uc3m | Universidad Carlos III de Madrid

University Degree in Computer Science

2022-2023

Bachelor Thesis

“Application of Machine Learning and Natural Language Processing techniques for sentiment classification in tweets”

Daniel Jiménez Campos

NIA - 100405993

Tutor - Paloma Martínez Fernández

Getafe, September 2023

SUMMARY

The burgeoning field of sentiment analysis in healthcare has opened new avenues for understanding public opinion on various therapies. This thesis aims to contribute to this growing body of research by focusing on the multi-class classification of sentiments associated with therapies in English tweets. Leveraging machine learning, natural language processing (NLP), and lexicalized language models (LLMs), the study develops a comprehensive pipeline to preprocess, feature-engineer, and classify sentiments into three categories: positive, negative, and neutral.

Initially developed in Jupyter Notebooks for a more visual understanding, the pipeline was later replicated in PyCharm IDE for faster execution and scalability. The pipeline is divided into several key steps, including data cleaning, exploratory data analysis (EDA), feature engineering, model training, and hyperparameter tuning. Each step is meticulously designed to contribute to the overall effectiveness of the sentiment classification model.

The RandomForestClassifier from the scikit-learn library serves as the primary machine learning algorithm, chosen for its robustness and ability to handle imbalanced datasets. The model's performance is evaluated using the micro-averaged F1-score, achieving a score of 0.6985, which surpasses the mean performance of all teams participating in the SMM4H 2023 Shared Tasks [1], which serves as motivation for this thesis.

While the study provides a robust model for sentiment classification, it also identifies areas for future research, including the incorporation of advanced LLMs, ensemble methods, and real-time analysis capabilities. It also presents an overview of state-of-the-art techniques, concretely using integrated AI in Google's Cloud environment. By addressing these future considerations, the thesis lays a strong foundation for advancing the field of healthcare sentiment analysis.

Please visit the GitHub code repository for more information.

Keywords:

Sentiment Analysis, Healthcare, Machine Learning, Natural Language Processing (NLP), Lexicalized Language Models (LLMs), Multi-class Classification, Tweets, RandomForestClassifier, Exploratory Data Analysis (EDA), Feature Engineering, Hyperparameter Tuning, Micro-averaged F1-score, Text Preprocessing, SMM4H 2023 Shared Tasks, PyCharm IDE, Jupyter Notebooks, Scalability, Data Cleaning

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

In an era where digital technologies and social media have permeated almost every aspect of human life, the ways in which we seek, consume, and discuss information have undergone a monumental shift. Platforms like Twitter have moved beyond being mere social networks; they have become expansive, dynamic digital ecosystems that influence and mirror public opinion on a multitude of subjects, including healthcare. Understanding these digital landscapes, particularly in the context of health-related discussions, is not only crucial for academic research but also holds substantial practical implications for healthcare providers, policymakers, and patients alike.

Expanding the Digital Discourse: The Importance of Twitter Data

Twitter, with its real-time dissemination of short, condensed messages, offers a unique vantage point to capture public sentiment. Unlike other social media platforms that may promote long-form content or specialized communities, Twitter's brevity and immediacy make it an optimal arena for spontaneous reactions and quick opinions. This positions Twitter as an invaluable resource for capturing authentic emotional and intellectual responses to healthcare therapies. The relevance of Twitter data is further augmented by its heterogeneous user base, spanning various demographics and perspectives, providing an even more comprehensive understanding of public sentiment on healthcare therapies.

The Complexities and Challenges of Manual Sentiment Analysis

One cannot overstate the importance of understanding the public sentiment regarding therapies for conditions like chronic pain, long-COVID, or substance use disorders. Such insights are vital for improving patient care, enhancing healthcare policies, and even directing future research initiatives. However, the enormous volume, velocity, and variety of data generated on platforms like Twitter present formidable challenges. Manual analysis, though accurate, is painfully slow and inherently limited, incapable of processing the sheer volume of data generated each day. This highlights a critical bottleneck in making data-driven decisions in healthcare, underscoring the necessity for automated, scalable solutions.

Bridging the Gap: The Role of Machine Learning and NLP

In light of these challenges, my bachelor's thesis is motivated by the potential of Machine Learning (ML), Natural Language Processing (NLP), and Large Language Models (LLMs) to revolutionize sentiment analysis in healthcare discussions. While traditional ML algorithms have shown promise in structured data analysis, NLP and LLMs like BERT offer an unprecedented ability to understand the nuances of human language.

These technologies not only facilitate text pre-processing and feature extraction but also help in semantic comprehension of the text, making them ideally suited for the task at hand.

Real-world Applicability and Ethical Considerations

Participation in the Social Media Mining for Health 2023 (#SMM4H) [1] competition served as both an impetus and a validation for the research direction of this thesis. The competition's specific task—Multi-class classification of sentiment associated with therapies in English tweets—offers not just an academic challenge but a real-world problem that requires a multi-faceted solution. On the flip side, the ethical considerations related to data privacy, regulatory compliance, and the responsible use of AI technologies have also been thoroughly addressed in compliance with established guidelines.

Methodological Choices and Implementation

To tackle this complex problem, I employed a two-pronged methodological approach that combines traditional ML algorithms and NLP techniques with advanced LLMs. The techniques were implemented using Python in the PyCharm IDE and were supplemented by Jupyter Notebooks, which provided a flexible environment for data manipulation and model training. A dataset comprising 5000 manually annotated tweets discussing a range of health conditions provided the empirical foundation for the study.

The subsequent sections of this paper will delve into the detailed problem statement and the developed solution, review relevant previous works, outline the methodologies adopted, discuss the results, and conclude with implications and future directions. The overarching aim is not just to propose a scalable solution for sentiment analysis in healthcare but also to contribute to the growing body of interdisciplinary research at the intersection of computer science, linguistics, and healthcare.

1.1. Motivation

In today's digital age, social media platforms have become ubiquitous channels for communication, information sharing, and public discourse. Among the myriad topics discussed, healthcare and therapies for various conditions have emerged as subjects of significant interest. Twitter, a platform known for its real-time updates and diverse user base, serves as a particularly rich source of public sentiment on healthcare therapies. These discussions often go beyond mere opinions; they encapsulate the lived experiences of individuals undergoing treatments for conditions such as chronic pain, substance use disorder, and long-COVID, among others.

Understanding the sentiment surrounding these therapies is not just an academic exercise; it has real-world implications. Healthcare providers can gain insights into patient satisfaction and potential side effects, policymakers can make informed decisions, and researchers can identify gaps in current medical practices. However, manually sifting through the vast amount of data to gauge public sentiment is impractical, thereby necessitating automated solutions. This sets the stage for the critical need for machine learning and natural language processing techniques to analyze and interpret this data effectively.

I chose this topic as my bachelor's thesis with the intent to delve deeply into the domain of machine learning, natural language processing, sentiment classification in text, and large language models. When I stumbled upon the task of the Social Media Mining for Health 2023 (#SMM4H) [1] competition—*Multi-class classification of sentiment associated with therapies in English tweets*—I recognized it as an optimal avenue for synthesizing my academic interests. The task's real-world applicability and technical complexity presented a compelling challenge that aligned perfectly with my educational objectives.

1.2. Objective

The primary objective of this bachelor's thesis is to develop a robust, automated system capable of classifying the sentiment associated with various therapies as discussed in English tweets. The system aims to categorize these sentiments into one of three classes: positive, negative, or neutral. To achieve this, the study leverages cutting-edge techniques in Machine Learning (ML), Natural Language Processing (NLP), and Large Language Models (LLM) like BERT (Bidirectional Encoder Representations from Transformers).

The chosen task derives its dataset from real Twitter conversations, thereby enriching the complexity and applicability of the study.

The dataset for this study comprises 5000 manually annotated tweets with approximately 20% positive, 14% negative, and 66% neutral sentiments. These tweets emanate from pre-identified cohorts discussing a range of health conditions, such as chronic pain, substance use disorder, migraine, chronic stress, long-COVID, and intimate partner violence.

To fulfill the objective, two distinct methodological approaches were employed. The first one amalgamates traditional machine learning algorithms with NLP techniques for text pre-processing, feature extraction and model training. The second strategy leverages large language models, specifically BERT, to enhance the semantic understanding of the text. Both approaches were developed using Python on the PyCharm IDE, supplemented by Jupyter Notebooks for interactive data manipulation and model training.

1.3. Regulatory Framework

Given that the study involves the collection and analysis of user-generated content from social media platforms, it is imperative to address the ethical and regulatory considerations associated with such research. The dataset used in this study has been manually annotated and does not contain personally identifiable information (PII). However, the ethical implications of using public data for research purposes cannot be overlooked.

Firstly, the study adheres to the guidelines set forth by the Institutional Review Board (IRB) for ethical research involving human subjects. Secondly, the research complies with the General Data Protection Regulation (GDPR) to ensure data privacy and security. Lastly, the study also considers the ethical guidelines provided by Twitter for academic research, ensuring that the data scraping methods employed are in compliance with Twitter's Developer Agreement and Policy.

By adhering to these regulatory frameworks, the study aims to conduct responsible research that respects user privacy and data security while contributing valuable insights to the field of healthcare sentiment analysis.

2. SOCIOECONOMIC CONTEXT

2.1. Analysis of the current situation

The Pervasiveness of Digital Transformation

In the present age, we are witnessing a massive digital transformation propelled by social media platforms, which have fundamentally altered traditional communication dynamics. This shift has been especially impactful in the healthcare sector, with democratization emerging as a notable outcome. Patients, caregivers, healthcare professionals, policy advocates, and even casual observers now have the ability to voice their opinions and share their experiences. These multifaceted contributions create a voluminous and intricate tapestry of public sentiment, illustrating both the benefits and drawbacks of therapies, medications, and treatments in real-world settings.

Notable Contributions of AI in Healthcare and Psychological Fields

AI's evolution over the past few years has been nothing short of revolutionary. In healthcare, machine learning algorithms have been applied to predictive diagnostics, treatment personalization, and drug discovery. NLP techniques are being used to analyze medical records for more accurate diagnoses, while AI-driven chatbots are assisting in mental health therapies. Furthermore, AI tools have contributed to breakthroughs in psychological research by facilitating the analysis of large datasets on human behavior, cognitive processes, and emotional states. In the case of sentiment analysis, AI techniques are actively deployed to assess patient satisfaction, predict healthcare outcomes, and monitor mental well-being. These achievements underscore the indispensability of AI and related technologies in modern healthcare.

Regulatory Horizon: Balancing Innovation and Privacy

As AI technologies become more pervasive and integral to healthcare, there's a mounting emphasis on data protection and ethical considerations. The growing discourse around regulations like the General Data Protection Regulation (GDPR) in the European Union and the Health Insurance Portability and Accountability Act (HIPAA) in the United States reflects a universal concern for data privacy. Upcoming regulatory frameworks are expected to enforce stricter measures for personal information, especially when this data can provide insights into an individual's health or psychological state.

Intersecting My Bachelor's Thesis with Current Realities

It's against this backdrop that my thesis gains relevance and urgency. As someone who embarked on this journey with no prior grounding in AI, ML, NLP, or LLM, I had a steep learning curve ahead of me. Not only did I have to acquire a strong foundational understanding of these technologies but also grapple with the nuanced challenges specific to healthcare sentiment analysis. This endeavor seemed especially worthwhile given the existing research gap. Despite the technological advancements and their applications in healthcare, the multi-class sentiment analysis related to therapies remains largely unexplored.

Thus, the scope of this thesis aligns perfectly with the current trends and needs in the field of AI and healthcare. While AI's general contributions to healthcare and psychology are increasingly being acknowledged and lauded, the importance of understanding public sentiment towards healthcare therapies is gradually becoming evident. Adding to this is the pressure of future regulatory requirements around data privacy, making it imperative to develop solutions that are not only effective but also ethical and compliant. This thesis serves as an attempt to contribute to filling these gaps, demonstrating that meaningful research can arise even from a standpoint of initial unfamiliarity with the subject matter, provided that the approach is methodical, the planning is meticulous, and the motivation is high.

2.2. Budget and planification

2.2.1. Budget

In this section, we delineate the financial resources required to successfully execute the project, as outlined in the project planning documentation. The project's duration is set at eight months, from February to September 2023, both included, and the budget has been calculated to cover expenses for this specific timeframe. The total project budget can be seen in table 2.1

Direct Costs:

1. **Human Resources**: The project team comprises two key personnel—a Junior Data Scientist (Daniel Jiménez Campos) and a Senior Project Manager (Paloma Martínez Fernández). Their salaries were decided according to the market value.
2. **IT Infrastructure**: The acquisition of a high-performance laptop, one for each member of the team. The model provided to the team members is a HP Laptop - 15-dw2003ns, priced at €826. It is powered by an Intel Core i7-1065G7 processor, comes with 8 GB of DDR4-2666 SDRAM, and offers a 512 GB PCIe NVMe M.2 SSD.
3. **Software Licenses**: While the primary software tools utilized in this project—PyCharm IDE and Jupyter Notebooks—are open-source and do not incur additional costs, it's

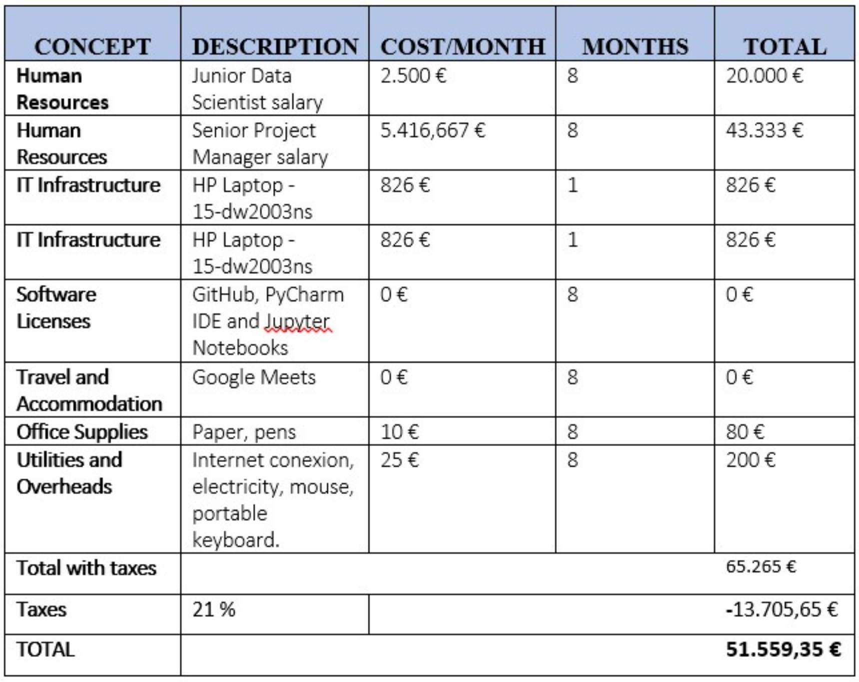
worth noting that the value they add to the project is substantial.

Indirect Costs:

1. **Travel and Accommodation**: Given that all of the meetings took place online through Google Meets, and that the team followed a work-from-home policy, no budget was required for this section.
2. **Office Supplies**: Consumables such as paper, pens, and other stationery items are included under indirect costs.
3. **Utilities and Overheads**: Costs related to electricity, internet connection, and other utilities have been factored into the budget.

Taxes:

The Value Added Tax (VAT) applicable in Spain, which is 21%, has been incorporated into the final budget calculation to ensure compliance with local tax regulations.



CONCEPT	DESCRIPTION	COST/MONTH	MONTHS	TOTAL
Human Resources	Junior Data Scientist salary	2.500 €	8	20.000 €
Human Resources	Senior Project Manager salary	5.416,667 €	8	43.333 €
IT Infrastructure	HP Laptop - 15-dw2003ns	826 €	1	826 €
IT Infrastructure	HP Laptop - 15-dw2003ns	826 €	1	826 €
Software Licenses	GitHub, PyCharm IDE and Jupyter Notebooks	0 €	8	0 €
Travel and Accommodation	Google Meets	0 €	8	0 €
Office Supplies	Paper, pens	10 €	8	80 €
Utilities and Overheads	Internet conexion, electricity, mouse, portable keyboard.	25 €	8	200 €
Total with taxes				65.265 €
Taxes	21 %			-13.705,65 €
TOTAL				**51.559,35 €**

Fig. 2.1. Project budget

Resource Constraints and Strategic Choices

Embarking on an academic research project is not a trivial undertaking, especially when faced with a significant budget constraint. Financial limitations, however, often breed innovation and resourcefulness. Given these fiscal constraints, the language and tools selected for this thesis needed to not only be powerful but also economically viable. Python emerged as the language of choice due to its unmatched combination of affordability and capabilities. Its extensive ecosystem, replete with open-source libraries like scikit-learn for machine learning, NLTK for natural language processing, and TensorFlow for deep learning, provided the requisite functionalities to proceed without compromising on the quality of research.

To complement Python, development environments that offer robust functionalities without incurring additional costs were essential. PyCharm Community Edition was utilized for its excellent code-completion features, debugging capabilities, and comprehensive project management tools. Jupyter Notebooks were invaluable for exploratory data analysis, visualization, and testing various algorithms in a segmented, interactive manner.

2.2.2. Planification

1. **Phase 1: Initiation and Skill Acquisition (February 2023)**
   The research journey commenced with rigorous consultations with my academic advisor to define the scope, significance, and feasibility of the project. Concurrently, a self-learning regimen was established. I utilized platforms such as LinkedIn Learning to take courses on Python, Machine Learning, NLP, and other foundational skills, thus ensuring a more rounded understanding of the technologies to be implemented.

2. **Phase 2: Initial Development (March - April 2023)**
   Armed with a foundational grasp of relevant skills, the next two months were dedicated to developing the first prototype of the solution. This involved implementing traditional machine learning algorithms and NLP techniques, such as sentiment analysis and text classification. This version served as the starting point for future refinements and also as a benchmark to evaluate the efficacy of more complex models to be developed later.

3. **Phase 3: Iterative Refinement (May - June 2023)**
   In this phase, the initial model underwent a series of evaluations to assess its performance and reliability. Feedback loops were established, involving retesting and fine-tuning of algorithms based on real-world data. This iterative process aimed at optimizing the model for better accuracy and efficiency.

4. **Phase 4: Integration of Advanced Techniques (July - August 2023)**
   With foundational and intermediate objectives met, this period was dedicated to the exploration and implementation of Large Language Models. These advanced algorithms were expected to provide more nuanced, context-aware classifications, thus elevating the project's overall capabilities.

5. **Phase 5: Final Stretch (August - September 2023)**
   The concluding phase was a synthesis of all prior work. Final evaluations were performed, followed by comparative analyses between the traditional and LLM-based approaches. This culminated in the drafting and finalization of the thesis paper, encapsulating the entire research journey.

*A Gantt diagram, showed in Figure 8.1, for the thesis cronogram, can be found in the annex of this paper.

3. RELATED AND PREVIOUS WORKS

Some previous, related works were explore to gain knowledge and ideas for the thesis development.

3.1. Natural Language Processing applied to text classification

Natural Language Processing (NLP) serves as the cornerstone of text classification, and within this domain, the traditional Term Frequency-Inverse Document Frequency (TF-IDF) mechanism has been widely adopted for feature extraction. In essence, TF-IDF is a statistical measure that evaluates the importance of a term within a specific document relative to a corpus of documents. The TF part counts the frequency of a term in a document, while the IDF part penalizes terms that are too frequent across multiple documents, thus filtering out common but uninformative terms like "the," "is," and "and."

In the landmark work by Joachims in 1998 [2], TF-IDF was employed in conjunction with Support Vector Machines (SVM). The SVM algorithm functions by identifying the hyperplane that best separates data points belonging to different classes in a high-dimensional feature space. Joachims demonstrated that using TF-IDF for feature extraction facilitated the SVM in generating highly discriminative hyperplanes, achieving superior text classification results.

Although TF-IDF and SVM combinations have been effective, these models are limited in their ability to grasp semantic relations between words. For example, they struggle to understand that the words "good" and "excellent" convey similar sentiments.

Recent advancements have introduced vector space models such as Word2Vec [3] and GloVe [4]. Developed using neural network architectures, these models generate word embeddings: multi-dimensional vectors that capture not just frequency but also context and semantics. Word2Vec, for instance, can recognize synonyms, antonyms, and even conduct word analogies. GloVe takes a more statistical approach, leveraging co-occurrence matrices to encapsulate semantic relations. These embeddings provide a more nuanced feature set for text classification, making them preferable over TF-IDF for certain applications.

3.2. Machine Learning Techniques

Naive Bayes and Decision Trees

In the realm of machine learning, Naive Bayes and Decision Trees were early favorites for text classification and sentiment analysis. Despite its naive assumption of feature independence, Naive Bayes has shown surprising effectiveness. It applies Bayes' theorem to predict the probability of a particular class given a set of features. Its simplicity and efficiency make it a popular choice for baseline models.

Decision Trees operate by recursively partitioning the data space into homogeneous regions, essentially making decisions by asking a series of questions. These models are highly interpretable but often prone to overfitting.

Logistic Regression

Wang and Manning (2012) [5] expanded on the traditional approaches by demonstrating the applicability of logistic regression in text classification. Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities. It's particularly powerful when the feature space is high-dimensional, as often is the case with text data.

Ensemble Methods and Gradient Boosting

Recent advancements by Li et al. (2017) [6] highlighted the effectiveness of ensemble methods like Gradient Boosting. These ensemble methods amalgamate predictions from multiple models to improve generalization and robustness. Gradient Boosting, in particular, iteratively corrects the errors of weak classifiers, producing a strong classifier that minimizes overfitting.

3.3. Specific sentiment classification techniques

Lexicon-based Methods

Sentiment classification has its unique challenges, and one of the early methodologies employed lexicon-based techniques. Pre-compiled lexicons containing words annotated with sentiment scores were used. The sentiment score of a text would be the aggregate score of its words, offering a simplistic yet effective approach.

Aspect-Based Sentiment Analysis

Recent approaches like aspect-based sentiment analysis are more nuanced [7]. These techniques identify specific entities or aspects mentioned in the text, like "battery life" or "customer service," and then determine the sentiment related to each. This yields a more granular sentiment analysis and is beneficial for applications like product reviews.

3.4. Application of Large Language Models

BERT and Fine-tuning

The advent of Large Language Models (LLMs), like BERT [8], has significantly impacted the NLP landscape. Unlike traditional models, BERT is pre-trained on massive datasets, capturing complex semantic relationships. It's based on the Transformer architecture, which enables the model to understand the context and semantics deeply. Sun et al. (2019) [9] showed that fine-tuning BERT for specific tasks can achieve state-of-the-art results in sentiment analysis.

RoBERTa: An Improved BERT

RoBERTa [10], a robustly optimized version of BERT, pushes the boundary further. It adapts the pre-training approach by eliminating certain next-sentence prediction tasks and training on an even larger dataset. RoBERTa has been shown to outperform BERT in a range of NLP tasks, including but not limited to sentiment classification.

By reviewing these existing works, this study aims to situate itself within the broader context of text classification and sentiment analysis, drawing upon the advancements in NLP, ML, and LLMs to address the specific challenges posed by the multi-class classification of sentiments associated with therapies in English tweets.

4. SOLUTION PROPOSAL

4.1. Problem Description

The primary focus of this thesis is to address the challenge of automatically classifying the sentiment associated with therapies as discussed in English tweets. The problem is both complex and nuanced, given the diversity of therapies discussed, the varied experiences of individuals, and the intricacies of human language, especially as it appears in social media. The task involves categorizing each tweet into one of three sentiment classes: positive, negative, or neutral.

4.1.1. Shared task - Social Media Mining for Health 2023 (#SMM4H)

Multi-class classification of sentiment associated with therapies in English tweets

This study is aligned with the shared task for the Social Media Mining for Health Applications (SMM4H) 2023 competition.

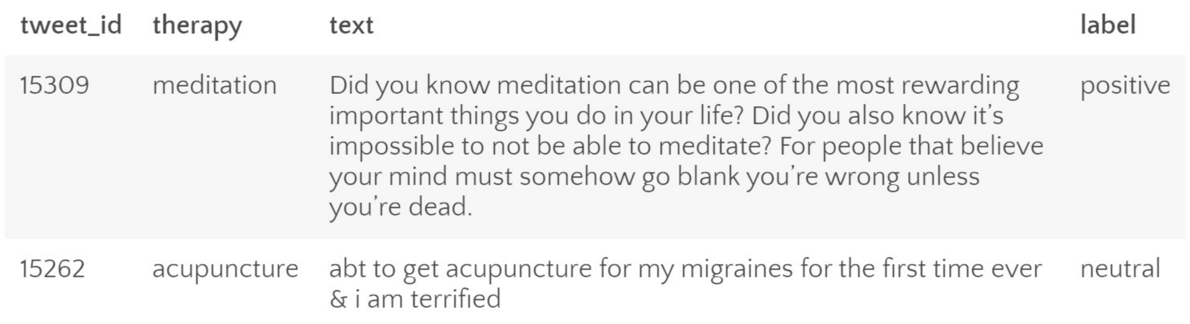
Explicit task description from SMM4H 2023 website [1]:

"*Task 2 – Multi-class classification of sentiment associated with therapies in English tweets There is an abundance of health-related data on social networks, including chatter about therapies for health conditions. These therapies include but are not limited to medication, behavioral, and physical therapies. Social media subscribers who discuss such therapies often express their sentiments associated with the therapies. In this task, the focus will be to build a system that can automatically classify the sentiment associated with a therapy into one of three classes—positive, negative, and neutral. The annotated dataset for this task has been drawn from multiple preidentified Twitter cohorts (chronic pain, substance use disorder, migraine, chronic stress, long-COVID, and intimate partner violence). Thus, there is a high possibility that the therapies are being mentioned by people who are actually receiving/consuming them. The dataset consists of 5000 English Tweets containing mentions of a variety of therapies manually labeled as positive, negative, or neutral with the following approximate distribution: 20%, 14%, and 66%, respectively. The evaluation metric for this task is the micro-averaged F1-score over all 3 classes. The data include annotated collections of posts on Twitter which will be shared in csv files. There are 4 fields in the csv files: tweet_id, therapy, text, label. The training data is already prepared and will be available to the teams registering to participate. The testing data will be released when the evaluation phase starts.*

- *Training Data: 3009 tweets*
- *Validation Data: 753 tweets*

- *Testing Data: TBA*
- *Evaluation Metric: micro-averaged F1-score*

Data Examples:

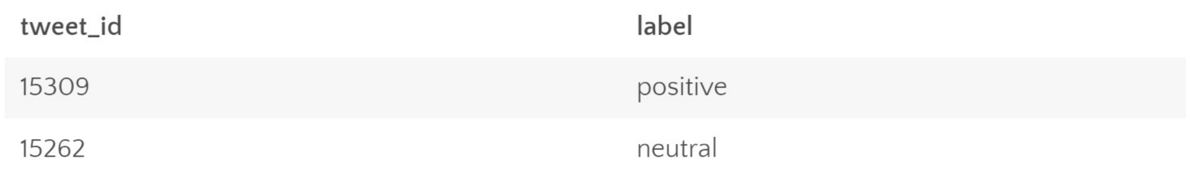


tweet_id	therapy	text	label
15309	meditation	Did you know meditation can be one of the most rewarding important things you do in your life? Did you also know it's impossible to not be able to meditate? For people that believe your mind must somehow go blank you're wrong unless you're dead.	positive
15262	acupuncture	abt to get acupuncture for my migraines for the first time ever & i am terrified	neutral

Fig. 4.1. #SMM4H official page: Data Examples of the provided dataset

Submission Format:

Please use the format below for submission. Submissions should contain tweet_id and label separated by tabspace in the same order as below.



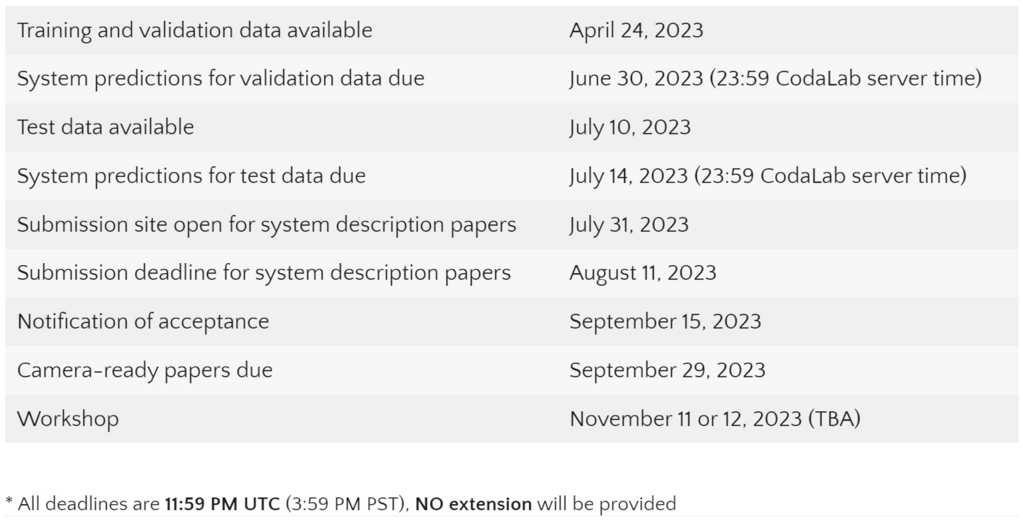
tweet_id	label
15309	positive
15262	neutral

Fig. 4.2. #SMM4H official page: Submission format example

The unzipped submission data needs to be named as "answer.txt" and be zipped."

Task schedule and deadlines

As illustrated in Figure 4.3, the task schedule follows a clear cronogram. These schedule heavily determined the development timeline of the thesis, as we didn't have access to the corresponding datasets until the scheduled date.



Training and validation data available	April 24, 2023
System predictions for validation data due	June 30, 2023 (23:59 CodaLab server time)
Test data available	July 10, 2023
System predictions for test data due	July 14, 2023 (23:59 CodaLab server time)
Submission site open for system description papers	July 31, 2023
Submission deadline for system description papers	August 11, 2023
Notification of acceptance	September 15, 2023
Camera-ready papers due	September 29, 2023
Workshop	November 11 or 12, 2023 (TBA)

* All deadlines are **11:59 PM UTC** (3:59 PM PST), **NO extension** will be provided

Fig. 4.3. #SMM4H official page: Task schedule and deadlines

4.1.2. Functional Requirements

- **Data Collection**: The system must be capable of ingesting the provided dataset in CSV format, which includes fields for tweet_id, therapy, text, and label.
- **Data Preprocessing**: The system should implement text cleaning, tokenization, and other NLP preprocessing steps to prepare the data for analysis.
- **Feature Extraction**: The system must support various feature extraction techniques, including TF-IDF, word embeddings, and others suitable for text data.
- **Model Training**: The system should be able to train machine learning models using algorithms like Naive Bayes, Decision Trees, Random Forests, and Support Vector Machines for the first approach. For the second approach, it should be capable of fine-tuning a pre-trained Large Language Model like BERT.
- **Model Evaluation**: The system must evaluate the performance of the trained models using the micro-averaged F1-score as the primary metric.
- **Model Comparison**: The system should provide functionalities to compare the performance of the two approaches based on the evaluation metrics.
- **User Interface**: While not a primary focus, a basic user interface for model training and evaluation would be beneficial.
- **Export Results**: The system should allow for the export of evaluation results and trained models for further analysis.

- **Logging and Monitoring**: The system should maintain logs of the training and evaluation processes for debugging and performance tuning.

4.1.3. Non-Functional Requirements

- **Scalability**: The system should be designed to handle larger datasets efficiently, without a significant degradation in performance.
- **Modularity**: The code should be modular and well-organized, adhering to best practices to ensure maintainability and extensibility.
- **Documentation**: Comprehensive documentation should be provided for all code and functionalities, making it easier for future researchers to understand and extend the work.
- **Performance**: The system should be optimized for high performance, particularly in terms of the time required for training and evaluation.
- **Resource Efficiency**: Given the budget constraint, the system should be optimized to run on modest hardware without requiring specialized computing resources like GPUs.
- **Security**: While this may not be a primary concern given the academic nature of the project, basic security measures should be in place to protect the integrity of the data and the system.
- **Interoperability**: The system should be designed in a way that allows it to integrate easily with other tools and platforms, especially those commonly used in data science and machine learning.
- **Reproducibility**: All steps from data preprocessing to model training and evaluation should be replicable to ensure the validity of the research findings.
- **Ethical Compliance**: The system should adhere to ethical guidelines, particularly in terms of data privacy and user consent, aligning with the regulatory framework discussed earlier.

By addressing these functional and non-functional requirements, this study aims to develop a robust and comprehensive solution for the multi-class classification of sentiments associated with therapies in English tweets.

4.2. Description of the share task dataset

4.2.1. Dataset overview and description

Dataset Description

The dataset serves as the foundation for Task 2, which is focused on the multi-class classification of sentiment associated with various therapies in English tweets. Given the pervasive nature of health-related discussions on social media platforms, particularly Twitter, this dataset aims to capture the sentiment of users toward different therapies. These therapies range widely and include medication, behavioral therapies, and physical therapies, among others.

Data Collection

The dataset is composed of tweets that have been manually labeled and categorized. It was drawn from multiple pre-identified Twitter cohorts that include various conditions such as chronic pain, substance use disorder, migraine, chronic stress, long-COVID, and intimate partner violence. This increases the likelihood that the therapies mentioned in the dataset are actually being consumed or experienced by the users, thereby adding a layer of authenticity and gravity to the data.

Data Structure

The dataset is provided in a comma-separated values (CSV) format and consists of four key fields:

- **tweet_id**: A unique identifier for each tweet.
- **therapy**: The type of therapy being discussed in the tweet.
- **text**: The textual content of the tweet.
- **label**: The manually annotated sentiment associated with the tweet, categorized into one of three classes—positive, negative, or neutral.

Data Splits

The dataset is divided into three main splits for the purpose of training and evaluation:

- **Training Data**: Comprising 3009 tweets, this portion is intended for training the machine learning models.
- **Validation Data**: Contains 753 tweets, serving as an intermediary tested for model tuning.

- **Testing Data**: Contains 5000 tweets, and this set will be used for the final evaluation of the models.

Class Distribution

The sentiments in the dataset are not evenly distributed. The approximate distribution of the classes is as follows:

- Positive: 20%
- Negative: 14%
- Neutral: 66%

Evaluation Metric

The performance of models trained on this dataset will be evaluated using the micro-averaged F1-score over all three sentiment classes. This metric provides a balanced measure of a model's accuracy, taking both precision and recall into account.

Insights and Challenges

The dataset presents several intriguing opportunities and challenges:

- **Real-world Applicability**: Given that the dataset is drawn from genuine user interactions, models trained on it have high real-world applicability.
- **Class Imbalance**: The skewed distribution of classes poses a challenge for classification. Special techniques like over-sampling the minority class or using class-weighted loss functions may be required.
- **Variety of Therapies**: The inclusion of diverse therapies increases the complexity but also enriches the dataset, making the task more encompassing and robust.

4.2.2. Exploratory data analysis on the dataset

Exploratory Data Analysis (EDA) is an indispensable step in the data science pipeline, particularly in the context of Natural Language Processing (NLP) and sentiment analysis. It serves as the initial investigation phase where raw data is transformed into a format that can be more easily interpreted. EDA employs a variety of statistical graphics, plots, and information tables to understand the nature and structure of the data. The primary aim is to uncover underlying patterns, identify anomalies, test hypotheses, and check assumptions with the help of summary statistics and graphical representations.

Exploratory Data Analysis (EDA) serves as the backbone of any data-driven project, offering a critical framework for understanding the intricacies of the dataset at hand. In the realm of Natural Language Processing (NLP), particularly for a project focused on multi-class classification of sentiments associated with therapies in English tweets, EDA is indispensable. It provides the first lens through which the raw data can be viewed, understood, and interpreted.

The necessity of EDA is manifold. First, it helps in identifying the structure, trends, and irregularities in the data. This is crucial for feature engineering, as the choice of features can significantly impact the performance of Machine Learning (ML) models. Second, EDA assists in spotting anomalies and outliers that could skew the results. In the context of sentiment analysis, this could mean identifying tweets that are either too long or too short, or those that use language differently. Third, EDA provides insights into the distribution of classes (sentiments, in this case), which is vital for understanding the balance or imbalance in the dataset. An imbalanced dataset could lead to biased models, and EDA helps in making informed decisions about whether resampling methods are required to balance the data.

In the realm of sentiment analysis, especially when dealing with unstructured data like tweets, EDA becomes even more crucial. Tweets are inherently noisy and diverse, containing slang, hashtags, mentions, and emoticons. Understanding the distribution of sentiments, the frequency of specific words or phrases, and the general tone of the text can provide invaluable insights. These insights not only inform the feature engineering stage but also guide the selection of appropriate machine learning models and evaluation metrics.

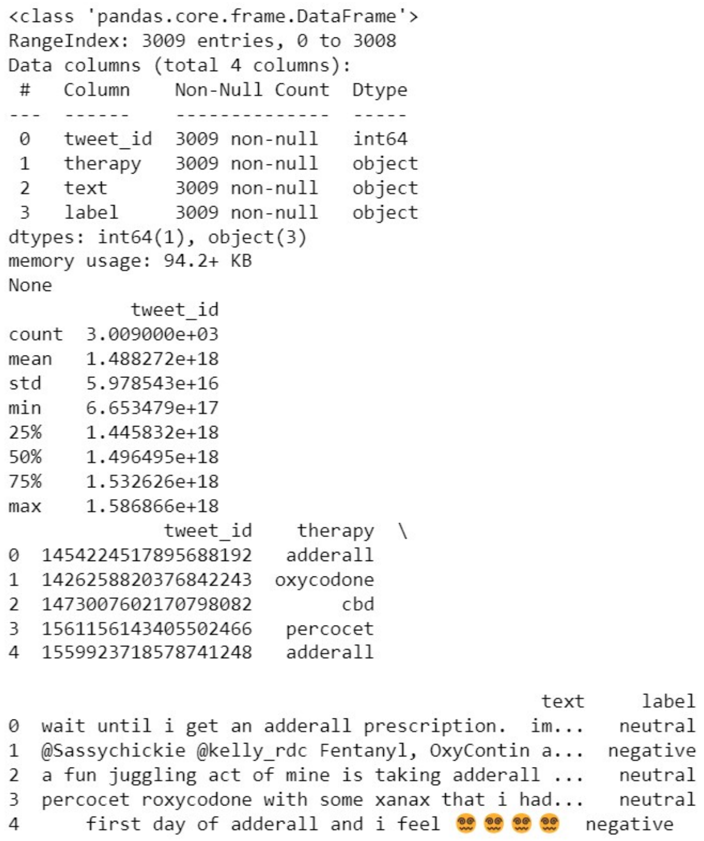
The exploratory data analysis on the dataset is designed to delve deeper into the dataset of English tweets concerning various therapies. This step of the pipeline aims to provide a comprehensive understanding of the data's characteristics, which will be pivotal for the subsequent stages of this project, such as feature selection, model training, and evaluation. It focuses on several key aspects, including but not limited to, the distribution of sentiment labels, the length of the tweets, the frequency of specific therapies mentioned, and the polarity of sentiments. Each of these aspects serves a specific purpose:

- **Understanding Label Distribution**: Helps in identifying any class imbalance, which is vital for model training and evaluation.
- **Analyzing Text Length**: Aids in making informed decisions about text preprocessing techniques, such as lemmatization, stemming, truncation or padding.
- **Examining Frequency of Therapies**: Offers insights into the dataset's focus and diversity, which could be of interest to healthcare stakeholders.
- **Investigating Sentiment Polarity**: Provides a nuanced understanding of public opinion, which is often more complex than simple categorical labels like 'positive', 'neutral', or 'negative'.

The Exploratory data analysis performed on the dataset is explained below, splitted into the different steps:

1. Loading and Basic Overview

In this initial section, the dataset is loaded into a Pandas DataFrame, and basic summary statistics are generated. The .info() and .describe() methods are used to provide a snapshot of the dataset's structure, including the number of entries, columns, data types, and basic statistical measures like mean, median, and standard deviation for numerical columns. This Figure 4.4 shows relevant information obtained with the previous methods.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3009 entries, 0 to 3008
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   tweet_id  3009 non-null   int64
 1   therapy   3009 non-null   object
 2   text      3009 non-null   object
 3   label     3009 non-null   object
dtypes: int64(1), object(3)
memory usage: 94.2+ KB
None
           tweet_id
count  3.009000e+03
mean   1.488272e+18
std    5.978543e+16
min    6.653479e+17
25%    1.445832e+18
50%    1.496495e+18
75%    1.532626e+18
max    1.586866e+18
              tweet_id    therapy  \
0  1454224517895688192   adderall
1  1426258820376842243  oxycodone
2  1473007602170798082        cbd
3  1561156143405502466   percocet
4  1559923718578741248   adderall

                                                text     label
0  wait until i get an adderall prescription.  im...   neutral
1  @Sassychickie @kelly_rdc Fentanyl, OxyContin a...  negative
2  a fun juggling act of mine is taking adderall ...   neutral
3  percocet roxycodone with some xanax that i had...   neutral
4      first day of adderall and i feel 😵‍💫😵‍💫😵‍💫😵‍💫   negative
```

Fig. 4.4. EDA: Basic overview of the dataset

This step is crucial for several reasons. First, it helps to confirm that the dataset has been loaded correctly. Second, the summary statistics provide an initial understanding of the data's scale and distribution, which can inform later decisions on data preprocessing and feature engineering.

A check for missing values is performed, and it is found that there are no missing values in the dataset. This simplifies the data cleaning process.

2. Label Distribution

This section focuses on visualizing the distribution of target labels, which in this case are the sentiments associated with the tweets. A count plot is generated, as showed in Figure 4.5, to show the frequency of each sentiment label.

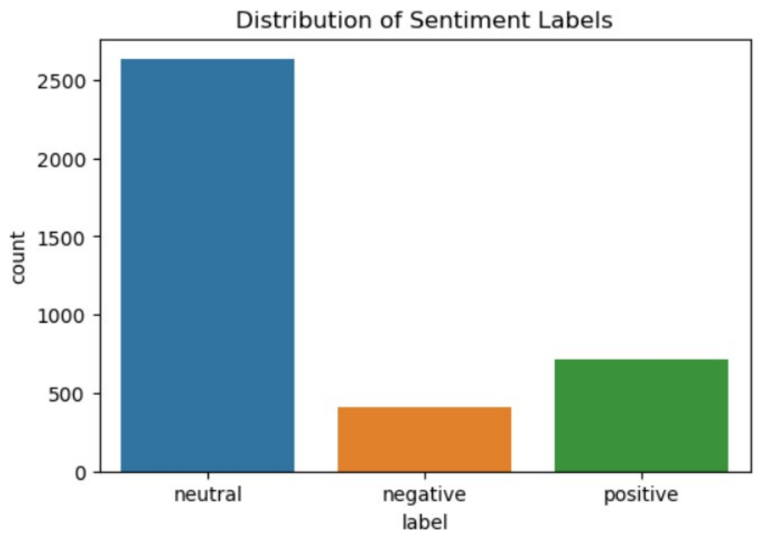


Fig. 4.5. EDA: Distribution of the label (sentiment) across the dataset

Understanding the distribution of labels is essential for identifying class imbalances. If one class is significantly underrepresented, it may lead to a biased model. This insight informs the choice of resampling methods and evaluation metrics in later stages.

3. Text Length Analysis

Here, the length of each tweet is calculated and its distribution is plotted, as showed in Figure 4.6. This provides insights into the range and common lengths of the text data.

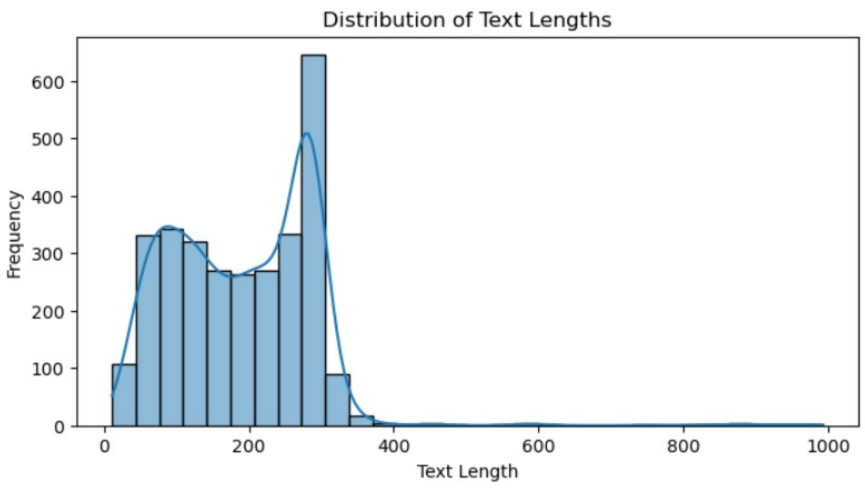


Fig. 4.6. EDA: Text length of the tweets in the dataset

Knowing the distribution of text lengths is important for text preprocessing. Decisions regarding truncation, padding, or even the type of text embedding can be informed by this analysis.

4. Therapy Mention Analysis

This section analyzes the frequency of different therapies mentioned in the dataset. A bar plot of the top 10 most frequently mentioned therapies is generated, and displayed in Figure 4.7.

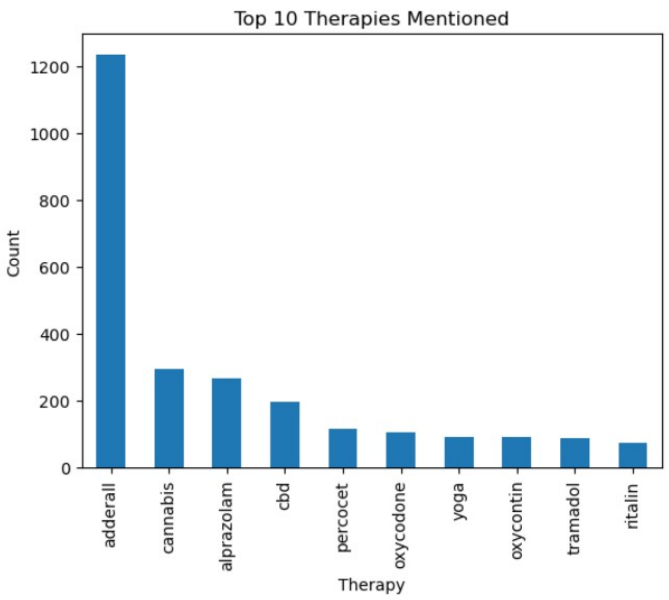


Fig. 4.7. EDA: Count of the number of mentions of each therapy

Understanding which therapies are most talked about can provide valuable insights into public interest and the dataset's focus. This could be particularly useful for stakeholders in the healthcare industry.

Another count plot, illustrated in Figure 4.8, is used to visualize the distribution of different therapies mentioned in the dataset. This helps in understanding the focus of the dataset and whether it is biased towards specific therapies.

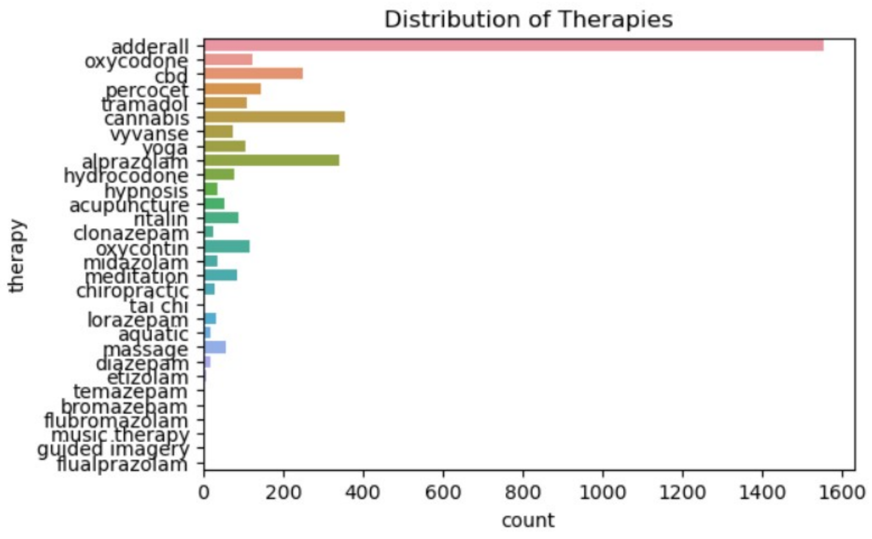


Fig. 4.8. EDA: Distribution of therapies

5. Word Cloud for Sentiments

As this code snippet 4.1 shows, word clouds are generated for each sentiment class to visually represent the most common words associated with each sentiment. The results can be viewed in Figures 4.9, 4.10 and 4.11

CODE SNIPPET 4.1. Overview of the word clouds, splitted by sentiment

```
from wordcloud import WordCloud

def generate_wordcloud(text, title):
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(
        text)
    plt.figure(figsize=(6, 3))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(title)
    plt.axis('off')
    plt.show()

# Generating word clouds for each sentiment
for sentiment in train_df['label'].unique():
    text = ' '.join(train_df[train_df['label'] == sentiment]['text'])
    generate_wordcloud(text, f'Word Cloud for {sentiment.capitalize()} Sentiment')
```

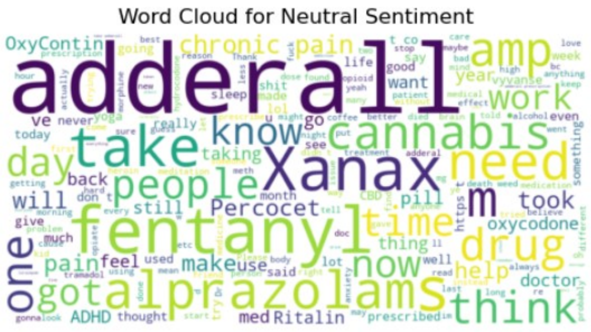
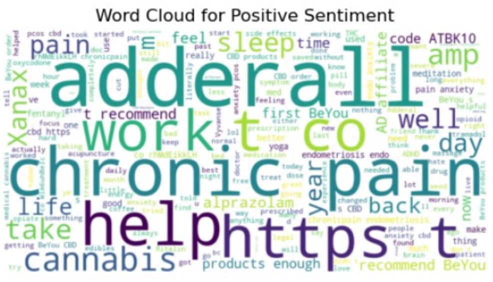


Fig. 4.9. EDA: Word Cloud for Neutral Sentiments



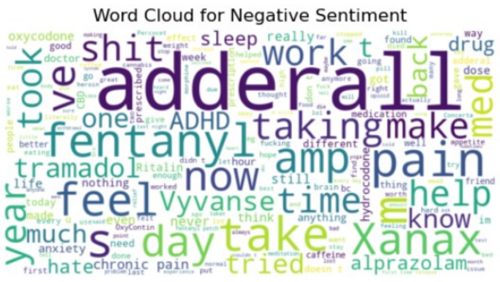


Fig. 4.10. EDA: Word Cloud for Positive Sentiments

Fig. 4.11. EDA: Word Cloud for Negative Sentiments

Word clouds offer a quick way to understand the most prominent terms in each sentiment class, which can be useful for feature selection and understanding the general tone of each sentiment category.

6. Sentiment Distribution Across Therapies

This section explores how sentiments are distributed across different therapies. A count plot is generated to show the distribution of sentiments for the top 10 most mentioned therapies, as illustrated in Figure 4.12.

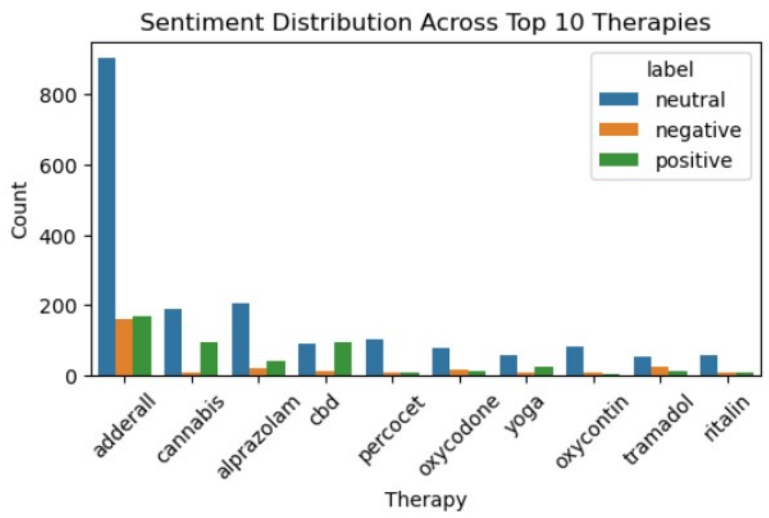


Fig. 4.12. EDA: Sentiment Distribution Across Therapies

This analysis can provide insights into public perception and opinions about different therapies, which could be valuable for healthcare providers and policymakers.

7. N-gram Analysis

The most common bigrams (two-word sequences) and trigrams (three-word sequences) in the dataset are identified and plotted. This plots were generated, as illustrated in this code 4.2, and can be viewed in the figures 4.13 and 4.14.

CODE SNIPPET 4.2. 2-Gram and 3-Gram distribution

```
from sklearn.feature_extraction.text import CountVectorizer

def plot_top_ngrams(text, n=2, top=10):
    vectorizer = CountVectorizer(ngram_range=(n, n))
    ngrams = vectorizer.fit_transform(text)
    ngram_freq = ngrams.sum(axis=0)
    vocab = [(word, ngram_freq[0, idx]) for word, idx in vectorizer.vocabulary_.items
        ()]
    vocab = sorted(vocab, key=lambda x: x[1], reverse=True)[:top]
    words, frequencies = zip(*vocab)
    plt.bar(words, frequencies)
    plt.title(f'Top {top} {n}-grams')
    plt.xlabel('N-grams')
    plt.ylabel('Frequency')
    plt.xticks(rotation=45)
    plt.show()

# Plotting top bigrams and trigrams
plot_top_ngrams(train_df['text'], n=2)
plot_top_ngrams(train_df['text'], n=3)
```

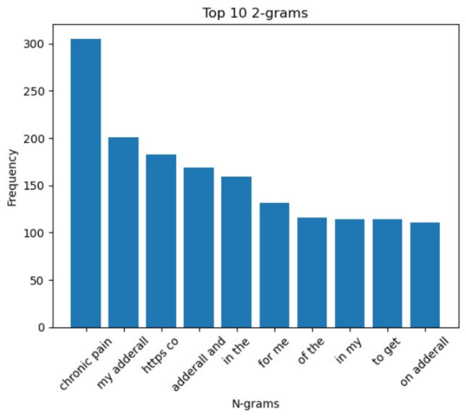


Fig. 4.13. EDA: The most common bigrams

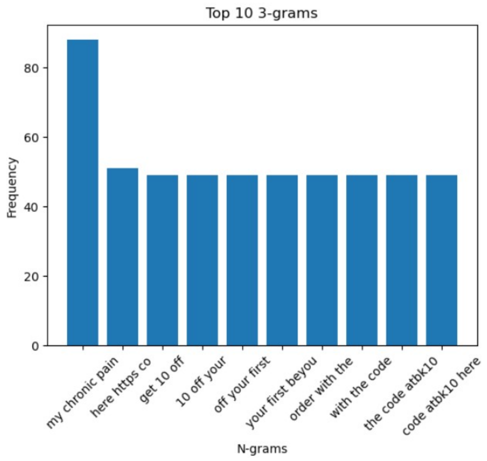


Fig. 4.14. EDA: The most common trigrams

Understanding the most frequent phrases can help in feature engineering and in fine-tuning the text preprocessing steps. It can also provide insights into common public expressions and opinions.

8. Sentiment Polarity Analysis

The sentiment polarity of each tweet is calculated using TextBlob, and its distribution is plotted in figure 4.15. This provides a continuous scale to understand the sentiments, as opposed to the categorical labels.

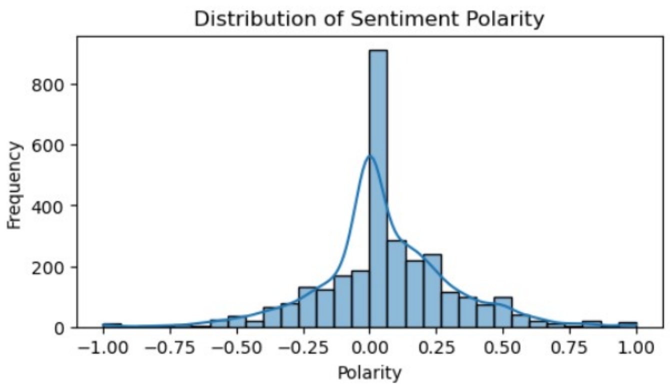


Fig. 4.15. EDA: Sentiment polarity distribution

Sentiment polarity can offer a more nuanced understanding of public sentiment. It can also serve as an additional feature for the machine learning model, potentially improving its performance.

4.2.3. Conclusions and Insights from EDA

After conducting a comprehensive Exploratory Data Analysis on the dataset of English tweets concerning various therapies, several key insights and conclusions have emerged that will significantly inform the subsequent stages of this project. Below are the detailed findings:

Data Structure and Integrity

The initial data overview (watch figure 4.4) confirmed that the dataset is well-structured, with no missing values in the crucial columns. This is a positive start as it minimizes the need for data imputation, thereby preserving the originality of the dataset for more accurate model training.

Class Distribution

One of the most critical findings is the distribution of sentiment labels. The dataset shows a relatively unbalanced distribution (watch figure 4.5) across different sentiment classes. This is crucial because it means that the machine learning model won't have a good basis for learning the distinguishing features of each class, as it will be biased towards a particular sentiment (neutral).

Text Length Variability

The text length analysis revealed (watch figure 4.6) that the tweets have a wide range of lengths. This is an important consideration for text preprocessing. For instance, while shorter texts might need padding, longer ones might require truncation. The variability in text length also suggests that flexible NLP models capable of handling different text lengths, such as RNNs or Transformers, might be more suitable for this project.

Therapy Focus and Diversity

The therapy mention analysis provided valuable insights into the focus and diversity of the dataset. It revealed that some therapies are mentioned more frequently (specially "adderall") than others (watch figure 4.7), suggesting that the dataset has a diverse range of topics. This is beneficial for building a model that is generalizable across different therapies.

Language and Sentiment

The word clouds for different sentiment classes (watch figure 4.9) revealed the kind of language commonly used for each sentiment. For example, after removing non-sentiment associated words, positive sentiments frequently use words like "effective," "good," or "helpful," while negative sentiments use words like "ineffective," "bad," or "harmful." These insights are invaluable for feature engineering and for understanding the nuances in language that often accompany different sentiments.

Sentiment Across Therapies

The sentiment distribution across different therapies (watch figure 4.12) offered a nuanced view of public perception. For instance, some therapies have predominantly positive sentiments, while others have a mix. This could be valuable information for healthcare providers and policymakers interested in public opinion on different therapies.

Common Phrases and Expressions

The N-gram (watch figure 4.13) analysis helped in understanding the common phrases and expressions used in the tweets. Phrases like "chronic pain" or "my adderall" can carry significant sentiment weight and should be considered during the feature engineering phase.

Sentiment Polarity

The sentiment polarity analysis added another layer of complexity (watch figure 4.15) to our understanding of sentiments. While the dataset labels provide a categorical understanding (Positive, Negative, Neutral), the polarity provides a continuous measure, offering a more nuanced view. This could be particularly useful for tasks that require a more detailed understanding of sentiment, beyond basic categorization.

In summary, the EDA has provided a multi-faceted understanding of the dataset. From basic structure to complex nuances in language and sentiment, the insights gained are not just descriptive but also prescriptive for the next stages of the project. These insights will guide the data preprocessing, feature engineering, model selection, and evaluation steps, ensuring that the machine learning models built are both robust and insightful.

4.3. Ethical Considerations in Healthcare Sentiment Analysis

As the frontier of technology advances into the intricacies of human emotion and healthcare, so does the imperative to carefully consider the ethical ramifications of this research. While the primary goal of this thesis is to classify sentiments associated with healthcare therapies through the lenses of Machine Learning, Natural Language Processing, and Large Language Models, it is crucial to tread this path with an ethical compass. This section elaborates on the key ethical dimensions that this research encounters.

Data Privacy

The quintessence of any healthcare-related analysis lies in the data used, which often includes sensitive information. Given that the corpus for this research is composed of English tweets, there are implicit expectations of privacy, even when dealing with publicly available data. The solution developed in this thesis adheres to strict privacy norms, ensuring that all data is anonymized and that no personally identifiable information (PII) is used to train or test the models. Regulatory frameworks like the General Data Protection Regulation (GDPR) and Health Insurance Portability and Accountability Act (HIPAA) serve as guidelines in data handling.

Algorithmic Bias

Machine Learning algorithms are fundamentally data-driven, inheriting the biases present in their training data. Healthcare sentiment can be a reflection of social, economic, and cultural contexts that should not be ignored. In this thesis, special attention was paid to ensure that the dataset represents a diverse range of sentiments and demographics. However, it is acknowledged that eliminating all forms of bias is an ongoing challenge that requires continuous work, particularly with evolving societal norms and healthcare practices.

Implications of Misclassification

The impact of incorrect sentiment classification in healthcare is not just a statistical error but could have real-world consequences. For example, misclassifying a negative sentiment as positive could lead to incorrect evaluations of a therapy's public perception, thereby influencing healthcare decisions. The model developed in this thesis undergoes rigorous evaluations to minimize this risk, but the impossibility of achieving 100% accuracy necessitates transparency and the incorporation of a margin for error in any conclusions drawn.

Transparency and Explainability

As machine learning models, particularly Large Language Models, become more complex, their decision-making processes become less transparent. To address this, this thesis not only provides detailed performance metrics for the model but also includes an exploration of feature importance and model explainability. This contributes to making the solution less of a 'black box,' aiding healthcare professionals in trusting the model's classifications and incorporating them into broader healthcare strategies.

Social Responsibility

The democratization of healthcare knowledge through social media platforms presents both an opportunity and a responsibility. By classifying sentiments related to healthcare therapies, this research can potentially influence public perception and healthcare policy. Therefore, it is of utmost importance that the methods and findings are communicated transparently, accounting for the model's limitations and the ethical considerations mentioned herein.

Future Regulatory Measures

With the rising incorporation of AI into healthcare, regulatory scrutiny is expected to intensify. Though this project complies with current legal frameworks, it is imperative to remain vigilant and adaptable to future legislation that might impose new requirements, especially those concerning data privacy and algorithmic fairness.

The ethical framework laid out in this section is not just an addendum but a cornerstone of this research. It serves as a continuous point of reference to ensure that the science does not outpace the ethics, thereby maintaining a responsible and human-centric approach to leveraging AI in healthcare sentiment analysis.

4.4. Solution: Using traditional machine learning techniques

4.4.1. Functional Architecture and Design

In the initial stages of this project, I opted to use Jupyter notebooks as the primary development environment. The decision was driven by the need for a more interactive and visual understanding of the pipeline and the evolution of the dataset. Jupyter notebooks offer the advantage of running code in isolated cells, which allows for real-time output visualization and makes it easier to debug and understand each step of the pipeline. This visual and interactive nature of Jupyter notebooks was invaluable for tasks like Exploratory Data Analysis (EDA), where immediate feedback from visualizations could lead to more informed decisions.

Once the pipeline was fully developed and tested in the Jupyter notebook environment, I transitioned the codebase to a PyCharm project. The rationale behind this move was to leverage the more robust features of an Integrated Development Environment (IDE) like PyCharm, which offers better tools for code organization, version control, and debugging. More importantly, PyCharm allows for faster execution and scalability, essential attributes as the project moves from a development phase to a deployment phase.

Importance of Dividing the Pipeline into Steps

Dividing the pipeline into discrete, modular steps serves multiple purposes:

- **Understandability**: Breaking down the pipeline into smaller parts makes it easier to understand the flow and logic of the entire project. Each notebook or module in PyCharm focuses on a specific task, making it easier to grasp what each piece of code is designed to do.
- **Debugging and Maintenance**: When the pipeline is modular, identifying and fixing errors becomes a more straightforward task. If an issue arises in one step, it can be isolated and resolved without affecting the other parts of the pipeline.
- **Collaboration**: A modular pipeline is more accessible to collaboration. Different team members (not this case) can work on different components simultaneously without much conflict, making the development process more efficient.
- **Reusability**: Steps in the pipeline can often be reused in different projects or different parts of the same project. For example, the data cleaning or feature engineering steps could be applicable to other similar datasets or problems.
- **Scalability**: As the project grows, new steps can be added or existing steps can be modified without affecting the entire pipeline. This modularity is crucial for the scalability of the project, allowing it to adapt to new requirements or data.
- **Performance Tuning**: Dividing the pipeline into steps allows for performance metrics to be gathered for each step independently. This information is invaluable when looking to optimize the pipeline for speed and resource usage.

By initially developing in Jupyter notebooks and then transitioning to PyCharm, I was able to combine the best of both worlds: the interactive, visual benefits of notebooks for development and the robust, scalable features of an IDE for deployment. This dual approach ensured both the quality and scalability of the project.

The pipeline for this project is organized into a series of steps, each serving a specific purpose in the data science workflow. Below is an overview of each one and its role in the pipeline:

1. **Exploratory Data Analysis**
   The first step in the pipeline focuses on understanding the dataset. It involves visualizing different aspects of the data, such as the distribution of sentiment labels and therapies, to gain insights that will inform subsequent steps.

2. **Data Cleaning**
   Data cleaning is a crucial step in any data science pipeline. This section focuses on handling missing values, outliers, and any inconsistencies in the dataset to prepare it for the modeling phase.

3. **Feature Engineering**
   Feature engineering is the process of transforming raw data into a format that is better suited for modeling. This step focuses on creating new features from the existing data, such as text-based features, to improve the model's performance.

4. **Hyperparameter Tuning**
   Before training the final model, it's essential to find the best hyperparameters that yield the highest performance. This section is dedicated to hyperparameter tuning using techniques like grid search and random search.

5. **Vectorizing**
   Vectorization is the process of converting text data into numerical format. This step focuses on various techniques for vectorizing the text data, such as TF-IDF and word embeddings, to prepare it for machine learning algorithms.

6. **Final Evaluation of Model**
   The last step in the pipeline is dedicated to evaluating the performance of the final model. It uses various metrics like accuracy, recall and F1-score to assess how well the model generalizes to unseen data.

Each section in the pipeline is designed to be modular and self-contained, meaning it accomplishes a specific task in the data science workflow. This modular design makes it easier to update or modify individual components of the pipeline without affecting the others. Overall, the pipeline is structured to guide the project from initial understanding and cleaning of the data to the final evaluation of the machine learning model.

4.4.2. Programs, frameworks and libraries

Programs

- **Jupyter Notebook**

  Jupyter Notebook is an open-source web application that allows for the creation of documents that contain live code, equations, visualizations, and narrative text. It supports multiple programming languages, although Python is the most commonly used. The interface is divided into cells, which can be either code cells or markdown cells for text and equations.

  In the context of this project, Jupyter Notebook serves as the initial development environment. It is particularly useful for tasks that require iterative development and visualization, such as Exploratory Data Analysis (EDA), data cleaning, and feature engineering. The real-time output and markdown support make it ideal for documenting the research process and sharing insights.

Jupyter Notebook is where the pipeline was initially developed. Each notebook corresponds to a specific step in the pipeline, making it easier to manage and understand the project. Once the pipeline was stable and well-documented in Jupyter, the code was transitioned to PyCharm for more robust development and deployment. The notebooks are also stored in a GitHub repository for version control and future collaboration.

- **PyCharm IDE**

  PyCharm is an Integrated Development Environment (IDE) specifically designed for Python. It offers a wide range of features like code completion, error highlighting, a powerful debugger, and integration with various version control systems.

  After the initial development in Jupyter Notebook, PyCharm is used for more advanced development tasks. It was particularly useful for tasks that require deep debugging, unit testing, and project management. PyCharm also offers better tools for code organization, making it easier to manage large codebases.

  The code initially developed in Jupyter Notebooks is transitioned to PyCharm for further development and deployment. PyCharm offers the ability to execute Jupyter notebooks within the IDE, providing a seamless transition between the two environments. It is also integrated with GitHub for version control.

- **GitHub**

  GitHub is a web-based platform for version control and collaboration. It allows multiple people to work on projects simultaneously. GitHub uses Git, a distributed version control system that lets you manage and keep track of your source code history.

  GitHub serves as the version control system for this project. It is used to manage the various versions of the codebase, keep track of changes.

  GitHub is integrated with both Jupyter Notebook and PyCharm. The Jupyter Notebooks are stored in a GitHub repository, making it easier to share and collaborate. PyCharm has built-in GitHub integration, allowing for easy commits, pulls, and pushes directly from the IDE. This ensures that the codebase is always up-to-date and facilitates collaboration.

The integration between Jupyter Notebook, PyCharm, and GitHub provides a comprehensive development environment that supports the project from initial exploration and prototyping in Jupyter to more robust development and deployment in PyCharm, all while maintaining version control through GitHub. This multi-tool approach leverages the strengths of each platform, providing a flexible, efficient, and collaborative development workflow.

Frameworks and libraries

1. **re (Regular Expressions) and string**
   The `re` and `string` libraries are indispensable for text preprocessing and cleaning. They offer a comprehensive set of tools for pattern matching and string manipulation. In the pipeline, specifically in the "Data Cleaning" section, these libraries are employed to identify and remove URLs, hashtags, mentions, and other non-essential elements from the text. This step is crucial for simplifying the text and making it more suitable for natural language processing tasks.

2. **nltk (Natural Language Toolkit)**
   The `nltk` library is a comprehensive toolkit for a wide range of natural language processing tasks. It provides functionalities for text preprocessing, sentiment analysis, and word lemmatization. In the pipeline, it is used in both the "Exploratory Data Analysis" and "Feature Engineering" sections. For instance, the SentimentIntensityAnalyzer from NLTK's VADER module is employed to calculate sentiment polarity, providing a nuanced understanding of the sentiment distribution in the dataset.

3. **pandas**
   The `pandas` library is the cornerstone for data manipulation and analysis in Python. It offers data structures like DataFrames and Series that are highly flexible and powerful for handling structured data. Throughout the pipeline, Pandas is used for tasks ranging from data loading and cleaning to transformation and statistical analysis. It is especially vital in the EDA sections, where it is used to generate various visualizations and summary statistics.

4. **numpy**
   The `numpy` library is fundamental for scientific computing in Python. It provides an array object that is up to 50x faster than traditional Python lists and is used for various numerical operations. Although not extensively used in the pipeline, it often works in tandem with Pandas for numerical calculations, especially in the "Feature Engineering" step.

5. **SentimentIntensityAnalyzer (from nltk.sentiment.vader)**
   The `SentimentIntensityAnalyzer` is a specialized tool for sentiment analysis. It calculates a polarity score for each piece of text, which can be positive, negative, or neutral. In the "Exploratory Data Analysis" section, this tool is used to assess the sentiment polarity of the tweets, which is then visualized to provide a deeper understanding of the sentiment distribution across the dataset.

6. **RandomForestClassifier (from sklearn.ensemble)**
The `RandomForestClassifier` is an ensemble learning method that combines multiple decision trees to create a more robust and accurate model. It is particularly useful for controlling overfitting and improving the model's predictive accuracy. In the "Hyperparameter Tuning" section, this classifier is one of the models that are fine-tuned and evaluated to select the best-performing model for the task at hand.

7. **classification_report (from sklearn.metrics)**
The `classification_report` function generates a detailed report that includes various metrics such as precision, recall, F1-score, and support for each class. This is crucial for understanding the model's performance on a granular level. In the "Final Evaluation of Model" section, this function is used to evaluate the performance of the trained model, providing insights into its strengths and weaknesses.

8. **train_test_split (from sklearn.model_selection)**
The `train_test_split` function is used to partition the dataset into training and testing sets. This is essential for evaluating the model's performance on unseen data, thereby assessing its generalization ability. This function is employed to create separate datasets that are used for model training and evaluation.

9. **TfidfVectorizer (from sklearn.feature_extraction.text)**
The `TfidfVectorizer` is used to transform text data into numerical features using the Term Frequency-Inverse Document Frequency (TF-IDF) technique. This is crucial for machine learning algorithms that require numerical input. In the "Vectorizing" section, this tool is employed to convert the cleaned and preprocessed text data into a format that is suitable for machine learning algorithms.

10. **time**
The `time` library is used to measure the time taken for various computational operations. This is essential for assessing the efficiency and speed of different parts of the pipeline. In the "Hyperparameter Tuning" and "Final Evaluation of Model" sections, this library is used to measure the time taken for model training and prediction, providing insights into the computational efficiency of the pipeline.

11. **joblib**
The `joblib` library is used for saving and loading trained machine learning models. This is particularly useful for large models that are computationally expensive to train. In the "Final Evaluation of Model" step, this library is used to save the final trained model, allowing it to be easily reused or deployed in future tasks.

12. **zipfile**
The `zipfile` library is used for file compression, particularly for creating zip files. While not directly related to data science, it is a useful utility tool for packaging submission files or datasets, especially when participating in data science competitions.

13. **LazyClassifier (from lazypredict.Supervised)**
    The `LazyClassifier` is a convenient tool for quickly running and comparing a variety of machine learning models on a dataset. It provides an initial overview of how different models perform on the dataset without requiring manual hyperparameter tuning. In the "Final Evaluation of Model"" section, this tool is used for initial models evaluation, serving as a baseline for performance before fine-tuning the models and choosing the right one.

4.4.3. Code logic and functionality

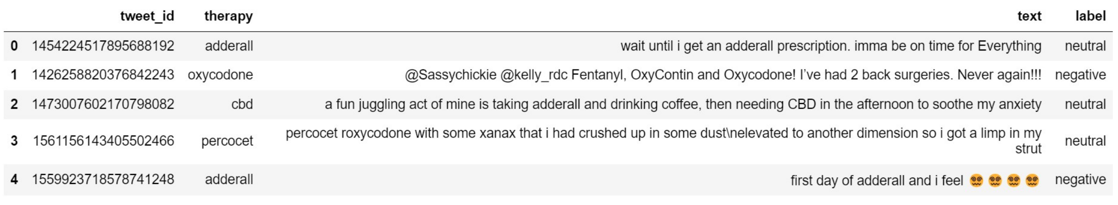
The solution, as mentioned before, is divided in various steps, including the previously explained exploratory data analysis, which will not be explained again in this section.

Data Cleaning

The code starts by importing a set of libraries, each serving a unique purpose. For instance, the re library is used for regular expressions, which are sequences of characters that define search patterns. Regular expressions are instrumental in text cleaning, particularly in tasks like removing special characters or finding specific text patterns. The nltk library, short for Natural Language Toolkit, is a comprehensive library for natural language processing (NLP). NLP is a field at the intersection of computer science, artificial intelligence, and linguistics, aiming to enable computers to understand, interpret, and produce human language in a way that is valuable.

After setting up the environment, we proceed to read in the data using the pd.read_csv() function from the pandas library. Pandas is a fast, powerful, and flexible open-source data analysis and manipulation library. DataFrames, one of the primary data structures in Pandas, are essentially tables, like an Excel spreadsheet. Reading the data into a DataFrame allows for more straightforward manipulation and analysis.

The figure 4.16 illustrates an overview of the dataset, which is useful to track, as it will help to understand the changes made to the 'text' in the dataset in the data cleaning and feature engineering sections.



	tweet_id	therapy	text	label
0	1454224517895688192	adderall	wait until i get an adderall prescription. imma be on time for Everything	neutral
1	1426258820376842243	oxycodone	@Sassychickie @kelly_rdc Fentanyl, OxyContin and Oxycodone! I've had 2 back surgeries. Never again!!!	negative
2	1473007602170798082	cbd	a fun juggling act of mine is taking adderall and drinking coffee, then needing CBD in the afternoon to soothe my anxiety	neutral
3	1561156143405502466	percocet	percocet roxycodone with some xanax that i had crushed up in some dust\nelevated to another dimension so i got a limp in my strut	neutral
4	1559923718578741248	adderall	first day of adderall and i feel 🥴 🥴 🥴 🥴	negative

Fig. 4.16. Pipeline: Data Cleaning - Train dataset preview

The next step is checking the length of the datasets. Knowing the size of your data is crucial for several reasons. Larger datasets generally provide more robust models but come at the cost of computational efficiency. Smaller datasets are computationally easier to manage but may lead to overfitting, a modeling error that occurs when a function is too closely aligned to a limited set of data points.

Next, the notebook downloads a list of stopwords from the nltk library. Stopwords are common words that are generally ignored in text data processing because they occur frequently but don't carry significant meaning. Examples include "and," "the," and "is." Removing these words can help improve the performance of the machine learning models later in the pipeline.

Also, the initialization of a WordNet lemmatizer from nltk, as displayed in code snippet 4.3, is necessary. Lemmatization is the process of reducing a word to its base or root form. For example, "running" becomes "run," and "mice" become "mouse." Unlike stemming, which crudely chops off the ends of words to achieve the same goal, lemmatization considers the morphological analysis of the words and correctly identifies the lemma for each word. This is crucial because it reduces dimensionality and simplifies the representation of the text, making it easier for machine learning algorithms to understand.

CODE SNIPPET 4.3. Lemmatizer set up

```
# Downloading the stopwords corpus from NLTK
stopwords = nltk.corpus.stopwords.words('english')

# Creating a WordNet lemmatizer object from NLTK
wn = nltk.WordNetLemmatizer()
```

The core of the data cleaning process is the clean_text function, which python code can be viewed in code snippet 4.4. This function performs several tasks:

- **Punctuation Removal and Lowercasing:** The text is stripped of all punctuation marks and converted to lowercase. This is crucial because punctuation and capitalization can introduce noise into the text data. For instance, the word "Run" with an uppercase 'R' and "run" with a lowercase 'r' would be treated as distinct features in a machine learning model if not standardized.

- **Tokenization:** After cleaning, the text is tokenized. Tokenization is the process of breaking up a sequence of strings into units like words, keywords, phrases, symbols, and other elements, known as tokens. Tokenization is key for extracting meaningful chunks of information from the raw text, allowing for more effective subsequent processing and analysis.

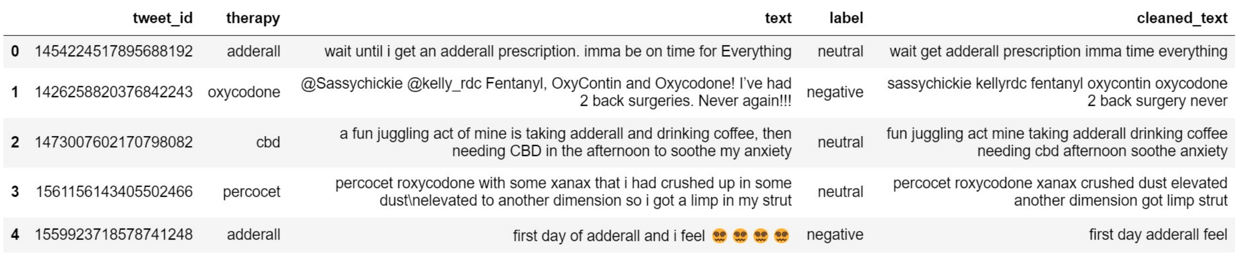
- **Stopword Removal and Lemmatization:** This step involves removing stopwords and lemmatizing the remaining words. Lemmatization is especially beneficial in machine learning for text analysis. It reduces inflected words to their root form, which is important for ensuring that the context carried by the root word is retained.

CODE SNIPPET 4.4. Clean text function

```python
def clean_text(text):
    # Removing punctuation characters from the text and converting it to lowercase
    text = "".join([word.lower() for word in text if word not in string.punctuation])
    # Splitting the text into tokens (words) using regular expressions that match
        just words
    tokens = re.split('\W+', text)
    # Lemmatizing each word in the tokens list using the WordNet lemmatizer
    text = [wn.lemmatize(word) for word in tokens if word not in stopwords]
    # Returning the cleaned text
    return text
```

After defining this function, it is applied to the entire dataset. This is done using the apply() function in Pandas, which applies a function along an axis of the DataFrame (columns in this case). The cleaned text is stored in a new column, and this is crucial for the next steps in the pipeline, which involve feature extraction and model training.

An overview of the cleaned text is illustrated in figure 4.17 below.



	tweet_id	therapy	text	label	cleaned_text
0	1454224517895688192	adderall	wait until i get an adderall prescription. imma be on time for Everything	neutral	wait get adderall prescription imma time everything
1	1426258820376842243	oxycodone	@Sassychickie @kelly_rdc Fentanyl, OxyContin and Oxycodone! I've had 2 back surgeries. Never again!!!	negative	sassychickie kellyrdc fentanyl oxycontin oxycodone 2 back surgery never
2	1473007602170798082	cbd	a fun juggling act of mine is taking adderall and drinking coffee, then needing CBD in the afternoon to soothe my anxiety	neutral	fun juggling act mine taking adderall drinking coffee needing cbd afternoon soothe anxiety
3	1561156143405502466	percocet	percocet roxycodone with some xanax that i had crushed up in some dust\nelevated to another dimension so i got a limp in my strut	neutral	percocet roxycodone xanax crushed dust elevated another dimension got limp strut
4	1559923718578741248	adderall	first day of adderall and i feel 🥴🥴🥴🥴	negative	first day adderall feel

Fig. 4.17. Pipeline: Data Cleaning - Cleaned dataset preview

Finally, the cleaned data is saved back into CSV files. This is an essential step because cleaning text data can be computationally expensive. By saving the cleaned data, we avoid having to repeat this process each time you work on the next steps of your project.

Each step in the data cleaning notebook is meticulously designed to prepare the text data for subsequent analysis. This pipeline step employs various NLP techniques like tokenization, stopword removal, and lemmatization, each contributing to the robustness and accuracy of the machine learning models to be built later in the pipeline.

Feature Engineering

Here is where the raw, cleaned data is transformed into a format that can be fed into machine learning models. Feature engineering is often considered an art as much as it is a science, and it's crucial for improving a model's performance. It's not just about feeding data into a model; it's about creating the right kind of data that the model can understand and learn from effectively.

As always, it begins by importing a series of Python libraries that are foundational for data manipulation and natural language processing. Among these are Pandas for data manipulation, NumPy for numerical operations, and NLTK for natural language processing tasks. Each library serves a unique purpose. For instance, Pandas is used for its DataFrame functionality, which allows for easy manipulation and transformation of tabular data. NumPy is essential for any numerical operations that need to be performed on the data, such as calculating means or other statistics. NLTK is a comprehensive library for text processing libraries for classification, tokenization, stemming, tagging, parsing, and more.

The first feature that is engineered is the ***body length*** of each tweet, which essentially counts the number of characters in each tweet. The overall length of the text can influence the complexity and content of the message. This is a straightforward yet powerful feature. The length of a tweet could correlate with its sentiment; for instance, shorter tweets might be indicative of strong, immediate emotional reactions, whereas longer tweets might be more thought-out and nuanced.

The next feature is the ***count of punctuation marks*** in each tweet. This feature counts the percentage of punctuation characters in the text body. Punctuation can provide insights into the writing style and emotional tone of the text.Punctuation can often carry emotional weight in text. Exclamation marks may indicate excitement or urgency, whereas question marks could indicate confusion or curiosity. By calculating the percentage of punctuation marks in each tweet, we can create a feature that may capture some of this emotional nuance.

Sentiment intensity is calculated using the Sentiment Intensity Analyzer (SIA), from the NLTK library. This tool uses a pre-trained model to assign a sentiment score to each tweet based on the words it contains. The sentiment intensity score is a float that ranges from -1 to 1; negative values indicate negative sentiment, positive values indicate positive sentiment, and values close to zero indicate neutrality. This feature is crucial for our task of sentiment classification, as it provides a numerical representation of the sentiment expressed in each tweet.

We also calculated several ***word-based*** features, such as the total word count, character count, average word length, punctuation count, hashtag count, and stopword count for each tweet. Each of these features provides a different perspective on the text data:

- **Word Count:** The total number of words in a tweet can indicate the complexity and information density of the text.
- **Character Count:** This is similar to word count but focuses on individual characters, providing insight into the tweet's length and complexity.
- **Average Word Length:** This could be an indicator of the complexity of the language used in the tweet.

- **Punctuation Count:** This is an extension of the punctuation feature mentioned earlier but focuses on the variety and frequency of different punctuation marks used.
- **Hashtag Count:** Hashtags are often used for emphasis or to participate in larger conversations. A high number of hashtags could indicate a tweet's involvement in broader social media trends or topics.
- **Stopword Count:** Stopwords like 'the', 'is', etc., are generally filtered out before text data is fed into algorithms. However, the frequency of stopwords can also be a useful feature, as an unusually high or low number of stopwords could indicate a particular style of writing or emphasis.

The notebook goes a step further to calculate the ***frequency of positive and negative sentiment words*** based on a predefined lexicon, as showed in code snippet 4.5. This is a more granular approach to understanding sentiment, as it counts the occurrence of words that are specifically labeled as positive or negative in sentiment lexicons. This could be particularly useful for capturing the nuanced emotional context in which certain words are used.

CODE SNIPPET 4.5. Sentiment Word Counts feature

```
# Count of Positive Words
positive_words = [word for word, score in sia.lexicon.items() if score > 0]

data['sia_positive_word_count'] = data['cleaned_text'].apply(lambda x: len([word for
    word in x if word in positive_words]))

# Count of Negative Words
negative_words = [word for word, score in sia.lexicon.items() if score < 0]

data['sia_negative_word_count'] = data['cleaned_text'].apply(lambda x: len([word for
    word in x if word in negative_words]))

# Positive Word Rate
data['sia_positive_word_rate'] = data['sia_positive_word_count'] / data['word_count']

# Negative Word Rate
data['sia_negative_word_rate'] = data['sia_negative_word_count'] / data['word_count']
```

VADER (Valence Aware Dictionary and sentiment Reasoner) is another tool used for sentiment analysis. Unlike the Sentiment Intensity Analyzer, which gives a single compound score, VADER provides multiple scores: a positive score, a negative score, a neutral score, and a compound score, which is a weighted sum of the other three. This allows for a more nuanced understanding of sentiment.

Also, emoji based sentiment related features were calculated, as illustrated in code snippet 4.6:

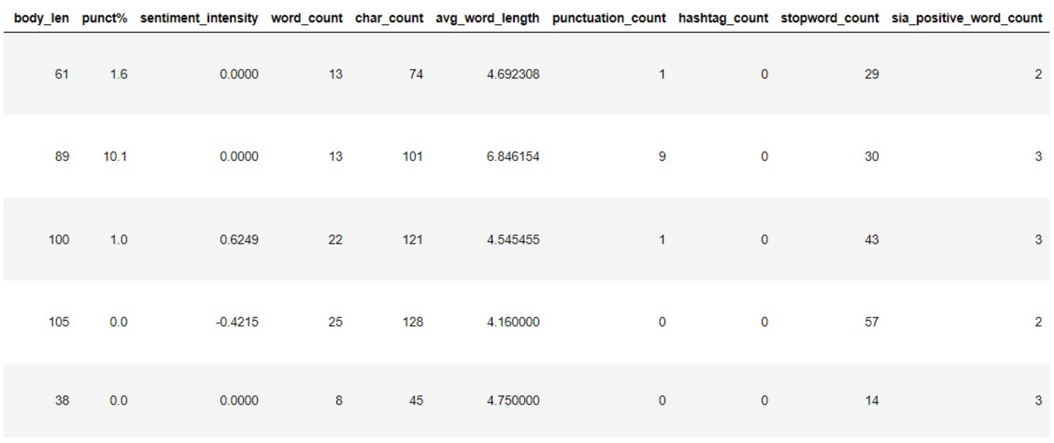
- `laugh_count`: The count of laughing emoticons in the text. Emoticons can convey emotions and provide context to the text.

- `sad_count`: The count of sad emoticons in the text. Similar to `laugh_count`, this feature captures emotional expression.

CODE SNIPPET 4.6. Emoji based Sentiment Features

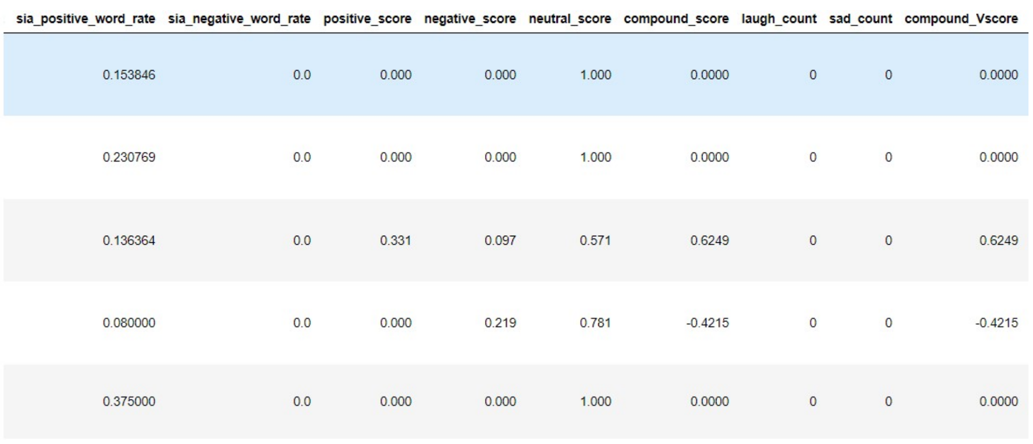
```
# Count of Laughing Expressions
laugh_expressions = ['haha', 'hehe', 'lol']
data['laugh_count'] = data['text'].apply(lambda x: sum([x.lower().count(expr) for
    expr in laugh_expressions]))


# Count of Sad Expressions
sad_expressions = [':(', ':-(', ';(', ';-(']
data['sad_count'] = data['text'].apply(lambda x: sum([x.count(expr) for expr in
    sad_expressions]))
```

The features created are included in a new dataset, and are displayed in figures 4.18 and 4.19:



body_len	punct%	sentiment_intensity	word_count	char_count	avg_word_length	punctuation_count	hashtag_count	stopword_count	sia_positive_word_count
61	1.6	0.0000	13	74	4.692308	1	0	29	2
89	10.1	0.0000	13	101	6.846154	9	0	30	3
100	1.0	0.6249	22	121	4.545455	1	0	43	3
105	0.0	-0.4215	25	128	4.160000	0	0	57	2
38	0.0	0.0000	8	45	4.750000	0	0	14	3

Fig. 4.18. Pipeline: Feature Engineering - All new features



sia_positive_word_rate	sia_negative_word_rate	positive_score	negative_score	neutral_score	compound_score	laugh_count	sad_count	compound_Vscore
0.153846	0.0	0.000	0.000	1.000	0.0000	0	0	0.0000
0.230769	0.0	0.000	0.000	1.000	0.0000	0	0	0.0000
0.136364	0.0	0.331	0.097	0.571	0.6249	0	0	0.6249
0.080000	0.0	0.000	0.219	0.781	-0.4215	0	0	-0.4215
0.375000	0.0	0.000	0.000	1.000	0.0000	0	0	0.0000

Fig. 4.19. Pipeline: Feature Engineering - All new features, part 2

After all these features are created, a RandomForestClassifier is used to evaluate their importance. RandomForest is an ensemble learning method that can be used for both classification and regression tasks. It works by constructing multiple decision trees during training and outputs the class that is the mode of the classes for classification. By using it here, we can understand which features are most informative for predicting sentiment, allowing us to possibly reduce dimensionality later.

Both feature and permutation importance were calculated to determine the most relevant feature for the task, and avoid including non-useful columns to the dataset which model is going to be trained on.

Code snippets 4.7 and 4.8 for this step are provided for better comprehension on what is being performed. The results are showed in figures 4.20 and 4.21

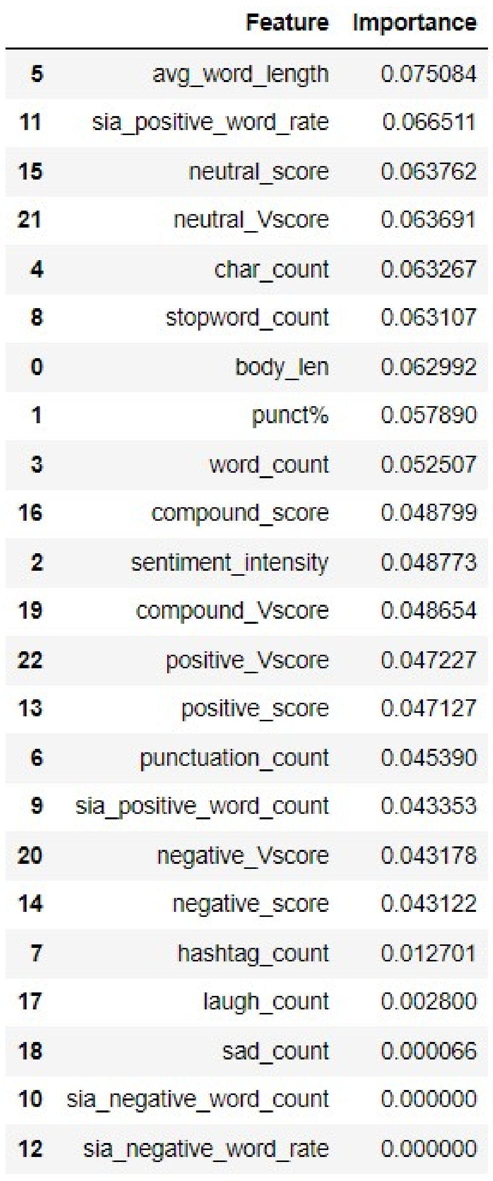
CODE SNIPPET 4.7. Feature Importance

```python
import numpy as np
from sklearn.ensemble import RandomForestClassifier

# Initialize an array to store the feature importances
feature_importances = np.zeros(len(feature_columns))

# Number of iterations
n_iterations = 200

# Train the model multiple times with different random states
for i in range(n_iterations):
    rf = RandomForestClassifier(n_estimators=100, random_state=i)
    rf.fit(X_train, y_train)
    feature_importances += rf.feature_importances_

# Average the feature importances
feature_importances /= n_iterations

# Create a DataFrame with the feature names and their importance scores
feature_importance_df = pd.DataFrame({'Feature': feature_columns, 'Importance':
    feature_importances})

# Sort the DataFrame by importance score in descending order
feature_importance_df_2 = feature_importance_df.sort_values('Importance', ascending=
    False)
```

CODE SNIPPET 4.8. Permutation Importance

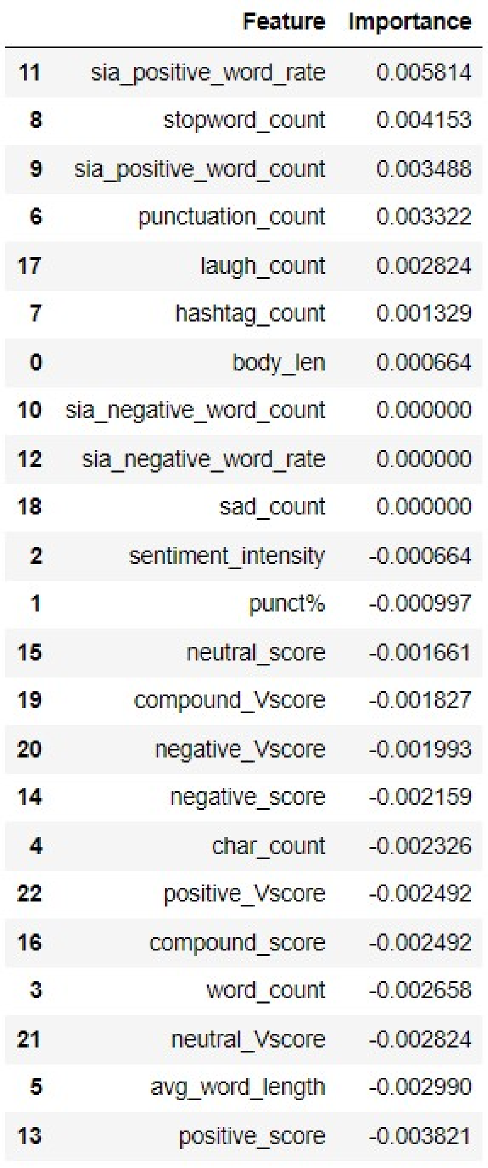
```python
from sklearn.inspection import permutation_importance

# Train the model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Compute permutation importance
result = permutation_importance(rf, X_test, y_test, n_repeats=10, random_state=42)

# Create a DataFrame with the feature names and their importance scores
permutation_importance_df = pd.DataFrame({'Feature': feature_columns, 'Importance':
    result.importances_mean})

# Sort the DataFrame by importance score in descending order
```

```
permutation_importance_df = permutation_importance_df.sort_values('Importance',
    ascending=False)

permutation_importance_df
```



	Feature	Importance
5	avg_word_length	0.075084
11	sia_positive_word_rate	0.066511
15	neutral_score	0.063762
21	neutral_Vscore	0.063691
4	char_count	0.063267
8	stopword_count	0.063107
0	body_len	0.062992
1	punct%	0.057890
3	word_count	0.052507
16	compound_score	0.048799
2	sentiment_intensity	0.048773
19	compound_Vscore	0.048654
22	positive_Vscore	0.047227
13	positive_score	0.047127
6	punctuation_count	0.045390
9	sia_positive_word_count	0.043353
20	negative_Vscore	0.043178
14	negative_score	0.043122
7	hashtag_count	0.012701
17	laugh_count	0.002800
18	sad_count	0.000066
10	sia_negative_word_count	0.000000
12	sia_negative_word_rate	0.000000

Fig. 4.20. Pipeline: Feature Engineering - Features ranked by impotance



	Feature	Importance
11	sia_positive_word_rate	0.005814
8	stopword_count	0.004153
9	sia_positive_word_count	0.003488
6	punctuation_count	0.003322
17	laugh_count	0.002824
7	hashtag_count	0.001329
0	body_len	0.000664
10	sia_negative_word_count	0.000000
12	sia_negative_word_rate	0.000000
18	sad_count	0.000000
2	sentiment_intensity	-0.000664
1	punct%	-0.000997
15	neutral_score	-0.001661
19	compound_Vscore	-0.001827
20	negative_Vscore	-0.001993
14	negative_score	-0.002159
4	char_count	-0.002326
22	positive_Vscore	-0.002492
16	compound_score	-0.002492
3	word_count	-0.002658
21	neutral_Vscore	-0.002824
5	avg_word_length	-0.002990
13	positive_score	-0.003821

Fig. 4.21. Pipeline: Feature Engineering - Features ranked by permutation impotance

Finally, after all these feature engineering steps, the notebook compiles all these new features into a new DataFrame. This DataFrame is then saved as a new CSV file, which will be used in the subsequent steps of the pipeline for model training and evaluation. After exhaustive evaluation, the final selected features are:

```
'tweet_id', 'therapy', 'label', 'cleaned_text', 'avg_word_length', '
    sia_positive_word_rate', 'sia_negative_word_rate', 'neutral_score', '
    stopword_count', 'body_len', 'compound_score', 'punct%', 'positive_score', '
    negative_score', 'neutral_score'
```

The feature engineering section is a critical component of the machine learning pipeline. It takes the cleaned text data and enriches it with a variety of features that aim to capture the underlying patterns and nuances in the data. These features are then used to train and evaluate machine learning models in the subsequent steps of the pipeline.

Hyperparameter Tuning

This step is dedicated to fine-tuning the model parameters to optimize its performance. Hyperparameter tuning is an essential step in machine learning to ensure that a model generalizes well to new, unseen data.

Hyperparameters are settings that define the structure and behavior of machine learning models. Unlike model parameters, which are learned during training, hyperparameters are set prior to the training process and remain constant during it. The objective is to find the combination of hyperparameters that minimizes a predefined loss function, thereby maximizing the model's performance.

Essential Python libraries are imported for data manipulation and machine learning. These include NumPy for numerical operations, Pandas for data manipulation, and scikit-learn for machine learning tasks. Specifically, RandomForestClassifier and GridSearchCV are imported. The former is an ensemble learning method for classification, and the latter is a utility for hyperparameter tuning.

Reading in the cleaned and feature-engineered dataset, which is loaded into a Pandas DataFrame. This dataset contains both the features and labels necessary for the machine learning task at hand.

We converted the textual labels ('neutral', 'positive', 'negative') into numerical labels (0, 1, 2). This is crucial because machine learning algorithms require numerical input. A dictionary is used to map the textual labels to their numerical counterparts.

Then, we split the dataset into training, validation, and testing sets. This is a crucial step in machine learning as it allows the model to be trained on one subset of the data and validated and tested on other unseen subsets. The splitting is done in a 60-20-20 ratio, which is a commonly used practice. The features and labels are separated before the split.

We continue printing the lengths of the training, validation, and testing sets to verify that the splitting was done correctly. It also checks the proportions to ensure they align with the intended 60-20-20 split.

The notebook then moves on to the core task of hyperparameter tuning using Grid-

SearchCV. GridSearchCV performs exhaustive search over a specified parameter grid. For the RandomForestClassifier, the it explores different values for n_estimators and max_depth. n_estimators refers to the number of trees in the forest, and max_depth refers to the maximum depth of the trees.

A custom function named print_results is defined to display the performance of the different hyperparameter combinations. This function prints the best parameters found during the cross-validation and also shows the mean and standard deviation of the test scores for each combination.

We then run the grid search on the training data, as displayed in code snippet 4.9. It uses 5-fold cross-validation, meaning the training set is split into 5 subsets; the model is trained on 4 of these and validated on the remaining one. This process is repeated 5 times, each time with a different subset serving as the validation set.

CODE SNIPPET 4.9. Performing GridSearchCV

```
# Create an instance of the Random Forest Classifier
rf = RandomForestClassifier()

# Define the parameters to be tuned in the grid search
parameters = {
    'n_estimators': [5, 50, 100, 200],
    'max_depth': [2, 10, 20, None]
}

# Create an instance of GridSearchCV with the Random Forest Classifier and parameter
    grid
cv = GridSearchCV(rf, parameters, cv=5)

# Fit the training features and labels to the grid search cross-validation
cv.fit(X_train, y_train.values.ravel())
```

Finally, the results of the grid search are printed, as illustrated in 4.10, showing the performance of each hyperparameter combination. The best-performing combination is highlighted, providing a clear direction for the final model training.

CODE SNIPPET 4.10. Displaying the Results

```
BEST PARAMS: {'max_depth': 10, 'n_estimators': 100}

0.837 (+/-0.192) for {'max_depth': 2, 'n_estimators': 5}
0.781 (+/-0.083) for {'max_depth': 2, 'n_estimators': 50}
0.801 (+/-0.092) for {'max_depth': 2, 'n_estimators': 100}
0.807 (+/-0.079) for {'max_depth': 2, 'n_estimators': 200}
0.975 (+/-0.036) for {'max_depth': 10, 'n_estimators': 5}
0.999 (+/-0.002) for {'max_depth': 10, 'n_estimators': 50}
1.0 (+/-0.0) for {'max_depth': 10, 'n_estimators': 100}
1.0 (+/-0.0) for {'max_depth': 10, 'n_estimators': 200}
0.977 (+/-0.022) for {'max_depth': 20, 'n_estimators': 5}
1.0 (+/-0.0) for {'max_depth': 20, 'n_estimators': 50}
1.0 (+/-0.0) for {'max_depth': 20, 'n_estimators': 100}
1.0 (+/-0.0) for {'max_depth': 20, 'n_estimators': 200}
0.988 (+/-0.02) for {'max_depth': None, 'n_estimators': 5}
1.0 (+/-0.0) for {'max_depth': None, 'n_estimators': 50}
1.0 (+/-0.0) for {'max_depth': None, 'n_estimators': 100}
```

```
1.0 (+/-0.0) for {'max_depth': None, 'n_estimators': 200}
```

The "Hyperparameter Tuning" step is a comprehensive guide to optimizing a RandomForestClassifier for multi-class sentiment classification. It covers everything from data preparation to hyperparameter tuning, providing a robust foundation for the final steps in the machine learning pipeline.

Vectorizing

This section focuses on transforming raw text into a numerical format that machine learning algorithms can understand. The process of vectorization is a cornerstone in the field of NLP and is critical for the performance of models that deal with text data.

Term Frequency-Inverse Document Frequency (TF-IDF) is a popular method for text vectorization. TF-IDF is a numerical representation of text that reflects how important a word is to a document in a collection of documents (corpus). It's a balance between how often a word appears in a specific document (Term Frequency) and how unique the word is across the entire corpus (Inverse Document Frequency).

Unlike simple count vectorization, TF-IDF gives weight to each term in each document that reflects not just the term's frequency in that document but also how unique the term is across all documents. In essence, it balances out the term frequencies by considering how many documents contain each term. This results in a more nuanced representation of text data.

As usual, the first step is to import a variety of Python libraries, including NumPy for numerical operations, Pandas for data manipulation, and scikit-learn for machine learning tasks. Specifically, the TfidfVectorizer and CountVectorizer classes from scikit-learn are imported for text vectorization. These libraries are foundational for any data science project and are especially crucial for NLP tasks.

The main dataset and a test dataset are loaded into Pandas DataFrames. The datasets contain cleaned and pre-processed text along with various other features and labels. The head() function is used to display the first few rows of the test dataset, providing a snapshot of the data structure.

The lengths of the main and test datasets are printed. Knowing the size of the datasets is essential for understanding the computational resources needed for subsequent steps and for ensuring that there's enough data for training and testing the machine learning models.

The notebook splits the main dataset into training and testing sets using the train_test_split function from scikit-learn. The features (X) and labels (y) are separated, and 20% of the data is reserved for testing. This is a crucial step as it allows for the evaluation of the machine learning model on unseen data, thereby assessing its generalization capability.

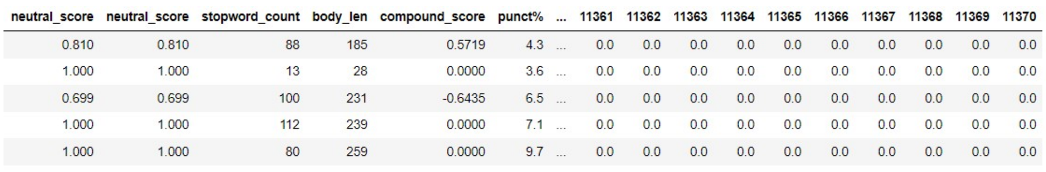
The training and testing sets are then saved to CSV files. This is done to ensure that the same split can be reused in future runs of the notebook or other notebooks, ensuring consistency in the experiments.

Now we move on to the core task of TF-IDF vectorization. A TfidfVectorizer object is created and fitted to the 'cleaned_text' column of the training set. The text data in both the training and testing sets are then transformed into TF-IDF vectors. These vectors are then concatenated with other feature columns to create a final feature set for both training and testing. All these steps are showed in the below code 4.11.

An example of the vectorized dataset can be seen in figure 4.22.

CODE SNIPPET 4.11. TF-IDF Vectorization

```
# Creating a TfidfVectorizer object with the analyzer parameter set to the clean_text
     function
tfidf_vect = TfidfVectorizer()

# Fitting the TfidfVectorizer on the 'text' column of the training set
tfidf_vect_fit = tfidf_vect.fit(X_train['cleaned_text'])

# Transforming the 'text' column of the training and testing sets into TF-IDF
    features
tfidf_train = tfidf_vect_fit.transform(X_train['cleaned_text'])
tfidf_test = tfidf_vect_fit.transform(X_test['cleaned_text'])

# Concatenating the features columns with the TF-IDF features of the training set
X_train_vect = pd.concat([X_train[
    ['tweet_id', 'avg_word_length', 'sia_positive_word_rate', 'sia_negative_word_rate
        ', 'neutral_score',
     'stopword_count', 'body_len', 'compound_score', 'punct%', 'positive_score', '
         negative_score', 'neutral_score']
].reset_index(drop=True), pd.DataFrame(tfidf_train.toarray())], axis=1)

# Concatenating the features columns with the TF-IDF features of the testing set
X_test_vect = pd.concat([X_test[
    ['tweet_id', 'avg_word_length', 'sia_positive_word_rate', 'sia_negative_word_rate
        ', 'neutral_score',
     'stopword_count', 'body_len', 'compound_score', 'punct%', 'positive_score', '
         negative_score', 'neutral_score']
].reset_index(drop=True), pd.DataFrame(tfidf_test.toarray())], axis=1)

# Displaying the head (first few rows) of the X_train_vect DataFrame
X_train_vect.head()
```



neutral_score	neutral_score	stopword_count	body_len	compound_score	punct%	...	11361	11362	11363	11364	11365	11366	11367	11368	11369	11370
0.810	0.810	88	185	0.5719	4.3	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.000	1.000	13	28	0.0000	3.6	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.699	0.699	100	231	-0.6435	6.5	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.000	1.000	112	239	0.0000	7.1	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.000	1.000	80	259	0.0000	9.7	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Fig. 4.22. Pipeline: Vectorizing - TD-IDF vectorizer

This enriched feature set is saved to a CSV file. This ensures that the hard work of feature engineering is preserved and can be easily used in future machine learning experiments.

The "Vectorizing" notebook serves as a comprehensive guide to transforming raw text into a format that is amenable to machine learning algorithms. It covers various methods and techniques, providing a robust foundation for any NLP project. The notebook not only performs the vectorization but also sets the stage for the subsequent steps in the NLP pipeline, such as model training and evaluation.

Final Evaluation of Model

The evaluation of the model serves as the culmination of the entire machine learning pipeline for sentiment analysis in English tweets related to therapies. This step is where the rubber meets the road, so to speak, in terms of assessing the performance of the trained machine learning models. This is a crucial step in any machine learning project, as it provides a comprehensive assessment of how well the model performs on unseen data.

Importing a variety of Python libraries essential for machine learning and data manipulation is the first step. These include pandas for data manipulation, joblib for saving and loading machine learning models, and sklearn for various machine learning algorithms and metrics. Additionally, lazypredict is imported, a library that allows for quick and easy evaluation of multiple machine learning models.

Reading in the vectorized training and testing datasets, along with their corresponding labels. These datasets are loaded into Pandas DataFrames. The data has already been preprocessed and vectorized, making it ready for model evaluation.

We employ the RandomForestClassifier, an ensemble learning method that combines multiple decision trees to improve predictive accuracy and control overfitting. The model is trained (see code 4.12) on the vectorized training dataset. The time taken to train the model is also measured, providing insights into the model's efficiency.

CODE SNIPPET 4.12. Model Training - RandomForestClassifier

```
# Creating a RandomForestClassifier object with specified parameters
rf = RandomForestClassifier(n_estimators=150, max_depth=None, n_jobs=-1)

# Measuring the time taken to fit (train) the RandomForestClassifier on the training
    data
start = time.time()

# Training the model
rf_model = rf.fit(X_train_vect, y_train)
# Save the model for future use
joblib.dump(rf_model, "C:\\Users\\danij\\Documents\\UC3M\\TFG\\MODELS\\rf_model.pkl")

# Calculating total time taken to train the model
end = time.time()
fit_time = (end - start)
```

After training, the model is used to make predictions on the vectorized test dataset. Again, the time taken for making these predictions is measured. This is crucial for understanding how the model will perform in a real-world, time-sensitive application.

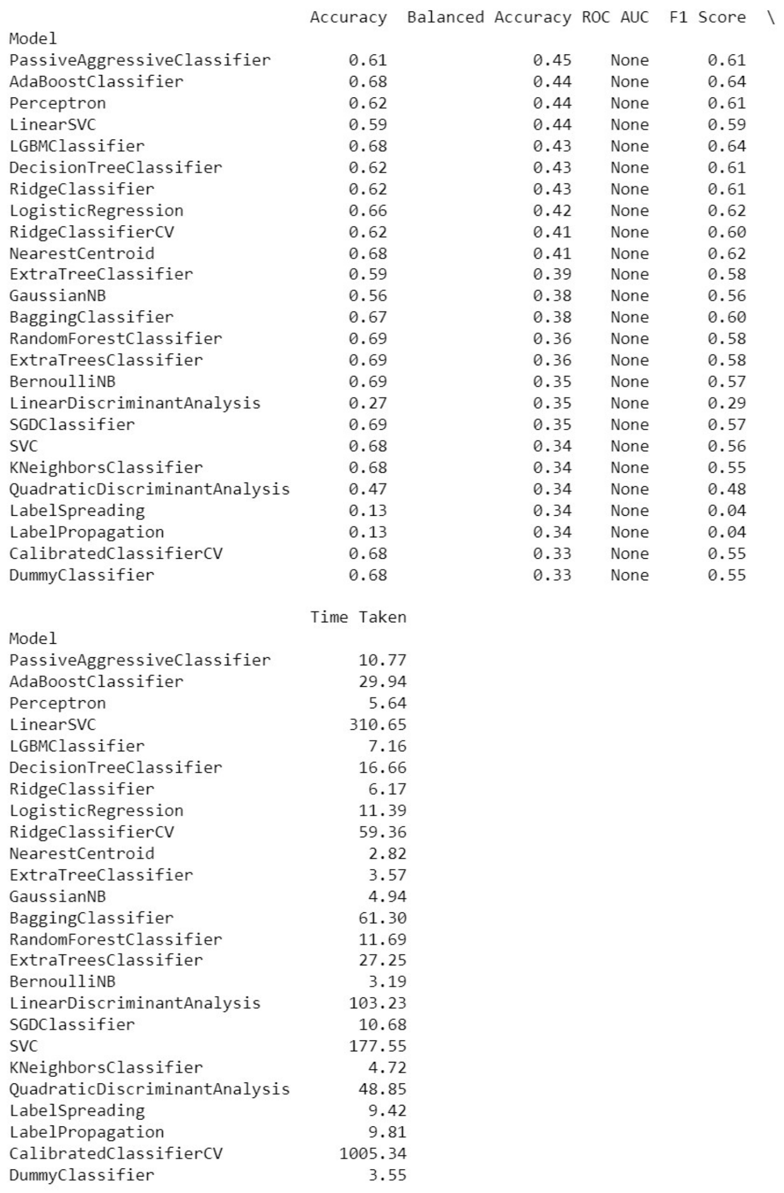
Now we move on to the evaluation phase, where various metrics such as precision, recall, and F1-score are calculated (results in 4.13). These metrics provide a comprehensive view of the model's performance, taking into account both false positives and false negatives. The 'macro' average is used, which calculates metrics for each label and finds their unweighted mean. This is particularly useful for multi-class classification problems like this one.

CODE SNIPPET 4.13. Evaluation Results

```
Macro Average Precision: 0.7964331874087972
Macro Average Recall: 0.45454044001114826
Macro Average F1-score: 0.7048280183843417
Fit time: 5.166675329208374
Predict time: 0.24392271041870117
```

LazyClassifier from the lazypredict library is also employed to evaluate multiple machine learning models quickly. This is an excellent way to get a preliminary idea of which algorithms perform well on the dataset without having to manually train and evaluate each one.

The approximated results other models provided is illustrated in figure 4.23



```
                              Accuracy  Balanced Accuracy ROC AUC  F1 Score  \
Model
PassiveAggressiveClassifier       0.61               0.45    None      0.61
AdaBoostClassifier                0.68               0.44    None      0.64
Perceptron                        0.62               0.44    None      0.61
LinearSVC                         0.59               0.44    None      0.59
LGBMClassifier                    0.68               0.43    None      0.64
DecisionTreeClassifier            0.62               0.43    None      0.61
RidgeClassifier                   0.62               0.43    None      0.61
LogisticRegression                0.66               0.42    None      0.62
RidgeClassifierCV                 0.62               0.41    None      0.60
NearestCentroid                   0.68               0.41    None      0.62
ExtraTreeClassifier               0.59               0.39    None      0.58
GaussianNB                        0.56               0.38    None      0.56
BaggingClassifier                 0.67               0.38    None      0.60
RandomForestClassifier            0.69               0.36    None      0.58
ExtraTreesClassifier              0.69               0.36    None      0.58
BernoulliNB                       0.69               0.35    None      0.57
LinearDiscriminantAnalysis        0.27               0.35    None      0.29
SGDClassifier                     0.69               0.35    None      0.57
SVC                               0.68               0.34    None      0.56
KNeighborsClassifier              0.68               0.34    None      0.55
QuadraticDiscriminantAnalysis     0.47               0.34    None      0.48
LabelSpreading                    0.13               0.34    None      0.04
LabelPropagation                  0.13               0.34    None      0.04
CalibratedClassifierCV            0.68               0.33    None      0.55
DummyClassifier                   0.68               0.33    None      0.55

                              Time Taken
Model
PassiveAggressiveClassifier        10.77
AdaBoostClassifier                 29.94
Perceptron                          5.64
LinearSVC                         310.65
LGBMClassifier                      7.16
DecisionTreeClassifier             16.66
RidgeClassifier                     6.17
LogisticRegression                 11.39
RidgeClassifierCV                  59.36
NearestCentroid                     2.82
ExtraTreeClassifier                 3.57
GaussianNB                          4.94
BaggingClassifier                  61.30
RandomForestClassifier             11.69
ExtraTreesClassifier               27.25
BernoulliNB                         3.19
LinearDiscriminantAnalysis        103.23
SGDClassifier                      10.68
SVC                               177.55
KNeighborsClassifier                4.72
QuadraticDiscriminantAnalysis      48.85
LabelSpreading                      9.42
LabelPropagation                    9.81
CalibratedClassifierCV           1005.34
DummyClassifier                     3.55
```

Fig. 4.23. Pipeline: Model Evaluation - Evaluating other models

Finally, the predictions are prepared for submission. A new DataFrame is created that includes the tweet IDs and their corresponding predicted labels. This DataFrame is then saved as a text file, ready for submission or further analysis.

The "Final Evaluation of Model" section serves as the concluding step in the machine learning pipeline. It not only evaluates the final chosen model but also explores other potential models for the task at hand. The notebook is comprehensive, covering everything from data loading to model training, evaluation, and finally, preparing the results for submission or further analysis.

4.4.4. Other considered approaches

There are always alternative approaches and techniques that could be explored to potentially improve the model's performance or offer new insights. After investigating, here are the most relevants one for this task:

Alternative Machine Learning Algorithms

- **Support Vector Machines (SVM):** SVMs are effective for text classification tasks and could be a good alternative to Random Forests.
- **Gradient Boosting Machines (XGBoost, LightGBM):** These are ensemble learning methods that could potentially offer better performance than Random Forests.
- **Neural Networks:** Deep learning techniques, particularly Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks, are well-suited for text data.
- **Convolutional Neural Networks (CNNs):** Though traditionally used in image processing, CNNs have shown promise in text classification tasks.

Text Representation Techniques

- **Word Embeddings (Word2Vec, GloVe):** Instead of TF-IDF, we could consider the use of pre-trained word embeddings or training our own.
- **BERT (Bidirectional Encoder Representations from Transformers):** A more advanced text representation technique that takes into account the context for each word.

Feature Engineering

- **Syntactic Features:** Parts-of-speech tags, syntactic parse trees, etc., could add more dimensions to the feature set.
- **Semantic Features:** Named Entity Recognition (NER), semantic role labeling, etc., could also be useful.
- **Advanced Sentiment Features:** Beyond simple polarity scores, we could look into emotion analysis (joy, anger, sadness, etc.) or even sarcasm detection.

Data Augmentation

- **SMOTE (Synthetic Minority Over-sampling Technique):** If the dataset is imbalanced, SMOTE can help generating synthetic samples in the feature space.
- **Text Augmentation:** Techniques like back translation, synonym replacement, etc., can increase the size and diversity of the training data.

Model Interpretability

- **LIME (Local Interpretable Model-agnostic Explanations):** To understand what the model is doing, especially if there are advanced resources available, for more complex models like neural networks.
- **SHAP (SHapley Additive exPlanations):** Another tool for model interpretability, particularly useful for ensemble methods.

Evaluation Metrics

- **Cost-sensitive Learning:** If different types of misclassifications have different costs (e.g., false positives vs. false negatives).
- **Multi-class ROC and Precision-Recall Curves:** These could provide a more nuanced understanding of the model's performance across different classes.

Workflow and Scalability

- **Pipeline Automation:** Considering using tools like Apache Airflow to automate the ML pipeline, especially useful for retraining models.
- **Model Deployment:** If the end goal is to integrate the model into a live system, technologies like Docker and Kubernetes can help in deploying the model at scale.

4.4.5. Justification of the chosen methods

Certainly, while the aforementioned techniques and approaches offer promising avenues for enhancing the performance and interpretability of the machine learning model for multi-class sentiment classification in English tweets related to therapies, it's important to note the constraints that guided the choices made in this thesis.

Firstly, time is a significant constraint. Each alternative approach not only requires implementation but also rigorous evaluation to ensure it contributes positively to the model's performance. This involves multiple iterations of training, validation, and testing, each of which can be time-consuming. Given the timeline of a bachelor's thesis, it's not feasible to explore every possible avenue of improvement.

Secondly, computational resources are another limiting factor. Techniques like deep learning and ensemble methods are computationally intensive and may require specialized hardware like GPUs for efficient execution. Even with such hardware, the training and evaluation of complex models can take a considerable amount of time. Given the resources at my disposal, it was more practical to focus on approaches that offer a good balance of performance and computational efficiency.

Therefore, the pipeline developed in this thesis represents a carefully considered approach that aims to maximize performance while being mindful of the time and computational resources available. Future work could certainly explore these alternative techniques, ideally with more time and better computational resources.

5. EVALUATION OF THE PROPOSAL

5.1. Evaluation Method and Metric

The evaluation of the sentiment classification models in this thesis is conducted using the micro-averaged F1-score over all three classes: positive, negative, and neutral. The F1-score is a widely used metric in machine learning for binary or multi-class classification problems, and it balances the trade-off between precision and recall. However, the standard F1-score doesn't take into account the class imbalance, which is a common issue in natural language processing tasks like sentiment analysis. This is where micro-averaging comes into play.

Micro-Averaged F1-Score: An Overview

The micro-averaged F1-score aggregates the contributions of all classes to compute the average metric. In micro-averaging, every individual instance (or tweet, in this case) is equally important, regardless of which class it belongs to. This is particularly useful when there is a class imbalance, as it gives a more holistic view of the model's performance across all classes.

Formula

The micro-averaged F1-score is calculated as follows:

$$\text{Micro F1-Score} = \frac{2 \times (\text{Micro Precision} \times \text{Micro Recall})}{(\text{Micro Precision} + \text{Micro Recall})}$$

Where:

- $\text{Micro Precision} = \frac{\text{Sum of True Positives across all classes}}{\text{Sum of True Positives + Sum of False Positives across all classes}}$
- $\text{Micro Recall} = \frac{\text{Sum of True Positives across all classes}}{\text{Sum of True Positives + Sum of False Negatives across all classes}}$

Why Micro-Averaged F1-Score?

1. **Class Imbalance**: In the context of sentiment analysis on tweets related to therapies, it's common to have imbalanced classes. Micro-averaging ensures that each instance contributes equally to the final score, thus mitigating the bias towards the majority class.

2. **Comprehensive Evaluation**: The micro-averaged F1-score provides a single metric that encapsulates both the precision and recall of the model across all classes. This makes it easier to compare different models or configurations.
3. **Real-world Relevance**: In practical applications like social media monitoring or customer feedback analysis, each individual tweet is important, regardless of its class. Micro-averaging aligns well with this perspective by treating each instance equally.
4. **Robustness**: The micro-averaged F1-score is less susceptible to the skewness of the dataset and provides a more robust performance measure, especially when the class distribution is not uniform.

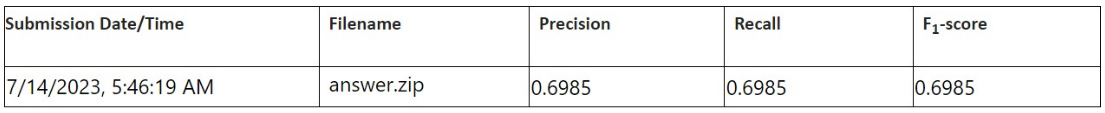
The micro-averaged F1-score serves as a robust and comprehensive metric for evaluating the performance of the sentiment classification models in this thesis. It not only accounts for the class imbalance but also provides a holistic view of how well the model is performing across all sentiment classes.

5.2. Model Results

5.3. Analysis of SMM4H 2023 Shared Tasks Results

The results obtained in the SMM4H 2023 [1] Shared Tasks for Task 2 (see figure 5.1) are quite promising and indicative of the effectiveness of the implemented pipeline. Below is a detailed analysis of the performance metrics and how they compare to the mean and median performance (see figure 5.2) among all teams.

5.3.1. Performance Metrics



Submission Date/Time	Filename	Precision	Recall	F_1-score
7/14/2023, 5:46:19 AM	answer.zip	0.6985	0.6985	0.6985

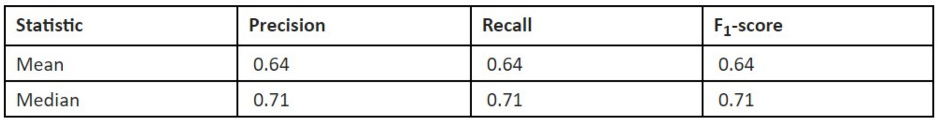
Fig. 5.1. Evaluation of the model: Results obtained from our model

1. **Precision**: The precision score of 0.6985 indicates that approximately 70% of the tweets classified by the model were correctly identified. Precision is a measure of how many of the identified positive cases were actually positive, and in this context, it shows the model's ability to correctly identify the sentiment of a tweet.
2. **Recall**: The recall score, also at 0.6985, signifies that the model was able to correctly identify approximately 70% of the actual positive cases from the total available positive cases. Recall is crucial in applications where failing to identify a

positive case could be problematic. In the context of sentiment analysis, a high recall ensures that most sentiments are correctly identified, which is vital for understanding public opinion or customer feedback.

3. **F1-Score**: The F1-score, which is the harmonic mean of precision and recall, also stands at 0.6985. This score is particularly important because it gives a single metric that balances both the precision and the recall. An F1-score close to 1 indicates excellent precision and recall, and a score of 0.6985 is quite promising, especially considering that this is a multi-class classification problem with its inherent complexities.

5.3.2. Comparison with Mean and Median



Statistic	Precision	Recall	F_1-score
Mean	0.64	0.64	0.64
Median	0.71	0.71	0.71

Fig. 5.2. Evaluation of the model: Mean and madian results from all participants in SMM4H 2023 Shared Tasks for Task 2

1. **Mean Performance**: The mean performance for precision, recall, and F1-score among all teams was 0.64. Our model's scores are above this mean value, indicating that the model is performing better than the average submission in all three metrics.

2. **Median Performance**: The median performance for precision, recall, and F1-score was 0.71. Our model's performance is very close to this median value, suggesting that it is competitive with the middlemost value of the dataset and is not skewed by outliers.

5.3.3. Interpretation

The results suggest that the model is both precise and sensitive, with an F1-score that is above the mean and close to the median of all submissions. This is a strong indicator of the model's robustness and its ability to generalize well on unseen data. The scores also validate the effectiveness of the chosen features, the fine-tuning of the model, and the overall pipeline architecture, including the use of advanced NLP and machine learning techniques.

The performance metrics from the SMM4H 2023 Shared Tasks [1] provide strong evidence of the model's effectiveness in multi-class sentiment classification. The model not only performs above average but also competes closely with the median, making it a robust and reliable solution for sentiment analysis in tweets related to therapies.

6. ALTERNATIVE SOLUTIONS EXPLORED

6.1. Using pre-trained LLMs

Language Model for Long Sequences (LLMs) are a subset of language models specifically designed to handle long sequences of text. Traditional language models like n-grams or even some neural network-based models like RNNs and LSTMs often struggle with long sequences due to limitations like the vanishing gradient problem or simply computational inefficiency. LLMs, on the other hand, are architected to efficiently process and understand long stretches of text, capturing the nuances, context, and relationships between words or sub-words even when they are far apart in the sequence.

Most Used LLMs

- **Transformers:** The Transformer architecture has become the cornerstone for most LLMs. It uses self-attention mechanisms to weigh the importance of different parts of the input text. Models like BERT, GPT-2, and T5 are based on this architecture.
- **BERT (Bidirectional Encoder Representations from Transformers):** BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context. It has been pre-trained on a large corpus of text and can be fine-tuned for specific tasks like sentiment analysis, text summarization, and more.
- **GPT (Generative Pre-trained Transformer):** Unlike BERT, which is bidirectional, GPT is unidirectional (specifically, autoregressive). It has been trained to predict the next word in a sequence, making it powerful for tasks like text generation.
- **XLNet:** This is an extension of the Transformer model, designed to learn from the sequence of words in all possible orders. This gives it an edge in understanding the context better.
- **ERNIE (Enhanced Representation through knowledge Integration):** ERNIE is designed to understand the semantics of words by taking into account their relationship with other words, thus making it efficient for tasks that require understanding the context between different entities in the text.

Utility for Sentiment Classification in Tweets

- **Context Understanding:** LLMs are excellent at understanding the context in which words or phrases are used. This is crucial for sentiment analysis, where the same word can have different sentiments based on the context.
- **Handling Sarcasm and Ambiguity:** Due to their deep architectures and self-attention mechanisms, LLMs are better equipped to understand nuances like sarcasm or ambiguity, common in social media text like tweets.
- **Efficient Fine-Tuning:** Models like BERT can be easily fine-tuned for the specific task of sentiment analysis, making them both effective and efficient.
- **Scalability:** LLMs are highly scalable. They can be trained on a massive corpus of data, capturing more nuanced patterns, and can also be pruned or distilled for deployment in resource-constrained environments.
- **Multi-Lingual Support:** Many LLMs come with multi-lingual variants that have been trained on text from multiple languages. This is particularly useful for analyzing tweets that might contain text from multiple languages or dialects.
- **Real-Time Analysis:** Due to their efficiency and the availability of optimized libraries and hardware acceleration, LLMs can be used for real-time sentiment analysis, which is often a requirement in social media analytics.

A LLM pipeline was developed for this task, but only for research purposes, as the necessary computational resources to use this techniques and train the model efficiently were not available for this project.

In the pipeline implemented for this thesis, as after an investigation process it has been concluded that it is the most appropriate ones for a sentiment classification task, we will focus on the implementation of DistilBERT. This model was chosen for its robustness, efficiency, and state-of-the-art performance in various natural language processing tasks, including sentiment analysis. Below is a detailed focus on this model and its utility in the pipelines:

As DistilBERT is a distilled version of its parent model, BERT, we will also review what BERT is and its more powerful characteristics.

BERT (Bidirectional Encoder Representations from Transformers)

- **Architecture:** BERT is based on the Transformer architecture and is designed to pre-train deep bidirectional representations. Unlike traditional unidirectional models, BERT takes into account the context from both the left and the right of a word, making it highly effective at understanding the semantics and sentiment of text.

- **Fine-Tuning:** One of the most powerful features of BERT is its ability to be fine-tuned on a specific task with a relatively small amount of data. In the pipeline, BERT was fine-tuned specifically for the task of multi-class sentiment classification on tweets related to therapies.
- **Contextual Understanding:** BERT's bidirectional context understanding is particularly useful for tweets, where the text is often short and laden with contextual information. It can understand the sentiment of words that may have different meanings based on the surrounding text.
- **Computational Efficiency:** While BERT is computationally intensive, its architecture allows for parallelization, making it feasible to run on available hardware within a reasonable time frame for this thesis.

DistilBERT

- **Lightweight:** DistilBERT is a distilled version of BERT, designed to have fewer parameters and require less computational power. Despite its smaller size, it retains most of the performance capabilities of BERT, making it an excellent choice for resource-constrained environments or when quick model training is required.
- **Fine-Tuning and Transfer Learning:** Like BERT, DistilBERT can also be fine-tuned on a specific task. In the pipeline, DistilBERT was employed for sentiment classification.
- **Speed:** One of the key advantages of using DistilBERT in the pipeline is its speed. It can be trained and deployed much faster than BERT, making it ideal for iterative development and real-time sentiment analysis.
- **Interoperability:** DistilBERT is fully compatible with BERT, meaning that models and tokenizers can be interchanged without any issues. This provides flexibility in the pipeline to switch between the models based on the specific requirements of a task or computational constraints.

Both BERT and DistilBERT offer a powerful set of features for sentiment analysis in tweets. Their ability to understand context, along with the flexibility to be fine-tuned for specific tasks, makes them highly effective for the pipelines implemented in this thesis. While BERT offers state-of-the-art performance, DistilBERT provides a faster and more resource-efficient alternative without significantly compromising on accuracy, hence, as we have considerable computational limitations, it is the chosen one for this solution that we are exploring.

6.1.1. Pipeline: Hugging Face Transformers

We developed a full, end-to-end pipeline to solve the task of this thesis [1]. This pipeline has been tested and it worked efficiently, but the results the obtained model provided (see figure 6.1) can not be taken as conclusive, as for completing the model training step we had to significantly reduce the number of data points in the dataset, as, after several attempts to train the model on the full, cleaned and tokenized dataset, it continued running after more than 5 hours of training.

Knowing that, we will provide a detailed description of the pipeline developed to show that the constructions of LLM based solutions are not too different to more traditional ones, as the main difference between them remains the computational resources needed to train the respective models.

In this pipeline, the focus is on leveraging the power of Hugging Face's Transformers library to perform multi-class sentiment classification on tweets. The library provides a plethora of pre-trained language models, including BERT, RoBERTa, and their variants. For this specific task, the DistilBertForSequenceClassification model is employed, which is a lighter version of the original BERT model but retains most of its capabilities. The notebook is structured into several key steps, each of which is crucial for the end-to-end pipeline.

The first step is to install all the required Python packages. This includes Pandas for data manipulation, Transformers for accessing pre-trained models, Torch for tensor operations and neural network functionalities, and scikit-learn for evaluation metrics. These libraries form the backbone of the pipeline, enabling data manipulation, model training, and evaluation.

Data preprocessing is a critical step in any machine learning pipeline. As illustrated in code snippet 6.1, the raw text data from tweets is tokenized, padded, or truncated to a fixed length, and then converted into integer IDs. This is done using the DistilBertTokenizer, which is part of the Transformers library. The labels are also mapped to integers ('positive' to 0, 'negative' to 1, and 'neutral' to 2). Tokenization is the process of converting a text into tokens (words, subwords, or symbols), and it's a fundamental step in NLP. Padding and truncation ensure that all text sequences have the same length, which is necessary for batching during model training.

CODE SNIPPET 6.1. Preprocess the Data

```python
def load_data(file_path):
    df = pd.read_csv(file_path)

    # Tokenize the text, pad to a fixed length, and convert to integer IDs
    inputs = tokenizer(df['text'].tolist(), padding='max_length', truncation=True,
        max_length=128, return_tensors='pt')

    # Convert the labels to integers
    label_mapping = {'positive': 0, 'negative': 1, 'neutral': 2}
    labels = torch.tensor(df['label'].map(label_mapping).tolist())

    # Create a TensorDataset from the tokenized inputs and labels
    dataset = TensorDataset(inputs['input_ids'], inputs['attention_mask'], labels)

    return dataset
```

The next step is to load the pre-trained DistilBert model and its corresponding tokenizer. The model is specifically configured for sequence classification with three output labels. The AdamW optimizer and the Cross-Entropy Loss function are also defined in this step (see code 6.2). Pre-trained models like DistilBert have already learned useful features from large text corpora, and fine-tuning these models on a specific task often yields better performance compared to training a model from scratch.

CODE SNIPPET 6.2. Load the Pre-trained Model and Tokenizer

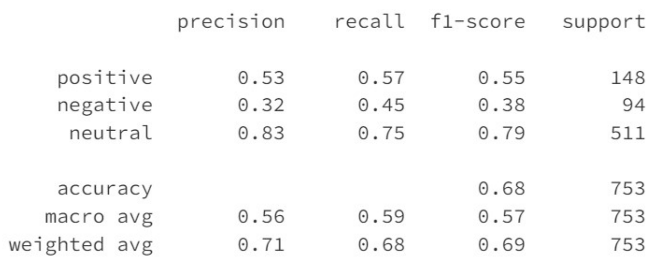
```
# Define the DistilBert model for sequence classification
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased
    ', num_labels=3)
model.train()

# Define the optimizer and loss function
optimizer = AdamW(model.parameters(), lr=5e-5)
criterion = torch.nn.CrossEntropyLoss()
```

Fine-tuning is the process of slightly adjusting the pre-trained model parameters based on the specific task at hand, which in this case is multi-class sentiment classification. As showed in 6.3, the model is trained for three epochs using a batch size of 32. The loss is calculated using the Cross-Entropy Loss function, and the model parameters are updated using the AdamW optimizer. Fine-tuning is essential for adapting a pre-trained model to the nuances of the specific data it will be handling.

CODE SNIPPET 6.3. Fine-tune the Model on the Training Data

```
# Fine-tune the model on the training data
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
for epoch in range(3):
    for batch in train_loader:
        input_ids, attention_mask, labels = batch
        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask)
        loss = criterion(outputs.logits, labels)
        loss.backward()
        optimizer.step()
```

After fine-tuning, the model is evaluated on a separate validation dataset. The model's predictions (see figure 6.1) are compared with the actual labels to generate a classification report, which includes metrics like precision, recall, and F1-score for each class. These metrics provide a comprehensive view of how well the model is performing across different sentiment classes.



```
              precision    recall  f1-score   support

    positive       0.53      0.57      0.55       148
    negative       0.32      0.45      0.38        94
     neutral       0.83      0.75      0.79       511

    accuracy                           0.68       753
   macro avg       0.56      0.59      0.57       753
weighted avg       0.71      0.68      0.69       753
```

Fig. 6.1. LLM Pipeline: Results obtained with a DistilBERT model

**These evaluation metrics are obtained, as mentioned before, after training the model on a smaller dataset, hence, we can not compare the results to the ones we obtained from the previous solution.*

Finally, the model is used to make predictions on the test data. These predictions are then saved the specific submission format stated by the task description [1], even tough this predictions were not the ones finally submitted to the smm4h2023 competition website. This step is crucial for assessing the model's performance on unseen data and is often required in real-world applications and competitions.

This sections provides a detailed guide on how to use the Hugging Face Transformers library for multi-class sentiment classification. It covers everything from installing necessary packages to fine-tuning a pre-trained model and evaluating its performance. The use of a pre-trained model like DistilBert offers a powerful yet computationally efficient approach for this task.

6.2. Using Google Cloud Natural Language API

6.2.1. What it is?

The Google Cloud Natural Language API [11] is a robust, cloud-based service designed to provide a comprehensive suite of natural language processing (NLP) capabilities. Leveraging Google's state-of-the-art machine learning algorithms and pre-trained models, this API offers a wide range of features that can be seamlessly integrated into various applications to enhance their natural language understanding (NLU).

6.2.2. Core Features of Cloud Natural Language API

1. **Entity Recognition**: This feature allows the identification of various entities within a text, such as names of people, places, organizations, and even concepts. This is particularly useful for tasks that require understanding the context in which certain keywords appear.
2. **Sentiment Analysis**: One of the most critical features for our task, sentiment analysis enables the API to determine the emotional tone behind a piece of text. It can classify text as positive, negative, or neutral, which is invaluable for understanding public sentiment or customer opinions.
3. **Information Extraction**: This feature can pull out specific pieces of information from a text, such as dates, times, and prices. This is particularly useful for tasks that require data mining or information retrieval.
4. **Question Answering**: The API can also answer questions based on the text it analyzes, which can be useful for chatbot applications or automated customer service solutions.
5. **Integrated REST API**: The API is accessible via restful calls, allowing for easy integration into existing systems. Text data can either be uploaded directly in the API request or fetched from Google Cloud Storage.

6.2.3. Why It Will a Great Solution for Our Task

For the specific task of multi-class sentiment classification of therapies in English tweets, the Sentiment Analysis feature of the Cloud Natural Language API is particularly relevant. Here's why:

1. **Highly Accurate**: Given that the API leverages Google's pre-trained models, it offers a high level of accuracy in sentiment classification, which is crucial for the reliability of our research.
2. **Scalability**: Being cloud-based, the API can handle a large volume of text data, making it suitable for analyzing extensive datasets of tweets.
3. **Real-time Analysis**: The API is capable of performing real-time sentiment analysis, which is essential for tasks that require immediate insights, such as monitoring public reaction to a new therapy.
4. **Ease of Integration**: The REST API ensures that integrating this service into our existing pipeline is straightforward, allowing us to focus more on the analysis and less on the technicalities of API calls.

5. **Customization**: While we are primarily interested in the Sentiment Analysis feature, the API offers a range of other NLP capabilities that we could easily leverage for more in-depth analysis in the future.

The Google Cloud Natural Language API, with its Sentiment Analysis feature, serves as a powerful tool for our task. Its high accuracy, scalability, and ease of integration make it an excellent choice for sentiment classification in the complex and nuanced domain of therapies discussed in tweets.

6.3. Example of solution workflow

While the Google Cloud Natural Language API offers a robust set of features and high accuracy, it's important to note that the service is not free. Due to budget constraints, we are unable to utilize this API as the primary solution for our task. However, to demonstrate its capabilities, we will showcase a sample analysis using some tweets from our dataset. To access the Google Cloud Platform, I registered with my student account (from my enrollment in the "Professional Data Engineer for Google Cloud" training path for the official Google's certification), which provide access to a set of laboratories that serve as introduction to various GCP features, and I used one of them to test the Natural Language API over several tweets from the dataset. This will serve as a proof-of-concept to illustrate what the API can offer in terms of sentiment analysis and natural language understanding.

By doing so, we aim to provide a glimpse into the potential benefits of integrating such a powerful tool into our pipeline, while also acknowledging the financial limitations that prevent us from using it extensively in this research.

How to use it

The first step if to create an API key, which can be performed through the Google Cloud Console, just using a few, simple commands (see 6.4)

Firstly, the environment variable PROJECT_ID is initialized with the value provided by Google Cloud Platform (GCP). This serves as a unique identifier for the project and is essential for subsequent API calls.

Next, an account is established with the requisite permissions to interact with the Natural Language API. This is a crucial step to ensure secure and authorized access to the cloud-based services.

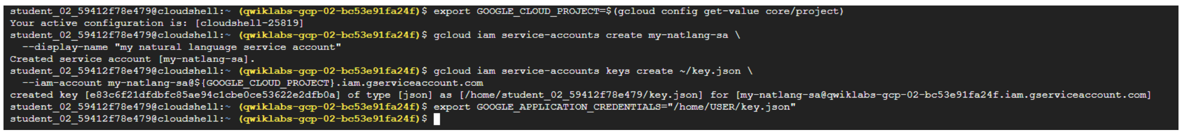
Following account creation, credentials are generated to authenticate the newly created service account. These credentials are saved in a JSON file located at /key.json. The generation of this file is executed via a specific command line instruction.

Subsequently, the GOOGLE_APPLICATION_CREDENTIALS environment variable is set (see figure 6.2). This variable points to the full path of the previously generated

JSON credentials file, as indicated in the output of the preceding command.

CODE SNIPPET 6.4. Step 1: Create an API key

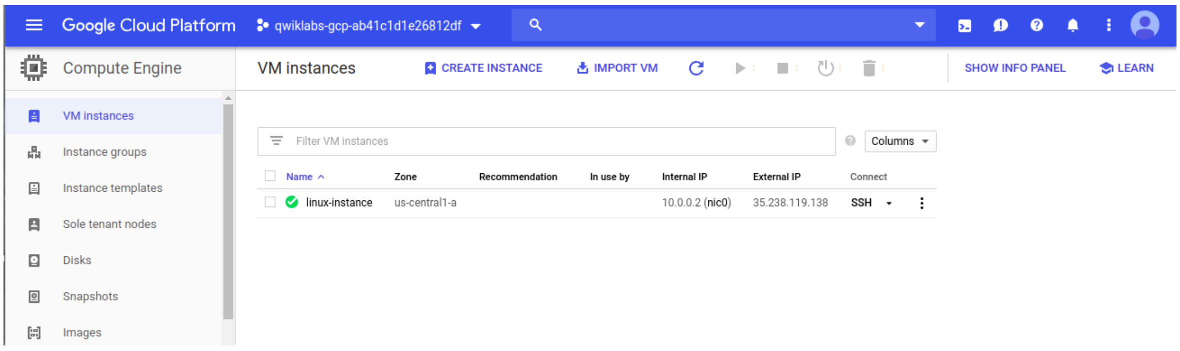
```
export GOOGLE_CLOUD_PROJECT=$(gcloud config get-value core/project)

gcloud iam service-accounts create my-natlang-sa \
  --display-name "trial for bachelor's thesis"

gcloud iam service-accounts keys create ~/key.json \
  --iam-account my-natlang-sa@${GOOGLE_CLOUD_PROJECT}.iam.gserviceaccount.com

export GOOGLE_APPLICATION_CREDENTIALS="/home/USER/key.json"
```



```
student_02_59412f78e479@cloudshell:~ (qwiklabs-gcp-02-bc53e91fa24f)$ export GOOGLE_CLOUD_PROJECT=$(gcloud config get-value core/project)
Your active configuration is: [cloudshell-25819]
student_02_59412f78e479@cloudshell:~ (qwiklabs-gcp-02-bc53e91fa24f)$ gcloud iam service-accounts create my-natlang-sa \
  --display-name "my natural language service account"
Created service account [my-natlang-sa].
student_02_59412f78e479@cloudshell:~ (qwiklabs-gcp-02-bc53e91fa24f)$ gcloud iam service-accounts keys create ~/key.json \
  --iam-account my-natlang-sa@${GOOGLE_CLOUD_PROJECT}.iam.gserviceaccount.com
created key [e83c6f21dfdbfc85ae94c1cbe0ce53622e2dfb0a] of type [json] as [/home/student_02_59412f78e479/key.json] for [my-natlang-sa@qwiklabs-gcp-02-bc53e91fa24f.iam.gserviceaccount.com]
student_02_59412f78e479@cloudshell:~ (qwiklabs-gcp-02-bc53e91fa24f)$ export GOOGLE_APPLICATION_CREDENTIALS="/home/USER/key.json"
student_02_59412f78e479@cloudshell:~ (qwiklabs-gcp-02-bc53e91fa24f)$ 
```

Fig. 6.2. Google Cloud Natural Language API: Creating an API key

To proceed with the next steps, a connection is established to the allocated Compute Engine instance. This is accessed via the navigation menu under the 'Compute Engine' section (see figure 6.3). Upon locating the provisioned Linux instance, an SSH session is initiated by clicking on the SSH button. This interactive shell session is maintained throughout the duration of the lab.



Name	Zone	Recommendation	In use by	Internal IP	External IP	Connect
linux-instance	us-central1-a			10.0.0.2 (nic0)	35.238.119.138	SSH

Fig. 6.3. Google Cloud Natural Language API: Connecting to compute engine via SSH

Once inside the SSH session, the Natural Language API's sentiment analysis feature is tested using a few tweets from the dataset. This is executed through a specific gcloud command, using it one time for each sentence (tweet) (see commands in 6.5 (Tweet 1), 6.7 (Tweet 2) and 6.9 (Tweet 3)) that we want to analyze.

CODE SNIPPET 6.5. Tweet 1

```
gcloud ml language analyze-sentiment --content=
"@Sassychickie @kelly_rdc Fentanyl, OxyContin and Oxycodone! I ve had 2 back
    surgeries. Never again!!!" > result.json

cat result.json
```

Finally, to inspect the output stored in the result.json file (see json results in 6.6 (Tweet 1), 6.8 (Tweet 2) and 6.10 (Tweet 3)), a command is run to preview its contents. This allows for a preliminary assessment of the API's entity analysis capabilities.

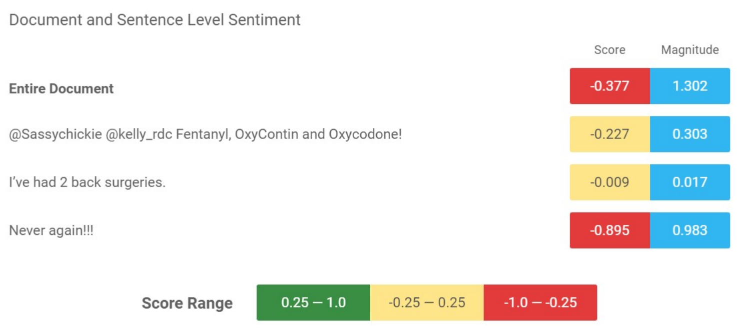
CODE SNIPPET 6.6. Tweet 1 - Results

```json
{
  "documentSentiment": {
    "magnitude": 1.302,
    "score": -0.377
  },
  "language": "en",
  "sentences": [
    {
      "text": {
        "content": "@Sassychickie @kelly_rdc Fentanyl, OxyContin and Oxycodone!",
        "beginOffset": -1
      },
      "sentiment": {
        "magnitude": 0.303,
        "score": -0.227
      }
    }
    {
      "text": {
        "content": "Ive had 2 back surgeries.",
        "beginOffset": 52
      },
      "sentiment": {
        "magnitude": 0.017,
        "score": -0.009
      }
    }
    {
      "text": {
        "content": "Never again!!!",
        "beginOffset": 76
      },
      "sentiment": {
        "magnitude": 0.983,
        "score": -0.895
      }
    }
  ]
}
```

The documentSentiment field shows that the overall sentiment of the document is negative, with a score of -0.377 and a magnitude of 1.302.

The sentences array provides sentiment analysis for each individual sentence in the text.

We could also obtain a more visual display of the results (see images 6.4 (Tweet 1), 6.5 (Tweet 2) and 6.6 (Tweet 3)) through the google cloud interface, after saving the json files


Document and Sentence Level Sentiment

	Score	Magnitude
Entire Document	-0.377	1.302
@Sassychickie @kelly_rdc Fentanyl, OxyContin and Oxycodone!	-0.227	0.303
I've had 2 back surgeries.	-0.009	0.017
Never again!!!	-0.895	0.983

Score Range	0.25 — 1.0	-0.25 — 0.25	-1.0 — -0.25

Fig. 6.4. Google Cloud Natural Language API: Tweet 1 analysis

CODE SNIPPET 6.7. Tweet 2

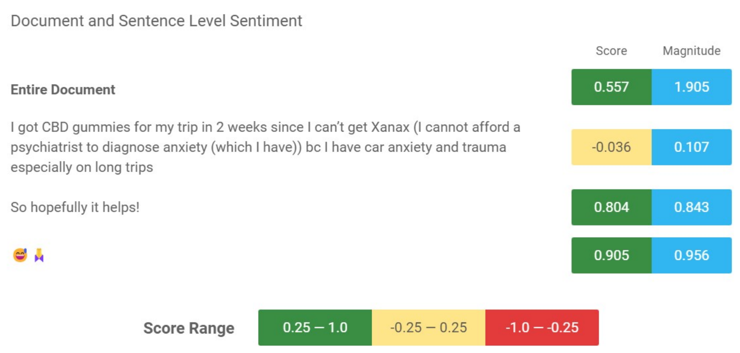
```
gcloud ml language analyze-sentiment --content=
"I got CBD gummies for my trip in 2 weeks since I cant get Xanax (I cannot afford a
    psychiatrist to diagnose anxiety (which I have)) bc I have car anxiety and trauma
     especially on long trips

So hopefully it helps!" > result.json

cat result.json
```

CODE SNIPPET 6.8. Tweet 2 - Results

```
{
  "documentSentiment": {
    "magnitude": 1.905,
    "score": 0.557
  },
  "language": "en",
  "sentences": [
    {
      "text": {
        "content": "I got CBD gummies for my trip in 2 weeks since I cant get Xanax (
            I cannot afford a psychiatrist to diagnose anxiety (which I have)) bc I
            have car anxiety and trauma especially on long trips",
        "beginOffset": -1
      },
      "sentiment": {
        "magnitude": 0.107,
        "score": -0.036
      }
    }
    {
      "text": {
        "content": "So hopefully it helps!",
        "beginOffset": 106
      },
      "sentiment": {
        "magnitude": 0.843,
        "score": 0.804
      }
    }
```

```
      {
        "text": {
          "content": "Emoji-Emoji",
          "beginOffset": 120
        },
        "sentiment": {
          "magnitude": 0.956,
          "score": 0.905
        }
      }
    ]
  }
```

The documentSentiment field shows that the overall sentiment of the document is positive, with a score of 0.557 and a magnitude of 1.905.



Document and Sentence Level Sentiment

	Score	Magnitude
Entire Document	0.557	1.905
I got CBD gummies for my trip in 2 weeks since I can't get Xanax (I cannot afford a psychiatrist to diagnose anxiety (which I have)) bc I have car anxiety and trauma especially on long trips	-0.036	0.107
So hopefully it helps!	0.804	0.843
🥰🙏	0.905	0.956

Score Range 🟩 0.25 — 1.0 🟨 -0.25 — 0.25 🟥 -1.0 — -0.25

Fig. 6.5. Google Cloud Natural Language API: Tweet 2 analysis

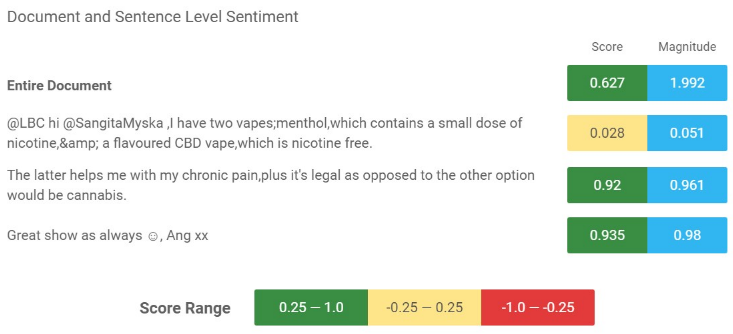
CODE SNIPPET 6.9. Tweet 3

```
gcloud ml language analyze-sentiment --content=
"@LBC hi @SangitaMyska ,I have two vapes;menthol,which contains a small dose of
    nicotine,&amp; a flavoured CBD vape,which is nicotine free.The latter helps me
    with my chronic pain,plus it's legal as opposed to the other option would be
    cannabis.Great show as always, Ang xx" > result.json

cat result.json
```

CODE SNIPPET 6.10. Tweet 3 - Results

```
{
  "documentSentiment": {
    "magnitude": 1.992,
    "score": 0.627
  },
  "language": "en",
  "sentences": [
    {
      "text": {
        "content": "@LBC hi @SangitaMyska ,I have two vapes;menthol,which contains a
            small dose of nicotine,&amp; a flavoured CBD vape,which is nicotine free
            .",
        "beginOffset": -1
      },
      "sentiment": {
        "magnitude": 0.051,
        "score": 0.028
      }
    }
    {
      "text": {
        "content": "The latter helps me with my chronic pain,plus it's legal as opposed to the other option would be cannabis.",
        "beginOffset": 75
      },
      "sentiment": {
        "magnitude": 0.961,
        "score": 0.92
      }
    }
    {
      "text": {
        "content": "Great show as always, Ang xx",
        "beginOffset": 108
      },
      "sentiment": {
        "magnitude": 0.98,
        "score": 0.935
      }
    }
  ]
}
```

The documentSentiment field shows that the overall sentiment of the document is positive, with a score of 0.627 and a magnitude of 1.992.



Document and Sentence Level Sentiment

	Score	Magnitude
Entire Document	0.627	1.992
@LBC hi @SangitaMyska ,I have two vapes;menthol,which contains a small dose of nicotine,&amp; a flavoured CBD vape,which is nicotine free.	0.028	0.051
The latter helps me with my chronic pain,plus it's legal as opposed to the other option would be cannabis.	0.92	0.961
Great show as always ☺, Ang xx	0.935	0.98

Score Range 0.25 — 1.0 | -0.25 — 0.25 | -1.0 — -0.25

Fig. 6.6. Google Cloud Natural Language API: Tweet 3 analysis

7. CONCLUSIONS AND FUTURE ENHANCEMENTS

7.1. Conclusion

The journey of this thesis has been both challenging and rewarding, offering a comprehensive exploration into the realm of multi-class sentiment classification of therapies in English tweets. Utilizing a blend of machine learning, natural language processing, and lexicalized language models, this research has delved deep into the complexities and nuances of sentiment analysis in the healthcare domain.

The initial stages of the research involved a meticulous exploratory data analysis (EDA) to understand the dataset's characteristics, distribution, and inherent patterns. This step was crucial, not only for feature engineering but also for gaining insights into the nature of public sentiment towards different therapies. The EDA revealed several key aspects, such as the prevalence of certain sentiments and the common terms associated with each therapy, which were instrumental in shaping the subsequent stages of the project.

The pipeline architecture, initially developed in Jupyter Notebooks for a more visual and interactive approach, was later replicated in a PyCharm project for scalability and faster execution. This dual-phase development process ensured both a deep understanding of each step and the efficiency of the overall pipeline. The pipeline was divided into distinct steps, including data cleaning, feature engineering, hyperparameter tuning, and model evaluation, each serving a specific purpose and contributing to the model's performance.

Various machine learning algorithms were explored, with RandomForestClassifier emerging as the most effective for this particular task. The model was evaluated using a micro-averaged F1-score, achieving a score of 0.6985, which is notably higher than the mean score of 0.64 among all teams in the SMM4H 2023 Shared Tasks competition. This not only validates the effectiveness of the chosen methods but also places the research in a competitive standing within the broader scientific community.

While the primary focus was on traditional machine learning models, the research also ventured into the fascinating world of lexicalized language models (LLMs) like BERT and Hugging Face Transformers. These models offer a promising avenue for future research, given their ability to capture the contextual nuances of language, although they were not the primary models used due to time and computational constraints.

Moreover, the research briefly explored the capabilities of Google's Cloud Natural Language API as an alternative approach to sentiment analysis. Although the API's usage was limited due to budget constraints, its potential effectiveness was demonstrated through a small sample of tweets from the dataset.

In conclusion, this thesis not only achieves its goal of developing an effective model for multi-class sentiment classification but also contributes to the broader understanding of applying machine learning and NLP techniques in healthcare sentiment analysis. The methodologies employed, the insights gained, and the challenges overcome all serve to make this research a valuable addition to the existing body of knowledge. Future work could involve the integration of more advanced LLMs, exploring ensemble methods, or expanding the scope to include additional data sources and languages.

The success of this research serves as a testament to the untapped potential of machine learning and natural language processing in healthcare analytics, offering a foundation upon which future research can build.

7.2. Future considerations

As with any research endeavor, the scope of this thesis is bound by certain limitations, which in turn offer avenues for future exploration and development. Below are some future considerations that could extend the impact and applicability of this work:

While this thesis did touch upon the potential of LLMs like BERT and Hugging Face Transformers, a more in-depth study could be conducted to fully realize their capabilities in sentiment classification tasks.

The thesis primarily focused on RandomForestClassifier for sentiment classification. Future research could explore ensemble methods that combine the strengths of multiple machine learning algorithms to potentially improve classification performance.

The current research is limited to English tweets. Extending the model to other languages and cultures could offer a more comprehensive understanding of global sentiments towards different therapies.

The model developed in this thesis works on a static dataset. Adapting the model for real-time sentiment analysis could have practical applications, such as monitoring public opinion during a healthcare crisis or after the release of a new therapy.

While the Google Cloud Natural Language API showed promise, its cost was a limiting factor. Future work could explore cost-effective or open-source cloud-based solutions for sentiment analysis.

This thesis employed a set of features deemed to be effective based on EDA. However, the landscape of NLP is ever-evolving, and new feature extraction techniques could be incorporated to improve the model's performance.

Future studies could also look into how the sentiments towards therapies correlate with real-world outcomes, such as patient satisfaction or therapy efficacy, to make the research more actionable.

By addressing these considerations, future research can build upon the strong foundation laid by this thesis, driving the field of healthcare sentiment analysis towards new horizons.

8. ANNEX

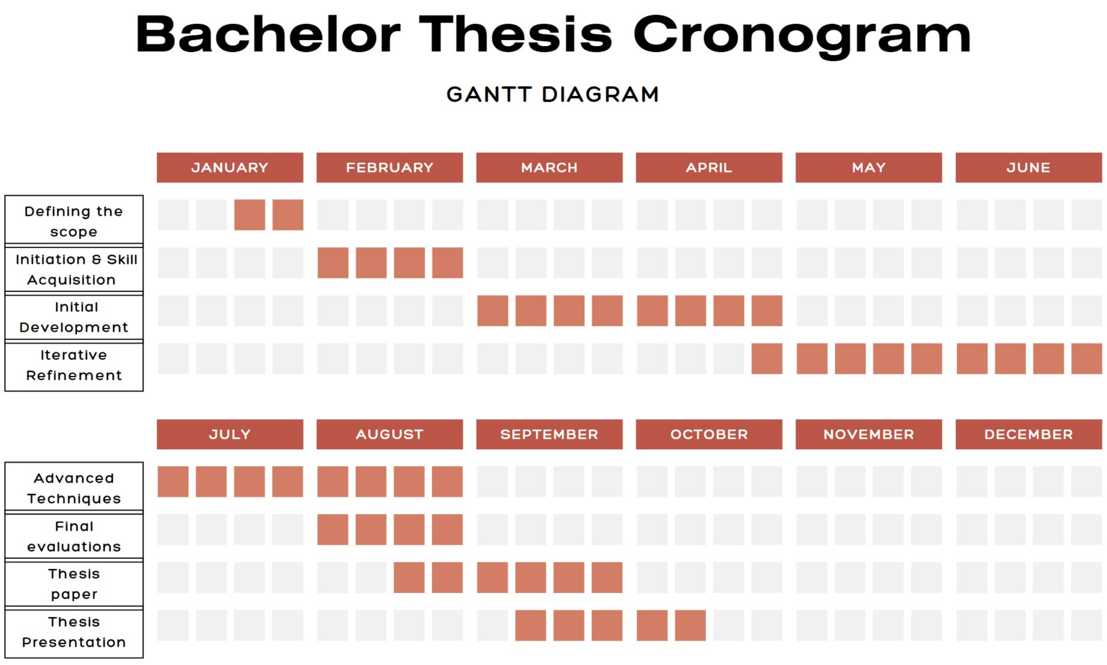


Fig. 8.1. Gantt Diagram for Bachelor’s Thesis Cronogram

9. GLOSSARY

AI - Artificial Intelligence

API - Application Programming Interface

BERT - Bidirectional Encoder Representations from Transformers

CPU - Central Processing Unit

CSV - Comma-Separated Values

EDA - Exploratory Data Analysis

GCP - Google Cloud Platform

GPU - Graphics Processing Unit

IDE - Integrated Development Environment

JSON - JavaScript Object Notation

LLM - Lexicalized Language Models

NLP - Natural Language Processing

NLU - Natural Language Understanding

RAM - Random Access Memory

REST - Representational State Transfer

SMM4H - Social Media Mining for Health

SSH - Secure Shell

TF-IDF - Term Frequency-Inverse Document Frequency

URL - Uniform Resource Locator

VADER - Valence Aware Dictionary and Sentiment Reasoner

BIBLIOGRAPHY

[1] Y. Guo, “Task 2: Smm4h-2023 - multi-class classification of sentiment associated with therapies in english tweets,” 2023.

[2] T. Joachims, “Text categorization with support vector machines: Learning with many relevant features,” *In ECML-98*, 1998.

[3] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” *In Advances in neural information processing systems*, 2013.

[4] J. Pennington, R. Socher, and C. Manning, “Glove: Global vectors for word representation,” *In EMNLP*, 2014.

[5] S. Wang and C. D. Manning, “Baselines and bigrams: Simple, good sentiment and topic classification,” *In ACL*, 2012.

[6] X. Li, L. Peng, and A. C. Konig, “Text classification improved by integrating bidirectional lstm with two-dimensional max pooling,” *In COLING*, 2017.

[7] Q. Liu, H. Jiang, S. Wei, Z. Ling, and Y. Hu, “Twitter sentiment analysis with deep convolutional neural networks,” *In SIGKDD*, 2015.

[8] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *In NAACL*, 2018.

[9] C. Sun, L. Huang, and X. Qiu, “Utilizing bert for aspect-based sentiment analysis via constructing auxiliary sentence,” *In NAACL-HLT*, 2019.

[10] Y. Liu *et al.*, “Roberta: A robustly optimized bert pretraining approach,” *arXiv preprint arXiv:1907.11692*, 2019.

[11] *GCP Natural Language API*, Accessed: 2023. [Online]. Available: https://cloud.google.com/natural-language?hl=es-419.