# Detecting Sentiment in Online Discourse: A Machine Learning Approach for Brand Monitoring


UNIVERSITY OF
LINCOLN

Alfie Atkinson
25715017

25715017@students.lincoln.ac.uk

School of Computer Science

College of Science

University of Lincoln

Submitted in partial fulfilment of the requirements for the
Degree of MComp Computer Science

*Supervisor* Dr. Zied Tayeb

May 2024

# Acknowledgements

Firstly, I want to thank somebody, and somebody else. Here is another thing.

# Abstract

Here is the abstract for this project report.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Project Overview

This project aims to enable users to determine the collective sentiment surrounding a specific product or brand through the use of Natural Language Processing (NLP).

Sentiment analysis, also known as opinion mining, is the process of using NLP techniques in order to determine the sentiment or emotion expressed in a passage of text. The goal of this process is to classify a text as either positive, negative, or neutral. The key steps of this process are as follows:

- Taking a text input, this could be a sentence, paragraph, or document. In the case of this project, the input will be a social media post.
- Before the text can be analysed, it must undergo preprocessing. This involves removing punctuation, stop words (words that don't carry much meaning, such as "the", "is", or "and"), converting to lowercase, and possibly tokenising (splitting the text into words or tokens).
- To analyse the text, it is inputted into a pre-trained sentiment classification model. This model will return a prediction of the sentiment within this text and assigns a classification to it based on the patterns it has learned in training.

This project will be testing and harnessing various machine learning models to analyse the sentiment of posts on social media. These posts will be scraped from the internet using various Application Programming Interfaces (APIs), and the process will be made accessible for a layman via an application with a user friendly interface.

## 1.2 Context

With the rise in use of the internet as both a place for discussion and a news source, it is becoming increasingly important for companies to monitor and understand what is being said about their brand online.

The swift expansion of the internet and social media has lead to customers abandoning company sites in favour of other networks for the purpose of sharing their opinions of a brand. This is an issue for brand monitoring, as it can be difficult to identify and understand problems that users may have with a product before it has been spread to millions of people.

In general, it now seems more effective to collect and analyse data from social media rather than investing in other means. This projects significance cannot be understated for the purpose of brand monitoring.

## 1.3 Motivations

Traditional methods of gathering public opinions are through the use of surveys and focus groups. These methods can be tedious and expensive to set up, as well as not really being as effective as may be necessary.

Surveys involve preset questions which can be interpreted differently to how the surveyor intended, it is also hard to detect emotions without the survey being in the form of an in-person interview, which is a slow process and more expensive to conduct.

Focus groups can be susceptible to 'groupthink' and dishonest responses as a by-product of the group setting. There is also the possibility of the results of the focus group being skewed towards one member who is louder than the rest, and even the focus group as a whole could skew results by not accurately representing the target audience.

On top of this, even existing sentiment analysis approaches have their limitations, many are not accessible to the average user as they have complicated or no Graphical User Interface (GUI). They also do not monitor sentiment in real-time, and do not

have a high level of user customisation. I also want to be sure to mitigate any ethical issues related to data privacy and bias by ensuring that my data is abstracted upon collection, and bias from things such as bot accounts is prevented.

There are several other issues with these approaches and they will be addressed in depth later in the report, where it will provide the advantages of using machine learning for sentiment analysis and how it tackles the issues with traditional methods.

## 1.4 Objectives

The aim of this project is to create a piece of software that is easy to set up and use, which will provide live insights into online sentiment pertaining to a specific brand. It has several distinct objectives which must be met for it to be considered a success.

Firstly, the program will implement one or more machine learning models for the use of sentiment analysis. It will allow the user to change between different models, and also provide their own training data rather than using the pre-trained models. I will aim to have at least 85% accuracy with the models, however this will be subject to the quality of the training data.

Secondly, the program will provide a way for the user to input their brand or product name. It will then use the X (formerly known as Twitter) API to collect thousands of tweets about the product, abstract them into just text, and then predict their sentiment with the trained models. I also wish to allow the user to provide their own collected text passages about their product.

Finally, an accessible GUI must be created which allows the user to easily access all the capabilities and features of the program. This GUI will give the user the ability to search their brand or product name, review the results of the search, and save the results of the search. The user will also be able to change various settings, such as the model and training data used. This aims to create a user friendly environment so that the programme is easy to use.

Achieving these objectives is critical to the success of this project, as they all contribute to the effectiveness and efficiency of the programme.

## 1.5 Report Structure

The next chapter of this report is a critical evaluation and review of academic literature that is relevant to this project. It will provide a strong foundation for my work and will establish clear links between existing research and the objectives of my project.

Following this, the third chapter of my report will present a further analysis of the specific requirements that my solution must satisfy in order to achieve the desired results. This will encompass both the functional and non-functional requirements of the solution, as well as a risk analysis and the testing/evaluation methodologies for the artefact.

Chapter 4 will outline the design and methodology implemented in this project. It will discuss the project management approaches and the software development methodologies used. It will also discuss algorithm and model choices, design considerations, analysis methods, performance evaluation strategies, and other relevant aspects of the project design and execution.

Chapter 5 will provide a clear, detailed description of every component which contributes to the software's implementation, documenting important sections of code where relevant. Here It will also show the results of any testing and evaluation results that are gathered during the development of the artefact.

Chapter 6 will then present the results of the implementation. Here I will reflect on the project and discuss how it meets the original objectives that were set out to be achieved, and the project's significance within the world of brand monitoring.

Finally, the project will be summarised within a conclusion, highlight its achievements, limitation, and implications. It will reflect on the accomplishments of this project and address any unresolved issues.

*Note: throughout this report there will be references to work from my previous project proposal (Atkinson, 2023), and my interim report (Atkinson, 2024a).*

# Chapter 2

# Literature Review

## 2.1 Introduction to Sentiment Analysis and NLP

Sentiment analysis is a key area of Natural Language Processing (NLP). It is used to extract and analyse the opinions and emotions from written passages (Dhuria, 2015; Mishra et al., 2022). This technique is useful in social media and e-commerce sites (Anny and Islam, 2022), as there is an abundance of data with over 500 million daily posts on X (formerly known as Twitter) alone (Sayce, 2022).

## 2.2 Instances of Public Sentiment Impacting Brands

Public sentiment historically has a significant impact on a brand's credibility, as evidenced by a range of studies. Berestova, D.-Y. Kim and S.-Y. Kim (2022) found that consumers prefer to engage with brands which take public stances on real world issues. Mostafa (2013) further supports this, writing about the generally positive sentiment surrounding various large brands on social media.

In 2016 Samsung released the Samsung Galaxy Note 7 Smartphone. Shortly after its release there were cases of the phone overheating and sometimes 'exploding. This caused massive backlash online, with many users of X posting negatively about the incident (Kang, Shim and J. Kim, 2019).

Lange and Sethi (2011) emphasizes the importance of sentiment analysis for managing reputational risk, especially in response to incidents like the Galaxy Note 7. Hu et al. (2017) also highlights the part social media users play in shaping a brand's perception, user loyalty, and advocacy.

These studies all show the crucial role that public sentiment plays in shaping a brand's reputation and success.

## 2.3 Nuanced Issues in Sentiment Analysis

While catastrophic events such as the Galaxy Note 7 incident may capture immediate attention, the use of sentiment analysis to identify more subtle issues has been the interest of a range of studies. Pang and Lee (2004) proposed a machine learning method which focused on the subjective parts of a given text, that may identify specific complaints or criticisms. Zhou and Ganesan (2016) highlighted the distinct properties of complaints and praises, while Nasukawa and Yi (2003) emphasized the need to identify semantic relationships between expressions and the subject matter.

However, D'souza and Sonawane (2019) gave a new perspective on sentiment analysis by discussing it's limitations and proposing a dual sentiment analysis approach. This approach would create sentiment-reversed reviews, meaning that for each training and test review, a corresponding reversed review would be generated. This technique aims to capture the dual nature of a given sentiment by considering both positive and negative perspectives. A technique such as this can be particularly useful for social media posts, as there is a strong likelihood that the posts contain both positive and negative sentiment.

## 2.4 Explosive Growth of Social Media Data

The surge in social media usage, as highlighted by Dean (2023), underscores the increasing need for efficient data analysis techniques for market research, where social media data can provide valuable insight (Camacho, Luzón and Cambria, 2021; Ausat et al., 2023).

To meet this need, Mali'c (2019) emphasizes the importance of systematic data collection, secure storage technology, and efficient querying of said data. This can be done through the parallelisation of data extraction using various web scraping tools and APIs.

## 2.5 Goals and Challenges of Market Research

Market research plays a crucial role in providing information for management decision-making, with the primary goals being to analyse market conditions, determine consumer segments, and assess sales possibilities (Cvijanovic, Mihailovic and Nikolic, 2014). However, the increasing volume and complexity of data, changes in consumer preferences, and growing competitive pressure present significant challenges with traditional techniques (Shikovets, Kvita and Bebko, 2023). Traditional data collection methods such as surveys and focus groups are also limited in their ability to accurately capture sentiment due to issues with polarity shift, data sparsity, and binary classification (Abirami and Gayathri, 2017).

To address these challenges, the industry is undergoing significant changes, including the integration of digital tools, the use of advanced technology, and the adoption of new research techniques (Barbu, 2013). Vikram and Kumar (2020) talk about how brands are increasingly relying on user feedback to improve their products and services. This feedback can take various forms, such as social media posts and product reviews (Kanev, Mladenova and Valova, 2023). Kupec et al. (2015) talks about how clients are increasingly seeking innovative approaches that provide higher value information, and Gerdes, Stringam and Brookshire (2008) highlights how valuable the integration of both qualitative and quantitative feedback is, providing a more comprehensive understanding of consumer sentiment.

Overall, brands are interested in feedback that can help them enhance product quality, service quality, and design, and that can provide insights into user experiences and perceptions efficiently and effectively.

## 2.6 Machine Learning Models

Both traditional machine learning models and deep learning models can be used in sentiment analysis. Traditional models such as logistic regression and decision trees, while robust and allowing for rapid prototyping, may not perform as well as deep

learning models when it comes to large datasets (Rajnayak, Moitra and Nahata, 2017).

In fact, Kansara and Sawant (2020) and Kamruzzaman et al. (2021) revealed the superiority of deep learning models over traditional machine learning models. Both found that deep learning models often outperformed traditional models in accuracy and performance.

To further support this, Dhola and Saradva (2021) demonstrated the effectiveness of various deep learning models such as Bidirectional Encoder Representations from Transformers (BERT) and Long Short-Term Memory (LSTM). Kamruzzaman et al. (2021) also highlighted the success of models like Single Convolutional Neural Network (CNN) + FastText and 2-L CNN + Gated Recurrent Unit (GRU) in sentiment classification specifically.

While BERT has been widely acclaimed for its performance in NLP tasks, its superiority over LSTM is not universal. Ezen-Can (2020) found that for intent classification with small datasets, LSTM outperformed BERT. However, Rangila et al. (2022) reported that BERT achieved higher accuracy and efficiency in sentiment analysis tasks. It has also been demonstrated that BERT is very effective in lexical simplification (Qiang et al., 2021), which may be useful for analysing social media posts.

Linking back to D'souza and Sonawane (2019), Yin et al. (2023) talk about their progress using BERT for Aspect-Based Sentiment Analysis (ABSA). This technique aims to predict sentiment polarity to specific aspects within the given sentence, allowing for more nuanced feedback to be detected. Yin et al. (2023) proposed a Prompt-oriented Fine-tuning Dual BERT (PFDualBERT) model which can capture complex semantic relevance and consider scarce data samples simultaneously.

## 2.7 Data Collection Through Web Scraping

Web-scraping is a data collection technique for automatically capturing data surrounding certain topics, which is used more frequently than ever in social research (Marres and Weltevrede, 2013).

Hernandez-Suarez et al. (2018) presented a methodology for collecting historical tweets by using web-scraping, and Baskaran and Ramanujam (2018) proposed a way to extract structured data records from health discussion forums using semantic analysis. On top of this, Dewi, A. Chandra et al. (2019) developed a system to scrape social media websites such as Facebook and X using their respective APIs, along with regular expressions to suppress overload information, presenting relevant information.

## 2.8 Ethical Considerations

Systems for Automatic Emotion Recognition (AER) and sentiment analysis can facilitate amazing progress (e.g., improving public health and commerce), however it can also be used maliciously (e.g., suppressing dissidents and manipulating voters) (Mohammad, 2022). Veeraselvi and Saranya (2014) and Basiri, Ghasem-Aghaee and Naghsh-Nilchi (2014) both emphasize the importance of considering context and history of user comments, as well as the potential bias in the data. Liu (2010) and C. N. Chandra, Gupta and Pahade (2015) further highlight the need for transparency and accountability when using sentiment analysis tools, particularly in the face of challenges such as manipulation, privacy concerns, and economic impact.

Krotov, Johnson and Silva (2020) provide a specific list of questions that a researcher must consider when web scraping, such as Terms of Service (ToS) compliance, data protection, and privacy policies. Fiesler, Beard and Keegan (2020) and Mancosu and Vegetti (2020) both talk about the ambiguity and inconsistency that is present in the ToS for social media platforms, which can make it difficult to be certain that rules are followed. Whereas Zia et al. (2020) focuses more on the fact that harvesting online data could infringe on a user's privacy, emphasizing the importance of anonymity when storing data.

Steinmann, Matei and Collmann (2016) propose a theoretical framework for ethical reflection in big data research, providing a privacy matrix that intersects ethical concerns with different contexts of data use. This approach offers a practical way to assess the ethical implications of data collection and use.

## 2.9 Challenges and Mitigation Strategies

There are many challenges in data analysis, and those challenges are multifaceted. It is difficult to be sure to extract only relevant, valuable information from large datasets, all while abiding by legal and ethical obligations and responsibly using the findings of data analysis (Nayak, 2002; Choenni et al., 2021).

Due to the significant amount of data that can be contained within these datasets, there are also various resource and scalability issues that may arise (Duffield and Wu, 2014). The possibility of failure in data analysis makes the importance of mitigating these challenges very apparent (Staegemann et al., 2020).

To address these problems, data sampling strategies, responsible usage of analyses, and planning for and understanding potential failures are imperative.

# Chapter 3

# Requirements Analysis

## 3.1 Target Audience Identification

The main target audience for this application consists of marketing professionals, product managers, and business owners in various industries. The main industries being the technology, retail, and hospitality sectors. However, it should be made available and accessible to other industries and audiences, including public users.

A study by MarketingWeek (2023) found that 74.6% of people working in marketing are between the ages of 26 and 45, meaning that they tend to be within the 'millennial' age group, a group which seems to be more in touch with technology and social media than older generations (Pew Research Center, 2019). However, a study by Harvard Business Review (2018) found that the top 0.1% of startup companies based on growth in the first 5 years have founders aged 45 on average, only getting older as the company progresses. This means that the program must be accessible to a broad age range, even those who may be unfamiliar with technology.

In conclusion, while there are some demographics that this application should cater towards, it should also be made with all demographics of people in mind.

## 3.2 Functional Requirements

### 3.2.1 Data Collection and Preprocessing

The program should be able to collect data from various social media platforms, such as X and Reddit. It should also be able to accept `.json` files containing reviews for the brand. The data should be effectively stripped of any personal data, and made

completely anonymous before being stored. The program should also remove noise within the text such as punctuation, stop words, and special characters. Finally, text normalisation techniques such as tokenisation, stemming, and lemmatisation should be applied to ensure consistency in the data.

The aim is to have the program remove 100% of any personal data and noise that may be present within the social media posts or reviews. It is imperative that this goal is met in order for the application to be effective ethically sound.

Both X and Reddit have well documented APIs which will be used to scrape posts relevant to the search term. There are python libraries which make it possible to access these APIs, and also libraries which allow for the reading of `.json` files.

It should be ensured that the collected data is definitely related to the specified search term in order to provide meaningful insights.

This requirement should be met within the first week of development.

### 3.2.2 Sentiment Analysis

The program will implement BERT for sentiment analysis and the proposed PFDualBERT for capturing complex semantic relevances and consider scarce data samples simultaneously. It will be able to predict whether the general sentiment around a given search term is positive, neutral, or negative; it will be able to predict sentiment polarity about specific aspects within the texts.

The predictions should have a sentiment classification accuracy greater than 85%, to ensure that it is overall robust and effective for analysing public sentiment.

State-of-the-art sentiment analysis such as LSTM, BERT and the proposed PFDualBERT will be explored.

The sentiment analysis results should provide valuable insights for brand monitoring and decision-making.

This requirement should be met within the first three weeks of development.

### 3.2.3 Real-Time Analysis

The program will be able to analyse social-media data in real-time; the user will be able to set how often they would like sentiment to be re-evaluated, and the data will be updated accordingly.

The program should be able to find all new social-media posts since the last update with no more than 1 minute delay, ensuring swift and consistent real-time results.

Data that has already undergone analysis will be saved to be recalled upon when the user wishes to view the data, this means that the program only needs to analyse new posts every interval.

The user will be provided with timely insights into the evolving sentiment surrounding their brand/product.

This requirement should be met within the first four weeks of development.

### 3.2.4 Graphical User Interface

The application will have a user-friendly interface which allows the user to interact with the system, change settings, and view various graphs containing the sentiment analysis data over chosen time periods.

A minimum usability score of 8 out of 10 should be achieved in user testing surveys conducted with targeted users.

Modern GUI design principles will be implemented, alongside iterative usability testing with target users.

Ensuring that the user interface effectively facilitates any user interactions is imperative for it to be a desired tool. Data being represented correctly within an interface is necessary for the results to be useful to the end-user.

This requirement should be met within the first eight weeks of development

## 3.3 Non-Functional Requirements

### 3.3.1 Performance

System performance will be optimised to handle a minimum of 10,000 social media posts per hour, ensuring that a large amount of data can be captured.

It should be able to achieve an average response time of less than 360 milliseconds for data retrieval and sentiment analysis processing.

Algorithms will be optimised to take as little time as possible while still performing tasks correctly.

This assures that the program can meet the demands of processing large volumes of social media data in real-time.

This requirement should be met within the first four weeks of development.

### 3.3.2 Usability

User testing sessions will be conducted with target users in order to evaluate the usability of the system interface.

A minimum usability score of 8 out of 10 should be achieved in user testing surveys conducted with targeted users.

Feedback will be gathered from users through usability tests and this feedback will be iterated on until the desired score is achieved.

This will ensure that the application GUI is intuitive and easy to navigate for users of varying levels of technical experience.

This requirement should be met within the first ten weeks of development.

### 3.3.3 Security

A secure login mechanism will be implemented to control access to the system and protect stored data. All saved data will be encrypted.

Encryption methods such as Advanced Encryption Standard (AES) encryption will be employed within the program along with a secure login system that makes data inaccessible without the correct password.

The program will use cryptography for AES encryption to implement secure login and data encryption features.

By making sure the program and data are secure, users can be sure that data is kept confidential and secure.

This requirement should be met within the first six weeks of development.

### 3.3.4 Reliability

The application will function reliably without crashes or unexpected errors, providing consistent performance under all usage scenarios.

Error handling and logging mechanisms will be used to track and address any issues that may be raised during the application's execution. The performance and uptime of the application will be monitored to achieve at least a 99.5% reliability rate.

Python's built-in exception handling will be used, and logging modules will capture errors. Testing frameworks such as pytest will be used to validate the application's reliability.

User trust and satisfaction will be maintained by ensuring the program operates reliably, minimising downtime and preventing data loss or corruption.

This requirement should be met within the first eight weeks of development.

### 3.3.5 Accuracy

The program should accurately categorise sentiment in text data, providing the user with reliable sentiment scores and classification.

Data validation checks will be used to ensure the accuracy of the sentiment analysis is at least 85%.

Well-established sentiment analysis libraries will be used, such as BERT. Unit tests and validation checks will be used to ensure that the desired accuracy is achieved.

This will allow users to trust the program and its analysis of social media data surrounding the search term.

This requirement should be met within the first four weeks of development.

## 3.4 Risk Analysis

### 3.4.1 Data Security

There is risk of data leaks or unauthorised access to sensitive information. This can be overcome by implementing robust encryption techniques such as AES encryption and secure storage to protect data. All data will be encrypted before ever being stored, and it will require a password to decrypt for viewing.

### 3.4.2 Performance Issues

It is possible that the application may crash or have slow response times during sentiment analysis. To overcome this, code should be optimised, error handling should be implemented, and performance testing should be conducted to ensure that the application operates efficiently under all conditions.

### 3.4.3 Algorithm Accuracy

There is the possibility that the sentiment analysis results may lead to incorrect insights by not being accurate enough. This is easily mitigated by validating the sentiment analysis algorithms using a diverse dataset, and refining algorithms based on these test results until an accuracy above 85% is achieved.

### 3.4.4 Poor User Experience

The GUI could have poor usability or be seen as unintuitive, which would affect user adoption. To mitigate against this, user testing will be conducted to gather feedback on the GUI design. This feedback will then be iteratively acted upon until user satisfaction is high enough.

### 3.4.5 Terms of Service Changes

Social media platforms frequently update their ToS, which could impact the programs ability to effectively scrape for social media posts or access the API. This

can be countered by regularly monitoring the chosen social media's ToS and by implementing flexibility in the program's design to be able to adapt to API changes.

### 3.4.6 Ethical Regulation Changes

There is always the possibility that changes with ethical regulations may impact the ways sentiment analysis can be used. In order to prevent this being a problem, it is important to stay informed about relevant regulations and ethical guidelines. It is also important to be sure to reflect on the ethics of the project at all stages.

## 3.5 Requirements Validation

Ensuring the robustness and reliability of the tool necessitates rigorous validation of both functional and non-functional requirements. To validate the functional requirements, the application will undergo a series of unit tests, integration tests, and system tests to be sure that each feature operates as intended and meets the specified criteria. For example, the login system will be tested to verify its functionality in securely authenticating users, while data encryption will undergo tests to confirm the safeguarding of sensitive information.

Non-functional requirements will be validated through comprehensive testing and performance monitoring. Security measures, including data encryption, will be rigorously evaluated to ensure compliance with industry standards and best practices. Moreover, the reliability and accuracy of the sentiment analysis algorithms will be assessed by comparing the tool's predicitions against manually annotated data sets to measure accuracy and consistency.

On top of this, User Acceptance Testing (UAT) will be conducted to validate that the application meets the expectations and needs of the target audience. This involves soliciting feedback from intended users of the program to gauge the tool's usability, effectiveness, and overall satisfaction.

# Chapter 4

# Design & Methodology

This section aims to outline the basic design and architecture of the artefact.

## 4.1 System Architecture

The sentiment analysis tool will be built as a desktop application developed in Python. The system architecture consists of three main elements: the GUI, the chosen machine learning model for sentiment analysis, and the integration with social media APIs for data retrieval. This section will go into more detail on the specific design and architecture of the application.

### 4.1.1 Architectural Style

The system uses a layered architecture, dividing the application into distinct layers containing components based on their function and responsibility. This style promotes modularity, scalability, and maintainability by separating concerns and enforcing clear boundaries between different parts of the system. The layered architecture also allows for easy integration of new features int he future.

### 4.1.2 Layers and Components

The system comprises of five main layers: presentation layer, business logic layer, services layer, persistence layer, and database layer.

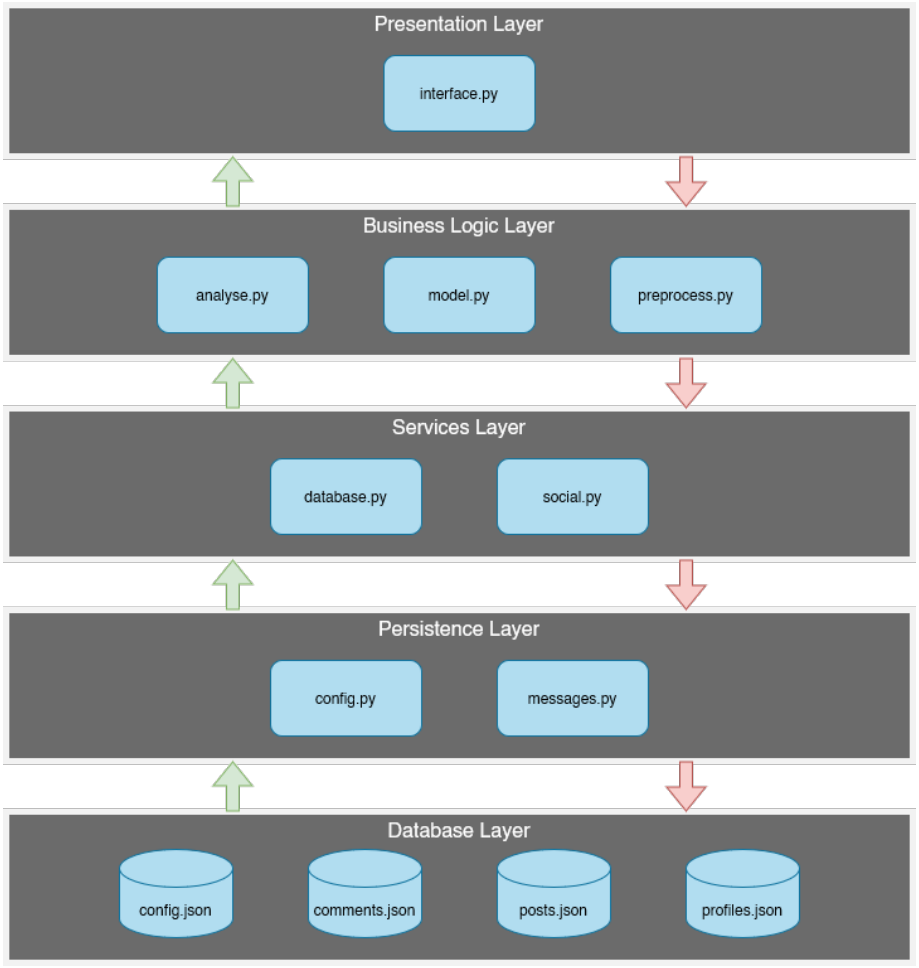


Figure 4.1: Proposed system layer architecture.

### Presentation Layer

The presentation layer is responsible for presenting the processed and analysed data to the user. In `interface.py` there will be various classes for building and handling the windows and their layouts; there will also be classes for handling functions on other threads in order to preserve the functionality of the GUI while long running operations are in progress.

### Business Logic Layer

The business logic layer is responsible for interpreting the data provided by the layers below. `analyse.py` will handle the analysis of data, for example the creation of a line chart which plots the sentiment over a given time frame. `model.py` will contain a class for the machine learning model which will handle training of the model as well

as making predictions using the model. Finally, `preprocess.py` will be responsible for processing and cleaning text before training/making predictions.

#### Services Layer

The services layer is responsible for interacting with the social media APIs and the data sources. In this layer there is `database.py`, which collects and manipulates post data from the APIs and any stored, previously collected data. Meanwhile, `social.py` will contain classes for the social media APIs, with functionality for scraping posts with given search terms.

#### Persistence Layer

The persistence layer is responsible for the persistence of configuration, settings, and logging errors and messages. It will consist of two modules: `config.py` and `messages.py`, which will have simple functions intended for retrieving data from the database layer and sending messages to the console respectively.

#### Database Layer

The database layer is responsible for storing persistent data used by the application. `config.json` will store all of the configuration settings of the app, while `profiles.json` will store the specific configurations for each profile in terms of searching social media. Lastly, `posts.json` and `comments.json` will store collected posts and comments respectively, with all the necessary information for analysis such as the body text, the amount of favourites, the predicted sentiment, and specific to comments, the parent post (to allow for contextual analysis).

### 4.1.3 Key Components

The key components and their important classes are as follows:

- `interface.py`: this module will handle everything to do with the GUI.
  - `MainWindow()`: this class will be responsible for building the main window and handling any interactions the user may have with it.

  - `SubWindow()`: there will be various classes for handling sub-windows such as profile editor and settings.
  - `Worker()`: this will be a class for handling multithreaded long-running tasks to keep the GUI responsive in the meantime.
- `model.py`: the module for handling everything to do with the machine learning models.
  - `BERTModel()`: this class will handle the training and predicting for a BERT model.
  - `LSTMModel()`: the class for handling the training and predicting for a LSTM model
  - `PFDualBERTModel()`: this class will handle training and predicting for a PFDualBERT model.
- `database.py`: this module will handle everything to do with the collection of data from the databases, as well as the merging of this data with new social media posts.
  - `PostCollector()`: a class to collect and merge data from social media with existing stored data.
- `social.py`: the module responsible for scraping social media's posts with given search parameters.
  - `XScraper()`: this class will interface with the X API to collect posts for a given search term.
  - `RedditScraper()`: this class will interface with the Reddit API to collect posts from given subreddits with given search parameters.

## 4.2 Data Model

As the artefact will involve the storage and retrieval of data, it is appropriate to model the relationships between different entities within the data. The data model consists of three entities: profile, post, and comment; each profile will have a number of posts collected for its specified subreddits, and each post will have a number of

comments collected with it if this setting is enabled. Both the posts and comments databases will be in a directory labeled with the profile ID.

For this project, it was decided that json format would be acceptable for storing data, this is due to the portable, efficient, and scalable nature of json, along with the lack of a need for anything with more complex functionality. The json data will be read and temporarily stored in a `pandas.DataFrame` structure while being used by the program.

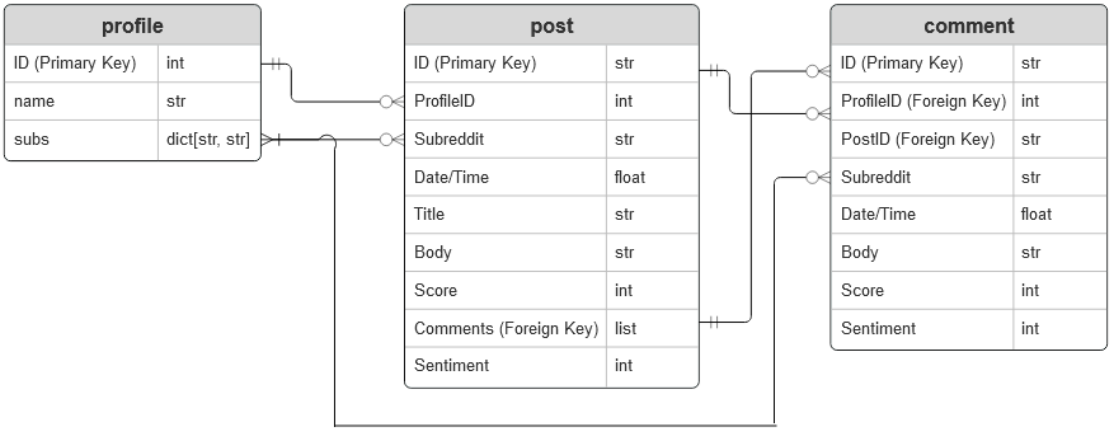


Figure 4.2: Relational diagram showing the database structure.

## 4.3 Graphical User Interface

The goal of this interface is to be easily accessible, clearly show the analysis of data, and look visually professional. The interface will consist of three main windows: the main window, where majority of the functionality happens and all the data is displayed, the profile editor window, where the user can add and edit brand profiles, and finally the settings window, where the user can change settings that affect the system functionality and visuals.

The main window will consist of three main areas: the main toolbar at the top of the window containing various buttons, the time-frame selector along with text describing the profile sentiment, and an analysis panel with tabs to visualise the data in various ways.

The profile editor window must allow the user to easily create and edit profiles, to

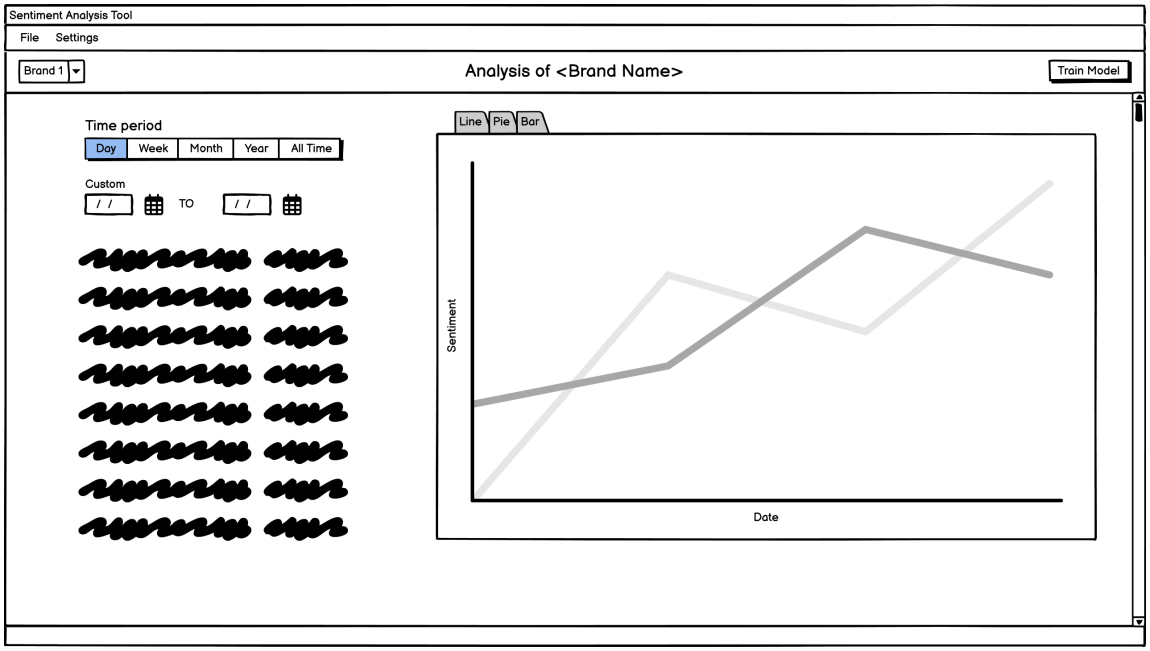


Figure 4.3: Wire-frame of the proposed main GUI.

do this it will have a text bar for choosing the profile name, along with two buttons "enter" and "delete", which will either save or delete the profile respectively. It must also allow them to add subreddits with search terms to this profile, to do this there will be two text boxes, one for the subreddit name, and one for the terms to search for in that subreddit. When a subreddit is added to the profile, it will appear in a row below these text boxes, here the user can click the "remove" button to remove a subreddit and its search terms from the profile.

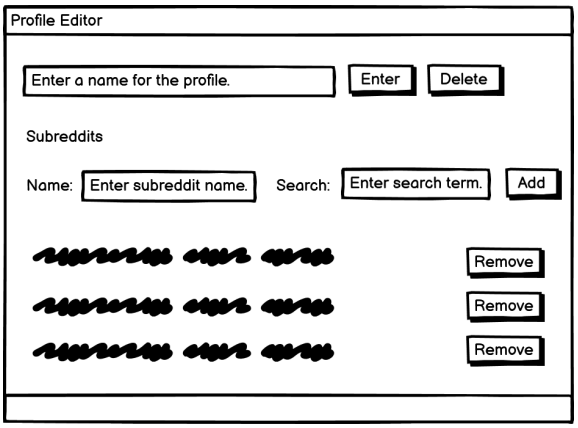


Figure 4.4: Wire-frame of the proposed profile editor GUI.

Finally, the settings window should be easily navigable and contain various settings for functionality and visuals of the application. The different types of settings

will be separated into tabs, and each setting will have its own row in the tab it belongs to.

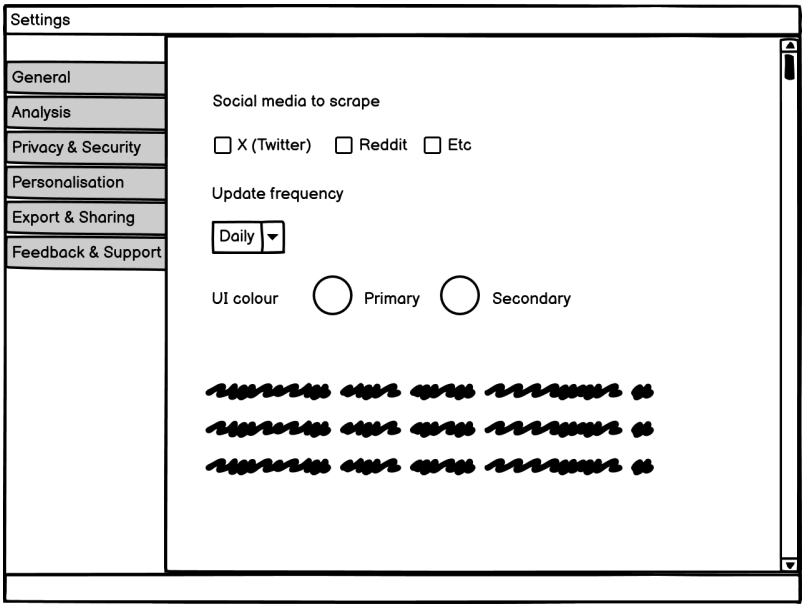


Figure 4.5: Wire-frame of the proposed settings GUI.

## 4.4 Model Selection

When selecting the model to use for this project, it was important to consider model performance, versatility, and the ability to handle complex linguistic patterns. While alternative models such as LSTM, Single CNN with FastText, and 2-L CNN with GRU have shown to be effective with NLP tasks, BERT was chosen as the preferred choice for several reasons.

### 4.4.1 Contextual Understanding

BERT's bidirectional architecture allows it to capture context information from preceding and succeeding words within the text, this addresses the complexities of language and sentiment expression highlighted by Nasukawa and Yi (2003) and D'souza and Sonawane (2019). Unlike LSTM, which processes text sequentially, BERT's bidirectional, contextual approach allows it to capture nuances and ambiguities in sentiment, making it perfect for the long-form posts prevalent in this task.

### 4.4.2 Pre-trained Language Representation

BERT is able to generalise well across diverse sentiment analysis tasks due to its pre-trained representation offering rich understanding of language semantics (Dhola and Saradva, 2021; Rangila et al., 2022). It's pre-trained knowledge reduces the need for extensive fine-tuning, facilitating transfer learning and aligning with Mali'c (2019)'s recommendation of systematic data collection and storage.

### 4.4.3 Transformers Architecture

The transformer architecture allows for parallel processing and hence efficient training on large-scale datasets (Kansara and Sawant, 2020). BERT's architecture and self-attention mechanism lets BERT capture long-range dependencies in long-form text, which overcomes issues with data sparsity and polarity shift often seen with traditional models (Abirami and Gayathri, 2017; Kamruzzaman et al., 2021).

### 4.4.4 Transfer Learning

Due to its transfer learning capabilities, BERT is adaptable to diverse sentiment tasks with minimal labelled data, which aligns with the need in market research for innovative approaches that provide high value information (Vikram and Kumar, 2020; Gerdes, Stringam and Brookshire, 2008). By fine-tuning BERT on task-specific datasets, its pre-trained knowledge can be harnessed to achieve high performance in sentiment analysis.

### 4.4.5 Polysemy and Ambiguity Handling

BERT's bidirectional context modeling allows it to disambiguate polysemous words and resolve ambiguities based on surrounding context (Zhou and Ganesan, 2016). This is particularly useful for analysing social media posts, where sentiment can vary based on the broader discourse and user engagement such as favourites and comments (Mostafa, 2013; Lange and Sethi, 2011).

## 4.5 Development Methodology

For the development of this tool, an Agile-inspired approach will be used and adapted to a single-person development team. This will allow for iterative development, allowing for flexibility and adaptability throughout the process. The development process will be divided into five key phases which each focus on a specific aspect of the application:

- **Sentiment Analysis Model**: This phase will encompass the training, fine-tuning, and prediction functionality of the chosen models.
- **Social Media Data Collection**: This phase will focus on collecting data from social media platforms for analysis. All interactions with social media APIs will be implemented in this phase.
- **Database Management**: With the data collection implemented, the focus will shift to setting up and managing the storage of collected and analysed data, and implementing secure encryption techniques.
- **Data Visualisation**: Once the analysed data is stored, the next phase will involve processing the data and visualising it in various charts, such as a line graph showing sentiment over time.
- **Graphical User Interface**: The final phase of development will revolve around designing and implementing a GUI for the application. The GUI will be built to be intuitive and linked to the previously created functionality.

Through each stage of development, an iterative approach will be taken. The work will be broken down into smaller, more manageable tasks, such as the creation of classes and implementation of specific functions, these tasks will be prioritised based on importance and dependencies, and completed in short iterations or 'sprints'. To facilitate this, the code will be split into many modules and objects, each with distinct functions to allow for code reusability.

Continuous Integration (CI) practices will be implemented, after each iteration the code will be tested through unit tests and integration tests to validate the interactions between different modules and functions. The codebase will then be pushed to a GitHub repository to allow for version control.

The code will be well documented through comments and type-hinting, ensuring clarity and consistency in the codebase, and allowing knowledge of the application's inner workings to be accessible should any other people add to the development in the future.

This Agile-inspired approach will facilitate flexibility and adaptability, allowing effective response to requirement and priority changes.

## 4.6 Testing Strategy

In order to ensure accuracy, reliability, and usability, the sentiment analysis tool must implement effective testing strategies. The primary objectives of these strategies include verifying the precision of sentiment analysis results, validating the functionality of the application components, and assessing the overall user experience provided by the GUI. To achieve this, various types of testing will be used:

- **Unit Testing**: the process of asserting the functionality of small, individual units or components of the application in isolation. It ensures that each component works as expected and verifies the functional integrity of the sentiment analysis tool.
- **Integration Testing**: focuses on validating interactions between different modules and components within the application. It tests the communication among all parts of the application, including data collection, database management, sentiment analysis algorithms, and the GUI functionality.
- **System Testing**: an evaluation of the sentiment analysis tool as a whole, assessing its compliance with the requirements and user expectations. This tests the end-to-end functionality of the application, encompassing all of its components and functions.
- **User Acceptance Testing**: the process of end-users assessing the usability, functionality, and overall user experience of the sentiment analysis tool. This makes sure that the application meets the needs and expectations of its intended users.

# Chapter 5

# Implementation

## 5.1 Sentiment Analysis Model

This section outlines the training, testing, and predicting of the BERT model, as well as the dataset chosen for fine-tuning and how it was pre-processed. Most of the functionality shown in this section is contained within the `BertModel` class under `toolkit/analysis/model.py`:

```
class BertModel(object):
    """
    Class for sentiment analysis using BERT model.
    Attributes:
        tokeniser: BERT tokenizer for tokenizing input text.
        model: BERT model for sentiment classification.
    """
    def __init__(self) -> None:
        self.tokeniser, self.model = None, None
        self.load_model(toolkit.get_dir() + '/models/')
```

### 5.1.1 Training

This section will outline the steps taken to train and fine-tune the pre-trained BERT model. The majority of the functionality in this section takes place in the `train()` and `cross_validate()` methods.

### Examining the Dataset

Before the model can be trained, an appropriate dataset for fine-tuning must be found. As this artefact focuses on social media sentiment analysis, a dataset that has pre-labelled social media posts based on sentiment is needed. For this use-case, the Sentiment140 dataset KazAnova (2017) was chosen as a widely used and praised dataset for fine-tuning BERT for social media sentiment analysis.

When calling `describe()` on the dataset loaded as a Pandas `DataFrame`, the results are as shown in the table below. This shows that the dataset labels negative sentiment as '0', and positive sentiment as '4', and also appears to have an equal number of negative and positive entries. This quality will be useful as it prevents issues like model generalisation, class imbalance, and overfitting, while allowing for enhanced performance and better feature representation learning.

Table 5.1: Dataset description.

| | Count | Mean | Std | Min | 25% | 50% | 75% | Max |
|---|---|---|---|---|---|---|---|---|
| Target | 1.60e6 | 2.00 | 2.00 | 0.00 | 0.00 | 2.00 | 4.00 | 4.00 |

We can confirm this by plotting the dataset on a chart using `pyplot`:

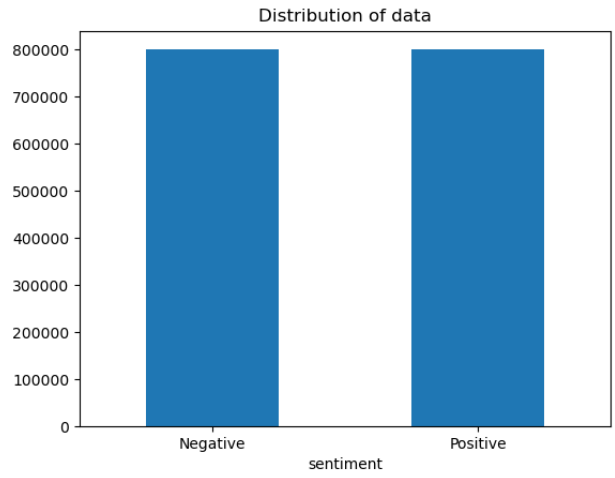


Figure 5.1: Classification distribution in the Sentiment140 dataset.

Further examination of the dataset can be accomplished by extracting a positively and negatively classified entry, the dataset provides six features for each post: Target (the sentiment polarity), ID, Date, Flag (the query), User, and Text (the body).

Table 5.2: Example of a positive post.

| Attribute | Value |
|---|---|
| Target | 4 |
| IDs | 1467822272 |
| Date | Mon Apr 06 22:22:45 PDT 2009 |
| Flag | NO_QUERY |
| User | ersle |
| Text | I LOVE @Health4UandPets u guys r the best!! |

Table 5.3: Example of a negative post.

| Attribute | Value |
|---|---|
| Target | 0 |
| IDs | 1467810369 |
| Date | Mon Apr 06 22:19:45 PDT 2009 |
| Flag | NO_QUERY |
| User | _TheSpecialOne_ |
| Text | @switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D |

### Pre-Processing the Data

Currently, the data is not fit for training purposes, to make it suitable the dataset must be cleaned up. Firstly, the dataset is loaded into a Pandas `DataFrame` using ISO-8859-1 encoding due to limitations with utf-8, then the unnecessary data can be removed from the dataset as it is loaded, as only the text and sentiment polarity are needed for training.

```
df = pd.read_csv(input_path, names=['sentiment', 'timestamp', 'datestring'
    , 'N/A', 'user', 'text'], encoding='ISO-8859-1')
df = df[['text', 'sentiment']]
```

Next, the way sentiment is classified in the dataset must be changed; currently negative sentiment is labelled as '0' and positive is '4', this is changed to a simple binary classification where '0' is negative and '1' is positive.

```
df['sentiment'] = df['sentiment'].replace(4, 1)
```

The dataset should also be shuffled, this ensures that there are no related or similar tweets next to each other in the dataset.

```
toolkit.console("Shuffling dataset...")
df = df.sample(frac=1).reset_index(drop=True)
toolkit.console("Dataset shuffled.")
```

Now that the dataset has been prepared, the text of each post needs to be processed, for this purpose a class `TextProcessor` is defined to handle all processing.

```
class TextProcessor(object):
    def __init__(self):
        self.lemmatiser = WordNetLemmatizer()
```

The first function `preprocess()` is to remove various parts that will not contribute, or may even be detrimental, to the training of the model, such as urls, user mentions, etc. Firstly, several regex patterns are defined:

```
def preprocess(self, text: str) -> str:
    x_mention_pattern = r'@\S{4,}'
    ampersand_pattern = r'&amp;'
    url_pattern = r'((http://)[^ ]*|(https://)[^ ]*|( www\.)[^ ]*)'
    reddit_user_mention_pattern = r'/?u/\S+'
    reddit_sub_mention_pattern = r'/?r/\S+'
    newline_pattern = r'(\r\n|\r|\n)'
    reddit_url_match = r'/\[.*?(?=\]\((.*?)\))/g'
    reddit_url_replace = r'/\[.*?\]\(.*?\)/g'
```

Then `re.sub()` is called on the text for each substitution:

```
text = re.sub(x_mention_pattern, 'USER', text) # @user -> USER
text = re.sub(ampersand_pattern, '&', text) # &amp; -> &
text = re.sub(reddit_user_mention_pattern, 'USER', text) # /u/user or u/
    user -> USER
```

```
text = re.sub(reddit_sub_mention_pattern, 'SUBREDDIT', text) # /r/sub or r
    /sub -> SUBREDDIT
text = re.sub(url_pattern, 'URL', text) # https://link or http://link ->
    URL
text = re.sub(newline_pattern, ' ', text) # \\n -> ' '
```

Finally, for URLs on Reddit, which are in the format [shown text](URL hyperlink), all matches of Reddit's URL format are found, and replaced with just the text that would be shown to the user.

```
# Find all tags (text inside square brackets)
tags = [match.group(1) for match in re.finditer(r'\[(.*?)(?=\]\(.*?\))',
    text)]
# Replace all markdown links with the extracted tags
text = re.sub(r'\[.*?\]\(.*?\)', lambda match: tags.pop(0), text) # [name
    ](link) -> name
```

There are also other optional functions that were experimented with for training in the `TextProcessor`: `lowercase()`, `lemmatise()`, and `soup()`, which toggle the text to lowercase, lemmatise and stem the text, and apply a `BeautifulSoup` html parser to the text, respectively.

These functions are toggled off by default, but may be turned on by the user in settings. For further insight into these, visit the GitHub repository for the project (Atkinson, 2024b) (under `toolkit/data/preprocess.py`).

### Training the Model

In the `BertModel`, a method `train()` for training the model is implemented, training on the Sentiment140 (KazAnova, 2017) dataset will fine-tune BERT to the specific linguistics and tone of social media. This function takes in a DataFrame containing the 'text' and 'sentiment' columns, along with the path to save the model to after training.

```
def train(self, dataset: pd.DataFrame, path: str) -> None:
```

Firstly, the data must be split into training, testing, and validation data. This is accomplished by leveraging Scikit-learn's `train_test_split()` method twice, initially for an 80/20 train/test split, then once more for a 80/10/10 split for separate testing and validation sets, ensuring robust evaluation metrics.

```
X_train, X_test, y_train, y_test = train_test_split(text, sentiment,
    test_size=1 - train_ratio, stratify=sentiment, random_state = 42)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=
    test_ratio/(test_ratio + val_ratio), stratify=y_test, random_state=42)
```

After this, `self.model` and `self.tokeniser` are re-initialised to avoid interference from any previously stored configurations. Then the data is encoded using `self.tokenise()` to transform the text inputs into token IDs.

```
self.tokeniser = BertTokenizer.from_pretrained('bert-base-uncased',
    do_lower_case=True)
self.model = TFBertForSequenceClassification.from_pretrained('bert-base-
    uncased', num_labels=2)
```

Each value is encoded using `self.tokenise()` which takes a string 'text' to transform the text inputs into token IDs.

```
self.tokeniser.batch_encode_plus(text, padding=True, truncation=True,
    max_length=128, return_tensors='tf')
```

Following this, `self.fit_model()` is called, the model is compiled with an optimiser, loss function, and evaluation metrics. The Adam optimiser is used with a learning rate of 2e-5, and a sparse categorical cross-entropy loss function is defined to compute the difference between predicted and actual labels, with accuracy chosen as the evaluation metric for assessing model performance.

```
optimiser = tf.keras.optimizers.Adam(learning_rate=2e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

```
metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')
self.model.compile(optimizer=optimiser, loss=loss, metrics=[metric])
```

Finally, the model is fitted to the encoded data using `self.model.fit()`. This process trains the model on the training data while simultaneously validating its performance on the validation set. The batch size, which is the number of samples processed per gradient update, is set to 32, and the training process iterates over the dataset for 3 epochs.

Through these iterations, the BERT model adapts its parameters to accurately classify the sentiment of the given social media post, enhancing its performance on this type of content.

```
history = self.model.fit([X_train_encoded['input_ids'], X_train_encoded['
    token_type_ids'], X_train_encoded['attention_mask']], np.array(y_train
    ), validation_data=([X_val_encoded['input_ids'], X_val_encoded['
    token_type_ids'], X_val_encoded['attention_mask']], np.array(y_val)),
    batch_size=32, epochs=3)
```

### Cross-Validation

Cross-validation is a crucial element to evaluate the performance and robustness of a machine learning model. By splitting the dataset into multiple folds, cross-validation ensures that the model generalises well for unseen data, reducing the likelihood of overfitting, while also providing insight into the model's consistency across different subsets of data.

This method is called at the end of the `train()` function instead of the single `self.fit_model()` call, if the user has enabled cross-validation. It takes in the training `DataFrame` and the number of splits `n_splits`, which defaults to 5. The text and sentiment columns of the dataset are converted into lists to be used later.

```
def cross_validate(self, dataset: pd.DataFrame, n_splits: int = 5) ->
    None:
    text = dataset['text'].tolist()
```

```
    sentiment = dataset['sentiment'].tolist()
```

Scikit-learn's `KFold` class is used to create cross-validation splits, shuffling the data before splitting to randomise the data.

```
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
```

After initialising several empty lists for test loss, test accuracy, train accuracy, predicted labels, and actual labels, the function enters a loop which iterates through each fold; every iteration splits the data into training and validation sets, tokenises them, then fits and tests the model.

```
for train_index, val_index in kf.split(text):
    X_train, X_val = [text[i] for i in train_index], [text[i] for i in
        val_index]
    y_train, y_val = [sentiment[i] for i in train_index], [sentiment[i]
        for i in val_index]
    X_train_encoded = self.tokenise(X_train)
    X_val_encoded = self.tokenise(X_val)
    history = self.fit_model(X_train_encoded, y_train, X_val_encoded,
        y_val)
    test_loss, test_accuracy, pred_labels, actual_labels = self.test(
        X_val_encoded, y_val)
```

The results of each fold are appended to their corresponding, predefined lists for further analysis.

### Testing the Model

In order to evaluate the model's performance on unseen data, it must be tested. This is done through the `test()` method, which takes the encoded test data in the form of a dictionary, and their corresponding labels in a list. The function returns key metrics such as test loss, test accuracy, and the predicted labels; these metrics help assess how well the model has generalised from the training data to the test data.

```
def test(self, X_test_encoded: dict, y_test: list[int]) -> tuple[float,
    float, list[str], list[str]]:
```

The model is evaluated on the test data using the `self.model.evaluate()` method.

```
test_loss, test_accuracy = self.model.evaluate([X_test_encoded['input_ids'
    ], X_test_encoded['token_type_ids'], X_test_encoded['attention_mask'
    ]], np.array(y_test))
```

Then predictions are made using the `self.model.predict()` method, which returns logits. Using TensorFlow's `argmax()` function, the predicted labels can be obtained, which are then converted to a NumPy array and then their sentiment as a string using list comprehensions.

```
pred = self.model.predict([X_test_encoded['input_ids'], X_test_encoded['
    token_type_ids'], X_test_encoded['attention_mask']])
logits = pred.logits
pred_labels = tf.argmax(logits, axis=1)
pred_labels = pred_labels.numpy()
labels = {1: 'Positive', 0: 'Negative'}
pred_labels = [labels[i] for i in pred_labels]
actual_labels = [labels[i] for i in y_test]
```

### 5.1.2 Predictions

The `predict()` function is designed to leverage the batch-processing and parallel-processing capabilities of BERT and TensorFlow, allowing for efficient sentiment prediction with a list of inputs of any length. It takes a list of strings to analyse, and returns the predicted sentiment labels for each input string.

```
def predict(self, text: list[str]) -> list[str]:
```

Initially, the input text is tokenised using the `self.tokenise()` method, which

encodes the text into input IDs, token type IDs, and attention masks. These encoded values are then passed to the `self.model.predict()` function to get the predictions.

```
input_ids, token_type_ids, attention_mask = self.tokenise(text).values()
prediction = self.model.predict([np.array(input_ids), np.array(
    token_type_ids), np.array(attention_mask)])
```

Next, the prediction logits are processed to obtain the probability distributions over the classes using TensorFlow's `softmax()` function, and the confidence for each prediction is determined by the maximum probability value.

```
pred_logits = prediction.logits
pred_probs = tf.nn.softmax(pred_logits, axis=1).numpy()
pred_confidence = np.max(pred_probs, axis=1)
```

Finally, the predicted labels are determined using TensorFlow's `argmax()` function on the logits. The labels are mapped to their corresponding sentiment labels 'Positive' or 'Negative'. However, if the confidence score of a prediction is less than the threshold defined by the user, it is labelled as 'Neutral'.

```
labels = {1: 'Positive', 0: 'Negative'}
pred_label_indices = np.argmax(pred_logits, axis=1)
pred_labels = [labels[i] if confidence >= confidence_threshold else '
    Neutral' for i, confidence in zip(pred_label_indices, pred_confidence)
    ]
```

The function then returns the predicted labels for further analysis.

## 5.2 Social Media Data Collection

The following section details the methods used to collect data from social media platforms for sentiment analysis. The data collection functionality is implemented within the `XScraper` and `RedditScraper` classes in `toolkit/data/social.py`. Unfortunately, due to new limitations with the free version of the X API which prevent

searching posts, the `XScraper` class is not fully implemented and will be excluded from this section.

The `RedditScraper` class is used to scrape post data from Reddit, including comments if the user's configuration calls for it, using the PRAW library. It interfaces with Reddit's API to retrieve data based on specified subreddits and search terms.

Firstly, the class initialises by setting up an instance of the Reddit API `praw.Reddit`. In this method, the necessary credentials are fetched (client ID, client secret, and user agent) from the environment variables. Storing these as environment variables prevents them from being hard-coded, increasing security and allows flexibility between users.

```
class RedditScraper(object):
    def __init__(self):
        self.api = praw.Reddit(client_id=os.getenv('REDDIT_CLIENT_ID'),
            client_secret=os.getenv('REDDIT_CLIENT_SECRET'), user_agent=
            os.getenv('REDDIT_USER_AGENT'))
```

Next, the class defines the `search_subs()` method, which is designed to search Reddit for posts and comments across any number of subreddits based on a list of given search terms. It takes a dictionary `subs` where the keys are the subreddits to search and the values are lists of search terms, and an integer `n`, which is the number of posts to retrieve per subreddit. Then it returns either a single `DataFrame` containing the scraped posts, or a tuple of two `DataFrame`s, containing both scraped posts and comments.

```
def search_subs(self, subs: dict[str, list[str]], n: int = 10) -> 'pd.
    DataFrame | tuple[pd.DataFrame, pd.DataFrame]':
```

Initially, the function defines two empty lists `posts` and `comments`, it then iterates through the given dictionary's key/value pairs, calling `self.search_sub()` on them and storing the returned values in temporary lists; these are then concatenated with the main lists before moving on to the next iteration.

```
posts = []
comments = []
for sub, search_terms in subs.items():
    temp_posts, temp_comments = self.search_sub(sub, search_terms, n)
    posts += temp_posts
    comments += temp_comments
```

As the Reddit API returns `Submission` and `Comment` objects, the function must now extract the needed attributes from these objects. To do this, two lists `post_attributes` and `post_comments` are created to be two dimensional lists. The function then iterates through the list of posts, initialising a list `post_comments`, and checks `post.is_self` to make sure the post is text-only. If it is, certain attributes are pulled from the `Submission` object:

- `id`: ID of the submission.
- `subreddit.display_name`: Provides an instance of `Subreddit` and then retrieves the name of that subreddit.
- `created_utc`: The time the submission was created, represented in Unix Time.
- `title`: The title of the submission.
- `selftext`: The body text of the submission.
- `score`: The number of 'upvotes' for the submission.

Also, while not an attribute of `Submission`, the tuple `tuple(comment for comment in post_comments)` is created as a list of comment IDs and added, with the other attributes, to a row in `post_attributes`.

```
for post in posts:
    post_comments = []
    if post.is_self:
        if scrape_comments:
            # Functionality will be covered next
        post_attributes.append([post.id, post.subreddit.display_name, post.
            created_utc, post.title, post.selftext, post.score, tuple(
            comment for comment in post_comments)])
```

If scraping comments is enabled by the user, this will also iterate over the comments of the post, pulling the same attributes as it did for the post, but along with the `submission.id` of the comment, referring to the parent post.

```
for comment in comments[posts.index(post)]:
    post_comments.append(comment.id)
    comment_attributes.append([comment.id, comment.submission.id, comment.
        subreddit.display_name, comment.created_utc, comment.body, comment.
        score])
```

Finally, the function defines column labels for the post and comment attributes, and adds them to `posts_df` and `comments_df`, returning the resulting `DataFrame`s.

```
columns = ['ID', 'Subreddit', 'Date/Time', 'Title', 'Body', 'Score', '
    Comments']
posts_df = pd.DataFrame(post_attributes, columns=columns)
if scrape_comments:
    columns = ['ID', 'PostID', 'Subreddit', 'Date/Time', 'Body', 'Score']
    comments_df = pd.DataFrame(comment_attributes, columns=columns)
```

The function `search_sub()`, which is called in `search_subs()`, searches Reddit for posts and comments within a specified subreddit based on the given search terms (none if the list is empty). It's parameters are the subreddit's name, the search terms to query, and the amount of posts to collect, returning a list of posts and comments.

```
def search_sub(self, sub: str, search_terms: list[str], n: int = 10) ->
    tuple[list, list]:
```

It then iterates over the search terms, requesting posts that contain the query in the specified subreddit, and adds the received post to the `posts` list. If comment scraping is enabled, it calls `self.search_comments()`, passing the post as an argument, and adds the returned list to the `comments` list.

```
if search_terms:
```

```
    for search_term in search_terms:
        for post in self.api.subreddit(sub).search(search_term):
            posts.append(post)
            if scrape_comments:
                post.comment_sort = 'hot'
                comments.append(self.search_comments(post))
```

If `search_terms` is empty, a near identical code block is executed, the difference being that it only has one iteration over the top ten posts in the subreddit after sorting by 'hot'.

```
for post in self.api.subreddit(sub).hot(limit=n):
# Rather than 'for post in self.api.subreddit(sub).search(search_term):'
```

The final function in `RedditScraper` is `search_comments()`. This method takes arguments `post` and `limit`, being the parent post and the amount of comments to retrieve respectively. This function simply iterates over the `CommentForest` returned by `post.comments`, and adds the results to a list of comments to be returned.

```
def search_comments(self, post, limit: int = 5) -> list:
post_comments = []
    post.comment_limit = limit
    post.comments.replace_more(limit=0) # Remove all MoreComments
    for comment in post.comments:
        post_comments.append(comment)
    return post_comments
```

## 5.3 Database Management

All management of the database is done within the `PostCollector` class found in `toolkit/data/database.py`. This class incorporates the functionality of `BertModel`, `RedditScraper`, and database management operations defined in the class itself.

The class initialises with the passed arguments `model` and `scraper`, an instance of `BertModel` and `RedditScraper` respectively, and a `profile`, which is a dictionary containing the details of subreddits and search terms to be scraped. `self.path` is then set to point at a directory based on the profile ID, which is where the collected data will be stored in `.json` format. Finally, two empty `DataFrame`s are created with the relevant column labels for posts and comments, and the object calls `self.from_json()` to load any data that may already exist for this profile.

```
class PostCollector(object):
    def __init__(self, model, scraper, profile: dict[str, any]) -> None:
        self.model = model
        self.scraper = scraper
        self.profile = profile
        self.path =f'{toolkit.get_dir()}/src/profiles/{self.profile["id"]}'
        self.posts = pd.DataFrame(columns=['ID', 'Subreddit', 'Date/Time',
            'Title', 'Body', 'Score', 'Comments', 'Sentiment'])
        self.comments = pd.DataFrame(columns=['ID', 'PostID', 'Subreddit',
            'Date/Time', 'Body', 'Score', 'Sentiment'])
        self.from_json()
```

The `scrape_posts()` method is responsible for collecting data from the subreddits and search terms configured in the loaded profile. It collects and processes posts and comments, cleans the text, predicts sentiment, and stores the data in the database.

Firstly, it initialises an instance of the `TextProcessor` object for later, and retrieves the subreddits to be scraped from `self.profile`. Then it calls `search_subs()` to retrieve posts and or comments using the `RedditScraper` depending on the current configuration; the resulting data is stored in `new_posts` and or `new_comments`.

```
def scrape_posts(self, n: int) -> None:
    text_processor = toolkit.TextProcessor()
    subs = self.profile['subs']
    if scrape_comments:
```

```
        new_posts, new_comments = self.scraper.search_subs(subs, n=n)
    else:
        new_posts = self.scraper.search_subs(subs, n=n)
```

Next, the method iterates over each post ID in `new_posts` to get the title and body of each one using `self.get_record()`, which returns the specific record from a `DataFrame` for the given ID and column. If the title or body is `None`, then the row is dropped from `new_posts`, otherwise the title and body are cleaned with the `TextProcessor` and concatenated with a BERT separator token `'[SEP]'`.

```
for ID in new_posts['ID']:
    title = self.get_record(new_posts, ID, 'Title')
    body = self.get_record(new_posts, ID, 'Body')
    if title is None or body is None:
        index = new_posts.index[new_posts['ID'] == ID]
        new_posts.drop(index)
        continue
    title = text_processor.clean(title)
    body = text_processor.clean(body)
    text = f"{title} [SEP] {body}"
    if not text.strip():
        print(f"Empty text for post ID: {ID}")
    else:
        all_texts.append(text)
```

All non-empty concatenated texts are then added to a list for batch sentiment prediction through `BertModel`'s `predict()` method, and once the sentiment is analysed, these values are inserted as a column into `new_posts`. Finally, `new_posts` is added to `self.posts`, removing any duplicate posts to ensure the integrity of the data.

```
post_sentiments = self.model.predict(all_texts)
new_posts.insert(6, 'Sentiment', post_sentiments, True)
```

```
self.posts = pd.concat([self.posts, new_posts], ignore_index=True)
self.posts.drop_duplicates(keep='last', inplace=True)
self.posts.drop_duplicates('ID', keep='last', inplace=True)
```

If comment search is enabled, there is similar behavior, the only difference is that the title and body of the parent post are concatenated with the body of the comment using the same BERT separator tokens. This allows BERT to predict the sentiment of the comment with the context of the parent post.

```
for ID in new_comments['ID']:
    post_id = self.get_record(new_comments, ID, 'PostID')
    # Get records of post title, body, and comment
    if title is None or body is None:
        # Drop as before
    # Clean the texts
    text = f"{title} [SEP] {body} [SEP] {comment}"
```

The new data is then stored in `.json` format for later use using `self.to_json()`.

## 5.4 Data Visualisation

The `Analyser` class in `toolkit/analysis/analyse.py` is crucial for interpreting sentiment data to the user via visual representation; large datasets could be overwhelming to the user, so by visualising the data and breaking it up into more digestible formats, the user is able to make quicker and more accurate insights. Furthermore, representing the data graphically makes it far easier to spot patterns that are not immediately obvious when looking at raw data, such as the change in sentiment over time or comparisons between different subreddits or queries.

Upon initialisation, the `Analyser` class prepares the passed `DataFrame` by converting the sentiment labels into a numerical format with `self._set_sentiment_score()`, where 'Positive', 'Neutral', and 'Negative' labels are converted to 1, 0, and -1 respectively. These new values are inserted as a new column, and if weighting by score is enabled in settings, these values are multiplied by the score (number of 'upvotes')

of the post; this allows the user to see how many people agree with each post, thus leading to a more in depth analysis.

```
class Analyser(object):
    def __init__(self, dataset: pd.DataFrame) -> None:
        self.dataset = dataset
        self._set_sentiment_score()

    def _set_sentiment_score(self) -> None:
        if toolkit.get_config('score_weighting'):
            self.dataset['SentimentScore'] = self.dataset['Score'] * self.
                dataset['Sentiment'].apply(self._sentiment_to_int)
```

This class defines three functions for chart creation, `generate_line()`, `generate_pie`, and `generate_bar`, for creating a line, pie, and bar chart respectively. They each take the `canvas` argument, which is a `FigureCanvas` object from `matplotlib` for use as a `QtPy` widget, along with a chart title, and start/end date for the plotted data. Both the line and bar chart also take the `labels` argument, which is a tuple containing the labels for the axis.

```
def generate_line(self, canvas: FigureCanvas, title: str, labels: tuple[
    str, str], start_date: float, end_date: float) -> None:
def generate_pie(self, canvas: FigureCanvas, title: str, start_date: float
     = None, end_date: float = None) -> None:
def generate_bar(self, canvas: FigureCanvas, title: str, labels: tuple[str
    , str], start_date: float = None, end_date: float = None) -> None:
```

The line chart is useful for visualising the changes in sentiment score over a given time-frame; line charts can be useful for identifying trends such as seasonal variations, spikes, or declines in sentiment. This specific implementation also allows for data from different subreddits to be visualised separately as independently plotted lines, if `split_subs` is configured to do so.

The pie chart represents the distribution of sentiment classifications within the

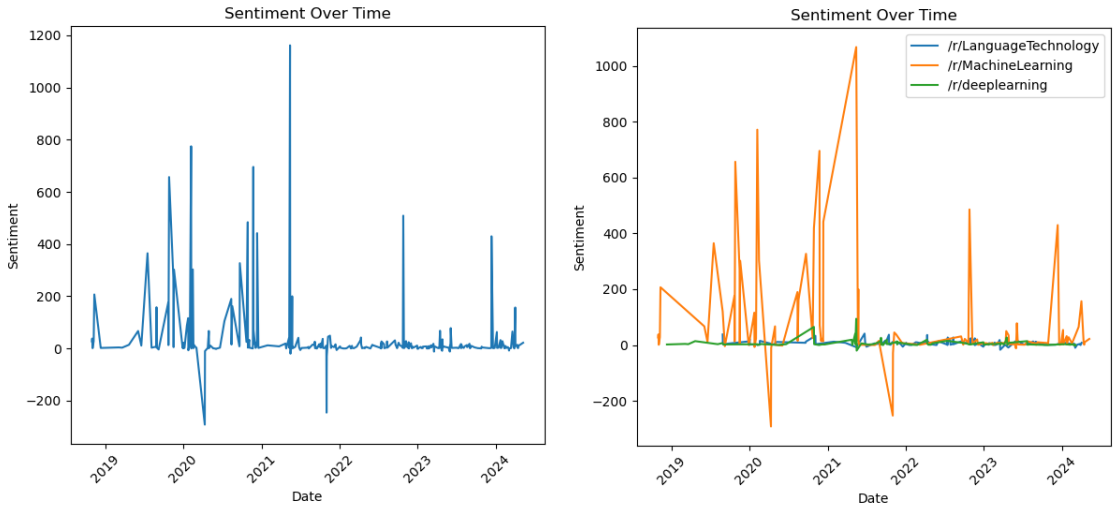


Figure 5.2: Example line charts with subreddit splitting enabled and disabled.

dataset as a whole. Pie charts are great for showing the proportions of sentiment classes within the whole dataset, which makes them intuitive to the user, giving easy insight into which sentiment category dominates and how sentiment is distributed.

Finally, the bar chart compares the total sentiment scores across different subreddits. They are ideal for comparative analysis as they clearly display proportional quantities for different groups (in this case, subreddits), allowing for clear visual indication of which subreddits and queries have the highest/lowest sentiment scores.

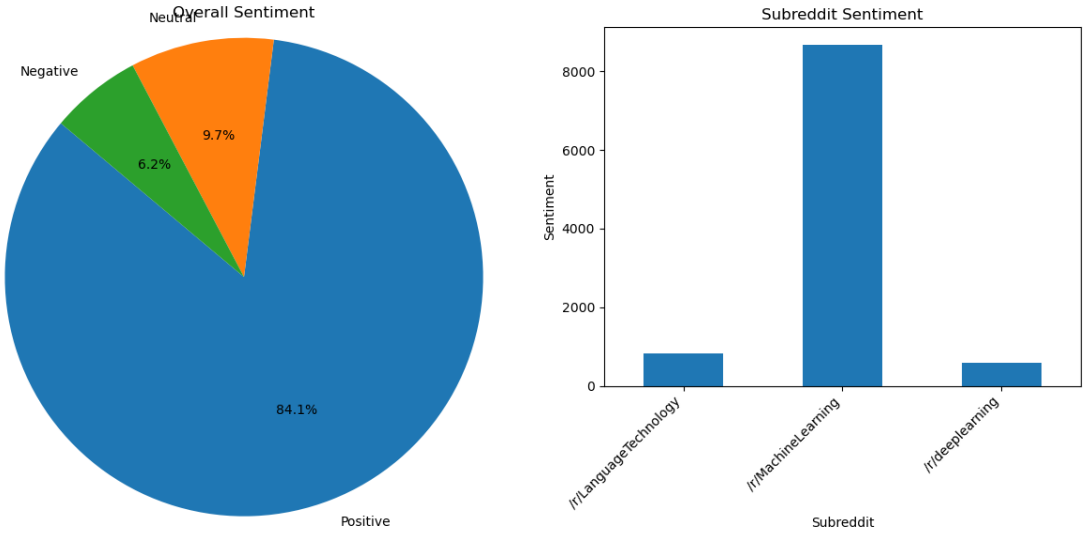


Figure 5.3: Example pie and bar chart.

For further insight into the implementation of these charts, visit the GitHub repository (Atkinson, 2024b).

## 5.5 Graphical User Interface

The GUI of the application (`toolkit/ui/interface.py`) serves as the means by which the user can interact with the functionality of the back-end processes. It is designed to provide an intuitive and user-friendly experience, while keeping performance and responsiveness to a high standard. This section outlines the architecture and components of the GUI, highlighting the core aspects such as the distinct windows and the use of parallel programming for long-running tasks. For the purpose of this application, the `QtPy` library was used, as it excels at creating complex, responsive, and user-friendly interfaces, as well as providing parallelism features.

### 5.5.1 Main Window

The `MainWindow` class, inheriting from `QtPy`'s `QMainWindow`, creates the main application, it displays widgets that link to the core functionalities of the program in an accessible format. The layout is built up by embedding other layouts and widgets within each other, creating an interface which is complex but pleasant to use.

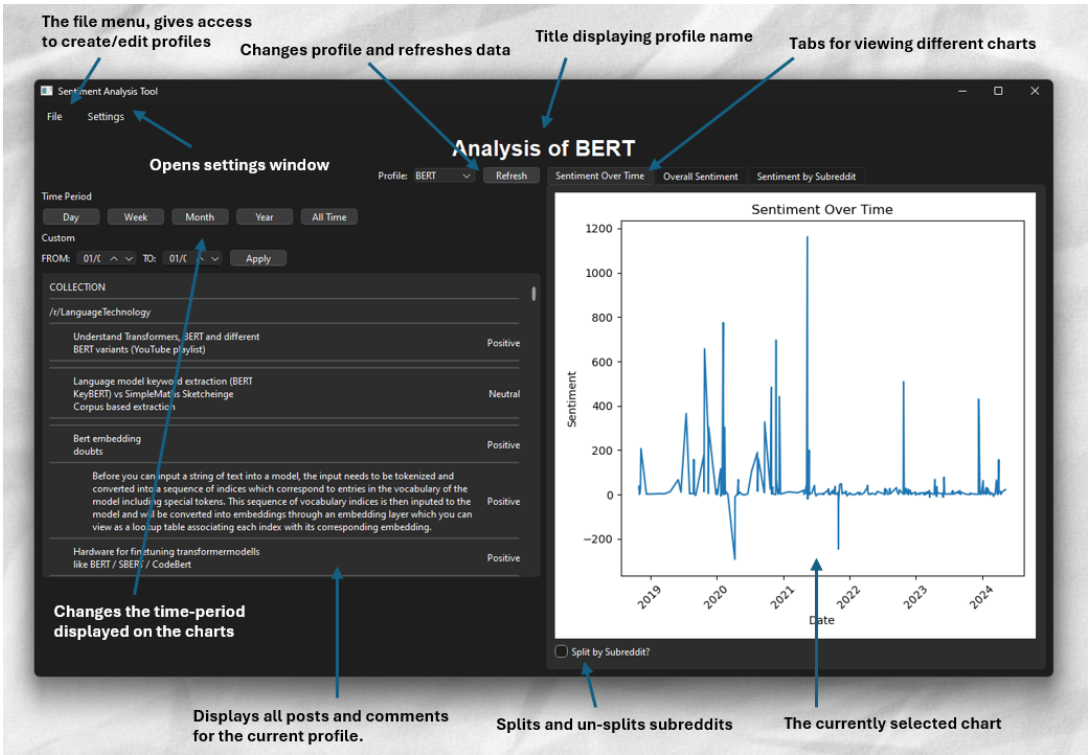


Figure 5.4: The main window of the application.

As can be seen in the provided figure, the window consists of several main elements: a menu bar, a profile changer, a time period editor, a scrollable widget featuring all posts for the profile, and an area with tabs displaying different charts. Each distinct area is built up using layouts and widgets embedded in parent layouts, which are used as building blocks for the GUI. For example, the time period editor is built up through several different `MainWindow` methods, namely:

- `self._make_time_period_layout()`
- `self._make_custom_time_layout()`
- `self._make_time_changer_layout()`

The time period layout consists of five buttons to change the time period to either the last twenty-four hours, week, month, year, or all-time.

The `QtPy` object `QHBoxLayout` is used to create a layout consisting of a row of horizontal 'boxes', which are used for placing widgets (or other layouts) into. Due to the one-dimensional layout of this, it is necessary to embed layouts `QHBoxLayout` and `QVBoxLayout` within each other to achieve a complex GUI. Once the layout is created, a `QPushButton` widget is created, connected to the `self._time_period_day()` function, and added to the layout as such (code reduced for simplicity):

```
def _make_time_period_layout(self):
    layout = QHBoxLayout()
    button_day = QPushButton("Day")
    button_day.clicked.connect(self._time_period_day)
    layout.addWidget(button_day)
    layout.addStretch()
    return layout
```

Following this, another layout is created to enable the user to select a more specific time period using selection boxes. This layout is implemented in a similar way, however, to prevent the charts updating every time the user increments or decrements the start/end date, a button labelled 'Apply' is also added. For this button to work and pass the new start/end date to its connected function, a lambda function must be created as the passed function.

```
def _make_custom_time_layout(self):
    layout = QHBoxLayout()
    selector_from = QDateEdit()
    selector_to = QDateEdit()
    apply = QPushButton("Apply")
    apply.clicked.connect(lambda: self._set_dates(selector_from,
        selector_to))
    # Add widgets and return layout
```

These are then added to their parent layout, itself being a child of other layouts.

```
def _make_time_changer_layout(self):
    layout = QVBoxLayout()
    time_period_layout = self._make_time_period_layout()
    custom_time_layout = self._make_custom_time_layout()
    layout.addLayout(time_period_layout)
    layout.addLayout(custom_time_layout)
def _make_left_layout(self):
    layout.addLayout(self._make_time_changer_layout())
def _make_main_layout(self):
    layout.addLayout(self._make_left_layout())
```

Finally, the finished layout, made up of dozens of child layouts and widgets, is set as the layout of the window itself in `MainWindow.__init__()`.

```
class MainWindow(QMainWindow):
    def __init__(self, *args, **kwargs):
        self.widget = QWidget(self)
        self.setCentralWidget(self.widget)
        self.layout = self._make_layout()
        self.widget.setLayout(self.layout)
```

To see the `MainWindow` in full detail, see the GitHub (Atkinson, 2024b).

### 5.5.2 Profile Editor

The `ProfileEditorWindow` is a dedicated interface for managing the brand profiles. It allows the user to create, edit, and delete profiles; each profile contains information about which subreddits to monitor and analyse, and which search terms, if any, to query.

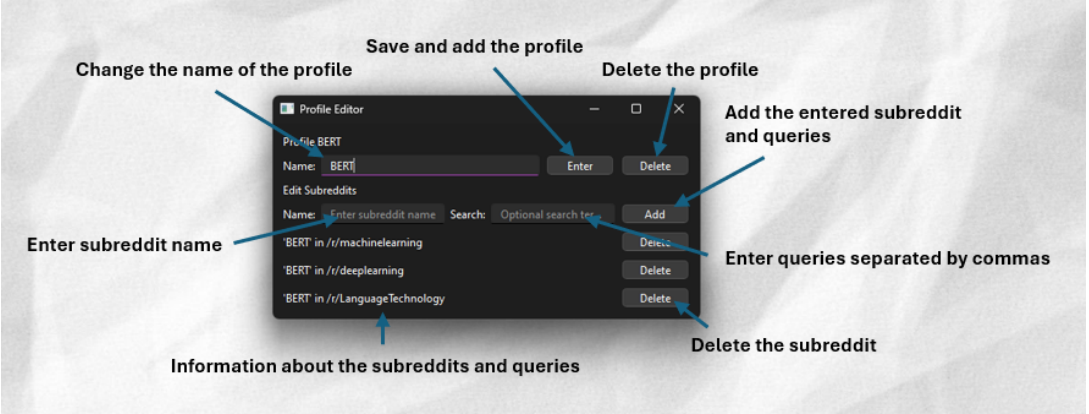


Figure 5.5: The profile editor window.

In order for this window to communicate with the main window, a `pyqtSignal` object must be defined, which allows the transmission of signals between different objects in a program. This signal object is then used in the `ProfileEditorWindow._update()` function to `emit()` a signal that is picked up by the `MainWindow`. This allows the main window to update every time the profile editor window does.

```
class ProfileEditorWindow(QWidget):
    submit_clicked = pyqtSignal()
    def _update(self):
        self.submit_clicked.emit()

# Inside MainWindow
self.window_profile_editor.submit_clicked.connect(self._update)
```

To gain more insight into the workings of the `ProfileEditorWindow`, please refer to the GitHub repository (Atkinson, 2024b).

### 5.5.3 Settings

The `SettingsWindow` class allows the user to customise various settings and preferences. It is divided into several distinct tabs for organisation and ease of use, such as General, Analysis, Personalisation, etc.

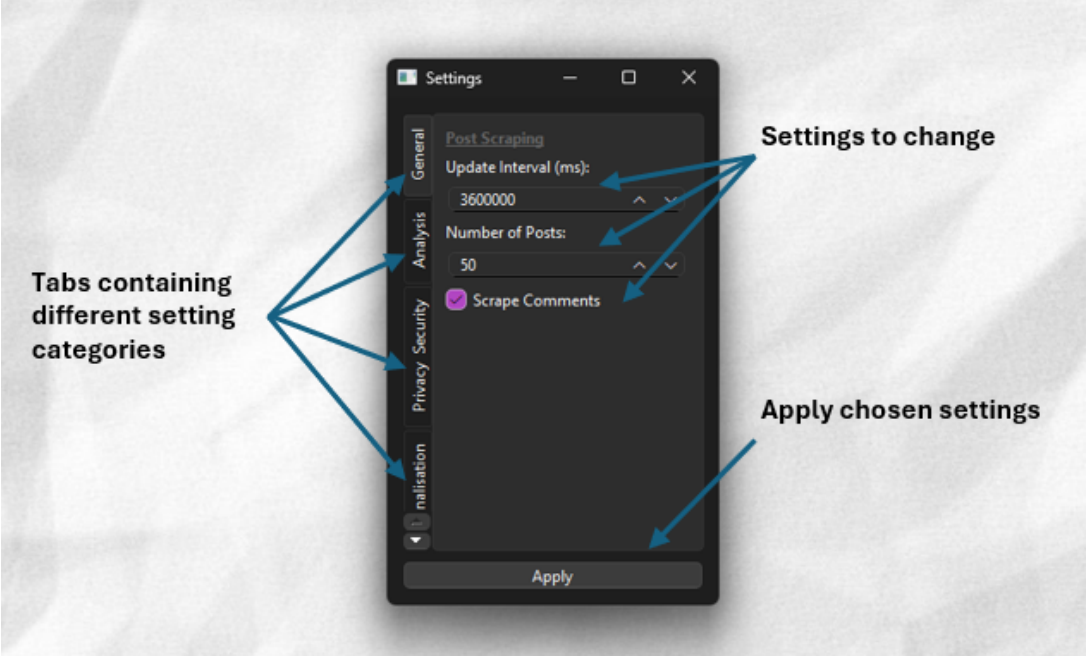


Figure 5.6: The settings window.

- **General**: user can change the update interval, number of posts to scrape, and toggle comment scraping.
- **Analysis**: provides options for text processing such as lowercase, lemmatisation, and `BeautifulSoup` parsing, as well as model configurations like toggling cross-validation and changing the confidence threshold.
- **Privacy & Security**: settings such as encryption can be toggled here (no functionality implemented).
- **Personalisation**: allows the user to customise the appearance of the application, such as font, colour, and chart styling (no functionality implemented).
- **Export & Sharing**: allows the user to initiate a data export or share analyses (no functionality implemented).
- **Feedback & Support**: provides the user with links to the creator's LinkedIn profile, send the creator an email, or visit the codebase on GitHub.

Each of these tabs have their own unique settings, for example, in the 'General' tab, the update interval can be set using a `QSpinBox` that sets the interval in milliseconds, allowing a range anywhere from one second to twenty-four hours.

```
update_interval_spinbox = QSpinBox()
update_interval_spinbox.setMinimum(1000)
update_interval_spinbox.setMaximum(86400000)
update_interval_spinbox.setValue(toolkit.get_config("update_interval"))
update_interval_spinbox.valueChanged.connect(lambda value, name="
    update_interval": self._update_setting(name, value))
```

To prevent constant updates to the `MainWindow`, all settings changes are deferred inside of the `self.settings` dictionary until the user clicks the 'Apply' button.

```
def _update_setting(self, name, value):
    self.settings[name] = value
```

This window uses similar functionality as the `ProfileEditorWindow` to communicate with the `MainWindow` using a `pyqtSignal` object, which this time is sent when the user clicks 'Apply' and `self._apply_settings()` is called.

```
class SettingsWindow(QWidget):
    submit_clicked = pyqtSignal()
    def _apply_settings(self):
        for setting_name, value in self.settings.items():
            toolkit.set_config(setting_name, value)
        self.submit_clicked.emit()
```

### 5.5.4 Parallelism for Long-Running Tasks

To ensure the application runs smoothly and efficiently, specifically during resource-intensive tasks such as social media scraping and analysis, `QtPy`'s parallelism features are used. Running tasks in parallel allows the application to perform multiple tasks

simultaneously, significantly reducing the time required to execute. It is also ensures that the GUI stays responsive during these tasks rather than freezing, meaning the user can continue to interact with and use the program unimpeded.

In the `MainWindow`, parallelism is employed in the execution of two functions: `self._collect_new_posts()` and `self._train_model()`, both of which are long-running tasks which would freeze the application for an undesirably extended period of time without parallelism. Initially, the class defines a `self.threadpool` storing a `QtPy.QThreadPool` object, which handles the queuing and execution of 'workers'.

```
class MainWindow(QMainWindow):
    def __init__(self, *args, **kwargs):
        self.threadpool = QThreadPool()
```

To be able to use the `QThreadPool`, a new class `Worker` is defined, inheriting from `QtPy.QRunnable`, with the function to execute passed as an argument to it's `run()` method, using the `@pyqtSlot()` decorator.

```
class Worker(QRunnable):
    def __init__(self, fn: callable, *args, **kwargs) -> None:
        super(Worker, self).__init__()
        self.fn = fn
        self.signals = WorkerSignals()
    @pyqtSlot()
    def run(self):
        try:
            result = self.fn(*self.args, **self.kwargs) # Execute the passed
                function
        except: # Emit exception
        else:
            self.signals.result.emit(result)
        finally:
            self.signals.finished.emit()
```

Finally, the two long-running functions are called using multithreading by passing them as a callable upon a `Worker` creation. The program starts the worker and waits for the 'finished' signal from to be emitted, so that it can call `self._update()`.

```
def _collect_new_posts(self):
    worker = Worker(lambda: self.collector.scrape_posts(toolkit.get_config
        ('n_posts')))
    worker.signals.finished.connect(self._update)
    self.threadpool.start(worker)

def _train_model(self):
    worker = Worker(self.model.train)
    worker.signals.finished.connect(self._update)
    self.threadpool.start(worker)
```

# Chapter 6

# Results & Discussion

The implementation and testing of the sentiment analysis tool (Atkinson, 2024b) have yielded valuable insights into the functionality and performance of the developed software. The purpose of this chapter is to present the results of functionality testing, encompassing the evaluation of key components of the system, such as sentiment analysis, social media interfacing, data management, and the GUI.

Furthermore, it discusses the details of fulfillment for the previously outlined functional and non-functional requirements of the application from the requirements analysis, followed by further reflection on the development process and possible areas for improvement.

## 6.1 Software Evaluation

### 6.1.1 Sentiment Analysis Model

### 6.1.2 Social Media Data Collection

### 6.1.3 Database Management

### 6.1.4 Data Visualisation

### 6.1.5 Graphical User Interface

## 6.2 Requirements Fulfillment

### 6.2.1 Functional Requirements

**Data Collection and Preprocessing (3.2.1)**

**Sentiment Analysis (3.2.2)**

**Real-Time Analysis (3.2.3)**

**Graphical User Interface (3.2.4)**

### 6.2.2 Non-Functional Requirements

**Performance (3.3.1)**

**Usability (3.3.2)**

**Security (3.3.3)**

**Reliability (3.3.4)**

**Accuracy (3.3.5)**

## 6.3 Reflection

# Chapter 7

# Conclusion

# References


Abirami, AM and V Gayathri (2017). 'A survey on sentiment analysis methods and approach'. In: *2016 Eighth International Conference on Advanced Computing (ICoAC)*. IEEE, pp. 72–76 (cit. on pp. 7, 25).

Anny, Fatema Tuz Zohra and Oahidul Islam (2022). 'Sentiment analysis and opinion mining on E-commerce site'. In: *arXiv preprint arXiv:2211.15536* (cit. on p. 5).

Atkinson, Alfie (2023). *Project Proposal*. University of Lincoln, School of Computer Science (cit. on p. 4).

– (2024a). *Project Interim Report*. University of Lincoln, School of Computer Science (cit. on p. 4).

– (2024b). *Sentiment Analysis Tool*. Available from https://github.com/alfieatkinson/sentiment-analysis-tool (cit. on pp. 32, 46, 49, 50, 55).

Ausat, Abu Muna Almaududi et al. (2023). 'Utilisation of Social Media in Market Research and Business Decision Analysis'. In: *Jurnal Minfo Polgan* 12.1, pp. 652–661 (cit. on p. 6).

Barbu, Alina (2013). 'Eight contemporary trends in the market research industry.' In: *Management & Marketing* 8.3 (cit. on p. 7).

Basiri, Mohammad Ehsan, Nasser Ghasem-Aghaee and Ahmad Reza Naghsh-Nilchi (2014). 'Exploiting reviewers' comment histories for sentiment analysis'. In: *Journal of Information Science* 40.3, pp. 313–328 (cit. on p. 9).

Baskaran, Umamageswari and Kalpana Ramanujam (2018). 'Automated scraping of structured data records from health discussion forums using semantic analysis'. In: *Informatics in Medicine Unlocked* 10, pp. 149–158 (cit. on p. 9).

Berestova, Anastasiia, Da-Yeon Kim and Sang-Yong Kim (2022). 'Consumers' active reaction to brands taking stands on public issues on Twitter'. In: *Sustainability* 14.1, p. 567 (cit. on p. 5).

Camacho, David, Ma Victoria Luzón and Erik Cambria (2021). *New trends and applications in social media analytics* (cit. on p. 6).

Chandra, Chandni Nav, Sarishty Gupta and Renuka Pahade (2015). 'Sentiment analysis and its challenges'. In: *International Journal of Engineering Research & Technology* 4.03, pp. 968–970 (cit. on p. 9).

Choenni, Sunil et al. (2021). 'Using data analytics results in practice: challenges and solution directions'. In: *Perspectives for Digital Social Innovation to Reshape the European Welfare Systems*. IOS Press, pp. 182–201 (cit. on p. 10).

Cvijanovic, Drago, Branko Mihailovic and Aleksandra Nikolic (2014). 'Market Research and Marketing Information Systems'. In: *International Journal of Economic Practices and Theories* 4.2, pp. 191–198 (cit. on p. 7).

D'souza, Stephina Rodney and Kavita Sonawane (2019). 'Sentiment analysis based on multiple reviews by using machine learning approaches'. In: *2019 3rd international conference on computing methodologies and communication (ICCMC)*. IEEE, pp. 188–193 (cit. on pp. 6, 8, 24).

Dean, Brian (2023). *Social Media Usage & Growth Statistics*. Available from https://backlinko.com/social-media-users [Accessed 12 Mar 2024] (cit. on p. 6).

Dewi, Lusiana Citra, Alvin Chandra et al. (2019). 'Social media web scraping using social media developers API and regex'. In: *Procedia Computer Science* 157, pp. 444–449 (cit. on p. 9).

Dhola, Kaushik and Mann Saradva (2021). 'A comparative evaluation of traditional machine learning and deep learning classification techniques for sentiment analysis'. In: *2021 11th international conference on cloud computing, data science & engineering (Confluence)*. IEEE, pp. 932–936 (cit. on pp. 8, 25).

Dhuria, Shabina (2015). 'Sentiment analysis: An approach in natural language processing for data extraction'. In: *International Journal of New Innovations in Engineering and Technology* 2.4, pp. 27–31 (cit. on p. 5).

Duffield, Nick and Jie Wu (2014). 'Challenges and opportunities for analysis based research in big data'. In: *2014 IEEE 33rd International Performance Computing and Communications Conference (IPCCC)*. IEEE, pp. 1–1 (cit. on p. 10).

Ezen-Can, Aysu (2020). 'A Comparison of LSTM and BERT for Small Corpus'. In: *arXiv preprint arXiv:2009.05451* (cit. on p. 8).

Fiesler, Casey, Nathan Beard and Brian C Keegan (2020). 'No robots, spiders, or scrapers: Legal and ethical regulation of data collection methods in social media terms of service'. In: *Proceedings of the international AAAI conference on web and social media*. Vol. 14, pp. 187–196 (cit. on p. 9).

Gerdes, John, Betsy Bender Stringam and Robert G Brookshire (2008). 'An integrative approach to assess qualitative and quantitative consumer feedback'. In: *Electronic Commerce Research* 8, pp. 217–234 (cit. on pp. 7, 25).

Harvard Business Review (2018). *Research: The Average Age of a Successful Startup Founder Is 45*. Available from https://hbr.org/2018/07/research-the-average-age-of-a-successful-startup-founder-is-45 [Accessed 12 Mar 2024] (cit. on p. 11).

Hernandez-Suarez, Aldo et al. (2018). 'A web scraping methodology for bypassing twitter API restrictions'. In: *arXiv preprint arXiv:1803.09875* (cit. on p. 9).

Hu, Guoning et al. (2017). 'Analyzing users' sentiment towards popular consumer industries and brands on twitter'. In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, pp. 381–388 (cit. on p. 5).

Kamruzzaman, Mahammed et al. (2021). 'A comparative analysis of sentiment classification based on deep and traditional ensemble machine learning models'. In: *2021 International Conference on Science & Contemporary Technologies (ICSCT)*. IEEE, pp. 1–5 (cit. on pp. 8, 25).

Kanev, Gabriel, Tsvetelina Mladenova and Irena Valova (2023). 'Leveraging User Experience for Enhancing Product Design: A Study of Data Collection and Evaluation'. In: *2023 5th International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. IEEE, pp. 01–06 (cit. on p. 7).

Kang, Seok, KyuJin Shim and Jiyoun Kim (2019). 'Social media posts on the Samsung Galaxy Note 7 explosion: A comparative analysis of crisis framing and sentiment in three nations'. In: *Journal of International Crisis and Risk Communication Research* 2.2, pp. 259–289 (cit. on p. 5).

Kansara, Dhvani and Vinaya Sawant (2020). 'Comparison of traditional machine learning and deep learning approaches for sentiment analysis'. In: *Advanced Computing Technologies and Applications: Proceedings of 2nd International Conference on Advanced Computing Technologies and Applications—ICACTA 2020*. Springer, pp. 365–377 (cit. on pp. 8, 25).

KazAnova, Marios Michailidis (2017). *Sentiment140 dataset with 1.6 million tweets*. Available from https://www.kaggle.com/datasets/kazanova/sentiment140 [Accessed 17 Apr 2024] (cit. on pp. 29, 32).

Krotov, Vlad, Leigh Johnson and Leiser Silva (2020). 'Tutorial: Legality and ethics of web scraping'. In: (cit. on p. 9).

Kupec, Václav et al. (2015). 'Marketing Research of Digital Life of Bank Clients'. In: *Marketing Identity* 3.1/2, pp. 116–125 (cit. on p. 7).

Lange, Kathy and Saratendu Sethi (2011). 'What Are People Saying about Your Company, Your Products, or Your Brand?' In: *SAS Institute Inc* (cit. on pp. 5, 25).

Liu, Bing (2010). 'Sentiment analysis: A multi-faceted problem'. In: *IEEE intelligent systems* 25.3, pp. 76–80 (cit. on p. 9).

Mali'c, Goran (2019). 'Analysis of social media'. In: URL: https://api.semanticscholar.org/CorpusID:88481375 (cit. on pp. 6, 25).

Mancosu, Moreno and Federico Vegetti (2020). 'What you can scrape and what is right to scrape: A proposal for a tool to collect public Facebook data'. In: *Social Media+ Society* 6.3, p. 2056305120940703 (cit. on p. 9).

MarketingWeek (2023). *2023 Career and Salary Survey*. Available from https://www.marketingweek.com/marketing-skews-young-stats/ [Accessed 12 Mar 2024] (cit. on p. 11).

Marres, Noortje and Esther Weltevrede (2013). 'Scraping the social? Issues in live social research'. In: *Journal of cultural economy* 6.3, pp. 313–335 (cit. on p. 8).

Mishra, Shreyash et al. (2022). 'Data extraction approach using natural language processing for sentiment analysis'. In: *2022 International Conference on Automation, Computing and Renewable Systems (ICACRS)*. IEEE, pp. 970–972 (cit. on p. 5).

Mohammad, Saif M (2022). 'Ethics sheet for automatic emotion recognition and sentiment analysis'. In: *Computational Linguistics* 48.2, pp. 239–278 (cit. on p. 9).

Mostafa, Mohamed M (2013). 'More than words: Social networks' text mining for consumer brand sentiments'. In: *Expert systems with applications* 40.10, pp. 4241–4251 (cit. on pp. 5, 25).

Nasukawa, Tetsuya and Jeonghee Yi (2003). 'Sentiment analysis: Capturing favorability using natural language processing'. In: *Proceedings of the 2nd international conference on Knowledge capture*, pp. 70–77 (cit. on pp. 6, 24).

Nayak, Richi (2002). 'Intelligent data analysis: Issues and challenges'. In: *6th World Multi Conferences on Systemics, Cybernetics and Informatics* (cit. on p. 10).

Pang, Bo and Lillian Lee (2004). 'A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts'. In: *arXiv preprint cs/0409058* (cit. on p. 6).

Pew Research Center (2019). *Millennials stand out for their technology use, but older generations also embrace digital life*. Available from https://www.pewresearch.org/short-reads/2019/09/09/us-generations-technology-use/ [Accessed 12 Mar 2024] (cit. on p. 11).

Qiang, Jipeng et al. (2021). 'Lsbert: Lexical simplification based on bert'. In: *IEEE/ACM transactions on audio, speech, and language processing* 29, pp. 3064–3076 (cit. on p. 8).

Rajnayak, Mamta A, Snigdha Moitra and Charu Nahata (2017). 'Traditional vs. Machine Learning Techniques: Customer Propensity'. In: *Intelligent Information and Database Systems: 9th Asian Conference, ACIIDS 2017, Kanazawa, Japan, April 3–5, 2017, Proceedings, Part II 9*. Springer, pp. 653–663 (cit. on p. 8).

Rangila, Mohammed Athar et al. (2022). 'Sentimental Analysis using Bert Algorithm over LSTM'. In: *International Journal of Advanced Research in Science, Communication and Technology* (cit. on pp. 8, 25).

Sayce, David (2022). *The number of tweets per day in 2022*. Available from https://www.dsayce.com/social-media/tweets-day/ [Accessed 12 Mar 2024] (cit. on p. 5).

Shikovets, Catherine, Halyna Kvita and Svetlana Bebko (2023). 'DIDGITAL TOOLS OF MODERN MARKETING RESEARCH'. In: *Market Infrastructure* (cit. on p. 7).

Staegemann, Daniel et al. (2020). 'Determining Potential Failures and Challenges in Data Driven Endeavors: A Real World Case Study Analysis.' In: *IoTBDS*, pp. 453–460 (cit. on p. 10).

Steinmann, Michael, Sorin Adam Matei and Jeff Collmann (2016). 'A theoretical framework for ethical reflection in big data research'. In: *Ethical reasoning in big data: An exploratory analysis*, pp. 11–27 (cit. on p. 9).

Veeraselvi, SJ and C Saranya (2014). 'Semantic orientation approach for sentiment classification'. In: *2014 international conference on green computing communication and electrical engineering (ICGCCEE)*. IEEE, pp. 1–6 (cit. on p. 9).

Vikram, G and FJ Peter Kumar (2020). 'Use of Social Feedback to Improve Product and Service Quality'. In: *International Journal of Recent Technology and Engineering* (cit. on pp. 7, 25).

Yin, Wen et al. (2023). 'Prompt-Oriented Fine-Tuning Dual Bert for Aspect-Based Sentiment Analysis'. In: *International Conference on Artificial Neural Networks*. Springer, pp. 505–517 (cit. on p. 8).

Zhou, Guangyu and Kavita Ganesan (2016). 'Linguistic understanding of complaints and praises in user reviews'. In: *Proceedings of the 7th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pp. 109–114 (cit. on pp. 6, 25).

Zia, Sadaf et al. (2020). 'There for the reaping: The ethics of harvesting online data for research purposes'. In: *Proceedings of the Annual Conference of CAIS/Actes du congrès annuel de l'ACSI* (cit. on p. 9).