected, here is how it works in a discord channel:

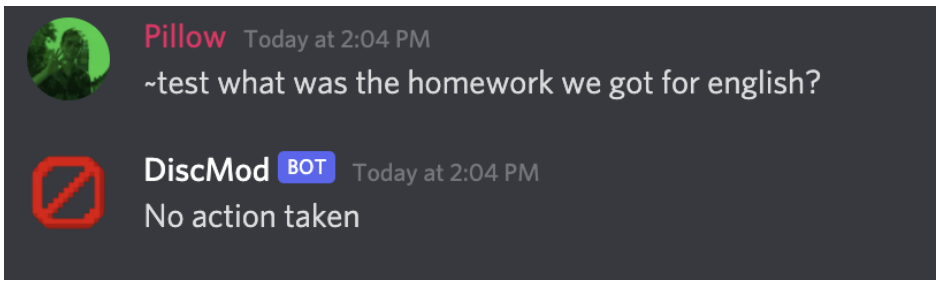


Figure 15 Porotype prediction

Prototype issues

I came across many issues while developing my prototype. I will discuss the issues in 3 parts:

1. Existing solution
2. Adapted solution
3. Prototype solution

Existing Solution

The first issue I came across was when I cloned the hate-speech-and-offensive-language repository was to get it to work. Both the Jupiter notebook file for python 3.6 and the final classifier would not run because the code was out of date. For the classifier the model files were also made with an older version of Scikit learn. The code was not maintained from 2019 so there were multiple things I had to change to get it to run. So, I began to edit the code to make the adapted solution

Adapted Solution

For the adapted solution, there were multiple changes I made, the simple fixes were the print statements and Unicode issues and the more complex issues where the performance issue, model training and then prediction. For the performance, the code would get the tweet tags every time the model was retrained so I solved this issue by saving the tweet tags to a txt file and then checking if the file existed every time the code ran. Code snippet on next page

```
tweet_tags = []
#if the tweet_tags.txt exists in the data folder open it and assign it to the tweet_tags
variable, else run the for loop
try:
    with open('data/tweet_tags.txt', 'r') as f:
        tweet_tags = f.read().splitlines()
except:
    #Get POS tags for tweets and save as a string
    for t in tweets:
        tokens = basic_tokenize(preprocess(t))
        tags = nltk.pos_tag(tokens)
        tag_list = [x[1] for x in tags]
        tag_str = " ".join(tag_list)
        tweet_tags.append(tag_str)
    #save the tweet tags to a file for later use
    with open("data/tweet_tags.txt", "w") as f:
        for t in tweet_tags:
            f.write(t + "\n")
```

Figure 16 Prototype tweet tags solution

Afterwards I had to fix an issue with the pipeline by updating the solver to libliner because the penalty was l1, another solution was changing the penalty to l2. Once these issues were solved, I was able to run the jupyter notebook file. The Accuracy however was significantly worse as seen by comparing both confusion matrixes:

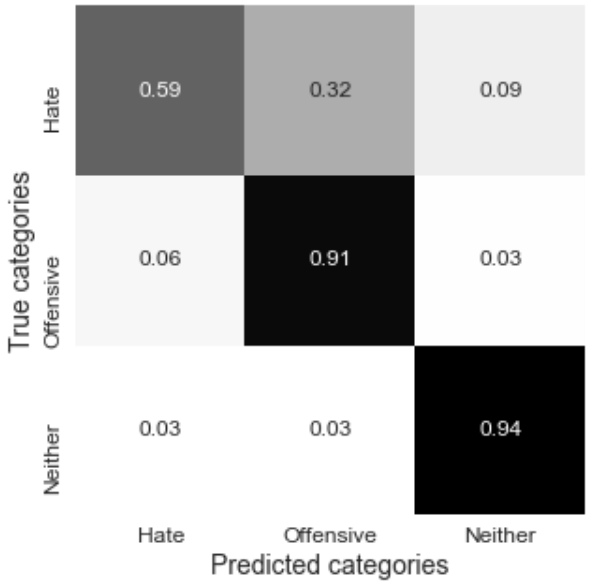
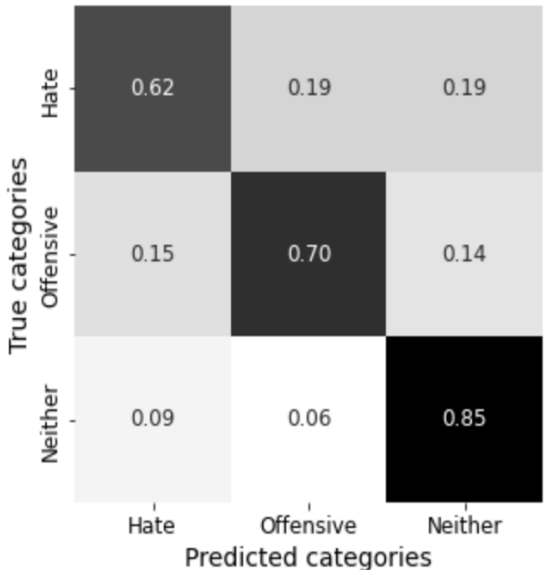


Figure 17 Original prototype confusion matrix(23)



and my confusion matrix

Prototype Solution

For the standalone classifier to the protype, I worked off the final classifier in the repository and off the adapted solution. I could not get any of the pickle files (for the vectorizer) or the model file to work. So, they are created every time the code is ran. From my testing this does not have a noticeable effect on the speed of the model. The prediction class I also had to completely remake myself. In the end the classifier does work however the predictions are not consistent and there are a lot of false positives and false negatives, so for the final solution I will investigate creating my own model and classifier.

Prototype Conclusion

In the end this prototype was completely random with the predictions and the only positives that came out of the project were the connection to the discord API and a general understanding of the machine learning method used, the dataset and how to work with it and knowing how in the end, it will all work and look. In the end, a lot was learned, and no time was wasted.

4.2.3. Prototype Two

The 2nd prototype was created to work the same with the discord API code so that was not changed. This time I created a model myself using TensorFlow. It was an unsupervised neural network that was trained for pattern matching. The way It worked was having 3 tags, for hate speech, offensive language and neither. Each of the sentences were taken and tokenized and assigned to the classes. Code below

```
ignore_words = ["HASHTAGHERE","MENTIONHERE","URLHERE"]
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    for pattern in intent['patterns']:
        #open pattern(it is a csv file)
        with open('../' + pattern ) as csvfile:
            csvreader = csv.reader(csvfile)
            for row in csvreader:
                # tokenize each word in the sentence
                # add to our words list
                words.extend(nltk.word_tokenize(row[0]))
                # add to documents in our corpus
                documents.append((row[0], intent['tag']))
                # add to our classes list
                if intent['tag'] not in classes:
                    classes.append(intent['tag'])
# stem and lower each word and remove duplicates
words = [stemmer.stem(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))
# remove duplicates
classes = sorted(list(set(classes)))
```

Figure 18 Prototype 2 word tokenisation

Once the sentences were tokenized and assigned to the classes. we created the model. the model was a neural network with 3 layers. The code below:

```
# reset underlying graph data
tf.compat.v1.reset_default_graph()
# Build neural network
net = tflearn.input_data(shape=[None, len(train_x[0])])
net = tflearn.fully_connected(net, 8)
net = tflearn.fully_connected(net, 8)
net = tflearn.fully_connected(net, len(train_y[0]), activation='softmax')
net = tflearn.regression(net)

# Define model and setup tensorboard
model = tflearn.DNN(net, tensorboard_dir='tflearn_logs')
# Start training (apply gradient descent algorithm)
model.fit(train_x, train_y, n_epoch=7, batch_size=128, show_metric=True)
model.save('models/model.tflearn')
```

Figure 19 Prototype 2 neural network

The model was trained with a batch size of 128 and took around 10 minutes to train per epoch. with testing it, the accuracy and training loss levelled off after 7- epochs. The results were not good with an accuracy of 73% and a training loss of 0.67.

Prototype issues

With further testing the model performed poorly, with most of the predictions, prediction messages to be offensive. This is because of a few reasons.

1) The dataset was large and lots of words were shared between all 3 of the classes
2) The split of the data with 77% being offensive,16% being neither and 5% being hate speech, just by the large number of offensive data, it always thought it was offensive content.

This model could be implemented to detect offensive content alone by filtering for offensive words. However, the dataset would have to be completely changed. These unsatisfactory results made me abandon this model as well and move onto a more complex solution. Using the BERT Model.

4.2.4 Prototype Three

The 3rd Machine learning prototype built was the one used in the final solution. This was created using TensorFlow and Tensorflow_Text which facilitate the use of the BERT model. Implementing this model took a few more steps as it involved finding, downloading, and then testing the dataset on the pretrained models. This was done through the TensorFlow hub website.(24) Here we can find pretrained encoders and pre-processors to use for the model. To create the prototype, I used the bert-base-encased for the pretrained model and tokenizer and for the pre-processor and encoder used the smallest bert encoder model and its corresponding pre-processing model. Code snippets below:

```
from transformers import BertTokenizer, TFBertForSequenceClassification
from transformers import InputExample, InputFeatures

model = TFBertForSequenceClassification.from_pretrained("bert-base-uncased")
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
```

Figure 20 Prototype 3 BERT models

```
import tensorflow_hub as hub
import tensorflow_text as text
preprocessor = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3")
encoder = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4")

def get_embeddings(sentences):
  '''return BERT-like embeddings of input text
  Args:
    - sentences: list of strings
  Output:
    - BERT-like embeddings: tf.Tensor of shape=(len(sentences), 768)
  '''
  preprocessed_text = preprocessor(sentences)
  return encoder(preprocessed_text)['pooled_output']

get_embeddings([
    "Hello, what are you doing"]
)
```

Figure 21 prototype 3 word embeddings

Once BERT was setup, the rest of the model was created using a similar process as with the other 2 prototypes. However, compared to the other models, this took longer to train and took approximately 35 minutes per epoch with around 10 -15 epochs before the accuracy and loss evened out at 87% and 0.35 respectively. This was very promising.

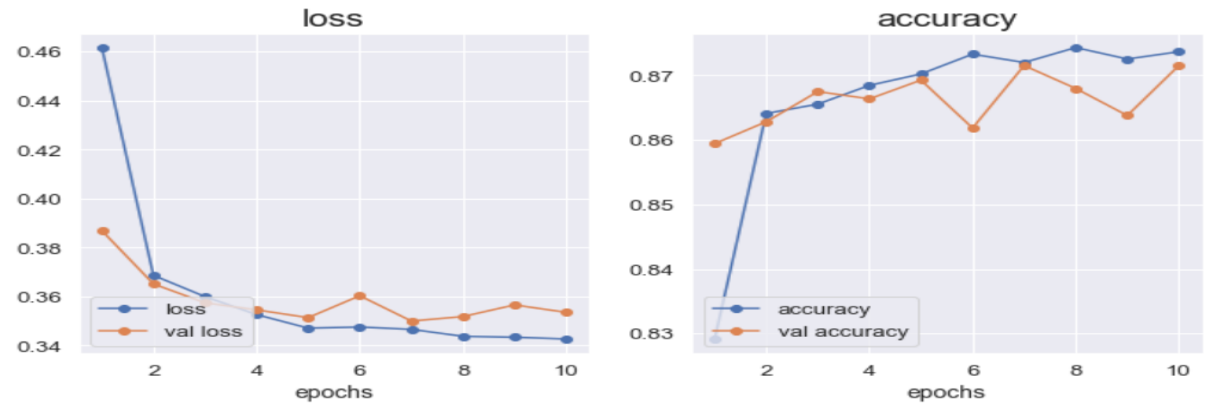


Figure 22 Prototype 3 training accuracy

Testing the model came next and putting in sentences that the other prototypes classified incorrectly, this model got them correct. Testing with various types of hate speech, offensive sentences, and just regular sentences. The model correctly classified the sentences at a satisfactory rate where it was comfortable to continue to use and work on the model to make it better.

```
27  reviews = [
        "hello world",
        "what the fuck",
        "Today is a good day"
    ]

    def predict_class(reviews):
        '''predict class of input text
        Args:
        - reviews (list of strings)
        Output:
         - class (list of int)
        '''

        print("----")
        return [np.argmax(pred) for pred in model.predict(reviews)]


    predict_class(reviews)

    ----
27  [2, 1, 2]
```

Figure 23 Prototype 3 Results

Where 0 is hate speech, 1 is offensive language and 2 is neither.

More information on the development on this model is covered in the project development chapter

5. Project Development

5.1. Introduction

This section will deal with all the different features of the project and how they were developed. The features that will be discussed in this section are:

- Discord API
- Further development and fine tuning of the model
- Development of the moderation tools
- Image to text extraction and classification
- Data collection and reporting false flags

5.2. Discord API

I started off development by creating the Discord bot. This was done by creating a developer account on the discord website.(25) Assigning the permissions for the bot. For this bot just giving it general administrator rights is sufficient. To add the bot to a server, you then go into the URL generator and give it the bot scope, administrator permissions and then click on the URL and select the server to add it to. You are then brought to the discord application and this pop up appears to add to a server

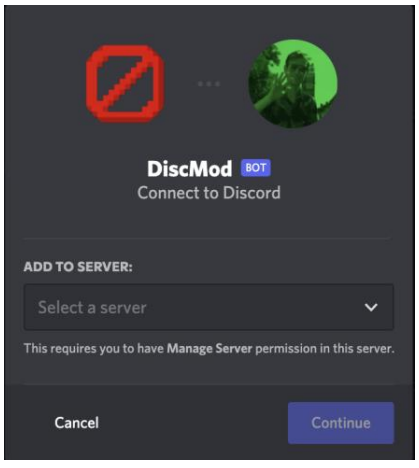


Figure 24 Add to server window

Once this completed. the work on coding the bot began. The first thing that was coded to make the bot come online. This was done using the following code:

```
import discord
from discord import opus
from discord.ext.commands import Bot
from discord.ext import commands
from dotenv import load_dotenv
load_dotenv()
TOKEN = os.getenv("DISCORD_TOKEN")

client = commands.Bot(command_prefix='~')
@client.event
async def on_ready():
    guilds = list(client.guilds)
    print(f'{client.user} is connected to the following guilds:\n')
    for guild in guilds:
        print(f'{guild.name}(id: {guild.id})')

client.run(TOKEN)
```

Figure 25 Discord client code

Each discord bot has a unique token that is used as a login key, and it is very important to keep it private as it could give other people access to your bot if it becomes available. To secure the token, the dotenv library is used and the token is put inside a file named. dotenv and the token is then passed to the code when the bot is ran. Once the code is ran, you will see the bot come online and in the console it prints out which servers it is connect to as seen below

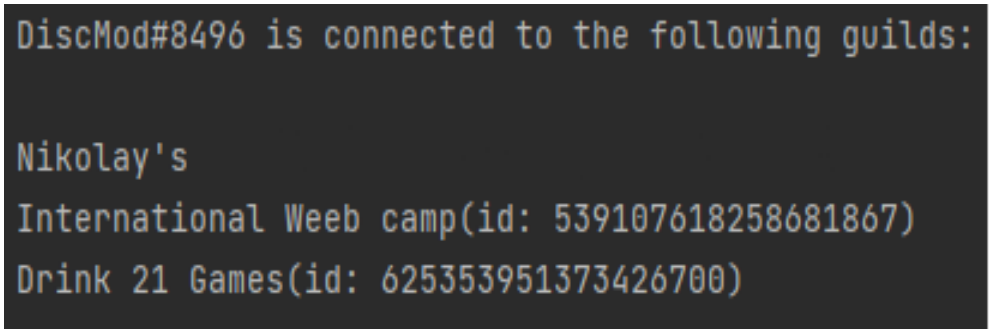


Figure 26 Screenshot showing what servers bot is connected to

With the discord bot up and running I moved onto the development of the model. Once the discord bot is in a server. Adding the bot to your own server is very simple. You can just click on the profile and click add to server. As seen below

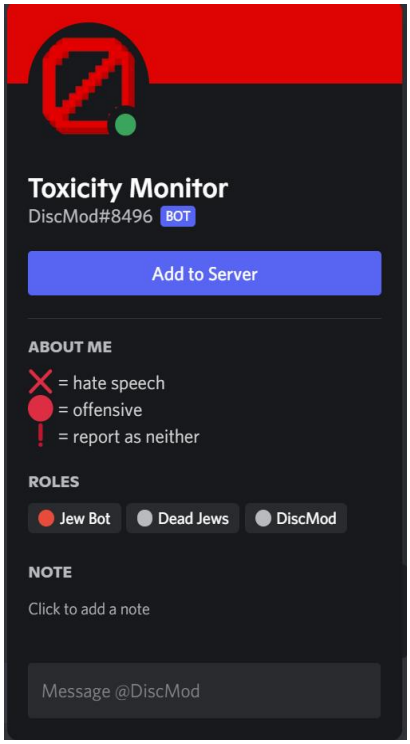


Figure 27 Bots profile

Instead of having each message passed to the bot through a command as it was in the prototype, to have the bot read all the messages itself, you create a on_message function with the event listener decorator and with the message as the parameter. Now every message will be read and can be passed onto the model. As seen below

```
@client.listen()
async def on_message(message):
    question = Classifier.process_msg(str(message.content))

    if not message.author.bot:
        response = Classifier.predict_class([question])
        if response:
            if response[0] == 0:
                await message.add_reaction(str('❌'))  # red x emoji
            elif response[0] == 1:
                await message.add_reaction(str('🟥'))  # red circle eomji
```

Figure 28 Discord prediction code

To deal with the clutter, instead of the bot replying to every message sent, it will put a reaction on the message with the ❌ signifying the message contains hate speech and the 🟥 signifying the message contains offensive content.

5.3. Model Development

With the Discord bot working and connected. I moved onto the development of the model. I took the prototype I developed and started to develop it further. It started with looking at the dataset to know what was to be expected of the model. I found that the dataset was large and was not evenly split between the 3 different classes.

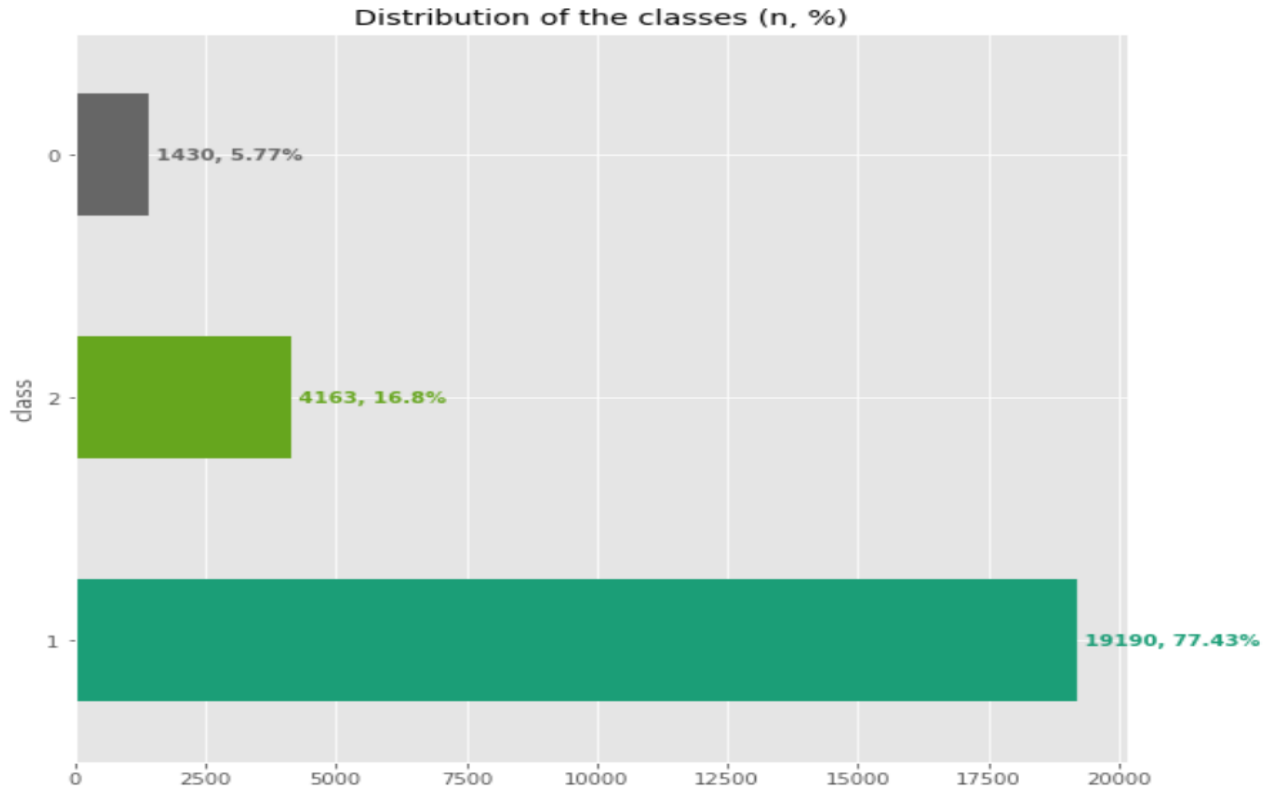


Figure 29 Distribution of data diagram

As seen in this graph. There were over 15 thousand more sentences with offensive content than the other 2 combined. Work was also done on the sentences in the dataset to make the result more accurate. Pre-processing consisted of removing words and objects in the sentence that were not needed for the model. The content that was removed was: user mentions, URLs, and extensive whitespace. This was done using the following code:

```
def process_tweets(df):
    space_pattern = '\s+'
    giant_url_regex = ('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|'
        '[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+')
    mention_regex = '@[\w\-]+'
    parsed_text = []
    for i in range(len(df)):
        text = df['tweet'][i]
        # 1) url
        text = re.sub(giant_url_regex, 'URLHERE', text)
        # 2) lots of whitespace
        text = re.sub(space_pattern, ' ', text)
        # 3) mentions
        text = re.sub(mention_regex, 'MENTIONHERE', text)
        parsed_text.append(text)
    df['tweet'] = parsed_text
    return df
```

Figure 30 Processing tweets code

Afterwards i began to understand the pretrained model and how it works and what the results would look like. I began with the embeddings and looking at the similarities between sentences as that's how the model will end up with predicting the classes. Using the following sentences

```
reviews = ["what is wrong with you",
           "i really dislike you",
           "are you ok",
           "i hate you",
           ]
```

I was able to plot the similarities between each sentence and got these results:

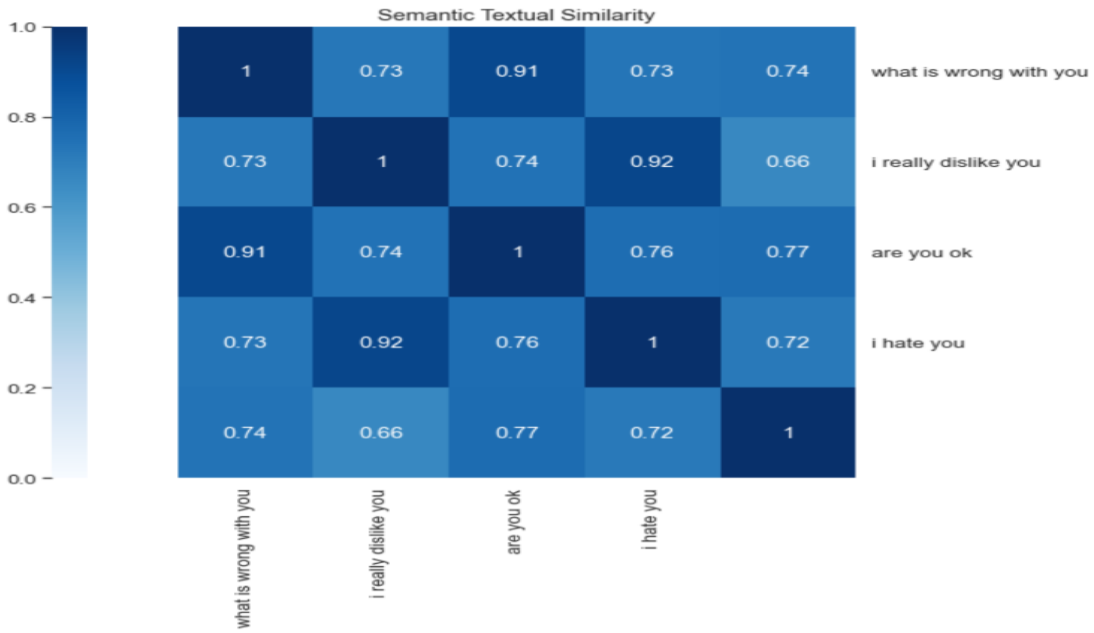


Figure 31 Semantic similarity diagram

With these results being satisfactory and the data all processed. It was time to move onto the creating and fitting the model. The model was created using the pretrained pre-processor and encoder and then fitted using the processed dataset.

Training the dataset involves some proper planning as on average it took 6-10 hours to train the model using my personal computer. With a Ryzen 5 2600 and 32gb of ram, CPU usage would be 100% and around 3-5gb of ram being used. Therefore, the computer would not be usable while training therefore the model would only be trained at night.

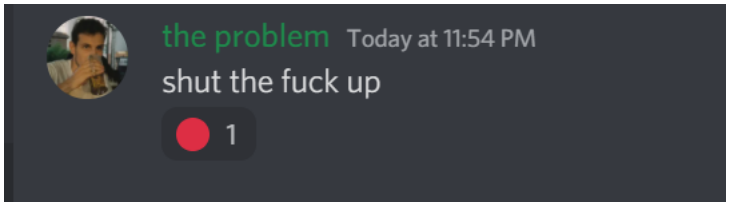
To not have to train the model every time the code is ran, the model is saved using model.save() and then loaded using keras with keras.models.load_model()

With the model fully trained and saved. It was time to connect it to the discord client. Using the model.predict() function I found that for best accuracy and to limit the number of false positives to only return the prediction if the prediction confidence is over 0.5 for hate speech and 0.80 – 0.90 for offensive language.

```
if model.predict(message)[0][0] > 0.5:
    return [np.argmax(pred) for pred in model.predict(message)]
elif model.predict(message)[0][1] > 0.85:
    return [np.argmax(pred) for pred in model.predict(message)]
```

Figure 32 prediction code

And on the discord side, if hate speech or offensive language is found within the message, the bot adds a reaction emoji to it as shown in the previous chapter. Some examples of the outputs: detecting hate speech and offensive language.




```
the prediction for "shut the fuck up" is:[5.759359e-02 9.417214e-01 6.850649e-04]
```

Figure 33 Offensive language Detection

The highest prediction was for offensive at 0.94 or 94%.

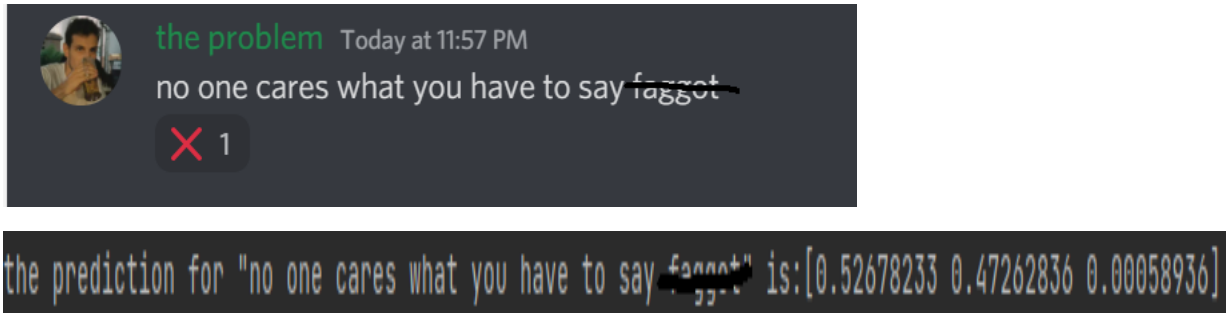


Figure 34 Hate speech detection

And the above sentence being flagged as hate speech as the model had the highest confidence on it and it was above the 0.5 threshold.

Messages that do not contain hate speech or offensive language do not have anything done to them and are left as is.

5.4. Moderation tools

The development of moderation tools was a small part compared to the other features as discord has very good moderation tools built. However, one issue with moderators is that they're human and they can't be online 24/7. Therefore, giving the users a little bit of power to deal with abusers was what I went out to achieve. Through various discussions and debates with members in my discord server. We came up with 2 solutions. Both involving allowing the members to vote. The users can vote to either mute the user for a minute or kick them from the server. The way to initiate vote kick is using the command ~votekick or ~votemute with the mention of the user.

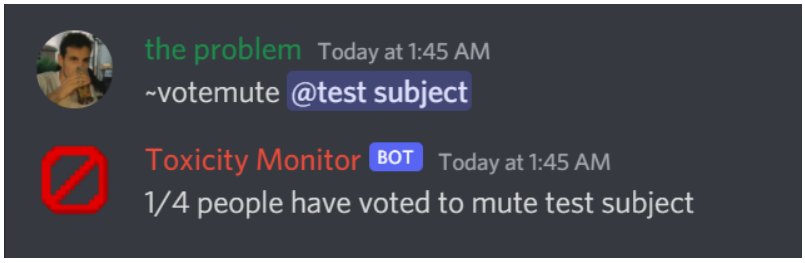


Figure 35 vote mute example

Once the command is run the bot informs that the vote was counted and waits for more votes. once the counter reaches the specified amount the user gets muted for a set amount of time. This works the exact same way for vote kick as well.

Setting up the muting command takes a bit more work as you must create a role that removes the permission for the user to send messages. This step does only take around 2 minutes to complete.

Once a user gets muted, the users text box changes from this:

To this:

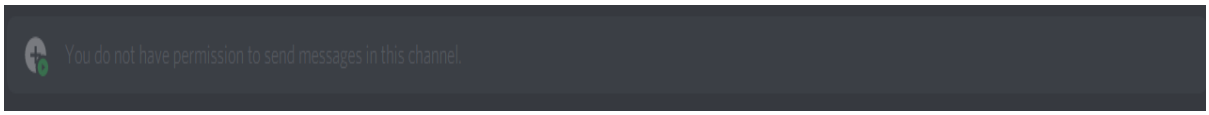


Figure 36 muted text box

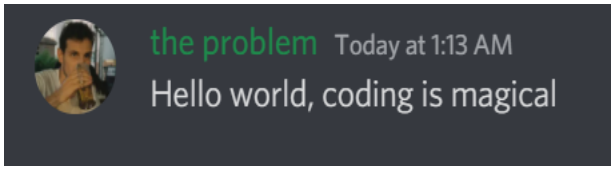
5.5. Image to text extraction and classification

Image to text extraction was implemented using OpenCV and pytesseract. OpenCV is an image processing library that allows you to edit images using code. Every time a message gets sent, the message is checked if it contains an image, if it does, the image is downloaded. The image is then processed to remove every pixel that is not white.

```
def setup_image(img):
    # convert the image to gray scale
    img = cv2.imread(img)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = gray
    # only keep the white pixels in the image
    img = cv2.threshold(img, 180, 255, cv2.THRESH_BINARY)[1]
    # save the thresh holded image
    cv2.imwrite("data/images/output.png", img)
    return img
```

Figure 37 Image setup code

This turns this screenshot of a message from discord



Into this:

Figure 38 image after being processed

And then running this image through pytesseract using this code:

```
text = pytesseract.image_to_string(img, config='--oem 3 --psm 12')
```

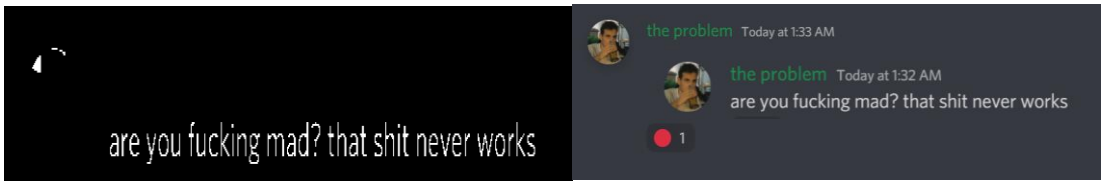
Turns the image into this output: “Hello world, coding is magical”

Which is then passed into the classifier which tests the text for hate speech and offensive content. The output of which is:

```
the prediction for "" is:[0.00440253 0.09184124 0.90375626]
the prediction for "Hello world, coding is magical " is:[0.01029026 0.03620496 0.9535048 ]
```

Figure 39 prediction on image

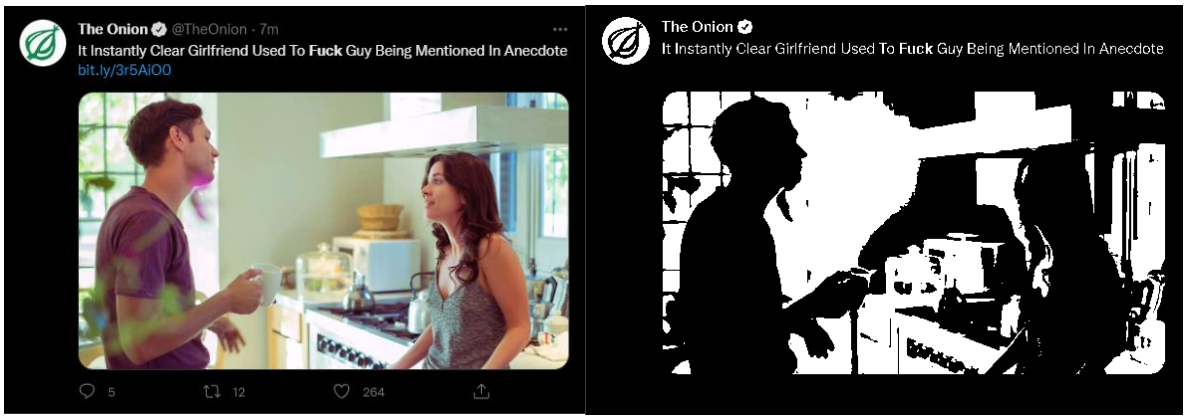
And with just sending the exact text through discord, the output is the same. Here is the result when the classifier detects offensive language (the red circle emoji signifying that the classifier detected offensive language):

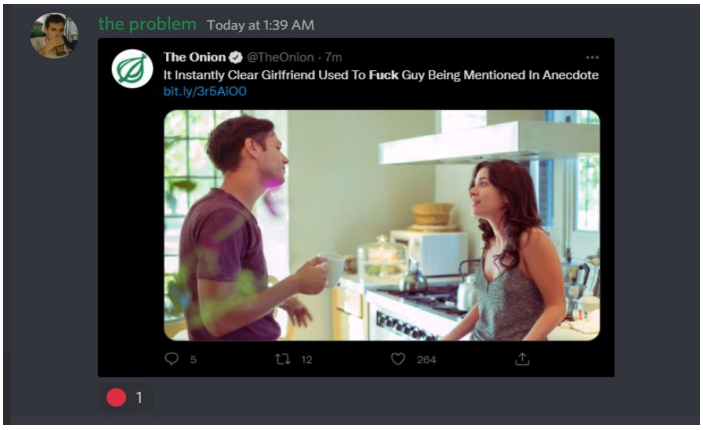



```
the prediction for "are you fucking mad? that shit never works " is:[0.0455769  0.9526624  0.00176069]
```

Figure 40 offensive language prediction on image

Furthermore, the image processing and text classification works with screenshots of tweets as shown below using a tweet from the onion:(26)






```
the prediction for "The Onion @ It Instantly Clear Girlfriend Used To Fuck Guy Being Mentioned In Anecdote @ rm ap" ai, a ' Q -f " is:[0.04043144 0.8889279  0.0706407 ]
```

Figure 41 Prediction on tweet screenshot

Prediction being offensive content at an 88% confidence

Without the pre-processing, pytesseract would pick up more of the text such as the username and time when the message was sent if it was a screenshot of a discord message. And with the tweet it would also pick up the numbers, how long ago the tweet was sent, the twitter handle and the URL.

Here is the code snippet of the code within the client code.

```
if message.attachments:
    img_data = requests.get(message.attachments[0].url).content
    with open('data/images/image_name.jpg', 'wb') as handler:
        handler.write(img_data)
    img = imageProcessing.setup_image('data\images\image_name.jpg')
    text = imageProcessing.convert_to_text(img)
    text = Classifier.process_msg(text)
    if not message.author.bot:
        response = Classifier.predict_class([text])
        if response:
            if response[0] == 0:
                await message.add_reaction(str('❌'))  # red x emoji
            elif response[0] == 1:
                await message.add_reaction(str('🔴'))  # red circle eomji
```

Figure 42 image prediction code

The performance of pytesseract in combination with the classification is very reasonable with it taking on average 1-2seconds to get the prediction.

5.6. Data collection and reporting false flags

Machine learning models are never 100% accurate and that is the same with my model. Through extensive use of the model in my discord server running and predicting hundreds of messages a day there were a small number of messages that were being false flagged as offensive when they did not contain any offensive language. To address this issue, I added a simple way users can report

messages that were incorrectly flagged. This was done by allowing the user to add an emoji reaction to a message and depending on the reaction, the bot saves the message and adds it to a collection. This is done using the on_raw_reaction_add() function in the discord API. The user has an option between 3 different emojis, with each emoji for the different classifications,

- ❗ if the bot misclassifies a sentence as hate speech or offensive and is neither
- 🔴 if the bot misclassifies a sentence as hate speech or neither and is offensive
- ❌ if the bot misclassifies a sentence as neither or offensive and contains hate speech

```
if not reaction.member.bot:
        if reaction.emoji.name == '❗':
            with open('data/Reports/neither.txt', 'r') as file:
                message = await channel.fetch_message(reaction.message_id)
                if str(message.content) not in file.read():
                    with open('data/Reports/neither.txt', 'a') as f:
                        f.write(f'{Classifier.process_msg(str(message.content))}\n')
                else:
                    print("Message already in file")
```

Figure 43 Report false flag code

There are also checks in place to make sure that duplicates are not added and that the bot's reactions are also not considered. To flag a message all that needs to be done is right click the message, click add reaction, and select the correct emoji. This data is then taken when the model gets retrained and is added to the dataset

```
dataset = datasetm.append(pd.read_csv('../data/Reports/neither.txt',
sep='\n', names=['tweet','class'],encoding='iso8859_2', index_col=False))
#where class = NaN, set it to 2 because the only classes where it would be
NaN is the recently added
dataset.loc[dataset['class'].isnull(), 'class'] = 2
dataset.reset_index(drop=True, inplace=True)
```

Figure 44 Adding false flags to training data

	tweet	class
24774	you really care bout dis bitch. my dick all in...	1.0
24775	you worried bout other bitches, you need me for?	1.0
24776	you're all ~~niggers~~	0.0
24777	you're such a ~~retard~~ i hope you get type 2 dia...	0.0
24778	you's a muthaf***in lie “MENTIONHERE: ME...	1.0
24779	you've gone and broke the wrong heart baby, an...	2.0
24780	young buck wanna eat!!.. dat ~~nigguh~~ like I ain...	1.0
24781	youu got wild bitches tellin you lies	1.0
24782	~~Ruffled | Ntac Eileen Dahlia - Beautiful col...	2.0
24783	oh nice	2.0
24784	oh ok	2.0
24785	whats that	2.0

Figure 45 Neither getting added to the data frame for training

This is done for hate speech and offensive afterwards. The original dataset is never altered. From approximately month of running and having this feature only around 50 sentences were flagged as neither and around 5 as offensive.

5.7. Multiple Language support

The last feature developed for this project was support for multiple languages. With there only being approximately 1.5 billion total English speakers.(27) Leaving over 70 percent of the population unable to take full advantage of this project. aim of this project was always to provide a system that would flag offensive text and having it limited to only 1/5th of the population did not seem enough. This implementation began by researching other datasets that would allow me to implement the multilingual support. However, I was not able to find any multilingual datasets. Doing more research into the pretrained BERT models I came across a universal sentence encoder that has support for over 100 languages. These sentence embeddings have support for test classification and semantic similarity.(28) Therefore it was perfect for this project.

Implementing this feature took very little changing of code however the big change was in training times. Changing from 40 minutes per epoch to 60-70 minutes per epoch. Once again with optimal epochs being at around 8-15. Model size also increased by approximately double from less than 1GB to 1.7GB. After adding this feature, I expanded the reach of this system from 1.5billion to over 6 billion people around the world. I understand that many of the people wont be able to use this system, however, I hope that it can be implemented in many places that can assist human moderation and allow the creation of healthier environment.

This feature was extensively tested on a discord server where the primary language was Croatian. Polish was also tested at a lesser extent. Putting the confidence threshold is also required to be but at a lower level from 85% to 75%. Here are some screenshots of the bot detecting offensive sentences written in Croatian:

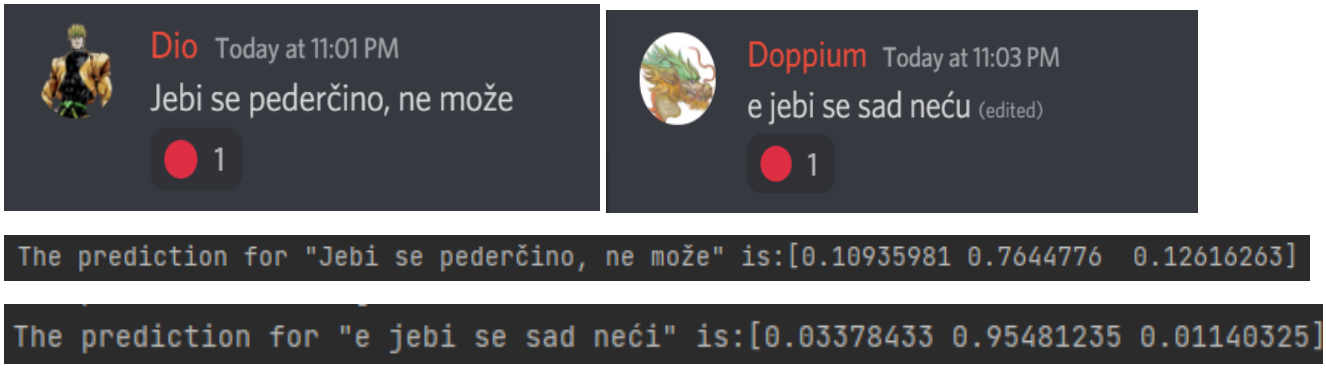


Figure 46 offensive content in Croatian

6. Testing and Evaluation

6.1. Introduction

In this section, I am going to discuss the method for testing and evaluating this project. From using multiple models to test accuracy and getting feedback from users, I will be attempting to make the model more accurate as the development continues.

6.2. Plan for Testing and Evaluation

The project will be tested all the time thought the entire development cycle. From the speed of the model training to the accuracy to make sure that it is working correctly.

To make sure that the project is never completely lost, I will be using GitHub for backup and version control. This will make sure that if there are any major issues that end up breaking the project, the changes could be rolled back. It will also prevent any issue occurring with any accidental deletion of any files.

The project will also be continuously tested and evaluated by user feedback. I plan on asking multiple large discord servers to use the bot and report with any feedback and issues that they see with the bot. Doing this will help me find issues that I alone will not be able to find. Furthermore, I will also be comparing if the solution is more effective than having human perform the exact same task.

6.2.1 User feedback

I will also be using user feedback to test the model. This will be for finetuning and making sure that the bot will not be under sensitive or over sensitive. For evaluation of the project, I will be taking a combination of user feedback, model accuracy and error scores, and finally comparing on whether the bot does a better job on detecting and dealing with messages than a human can. Both on a large scale and on a small scale.

6.3. Testing of project

Testing of the project was guided using a test plan along every process during development to make sure everything would work as expected in the end. The project was tested using both black and Gray testing depending on the user.

6.3.1 Blackbox Testing

Black Box Testing is a software testing method that involves testing the functions of software applications without looking at the written code, implementation method, or how the program works. Black Box Testing is a type of software testing that focuses on the input and output of software applications and is driven by software requirements and specifications. There are 3 types of black box testing

- Functional testing – testing different functions of the project
- Non - Functional testing – testing of performance, scalability, and usability
- Regression testing – testing after code changes to make sure new code does not affect existing code

For this project, all 3 types of testing were used due to the nature of this project. Functional testing was used mainly on the text classification as it was the main feature of the project. Making sure that the prediction was correct was extremely important and lots of time was invested into testing it. Non- functional testing was also used hand in hand with functional testing to make sure performance was always at an acceptable level and there were no delays. This was very important for the image processing as if done inefficiently, getting a prediction could take a long time. Lastly, regression testing was used every time the code was changed to make sure nothing was lost or effected negatively in the process.

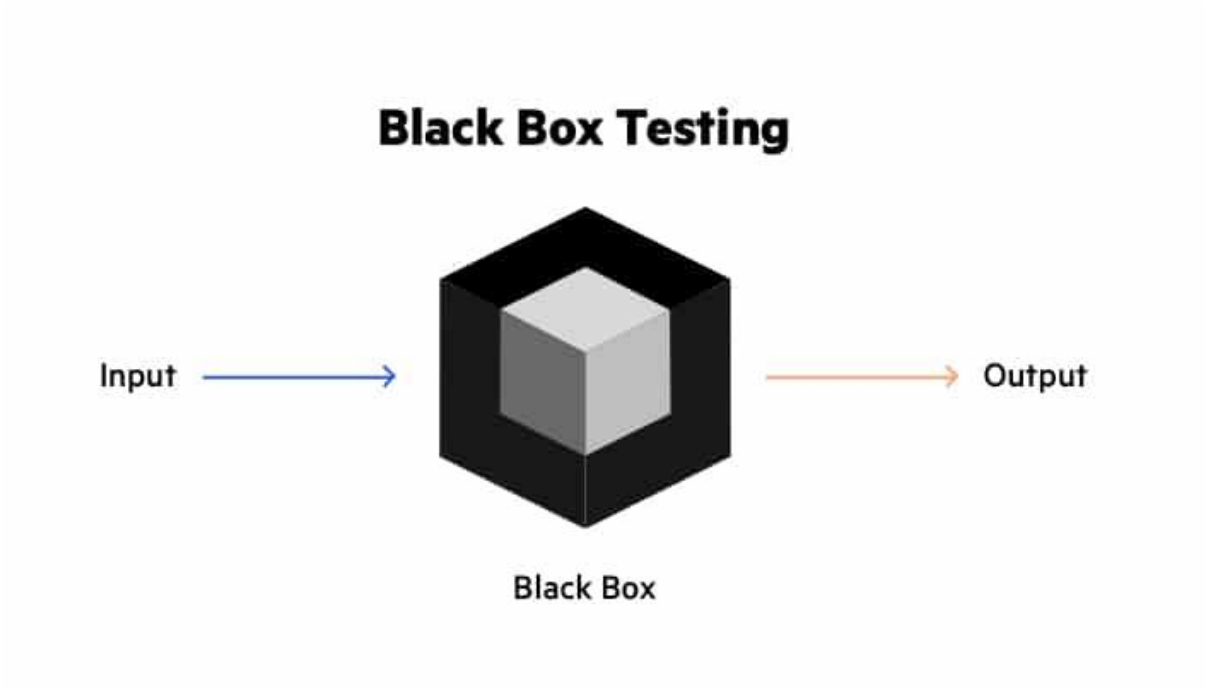


Figure 47 Black Box Testing(29)

6.3.2 Grey box Testing

Grey Box Testing is a software testing technique that involves testing a software product or application with just a limited view of the application's written code. Grey box testing is used to look for and discover flaws caused by poor code structure or application usage. Grey box testing uses a combination of both white box and black box testing. This gives it the ability to test both sides of an application; what the end user sees and what makes it all work. Gray box testing is the main method used for testing and understanding the predictions for hate speech and offensive language classification. Especially when dealing with false positives understanding where the cut off point for each class is. This is where the 85% and 50% confidence levels for offensive language and hate speech were found.

6.3.3 Test plan

A test plan was created to test the model to see if it was performing as expected

Test No.	**Test Description**	**Expected Outcome**	**Pass?**
1	Does the classifier predict hate speech	Will predict hate speech at an adequate speed and deal with it adequately. By either removing the message, warning the user, muting the user, etc	Yes
2	Does the classifier predict offensive language	Outcome will be like hate speech and punishment will be dealt depending on the severity and previous occurrences	Yes
3	Are there a low number of false positives?	User testing and feedback will give a yes	Yes
4	Are there a low number of false negatives?	User testing and feedback will give a yes	Yes
5	Test cross validation scores	They are at an adequate level	Yes
6	Test Accuracy	Make sure the model’s accuracy score is correct and not irregular	Yes

7	Test for overfitting or underfitting	The dataset is correctly fitted to the model	Yes

6.4. Model Evaluation

The model was evaluated in various ways, From user feedback to validation data from the model itself. Accuracy is a type of evaluation metric that is commonly used for classification tasks. It shows the percentage of correct predictions. It is calculated as a percentage of the total number of correct predictions to the total number of predictions generated by the model. Therefore, with an accuracy of 0.86 it means out of 100 predictions there would be 14 incorrect predictions. In machine learning there is a general consensus that great models get between 70-90% accuracy(30).

6.4.1 Overfitting and Underfitting

Validation loss (val loss) and loss are used to measure over fitting and under fitting. Overfitting is a arises when a statistical model fits perfectly to its training data. When this occurs, the algorithm is unable to execute adequately against unobserved data, therefore negating its objective. The ability to use a model on new data is ultimately what enables us to utilize machine learning algorithms to make predictions and categorize data. Overfitting occurs when loss is less than the validation loss.

Underfitting is where a data model is unable to effectively represent the connection between the input and output variables, resulting in a high error rate on both the training data and testing data. When a model is underfitted, it is unable to find the main correlation in the data, resulting in training mistakes and poor model performance. Therefore, if a model is underfitted it cannot be used for classification as it will not be able to accurately predict an output. Underfitting occurs when loss is greater than the validation loss. The closer both numbers are to each other, the better the model is.

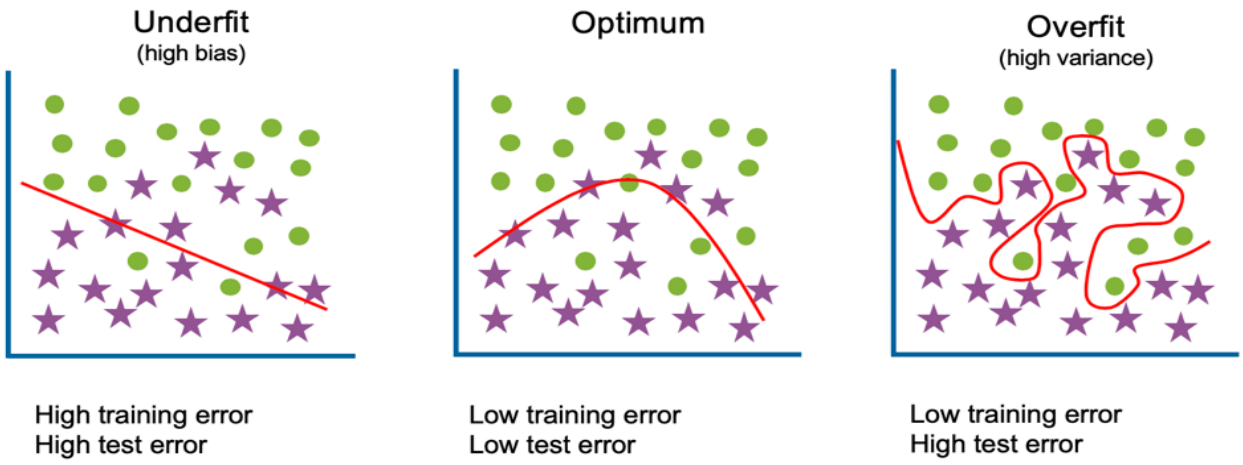


Figure 48 Overfitting vs Underfitting (31)

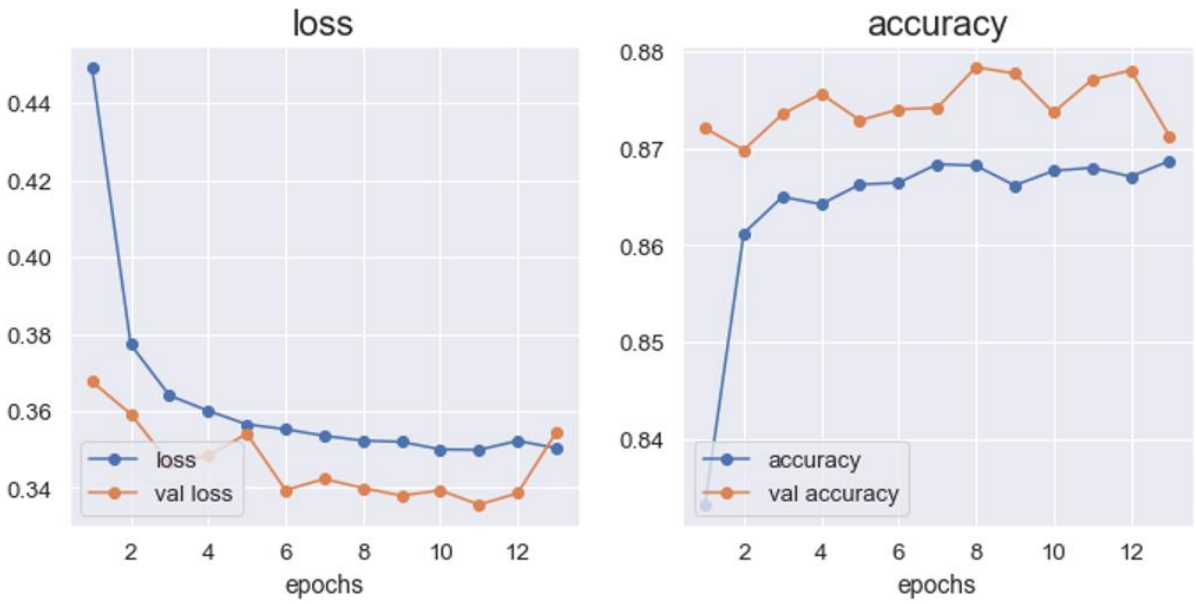


Figure 49 Results of final model

Looking at the data from the model training. Both the accuracy and losses are very good and should lead to very good results from the model. Which is achieved as the classification of hate speech and offensive language being very good.

6.5. User Feedback

User feedback is where you collect data from customers on their likes, dislikes and impressions on the product or service. User feedback can be collected in various ways that include email, phone surveys and online survey forms. For this project I created a google forms with a serious of questions that the users can answer and give their thoughts on. The users that gave feedback were all early 20s males that use discord on a daily basis and that used the bot in their servers for over a month.

The first question that was asked was their rating of the bot out of 10

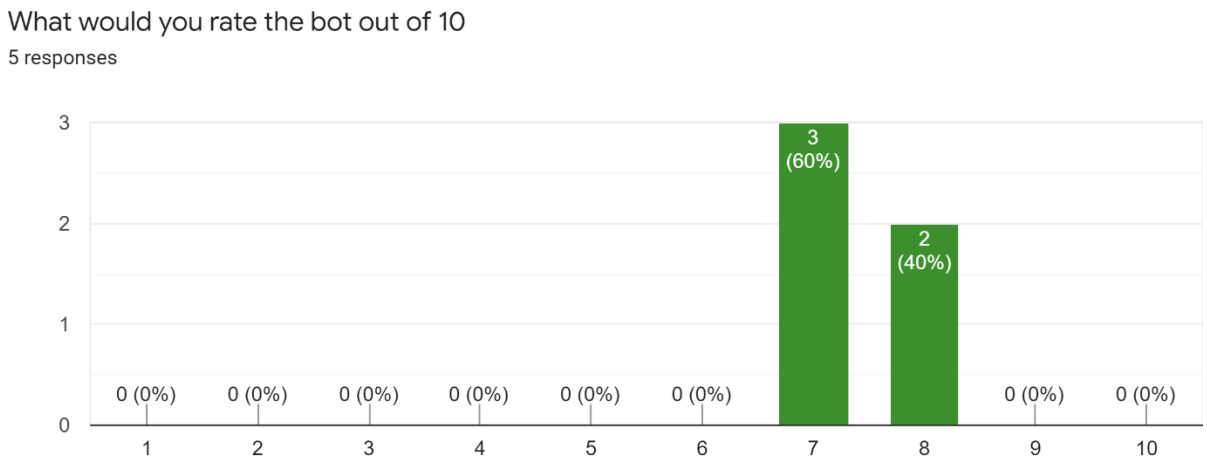


Figure 50 Rating of bot out of 10

The general consensus was that the bot in general was good and performed its job well.

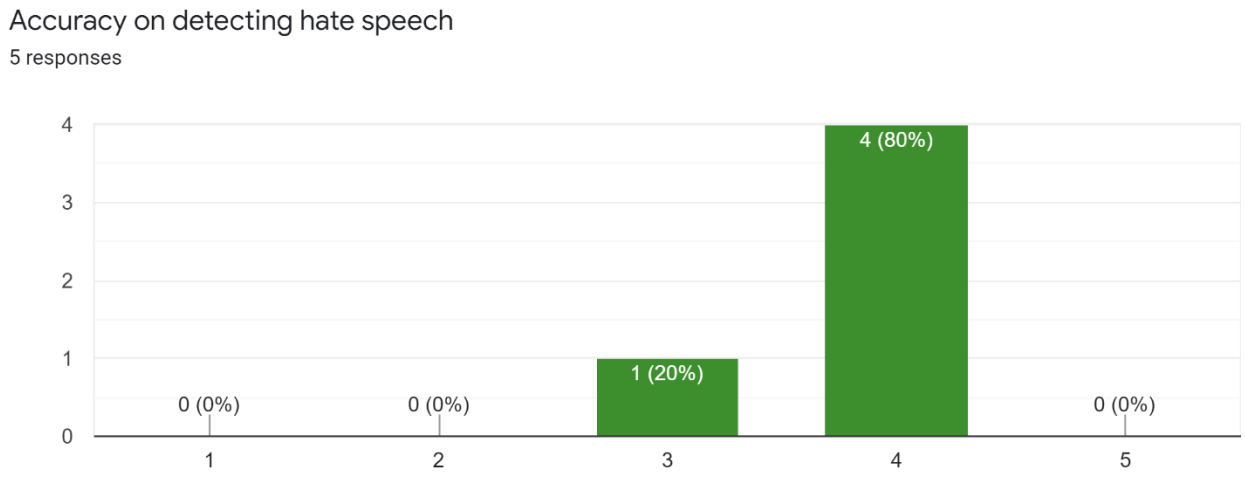


Figure 51 Feedback on hate speech detection

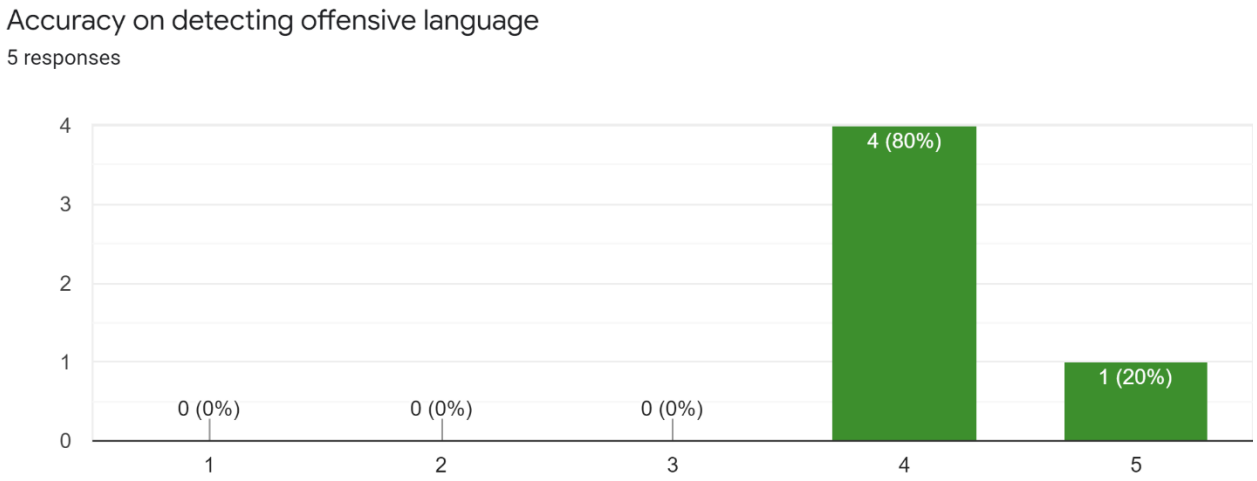


Figure 52 feedback on offensive language detection

The feedback on detecting hate speech and offensive language being also very highly rated with offensive language being more highly rated just based on the fact that it was witnessed experienced a lot more on a daily basis as the use of swear words was high in general and the detection of it was really good. Hate speech was never detected on a regular basis and was only flagged when specifically using hate speech to test the classification.

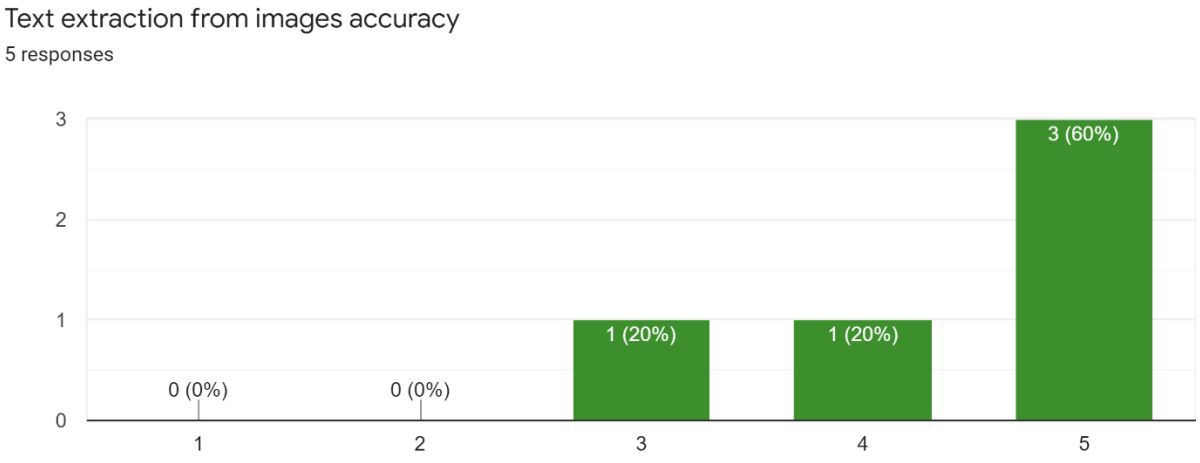


Figure 53 Feedback on text extraction

The feature which got most praise was the text extraction and then classification. With screenshots that are high-quality and have a clear distinction between the background and text, the text extraction is works really well.

The next question I asked was what they liked about the bot, The responses were

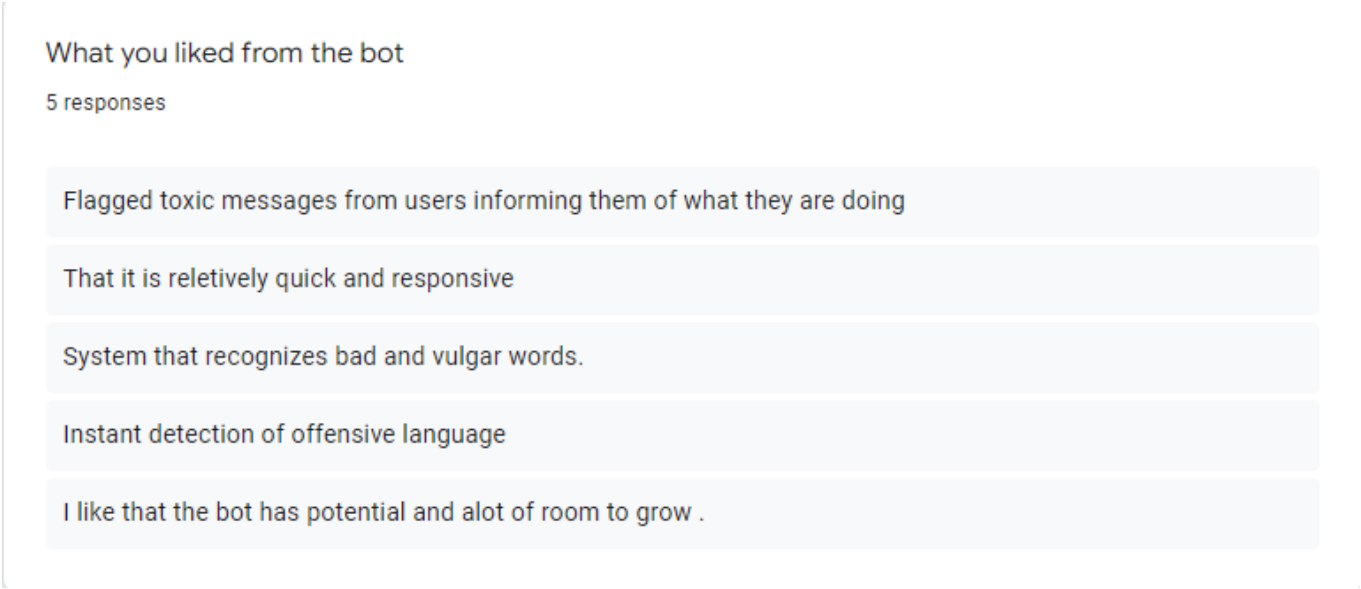


Figure 54 What users liked about the bot

The users really liked the accuracy and performance of the classification. This allows them to take action as soon as possible and deal with messages that could break any rules that they have in their discord server.

And the negatives:

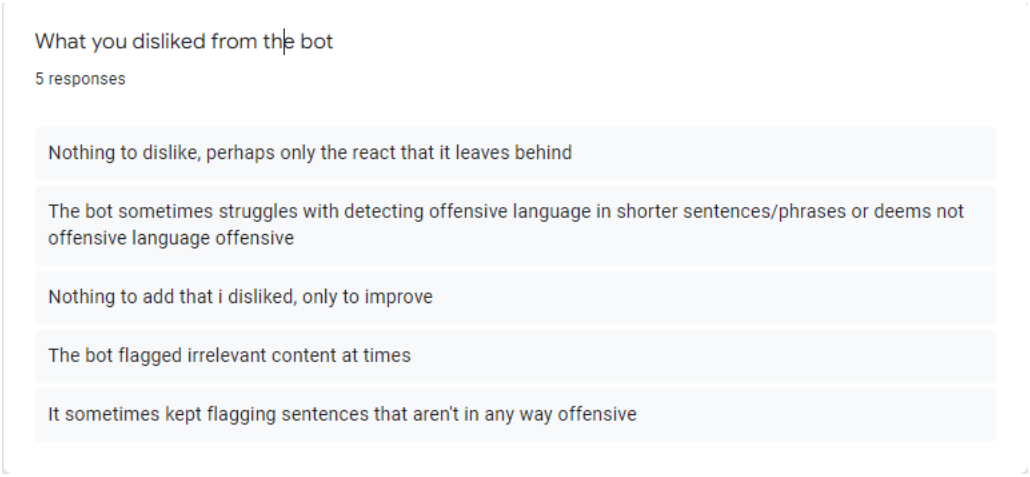


Figure 55 Feedback on dislikes

Most of the negatives came with the bot classifying sentences as offensive when they were not. From extensive testing it was discovered that this happened using extremely short sentences or when sentences included slang. A way to overcome this is to increase the amount of such sentences in the dataset. Furthermore, the reactions can also get annoying with extensive classifications which can be once again fixed with a more tailored dataset.

6.6. Conclusion

in conclusion the feedback was generally positive. The bot was able to classify hate speech and offensive language and the accuracy was very good. The bot was able to extract text from images and classify it as well. The biggest issue was the false flagging of sentences that did not contain any offensive language. This was mainly due to the use of slang and short sentences. which can be solved by flagging those sentences and then adding them to the dataset and then retrain the model.

7. Conclusion and Future Work

7.1. Introduction

In this section I am going to discuss everything I learned and achieved throughout this project. I will also discuss the issues that I encountered and the ways I solved them and any future work that that could be done to make the project better.

7.2. Issues and Risks dealt with in the project

The challenges arose during the development of the project:

- Developing the model and classifier
- Ensuring that the model is fast
- Working with the discord API and connecting all the features
- ethical and moral dilemmas in working on such a project

How the challenges were overcome:

- Completed adequate research and complete an online NLP course
- Testing between different training methods, train on different machines
- Continue working with the discord API and continue learning its features
- Read research papers and articles dealing with the issue

Risks that may have come up in the project:

- Developing a model with long training speeds
- Not completing the project in time.

How the risks were overcome:

- Train the model less frequently, save the model to a file so the model doesn't need to be retrained
- Sticked to the schedule and split the work into smaller chunks that were easier to do.

7.3. Plans and Future Work

The main part of the project that would still need working on is perfecting the model. this would be accomplished by continuing to use the bot and continue flagging the incorrectly classified sentences. Furthermore, proper development of moderation tools and dealing with users that get flagged frequently such as automatically muting them and or warning them. A big undertaking would be attempting to connect the BERT model to other instant messaging applications and see if it would work as expected.

Finishing the features that were not completed would also be important such as detecting spam, having to be able to chose how strict the detections are and lastly have every time a user gets flagged as being offensive or write hate speech be logged.

7.4. Conclusion

In conclusion the outcome of this project was very positive. The bot was able to classify hate speech and offensive language and the accuracy was very good. The bot was able to extract text from images and classify it as well. The biggest issue was the false flagging of sentences that did not contain any offensive language. This was mainly due to the use of slang and short sentences. which can be solved by flagging those sentences and then adding them to the dataset and then retraining the model. It is a very good starting point in the ever-growing field of machine learning and natural language processing. Especially when it comes to dealing with offensive and toxic content being sent in instant messaging applications.

GANTT Chart

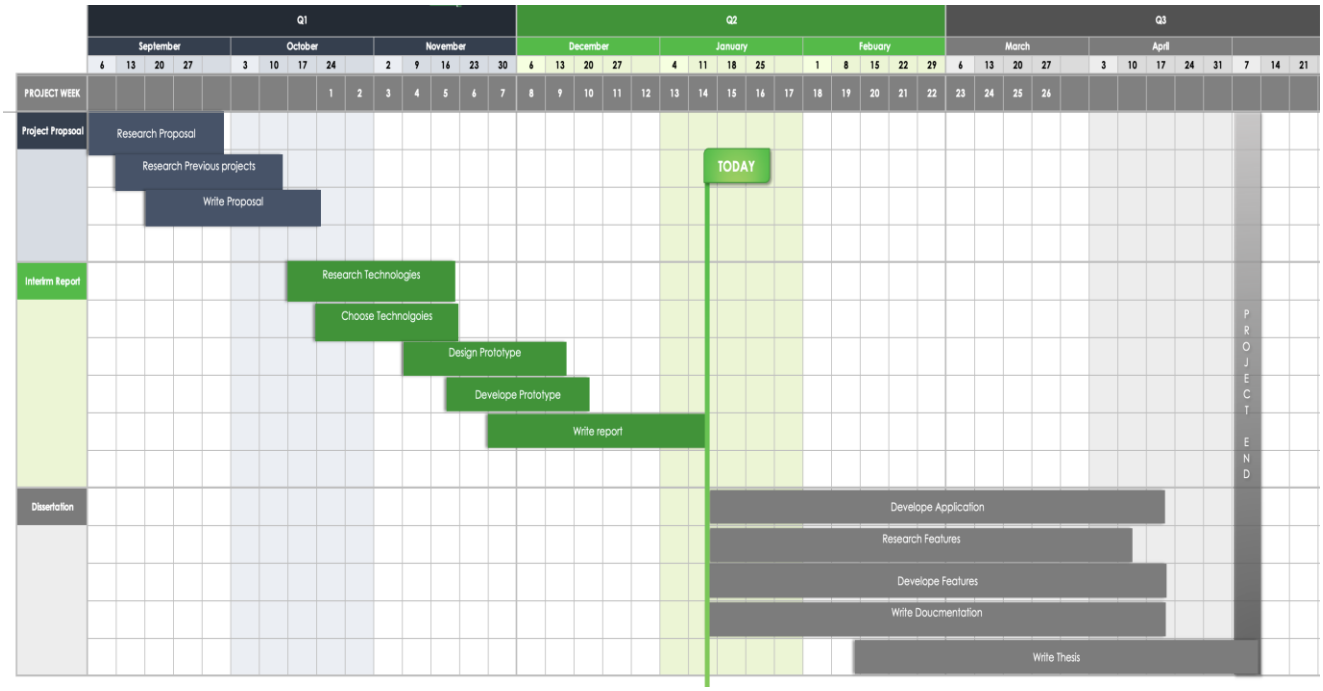


Figure 56 Gantt chart of work schedule

Bibliography

1. Müller K, Schwarz C. Fanning the Flames of Hate: Social Media and Hate Crime [Internet]. Rochester, NY: Social Science Research Network; 2020 Jun [cited 2022 Jan 2]. Report No.: ID 3082972. Available from: https://papers.ssrn.com/abstract=3082972

2. Saha K, Chandrasekharan E, De Choudhury M. Prevalence and Psychological Effects of Hateful Speech in Online College Communities. Proc ACM Web Sci Conf ACM Web Sci Conf. 2019 Jun;2019:255–64.

3. Danny. discord.py [Internet]. 2022 [cited 2022 Jan 2]. Available from: https://github.com/Rapptz/discord.py

4. Davidson T, Warmsley D, Macy M, Weber I. Automated Hate Speech Detection and the Problem of Offensive Language. In: Eleventh International AAAI Conference on Web and Social Media [Internet]. 2017 [cited 2022 Jan 2]. Available from: https://aaai.org/ocs/index.php/ICWSM/ICWSM17/paper/view/15665

5. What is FDD in Agile? [Internet]. Planview. [cited 2022 Jan 3]. Available from: https://www.planview.com/resources/articles/fdd-agile/

6. Antony. Hate speech prediction [Internet]. 2021 [cited 2022 Jan 3]. Available from: https://github.com/PyAntony/hate-speech

7. Top 5 Programming Languages and their Libraries for Machine Learning in 2020 [Internet]. GeeksforGeeks. 2020 [cited 2022 Jan 3]. Available from: https://www.geeksforgeeks.org/top-5-programming-languages-and-their-libraries-for-machine-learning-in-2020/

8. What is Supervised Learning? [Internet]. 2021 [cited 2022 Jan 14]. Available from: https://www.ibm.com/cloud/learn/supervised-learning

9. Calvo D. Supervised learning [Internet]. Diego Calvo. 2019 [cited 2022 Apr 8]. Available from: https://www.diegocalvo.es/en/supervised-learning/

10. What is Unsupervised Learning? [Internet]. 2021 [cited 2022 Jan 14]. Available from: https://www.ibm.com/cloud/learn/unsupervised-learning

11. A Beginners Guide to Unsupervised Learning | by Mathanraj Sharma | Analytics Vidhya | Medium [Internet]. [cited 2022 Apr 8]. Available from: https://medium.com/analytics-vidhya/beginners-guide-to-unsupervised-learning-76a575c4e942

12. What is Deep Learning? [Internet]. 2021 [cited 2022 Jan 14]. Available from: https://www.ibm.com/cloud/learn/deep-learning

13. What is Deep Learning? [Internet]. Databricks. [cited 2022 Apr 8]. Available from: https://databricks.com/glossary/deep-learning

14. Brownlee J. What Is Natural Language Processing? [Internet]. Machine Learning Mastery. 2017 [cited 2022 Jan 3]. Available from: https://machinelearningmastery.com/natural-language-processing/

15. What is Text Classification? [Internet]. MonkeyLearn. [cited 2022 Jan 3]. Available from: https://monkeylearn.com/what-is-text-classification/

16. Text Summarization Approaches for NLP - Practical Guide with Generative Examples [Internet]. Machine Learning Plus. 2020 [cited 2022 Jan 3]. Available from: https://www.machinelearningplus.com/nlp/text-summarization-approaches-nlp-example/

17. TensorFlow [Internet]. TensorFlow. [cited 2022 Jan 3]. Available from: https://www.tensorflow.org/

18. Team A. How-to Guide: Deploying Tesseract OCR With Python and OpenCV [Internet]. AI Oodles. 2020 [cited 2022 Apr 8]. Available from: https://artificialintelligence.oodles.io/blogs/tesseract-ocr-with-python/

19. How to fine-tune BERT on text classification task? | by Dhaval Taunk | Analytics Vidhya | Medium [Internet]. [cited 2022 Apr 8]. Available from: https://medium.com/analytics-vidhya/how-to-fine-tune-bert-on-text-classification-task-723f82786f61

20. Inappropriate or offensive content - Hwb [Internet]. [cited 2022 Jan 15]. Available from: https://hwb.gov.wales/zones/keeping-safe-online/inappropriate-or-offensive-content

21. O'Neill S. Anti-Bullying with Machine Learning Final Year Project Report. :67.

22. Top 4 software development methodologies | Synopsys [Internet]. Software Integrity Blog. 2017 [cited 2022 Jan 15]. Available from: https://www.synopsys.com/blogs/software-security/top-4-software-development-methodologies/

23. Davidson T. Automated Hate Speech Detection and the Problem of Offensive Language [Internet]. 2022 [cited 2022 Jan 15]. Available from: https://github.com/t-davidson/hate-speech-and-offensive-language/blob/507feccf0f3cf609409f765a08e0bbdac0a33d68/src/Automated%20Hate%20Speech%20Detection%20and%20the%20Problem%20of%20Offensive%20Language%20Python%203.6.ipynb

24. Home | TensorFlow Hub [Internet]. [cited 2022 Apr 5]. Available from: https://tfhub.dev/

25. Discord Developer Portal — API Docs for Bots and Developers [Internet]. Discord Developer Portal. [cited 2022 Apr 5]. Available from: https://discord.com/developers/applications

26. The Onion [@TheOnion]. It Instantly Clear Girlfriend Used To Fuck Guy Being Mentioned In Anecdote https://bit.ly/3r5AiO0 https://t.co/T9d2VqHFS3 [Internet]. Twitter. 2022 [cited 2022 Apr 6]. Available from: https://twitter.com/TheOnion/status/1511501449028640770

27. Which languages are most widely spoken? [Internet]. World Economic Forum. [cited 2022 Apr 7]. Available from: https://www.weforum.org/agenda/2015/10/which-languages-are-most-widely-spoken/

28. Yang Z, Yang Y, Cer D, Law J, Darve E. Universal Sentence Representation Learning with Conditional Masked Language Model. In: Proceedings of the 2021 Conference on Empirical

Methods in Natural Language Processing [Internet]. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics; 2021 [cited 2022 Apr 7]. p. 6216–28. Available from: https://aclanthology.org/2021.emnlp-main.502

29. What is Black Box Testing | Techniques & Examples | Imperva [Internet]. Learning Center. [cited 2022 Apr 8]. Available from: https://www.imperva.com/learn/application-security/black-box-testing/

30. How To Know if Your Machine Learning Model Has Good Performance | Obviously AI [Internet]. [cited 2022 Apr 8]. Available from: https://www.obviously.ai/post/machine-learning-model-performance

31. What is Overfitting? [Internet]. 2021 [cited 2022 Apr 8]. Available from: https://www.ibm.com/cloud/learn/overfitting