

FACULTY OF ENGINEERING OF THE UNIVERSITY OF PORTO

Automated User Feedback Mining for Software Requirements Elicitation - A Case Study in the Streaming Industry

Melissa Moreira da Silva



Master in Informatics and Computing Engineering

Supervisor: Prof. João Pascoal Faria

July 8, 2024

Automated User Feedback Mining for Software Requirements Elicitation - A Case Study in the Streaming Industry

Melissa Moreira da Silva

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

President: Prof. Rui Maranhão

Examiner: Prof. João Miguel Lobo Fernandes

Supervisor: Prof. João Pascoal Faria

July 8, 2024

Resumo

A Engenharia de Requisitos (ER) é uma fase crítica do ciclo de vida do desenvolvimento de software e é crucial para garantir o sucesso do projeto e a satisfação das partes interessadas. Apesar da sua importância, a ER enfrenta frequentemente desafios significativos, tais como requisitos ambíguos, envolvimento insuficiente das partes interessadas e a natureza evolutiva das necessidades dos utilizadores. Esta investigação aborda estas questões através da automatização de um processo abrangente de recolha de feedback das redes sociais, da extração de requisitos candidatos e do desenvolvimento de uma plataforma de visualização intuitiva para apresentar estes requisitos às partes interessadas e aos engenheiros de requisitos.

Os dados relativos às plataformas de serviços de streaming foram extraídos de quatro plataformas de redes sociais (**Reddit**, **Twitter**, **iTunes** e **Google Play**) utilizando web crawlers e APIs, garantindo um conjunto de dados diversificado e extenso que reflecte uma vasta gama de opiniões e experiências dos utilizadores. Foram utilizadas técnicas avançadas de Processamento de Linguagem Natural para pré-processar os dados em bruto, garantindo que os dados estavam limpos e adequados para análise, com atributos e características adicionais úteis para o processo de classificação. Foram testados vários classificadores de aprendizagem automática para determinar o método mais eficaz para classificar o feedback como relevante ou irrelevante. O feedback relevante foi posteriormente processado para criar características adequadas à visualização utilizando a sumarização e o agrupamento de texto, o que ajuda a identificar as necessidades e sugestões dos utilizadores. Os requisitos extraídos e o feedback são depois visualizados através de uma interface intuitiva e interactiva, permitindo que os intervenientes interpretem e atuem facilmente sobre a informação, criando problemas.

Esta plataforma foi validada pelas partes interessadas da MOG Technologies, onde esta investigação foi efetuada. O estudo dá vários contributos significativos para a literatura: ao contrário dos estudos puramente académicos, este trabalho integra a validação prática de peritos da indústria, garantindo que a plataforma satisfaz tanto o rigor académico como as necessidades da indústria do mundo real, colmatando eficazmente a lacuna entre a teoria e a aplicação. A extração automatizada de dados aumenta a eficiência e a escalabilidade, substituindo a recolha manual de dados e garantindo uma cobertura abrangente em diversas plataformas e línguas. A análise de dados de quatro plataformas proporciona uma perspetiva mais abrangente do que os estudos que se centram em menos plataformas. Foram também utilizadas técnicas inovadoras para processar o feedback dos utilizadores, como a utilização de Modelos de Linguagem de Grande Dimensão para uma sumarização eficaz. Além disso, implementámos o nosso próprio algoritmo de agrupamento. As funcionalidades da plataforma - tais como a agregação de dados, a visualização e a criação de problemas - fornecem informações accionáveis através de uma interface de fácil utilização, respondendo ao desafio de traduzir as informações em resultados accionáveis. Além disso, ao concentrar-se na plataforma Vizzi da MOG, o estudo capta indiretamente o feedback dos utilizadores de vários serviços de streaming, introduzindo um novo método de agregação e filtragem de feedback que não é explorado na literatura existente

Abstract

Requirement Engineering (RE) is a critical phase in the software development life cycle and is essential for ensuring project success and stakeholder satisfaction. Despite its importance, RE often faces significant challenges, such as ambiguous requirements, insufficient stakeholder engagement, and the evolving nature of user needs. This research addresses these issues by automating a comprehensive process of collecting feedback from social media, extracting candidate requirements, and developing an intuitive visualization platform to present these requirements to stakeholders and requirements engineers.

Data regarding streaming services platforms was extracted from four social media platforms (**Reddit**, **Twitter**, **iTunes**, and **Google Play**) using web crawlers and APIs, ensuring a diverse and extensive dataset that reflects a wide range of user opinions and experiences. Advanced Natural Language Processing techniques were employed to preprocess the raw data, ensuring the data was clean and suitable for analysis, with additional attributes and features useful for the classification process. Various machine learning classifiers were tested to determine the most effective method for classifying the feedback as relevant or irrelevant. Relevant feedback was further processed to create features suitable for visualization using text summarization and clustering, which helps identify user needs and suggestions. The extracted requirements and feedback are then visualized through an intuitive and interactive interface, allowing stakeholders to easily interpret and act upon the information by creating issues.

This platform was validated by stakeholders from MOG Technologies, where this investigation was conducted. The study makes several significant contributions to the literature: unlike purely academic studies, this work integrates practical validation from industry experts, ensuring that the platform meets both academic rigor and real-world industry needs, effectively bridging the gap between theory and application. The automated data extraction enhances efficiency and scalability, replacing manual data collection and ensuring comprehensive coverage across diverse platforms and languages. Analyzing data from four platforms provides a more comprehensive perspective than studies focusing on fewer platforms. Innovative techniques were also employed for processing user feedback, such as leveraging Large Language Models for effective summarization. Additionally, we implemented a custom clustering algorithm. The platform's features—such as data aggregation, visualization, and issue creation—deliver actionable insights through a user-friendly interface, addressing the challenge of translating insights into actionable outcomes.

Keywords: Requirements Engineering, Social Media, Mining Requirements, User Feedback.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Professor João Pascoal Faria, for his continuous support and guidance throughout this endeavor.

I would also like to extend my heartfelt thanks to MOG Technologies, particularly Hugo Neves, for providing me with the opportunity to develop my research in a stimulating environment.

To my parents, thank you for your love, understanding, and encouragement. Your constant support has been my greatest strength, and I am deeply grateful for everything you have done to help me reach this milestone.

To my friends, your unwavering faith in my success and constant support have been my guiding lights. It's nice to have a friend, and I'm grateful for each of you. Your presence has made even the most challenging days brighter. Please don't ever become a stranger. To those who helped me escape my coding dungeon and enjoy life outside of thesis land: I couldn't have done it without you. I had the time of my life fighting dragons with you. Thank you for being the incredible friends that you are. Your support, encouragement, and companionship have made all the difference. Long live all the magic we made.

With heartfelt gratitude,

Melissa Silva

*“Woah. How did we do that?
Magic.
No, for real.
Skill.
Definitely not that one.”*

Matthew and Stephanie Patrick

Contents

1	Introduction	1
1.1	Context and Motivation	1
1.2	Objectives	3
1.3	Innovative Aspects and Challenges	4
1.3.1	Gathering Indirect Feedback	4
1.3.2	Automating Process	4
1.3.3	Practical Validation	5
1.4	Document Structure	5
2	Requirements Engineering and Social Media	6
2.1	Requirements Engineering	6
2.2	Social Media	8
3	State of the Art	11
3.1	Research Methodology	11
3.1.1	Research Questions	11
3.1.2	Literature Search and Selection	12
3.2	Manual Extraction of Requirements from Social Networks	15
3.3	Automatic Extraction of Requirements from Social Media	18
3.3.1	Classification Algorithms	18
3.3.2	Categories	18
3.3.3	Preprocessing	19
3.3.4	Sentiment	23
3.3.5	Multi-Labeling versus Single	24
3.3.6	Feature Engineering	25
3.3.7	Multilingual	27
3.3.8	Data Imbalance	28
3.3.9	Grouping	28
3.3.10	Summarization	30
3.3.11	Training and Evaluation	31
3.4	Visualization of Candidate Requirements from Social Media	32
3.5	Conclusion	37
4	Analysis of Social Media Feedback	40
4.1	Data Extraction	40
4.1.1	Mobile Stores	41
4.1.2	Reddit	41
4.1.3	Twitter	42

4.1.4	Statistical sampling	43
4.1.5	New Strategy regarding Reddit	43
4.1.6	New version of Twitter	44
4.1.7	Search for “bugs”	45
4.2	Preprocessing	46
4.3	Training and Evaluating Classification Models	49
4.3.1	Predicting Relevance	49
4.3.2	Categorizing Posts	53
4.4	Summarization	55
4.4.1	Evaluation	55
4.5	Grouping	57
4.6	Conclusion	59
5	Platform Development	60
5.1	User Stories	60
5.2	Architecture	60
5.3	Features and User Interface	61
5.3.1	Landing Page	61
5.3.2	Dashboard Page	62
5.3.3	Suggested Feedback Page	64
5.3.4	Deleted Feedback Page	72
5.3.5	Accepted Feedback Page	73
5.3.6	Exported Feedback Page	79
5.4	Conclusion	81
6	Validation	82
6.1	User Feedback Evaluation	82
6.1.1	Questionnaire Design	82
6.1.2	Results	83
6.1.3	Participants’ Feedback	85
6.2	Active Usage Monitoring	86
6.2.1	Experiments and Results	86
6.2.2	Discussion	89
7	Conclusions and Future Work	91
A	Comprehensive Analysis of Research Papers	94
B	Machine Learning and NLP Techniques	95
B.1	Machine Learning	95
B.1.1	Classification Algorithms	95
B.1.2	Data Imbalance	96
B.1.3	Clustering	96
B.1.4	Training and Evaluation	97
B.2	NLP Techniques	99
B.2.1	Stemming	99
B.2.2	Sentiment	100
B.2.3	N-Gram Extraction	101
B.2.4	BoW	101

B.2.5	TF-IDF	102
B.2.6	Word and Sentence Embedding	102
B.2.7	Topic Modeling	104
B.2.8	Summarization	104
C	User Stories	106
D	Database Structure	108
E	Issue Templates	109
E.1	Bug Report Template	109
E.2	Feature Request/User Experience Template	110
F	Automated Execution Script	112
	References	114

List of Figures

3.1	Study search and selection diagram	13
3.2	Topic modeling for smartwatch domain [4].	30
3.3	Ardoc’s GUI [53].	32
3.4	OpenReq’s Dashboard [63].	33
3.5	OpenReq’s Focus Views [63].	33
3.6	CLAP’s GUI [70].	34
3.7	Generated Summary XML file.	35
3.8	Topics in the Generated Summary.	35
3.9	Most Relevant Review Sentences.	35
3.10	Complete Review.	35
3.11	App Reviews Evolution View [40].	36
3.12	App Store Comparison View [40].	36
3.13	Review Details View [40].	36
4.1	Social Media Feedback Processing Workflow Diagram.	40
5.1	Landing Page.	62
5.2	Dashboard.	63
5.3	Instructions for Dashboard.	63
5.4	Menu.	64
5.5	Suggested Feedback Page.	64
5.6	Instructions for Suggested Feedback Page.	65
5.7	Post.	66
5.8	View Post’s Comments.	67
5.9	View Post’s Full Text.	67
5.10	Confirm Category Modal.	68
5.11	Accepted Feedback Toast.	68
5.12	Post not Relevant Modal.	69
5.13	Filters.	69
5.14	Sorting Options.	70
5.15	<i>Clustered View.</i>	71
5.16	Expanded Cluster.	71
5.17	“Load More” Button.	72
5.18	Loading Symbol.	72
5.19	Deleted Feedback Page.	73
5.20	Instructions for Deleted Feedback Page.	73
5.21	Accepted Feedback Page.	74
5.22	Instructions for Accepted Feedback Page.	74

5.23	Selected Posts.	75
5.24	Select Post Warning.	75
5.25	Accepted Feedback <i>Clustered View</i>	76
5.26	Create Issue Modal.	76
5.27	Import CSV with Issues to GitLab.	77
5.28	Issue Template.	78
5.29	Remotely Create Issue for GitLab.	78
5.30	Instructions to Create Private Token.	79
5.31	Created Issue in GitLab.	79
5.32	Exported Feedback Page.	80
5.33	Instructions for Exported Feedback Page.	80
6.1	Usability Survey Section Answers.	84
6.2	Relevance Survey Section Answers.	85
B.1	ROC curve example, showcasing different AUCs.	99
D.1	Database Schema in UML.	108

List of Tables

3.1	<i>IEEE Xplore's Query</i>	12
3.2	<i>Scopus's Query</i>	12
3.3	Inclusion Criteria	14
3.4	Exclusion Criteria	14
3.5	First <i>IEEE Xplore's Visualization Query</i>	15
3.6	Second <i>IEEE Xplore's Visualization Query</i>	15
3.7	<i>ACM's Query</i>	15
4.1	Number of Extracted Posts.	43
4.2	Number of posts regarding each streaming service, using a new search strategy for Reddit	44
4.3	Number of posts regarding each streaming service, using a new search strategy for Twitter	45
4.4	Random Sampling of 1% of the Dataset.	46
4.5	Summary Matrix of Dataset Columns.	47
4.6	Number of Posts After Preprocessing.	49
4.7	Best Models and Techniques for Each Social Media Platform.	51
4.8	Evaluation Metrics for Different Algorithms Across All Social Media Platforms.	52
4.9	Relevant Posts Overview.	52
4.10	Summarization Algorithms Evaluation.	56
6.1	Summary of Results of Active Usage Monitoring.	86

Abbreviations

API	Application Programming Interface
ARdoc	App Reviews Development Oriented Classifier
AUC	Area Under the Curve
BERT	Bidirectional Encoder Representations from Transformers
BoW	Bag of Words
BTM	Biterm Topic Model
CBoW	Continuous Bag of Words
CLAP	Crowd Listener for releAse Planning
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DT	Decision Trees
FN	False Negatives
FP	False Positives
LDA	Latent Dirichlet Allocation
LLDA	Labelled Latent Dirichlet Allocation
MOG	Media Objects and Gadgets
NB	Naive Bayes
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
POS	Part-Of-Speech
PRAW	Python Reddit API Wrapper
RE	Requirements Engineering
RF	Random Forest
ROC	Receiver Operating Characteristic
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SBERT	Sentence-Bidirectional Encoder Representations from Transformers
SMOTE	Synthetic Minority Over-sampling Technique
SNS	Social Networking Sites
SURF	Summarizer of User Reviews Feedback
SUS	System Usability Scale
SVM	Support Vector Machines
TF	Term Frequency
TF-IDF	Term Frequency-Inverse Document Frequency
TP	True Positives
US	User Story
USE	Universal Sentence Encoder

Chapter 1

Introduction

This chapter serves as the gateway to the comprehensive exploration of the work presented in this dissertation. It intends to offer a clear understanding of the context, motivations, and objectives that underpin this research. In addition, we provide an overview of the document's structure, creating a navigational guide for readers to traverse the various sections and components within.

1.1 Context and Motivation

Requirements Engineering (RE) is a vital discipline within software development, encompassing activities such as elicitation, analysis, negotiation, documentation, and validation. However, traditional RE methods often assume direct access to end-users, which is not always true in today's market-driven product development landscape. Furthermore, the cost of conventional RE techniques can be prohibitive, particularly for small startups. A study by Ali and Hong [3] highlighted such companies' budget constraints, underscoring the need for cost-effective alternatives.

Enter social media, a vast and dynamic realm with great promise for RE. According to Statista (2023) [64], 59.4% of the global population uses social media, a significant figure when compared to 64.4% with internet access. Social media platforms host a wealth of information, including users' opinions, experiences with products and services, and expressions of desire for changes in those products and services. This user feedback can be a valuable resource for RE engineers, enabling them to gather requirements and learn more about users through observation.

Moreover, the richness of social media interactions extends beyond individual expressions. Users frequently engage with each other, sharing insights, recommendations, and critiques. This dynamic social environment provides an added layer of information for RE. By analyzing individual posts and user interactions, RE engineers gain a deeper understanding of collective sentiments, preferences, and emerging trends.

Furthermore, historical context in users' past posts offers valuable insights. By examining prior interactions, RE engineers can discern patterns, identify evolving needs, and gauge user sentiments over time, providing a comprehensive understanding.

In this context, the hypothesis is that social media can be a powerful tool for RE, offering requirements elicitation and prioritization opportunities. Previous studies have already shown the potential of social media in these activities. RE engineers have successfully used social media in requirements elicitation, negotiation, and prioritization [60].

With these insights, this research explores the role of social media in RE, offering a promising avenue for improving traditional practices and addressing the challenges faced in today's diverse product development landscape.

This research was conducted in a business environment, in the company MOG (Media Objects and Gadgets) Technologies. MOG, established over a decade ago, is a well-respected player in the broadcasting industry. The company specializes in delivering a diverse range of products tailored to the needs of this dynamic market. Their product offerings include mSDK, a professional media library; mDeck, a professional media deck; mRec, a video file recorder; mPlay, a play-out server; mLive, a streaming server; and mCoder, a transcoding appliance. In addition, MOG Technologies also provides cloud-based solutions, such as MAM4PRO, a cloud media production platform, and Vizzi, an Over-The-Top Television Platform. MOG Technologies primarily serves other businesses, specifically those within the entertainment industry.

Our initial goal was to gather feedback regarding Vizzi. However, MOG expressed interest in understanding user feedback about their competitors, prompting us to expand our scope to encompass the entire streaming service industry.

MOG's customer base presented a unique challenge in gathering feedback for Vizzi. Typically, businesses are less inclined to publicly share feedback or experiences with products on social media platforms. Such feedback is more commonly associated with individual consumers rather than corporate entities. Vizzi allows entertainment companies to tailor it to their brand's needs and share content with customers. While end-users may not know they are using Vizzi-powered platforms, they interact with Vizzi's features. Their experiences with these features, shared on social media, can provide valuable insights for MOG.

Any software product's success hinges on the RE process's effectiveness, as requirements are the cornerstone of software quality. However, it is not uncommon for requirements to contain errors, and rectifying them in later stages of development is time-consuming and expensive [3]. This underscores the critical importance of conducting the RE process meticulously.

In software development, many projects grapple with challenges and difficulties, often stemming from poorly defined requirements and low user involvement [3]. User feedback plays a pivotal role in the success of a product, given that the RE process revolves around understanding and realizing the vision of end-users and stakeholders. It is here that the significance of social media platforms comes into play.

Social media platforms have become reservoirs of valuable user feedback. Previous research has ventured into mining user feedback from diverse platforms, from app stores and Twitter to

Reddit and Facebook [28, 47, 51, 74]. The findings of these studies have been conclusive: the gathered feedback not only provides profound insights into user needs but also contributes to the enhancement of software quality. It encompasses invaluable information, spanning bug reports, feature requests, and more [28].

While the extraction of vital feedback began primarily through manual means, the sheer volume of data in social media platforms rendered this approach unsustainable. Consequently, researchers have proposed automatic or semi-automatic mechanisms for classifying and prioritizing feedback. However, these research endeavors have predominantly remained confined to academic purposes and have yet to find practical applications within real-world companies. As a result, a significant gap exists in the current literature: the absence of tools that enable stakeholders to visualize the amassed feedback and make informed decisions based on this wealth of information. It is precisely this gap in the existing literature that this dissertation seeks to explore and address.

1.2 Objectives

This dissertation aims to automate the comprehensive process of collecting feedback on digital products from social media, extracting candidate requirements, and developing an intuitive visualization platform to present this feedback to stakeholders and requirements engineers. This process empowers them to elicit, interpret, and act upon it effectively. The following objectives guide this endeavor:

1. Development of Social Media Feedback Processing Workflow:

- Collect data from a range of social media platforms, including app stores, to extract insights regarding MOG's clients' platforms that use Vizzi and other streaming platforms by employing web crawlers and APIs (Application Programming Interface);
- Employ advanced NLP techniques for thorough data preprocessing, ensuring data quality and consistency;
- Automatically classify data into two main categories: relevant and irrelevant feedback. Within the first category, further subdivide feedback into *bug report*, *feature request* and *other*;
- Incorporate sentiment analysis to evaluate the emotional tone and attitude expressed in user feedback, adding depth to the classification process;
- Utilize summarization techniques to generate concise yet informative reports, aiding stakeholders in grasping feedback patterns efficiently;
- Implement grouping techniques (clustering or topic modeling) to identify key themes and topics within the collected feedback, facilitating a deeper understanding of the gathered feedback;
- Visualize requirements and validate results with MOG's personnel.

2. Platform Development:

- Develop an intuitive visualization platform using data from the developed workflow;
- Implement platform refinements, such as integration with GitLab and other issue-tracking systems. This enhancement aims to facilitate the automatic creation of issues or offer proactive issue suggestions, editable by team members for clarity [40];
- Assessing the Usability and Intuitiveness of the Platform.

1.3 Innovative Aspects and Challenges

1.3.1 Gathering Indirect Feedback

While research in this domain has traditionally centered around academic pursuits, it predominantly revolves around software companies, all sharing a commonality—individual consumers as their end-users. However, a distinctive innovation in this dissertation is its focus on MOG’s product, Vizzi, among other aspects of the streaming service industry. Entertainment businesses employ Vizzi to create streaming platforms marketed to individual consumers. Consequently, these end-users, unaware they are interacting with Vizzi, share their experiences on social media, providing invaluable feedback for enhancing this platform.

This unique scenario introduces a novel layer of filtering that is unexplored in existing literature, rendering many conventional methods of information procurement presented in the literature invalid in this particular instance. For example, researchers would monitor the official social media accounts of the software companies under study. Since end-users are unaware of Vizzi’s involvement, these customary approaches face additional challenges since users would never directly address MOG or Vizzi. This dissertation thus confronts the task of navigating these novel challenges to extract meaningful insights from a distinctive social media landscape.

1.3.2 Automating Process

While existing literature predominantly focuses on creating pipelines for data extraction and testing their outcomes, a notable gap exists in tools replicating and automating this process. Furthermore, there is an even more significant void for tools that enable the visualization of candidate requirements, so needless to say, tools that combine the two are almost non-existent. Among the papers analyzed, only one described a platform with similar functionalities, albeit limited to data from mobile app stores.

The potential multilingual nature of the data introduces an additional layer of complexity. Since MOG has Vizzi users who are English speakers and others who are Portuguese speakers, we anticipated the data would include at least these two languages. Additionally, given the global user base of other streaming apps, feedback in various languages was possible. Therefore, we recognized the need to accommodate and process data in multiple languages.

This endeavor addresses an existing gap in the literature and presents unique challenges, including integrating data from diverse sources and incorporating multilingual capabilities into the automated platform.

1.3.3 Practical Validation

Beyond the theoretical exploration of social media's role in RE, the challenge persists in translating insights into actionable outcomes. The existing literature in this domain primarily serves academic purposes, where researchers leverage popular applications to obtain data and test hypotheses. However, a critical gap exists in the interaction of stakeholders involved in developing those products with the requirements derived from these studies. Moreover, most studies lack the creation of an interface accessible to non-technical stakeholders for easy observation of the extracted requirements.

Herein lies the distinctive contribution of our work. We are developing a platform that not only delves into the theoretical aspects but also incorporates a practical dimension. Our platform includes a visualization component, allowing all stakeholders involved in product development to easily observe and understand feedback and act upon it. Going beyond existing research, we plan to involve real-life users to test the platform's helpfulness and intuitiveness. The ultimate goal is to seamlessly integrate the platform into the practical workflows of companies, such as MOG, ensuring its practical utility and adoption in real-world scenarios.

1.4 Document Structure

In addition to the introduction, this document is structured as follows: **Chapter 2** provides background knowledge about RE and Social Media. **Chapter 3** describes the Research Methodology used and the State of the Art of the present topic. **Chapter B** provides insight into Machine Learning and NLP Techniques used by the papers analyzed in **Chapter 3**. **Chapter 4** details the development of a social media feedback processing workflow. **Chapter 5** describes the development of an intuitive visualization platform designed to empower stakeholders to act on the requirements gathered from social media. **Chapter 6** discusses the experiments conducted to obtain user feedback regarding the developed platform. Finally, **Chapter 7** concludes this report, summarizing the findings and insights obtained throughout the study.

Chapter 2

Requirements Engineering and Social Media

We decided to briefly introduce RE and Social Media in this chapter, as they constitute integral parts of our work.

2.1 Requirements Engineering

RE is a pivotal and critical phase in the software development life cycle, shaping decisions with enduring impacts throughout the process. Rationale management becomes essential during this phase to capture justifications for the decisions made [34].

Project activities may unfold in an interleaved, iterative manner, spanning the entirety of the software life cycle [60].

According to the Guide to Software Engineering Body of Knowledge (SWEBOK) [9] and the Glossary of RE Terminology [20], RE involves elicitation, prioritization, negotiation, specification, and management of requirements. Key activities include:

- **Requirements Elicitation:** Involves seeking, capturing, and consolidating requirements from stakeholders to identify information sources about the system and elicit requirements;
- **Requirements Prioritization:** Encompasses the analysis of collected requirements to establish priorities based on stakeholder importance. Stakeholders decide on the core requirements of the system, planning and selecting an order for implementation;
- **Requirements Negotiation:** Entails resolving conflicts by finding settlements that mostly comply with all stakeholders;
- **Requirements Specification:** Defines what the system should and should not do, providing a basis for agreement between customers and contractors;
- **Requirements Management:** Involves planning requirements and creating a traceability document to manage changes over time.

The success of a software project hinges on its RE process, with requirements playing a decisive role in software quality. Empirical evidence underscores that errors in requirements are common, turning them into a time-consuming and expensive task to rectify in later stages.

A research report from the Standish Group [21] indicates that 52.7% of projects faced challenges, with costs exceeding 189% of the estimated budget. The success rate was a mere 16.2%, and 31.1% were canceled. Poor requirements, unclear objectives, and lack of user involvement were identified as the primary sources of problems. Project success correlates with user involvement (15.9%), executive management support (13.9%), and a clear statement of requirements (13%). Challenges arise due to a lack of user input (12.8%), incomplete requirements and specifications (12.3%), and changing requirements and specifications (11.8%). Poor handling of system requirements emerges as a significant cause of project failures. As stated by Li et al. [38], past studies suggest that approximately 60% of all errors in software development projects originate during the requirements engineering phase.

The primary objectives of requirements engineering encompass fostering consensus among stakeholders regarding product desires and needs, coupled with meticulous documentation adhering to specified standards. Specification is a pivotal artifact in requirements engineering, significantly influencing software development. In crafting the requirements specification, it becomes imperative to gather the desires and needs of stakeholders towards the product, employing methods such as meetings or any other effective communication channels. This phase is commonly referred to as requirements elicitation, as stated previously.

Various approaches for requirements elicitation include [49]:

- **Interview:** A traditional approach involving end-users, domain experts, and stakeholders, conducted using predefined questions. Structured interviews utilize predefined questions, effectively guiding the interviewer's expected answers. This approach, however, tends to limit customers' opinions within the framework set by the interviewers. The target sample of customers may not represent most customers' opinions due to temporal and spatial constraints;
- **Observation:** Involves observing the actual execution of existing processes without direct user involvement. This approach is commonly used with other methods, such as task analysis and interviews. However, it demands high skills and effort from the analyst to interpret and understand the actions being performed, making it an expensive option;
- **Focus Group:** Brings groups together to discuss topics of interest with customers, practical for products with well-known features and trade-offs. This approach is expensive and complex;
- **Questionnaire:** Participants fill a form with questions.

However, these approaches depend on the availability of the stakeholders and on all stakeholders being simultaneously present in the same space. This does not bode well for the requirement elicitation process since end-users usually are not always available for meetings, which prolongs

this RE phase and delays software development. This might make companies tempted to skip RE altogether, which would be detrimental to the project in the long term [49]. The popularity of social media/network sites offers a potential solution to this communication challenge.

2.2 Social Media

As stated by Seyff et al. [60], social network sites (SNS) are examples of social software that facilitate connection and communication anytime and anywhere.

SNS are web-based services enabling individuals to:

- Develop a public, private, or semi-public profile;
- Maintain a list of connected people (friends);
- View and communicate with friends;
- Relate to each other based on common interests, friendship, or trust [6, 60].

According to Statista (2023) [64], 59.4% of the global population uses social media, a significant figure when compared to 64.4% with internet access. As a result, social media generates a massive amount of real-time and diverse data daily, revolutionizing research fields such as data science.

SNS can be valuable tools for RE, with successful studies focusing on open social media platforms like **Twitter**, **Reddit**, **Facebook**, **Instagram**, and app stores (**Apple Store**, also known as **iTunes**, and **Google Play**). While some works do not consider app stores social networks, we recognize their importance, as users can interact with each other and have profiles. It is not possible to connect with other users besides talking in comments. However, these platforms have been proven to have essential and valuable information for the study of our hypothesis, so we are considering them. For example, app stores allow users to rate the apps through a star system conveniently separated from the rest of the reviews. It allows users to gauge their experience with the app before even reading the text they have written. Specific forums like GitHub are not considered in our analysis.

Twitter and Reddit offer distinct features:

- **Twitter:** Facilitates bidirectional communication, enabling users to engage in conversations by replying to each other's tweets and adding supplementary information [22]. Direct interactions are a motivating factor for users, encouraging a consistent stream of high-quality feedback [25];
- **Reddit:** Allows bidirectional communication and includes an up or down vote system to show approval or disapproval (this system is equivalent to likes and dislikes in other social media; however, in most, this information is no longer available). Votes contribute to a karma score, which prioritizes posts and comments. Users also have a karma score that

serves as their reputation. User Karma is affected by the amount of community approval for comments and submissions the user has created. Both post and user karma scores can aid in the prioritization phase of RE. A high karma score from both posts and the user lets us know whether the opinion shared on the post is shared by many users, which means that it should be a high-priority requirement [32].

In open social media (excluding app stores), users can agree by liking content, allowing for the analysis of post popularity. Twitter further allows retweeting to boost post popularity and show user agreement. Most social media platforms support hashtags (originally from Twitter), facilitating grouping and retrieving posts discussing similar topics [32].

The optimal approach for extracting requirements from various platforms is through their Application Programming Interfaces (APIs). However, it is essential to note that not all platforms provide open access to their APIs, and some may have associated costs.

- **Reddit:** Previously offering a free API¹, Reddit introduced a charging model in July 2023, imposing a fee of 24 cents per 1,000 API requests;
- **Twitter:** also provides an API², and its pricing structure also changed in 2023. The free tier no longer allows the extraction of tweets, limiting usage to posting. The entry-level tier, named *Basic* incurs a cost of \$100 per month and enables tweet extraction;
- **Facebook:** Utilizing the *Graph API*³, Facebook offers free access, but with a rate limit. The limit is 200 calls per hour multiplied by the number of unique daily active users associated with the app. For developers extracting posts, this translates to 200 calls per hour;
- **Instagram:** While Instagram provides an API, *Instagram Basic Display API*⁴, it limits users to retrieving their profile information, photos, and videos. Extracting information from searches requires alternative methods, such as scraping;
- **Google Play:** The *Google Play API*⁵ facilitates the extraction of application reviews. However, this capability is restricted to apps created by the user; for other apps, scraping is necessary;
- **iTunes:** offers the *iTunes Search API*⁶, which is free but imposes a limit of approximately 20 calls per minute.

After conducting our analysis, we decided to use **Reddit**, **Twitter**, **Google Play**, and **iTunes** to extract user feedback. We excluded **Instagram** because its API only permits extracting profile

¹<https://www.techtarget.com/whatis/feature/Reddit-pricing-API-charge-explained>

²<https://developer.twitter.com/en>

³<https://developers.facebook.com/docs/graph-api>

⁴<https://developers.facebook.com/docs/instagram-basic-display-api>

⁵<https://developers.google.com/android-publisher>

⁶<https://developer.apple.com/library/archive/documentation/AudioVideo/Conceptual/iTunesSearchAPI/>

information, photos, and videos. The most valuable content from this platform for our endeavor would be the comments on users' posts. We were unsure how accessible they would be since we would need to make our own program to achieve the data, and it might not be helpful if it is mostly just agreeing or disagreeing with the post. It would not be useful because we would not have the context needed for the post since converting photos to text or processing videos for transcripts would introduce additional complexity beyond our project scope. Furthermore, we eliminated **Facebook** due to a manual ad-hoc search yielding limited user feedback relevant to our selected streaming services.

Chapter 3

State of the Art

In this chapter, we delve into the contemporary utilization of social media for collecting user feedback directly relevant to the themes explored in this dissertation. Our exploration begins by elucidating the methodology employed to grasp the present landscape of this subject. Subsequently, we present our discoveries and conclude the state-of-the-art practices, elucidating their implications on our research.

3.1 Research Methodology

We adhered to a systematic methodology [37] in exploring the state of the art. Our approach began by formulating research questions to serve as the compass for our investigation. Subsequently, we crafted a comprehensive review protocol, meticulously guiding our review process and data collection. Following the established protocol, we conducted a thorough literature search and selection, adhering to predefined criteria. The selected studies underwent meticulous reading, enabling a deep understanding of their content. Finally, we synthesized the gathered insights to form a cohesive narrative for reporting our findings.

3.1.1 Research Questions

This study sought to create an intuitive visualization platform that empowers stakeholders to interpret, act upon, and prioritize requirements by leveraging insights from social media data. With this objective, we aimed to learn the current state of using social media as a tool for RE, for starters. Subsequently, we aimed to delve into specific facets, such as the automation of this tool's usage and the state of visualizing requirements extracted through its application.

As a result, we had this overarching research question:

RQ1: What works exist about extracting requirements from social networks?

And more specifically:

RQ2: What works exist about automatically classifying candidate requirements extracted from social networks?

RQ3: What works exist about the visualization of candidate requirements extracted from social networks?

By addressing these questions, we aimed to comprehensively understand the landscape surrounding the extraction, classification, and visualization of requirements from social networks.

3.1.2 Literature Search and Selection

We implemented a systematic search and selection process to gather pertinent literature published between January 2015 and the current year of performing and writing the literature review, 2023. We considered papers from the current year if they had undergone peer review. This time frame was selected to ensure the inclusion of the most up-to-date information in the field. Our initial exploration involved utilizing *Google*, *Google Scholar*, and *IEEE Xplore* to familiarize ourselves with the thematic landscape and identify prominent keywords. We selected these databases based on our prior experience and knowledge. The primary digital libraries for collecting papers were *IEEE Xplore* and *Scopus*, chosen for our familiarity with their interfaces and the effectiveness of their filtering systems, particularly in the case of *Scopus*. To maximize the efficiency of our searches, we crafted distinct queries for each database, capitalizing on the specific filters available. The query for *IEEE Xplore* was:

Table 3.1: *IEEE Xplore*'s Query

Search	social network* and requirements engineering
Filters	2015 - 2023

Conversely, the query for *Scopus* was detailed as follows:

Table 3.2: *Scopus*'s Query

Search	social AND media AND requirements AND engineering
Search within	Article title, Abstract, Keywords
Year	2015 - 2023
Keywords	Social Networking (online); Social Media; Requirements Engineering
Language	English

The search on *IEEE Xplore* yielded 1205 papers, while the *Scopus* search resulted in 135 papers, a more manageable number (refer to [hyperref\[fig:methodology\]](#)Figure 3.1). Notably, the University of Porto extends an institutional subscription to students, encompassing both databases, thus influencing the results obtained. It is important to acknowledge that some of the collected references might necessitate a subscription, as the utilized databases permitted the amalgamation of freely accessible findings with those accessible exclusively through subscription. Our systematic

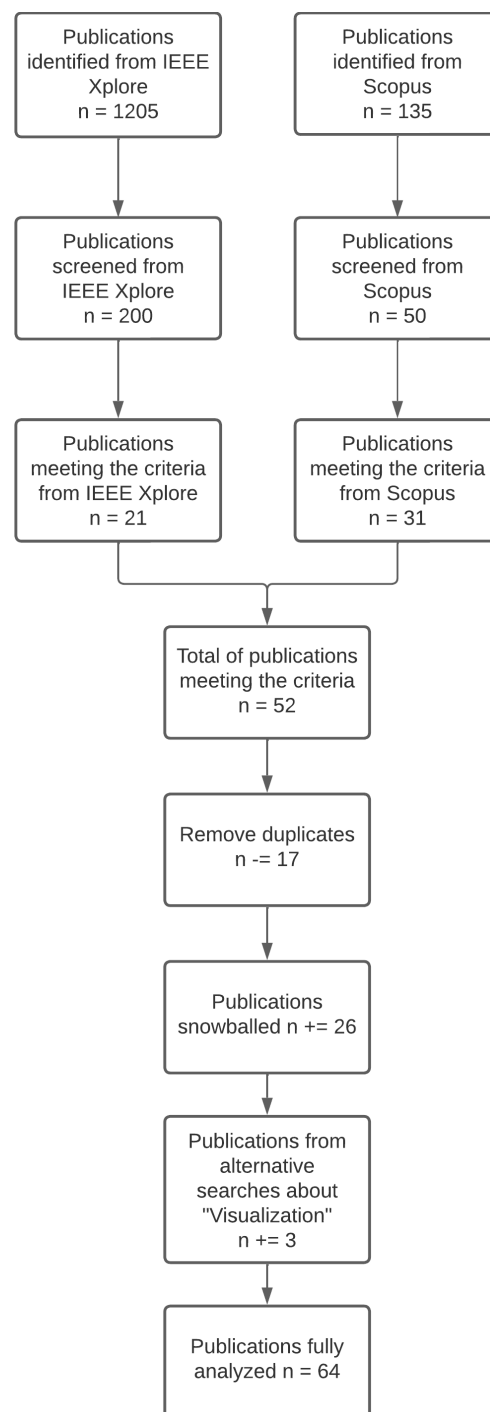


Figure 3.1: Study search and selection diagram

screening process involved a step-by-step analysis. We assessed the titles against a set of inclusion criteria detailed in the [hyperref\[table:inclusion_criteria\]](#)Chapter 3.3.

We proceeded to the next stage for papers meeting these title-based criteria, examining abstracts to validate inclusion further. However, there were exceptions for studies that lacked visu-

Table 3.3: Inclusion Criteria

Number	Inclusion Criteria
1	Reference of Social Media and RE
2	Investigation of the Interplay between Social Media and RE
3	Automation of the process of extracting requirements from Social Media
4	Mention of some visualization of the candidate requirements acquired
5	Peer-reviewed papers

alization of the extracted requirements. To gain a holistic understanding of the current state and industry perspectives on employing social media for RE, we also considered studies that did not explicitly mention automation. Due to the ambiguity of some abstracts, which led to the need for further investigation, we established exclusion criteria outlined in [hyperref\[table:exclusion_criteria\]](#) Table 3.4.

Table 3.4: Exclusion Criteria

Number	Exclusion Criteria
1	Papers not available
2	Papers not in English
3	Papers that don't mention in unison, Social Media and RE
4	Papers focused on Social Network Development (for example, gathering requirements to develop a social network)
5	Papers that merely comment on or make off-hand references to the use of Social Media in Software Engineering without in-depth analysis
6	Papers before 2015

We made an exception for exclusion criterion 6 due to multiple papers recommending the paper in question, emphasizing its significance despite being published before 2015.

Following the initial screening on *IEEE Xplore*, approximately 200 papers were reviewed, with only 21 meeting our specified criteria. On *Scopus*, we screened 50 papers, and 31 were deemed suitable according to our criteria. In total, our selection included 52 papers. During this process, we identified and removed 17 duplicated papers, resulting in 35.

We employed the *Snowball Method* to enhance the comprehensiveness of our paper pool [56]. The *Backward Snowball* involved tracing the references of initially selected papers. This method allowed us to identify additional relevant studies that may not have been captured in our initial search.

Concurrently, we utilized the *Forward Snowball* approach, exploring papers that cited the initially selected studies. This method highlighted the significance and impact of the chosen papers, as many were referenced and interconnected with other relevant works. Integrating *Backward* and *Forward Snowball* methods resulted in a selection pool of 26 publications, offering a holistic perspective on the literature landscape in the domain.

After an analysis of the initial selection pool, we identified a gap in information related to one of our specific research questions:

What works exist about the visualization of candidate requirements extracted from social networks?

To address this gap, we conducted additional queries focused on obtaining visualization-related information. Notably, we refined our searches with three new queries, two of which were executed on *IEEE Xplore*:

Table 3.5: First *IEEE Xplore*'s Visualization Query

Search	requirements engineering and data visualization
Filters	2015 - 2023

Table 3.6: Second *IEEE Xplore*'s Visualization Query

Search	requirements engineering user interface
Filters	2015 - 2023; user interfaces; software engineering; interactive systems

The final query was conducted on a new digital library, *ACM*, using the search:

Table 3.7: *ACM*'s Query

Search	requirements engineering user interface
Filters	2015 - 2023

This refined approach led to the inclusion of three additional papers in our analysis, offering a more comprehensive understanding of the landscape concerning the visualization of candidate requirements extracted from social networks. In total, we analyzed 64 papers.

The complete list of the research papers and their characteristics is available in [Appendix A](#), followed by a topic-wise analysis.

3.2 Manual Extraction of Requirements from Social Networks

The pervasive use of social networks has transformed our daily communication and become an emerging source for insights into user requirements in software development. In this section, we explore studies that delve into the manual extraction and analysis of requirements from social networks. Understanding the manual processes involved in gleaning valuable information from these platforms is essential before delving into the intricacies of automation. We begin by examining studies that employ manual approaches, where researchers actively engage with social network data, extracting and analyzing requirements through qualitative methods.

The contemporary use of social media as a tool in software development necessitates an exploration of its effectiveness and the perceptions of industry specialists regarding its application.

Al-Hunaiyyan et al. [1] conducted a comprehensive investigation to gauge these perspectives, employing a multi-faceted approach, including a questionnaire and a focused group discussion. Software engineering professionals actively participated in expressing their perspectives on leveraging social media for requirements gathering. The consensus among participants revealed a shared belief in the utility of SNS for requirements elicitation, highlighting the capacity of these platforms to stimulate innovative ideas and expedite the elicitation process.

Among the SNS discussed, *WhatsApp* emerged as a standout favorite. This direct messaging application, facilitating both text and voice communication, became the preferred choice for participants. Due to this choice, we can see that participants favored an active role in the RE process, akin to traditional requirements engineering methods. This active involvement implies direct interaction with stakeholders, contrasting with passive observation of their feedback about products. In the eyes of these participants, social media presents a solution to challenges posed by traditional methods, particularly in situations where face-to-face meetings with stakeholders are impractical. The ability to engage with stakeholders on platforms like *WhatsApp* addresses the limitations of geographical constraints, allowing professionals to maintain a direct line of communication even in the absence of in-person meetings. However, it is more suitable for interacting with already identified stakeholders rather than collecting extensive feedback.

Kanchev and Chopra [32] conducted a study involving a preliminary analysis of social media feedback, with a specific focus on information relevant to RE. The researchers directed their attention to user interactions with *Google Maps* on **Reddit**, a popular social media platform characterized by a forum-style discussion structure.

Google Maps, a widely-used navigation tool providing directions and maps to millions worldwide, became the subject of investigation due to its unique feature allowing users to execute commands verbally. This distinctive functionality was the central theme of the **Reddit** discussions scrutinized by Kanchev and Chopra.

The outcome of their study revealed a substantial reservoir of pertinent information for requirements engineering within social media channels. Key findings encompassed a spectrum of insights, including user requirements, observations articulated by end-users, a lack of awareness regarding existing solutions, expressions of sentiment towards previously discovered artifacts, and a sense of community support.

Notably, the researchers went beyond mere observation, constructing a goal model of *Google Maps*. Domain knowledge extracted from the nuanced discussions on **Reddit** informed this model. Through this comprehensive approach, Kanchev and Chopra affirmed the abundance of valuable information on social media for requirements engineering. They demonstrated the practical application of such insights in the form of a tangible goal model.

In their study focused on small software businesses, Ali and Hong [49] addressed the financial constraints often faced by startups in adopting conventional RE techniques, which can be cost-prohibitive. Recognizing the need for cost-effective alternatives, the researchers explored the potential of SNS in eliciting user requirements.

The researchers emphasized the dynamic nature of SNS in facilitating the observation of user discussions and conflicts, which is vital for identifying and prioritizing initial user requirements. Unlike a passive approach of merely monitoring social media feeds, Ali and Hong proposed an active engagement strategy. They suggested that companies establish dedicated discussion rooms on social media platforms, fostering direct interaction with stakeholders and end-users. This active participation involves moderating conversations, posing questions, and closely observing discussions.

Companies can address every stage of the requirements engineering process through this comprehensive approach. The process encompasses user requirement elicitation by inviting users to join, posting ideas, and encouraging brainstorming and discussion. Subsequently, determination involves interpreting these ideas into tangible requirements, while prioritization entails organizing requirements based on their importance using clustering techniques.

Crucially, this approach minimizes the need for extensive employee training and incurs negligible costs. Ali and Hong validated this methodology through a successful case study, where 33.06% of the total invited users actively participated within seven days. This active participation collected 345 user requirements, highlighting the effectiveness of an active and cost-efficient approach to requirements engineering in small software businesses.

Seyff et al. [60] employed social media, specifically **Facebook**, to convene like-minded individuals and conduct a virtual “focus group,” actively collecting requirements. The overarching discovery revealed the practicality of the gathered requirements, thereby validating social media as an excellent method for such endeavors. Furthermore, the paper explores social media’s application in various RE techniques beyond elicitation, prioritization, and negotiation.

In their study, Scanlan et al. [59] explore the application of social media in situations where end-users face limited availability. The specific focus was on gathering requirements for a mHealth (mobile health) tool designed to support caregivers of children with autism. Traditional RE methods often prove impractical for user groups like this due to time constraints, emotional pressure, potential geographical distance from RE engineers, and the challenges faced by primary caregivers in dislocating themselves for the benefit of the engineers.

The researchers adopted a unique approach by leveraging social media to overcome these challenges. Given the caregivers’ unavailability or limited time, Scanlan et al. employed a more passive strategy, gathering requirements by observing their social media posts rather than direct interaction. This approach enabled the researchers to delve into the caregivers’ needs and preferences with heightened sensitivity.

Scanlan et al. manually extracted posts from **Facebook**, utilizing APIs for the extraction process for **Twitter** and **Instagram**.

The research team manually analyzed each post, providing summaries and extracting the conveyed emotions within the text. This detailed examination unveiled distinct purposes for which caregivers utilized each social media platform. Through this nuanced analysis, the researchers gained richer and deeper insights into the users and their needs than what traditional RE methods alone could provide.

3.3 Automatic Extraction of Requirements from Social Media

The process of automatically classifying and extracting requirements from social media follows a well-established methodology. Typically, this involves data extraction through a scraper/API, pre-processing, categorization using classification algorithms, and subsequent algorithm performance evaluation.

In addition to these fundamental steps, many studies incorporate additional stages to enhance the overall process. These include sentiment analysis, which aids in the categorization and prioritization of candidate requirements. Grouping, performed post-classification, facilitates a better understanding of the gathered data. Summarization, also conducted after classification, helps distill critical insights. Feature engineering carried out before classification, assists in optimizing this step. Furthermore, methods to address this issue are applied before algorithm training in cases of imbalanced data.

Further details on the techniques discussed in this section can be found in [Appendix B](#).

3.3.1 Classification Algorithms

Within the established methodology, variations emerge across different studies, encompassing the selection of social media platforms for case studies, diverse algorithms, and distinct preprocessing techniques.

Notably, prevalent classification algorithms, such as (Multinomial) Naive Bayes (NB), Support Vector Machines (SVM), Decision Trees (DT), and Random Forest (RF), find widespread use due to their demonstrated effectiveness in categorizing text. The popularity of these algorithms is often attributed to their robust performance. Some authors cite this popularity as a contributing factor influencing their choice [25, 22, 35, 74, 28, 51].

3.3.2 Categories

All studies aim to categorize data into three fundamental groups, even if the terminology differs. These categories typically comprise *feature request* (users' suggestions on possible functionalities), *bug report* (describing problems found in software), and the *other* category, used to classify all information irrelevant for app improvement.

While the first two categories remain consistent across studies, the *other* category is often subdivided based on researchers' specific goals. For instance, Di Sorbo et al. [53] distinguished between *Information Giving* (sentences providing information about some aspect of the app) and *Information Seeking* (sentences asking for information and/or assistance). Maalej et al. [40] introduced a *rating* category reflecting stakeholder evaluations. Panichella et al. [52] proposed *Opinion Asking* and *Solution Proposal*. Guzman et al. [23] expanded the scope to include *Feature Strength*, *Feature Shortcoming*, *Praise*, *Complaint*, *Usage Scenario*, and more. Khan [35] focused on argument-related categories such as *claim-supporting* and *claim-attacking*. Guzman et al. [25] expanded to twenty-two categories, covering hardware and software constraints, advertisement,

dissuasion, questions, how-to guides, software pricing, compliance issues, software extensions, services, social interactions, and content-related comments.

McIlroy et al. [44] adopted a distinctive approach by further categorizing *bug report* based on issues encountered during the analysis of user reviews. Their categories included *Additional Cost*, *Functional Complaint*, *Compatibility Issues*, *Crashing*, *Feature Removal*, *Network Problems*, *Privacy and Ethical Issues*, *Resource Heavy*, *Response Time*, *Uninteresting Content*, *Update Issues*, and *User Interface*. Diverging from conventional methods, the authors scrutinized the performance of each label individually, revealing challenges with ambiguous issue types. The proliferation of categories negatively impacted result accuracy, prompting the authors to streamline and delete ambiguous categories. This underscores the importance of careful consideration when defining the number and type of categories in classification tasks.

In two distinct papers, Guzman et al. [22, 25] extended beyond defining feedback categories to identifying stakeholder groups for whom these categories hold relevance. The defined groups included:

- **Technical:** stakeholders directly involved in software development (e.g., requirement engineers, developers, and product owners);
- **Non-Technical:** stakeholders whose work impacts or is impacted by software development (e.g., individuals from sales and marketing);
- **General Public:** potential end-users.

This nuanced categorization allows for effective filtering of feedback for analysis. Focusing on categories relevant to technical stakeholders streamlines the study area and reduces the data computers need to process for requirements engineering activities. Nevertheless, it is noteworthy that in both papers, the outcomes for **Technical** stakeholders fell below the anticipated levels, whereas the other two groups yielded satisfactory results

Furthermore, Guzman et al. [22] conducted an exploratory study by categorizing their dataset into content generated by humans or bots. During data analysis, they identified many duplicate entries attributed to automated bots. Since stakeholders are primarily interested in content generated by humans—their actual clients—automatic filtering of this content becomes crucial. This categorization ensured a focus on genuine human-generated feedback, enhancing the relevance and significance of the study results.

3.3.3 Preprocessing

In data preprocessing, various techniques have gained prevalence across the analyzed papers. Common approaches include Stemming, Lemmatization, and Tokenization. In response to the informal nature of social media language, unconventional yet popular methods have emerged. These include correcting typos and removing URLs and special characters such as “@,” all aimed at normalizing the data.

Adopting these techniques reflects an acknowledgment of the unique challenges posed by social media data, where informal language and diverse formats are common. Correcting typos enhances textual content quality while removing URLs and special characters aids in standardizing the data for consistent analysis. The combination of traditional and innovative preprocessing methods underscores the importance of adapting techniques to the specific characteristics of social media content.

3.3.3.1 Stemming

Stemming stands out as a widely adopted preprocessing technique in machine learning algorithms, featuring prominently in a substantial percentage of the studies under analysis. This method involves reducing words to their morphological roots, reducing the number of features (words) as only one base form of the word is considered [4, 12, 28, 74].

A prevalent choice among researchers is the Snowball Stemmer, which implements the renowned Porter stemming algorithm. The popularity of this algorithm underscores its efficacy in various applications [18, 44, 52, 53].

In contrast, Phong et al. [71] took a unique approach by developing a customized stemmer. This bespoke solution considered the informal nature of social media language, accounting for potentially misspelled words and unique names. Notably, the algorithm incorporated a domain-specific dictionary of misspelled words and unique names to enhance its adaptability to the idiosyncrasies of social media content.

3.3.3.2 Lemmatization

Despite the widespread popularity of stemming as a preprocessing technique, many analyzed papers preferred lemmatization [14, 57].

Lemmatization involves reducing different inflected forms of a word to their fundamental lemma, treating them as a single item for analysis. Using Maalej et al.'s example [40, 42] for a better understanding, "fixing," "fixed," and "fixes" converge to the lemma "fix," providing a more contextually rich representation.

Unlike stemming, which removes postfixes to reduce terms to their basic form, lemmatization considers the linguistic context of the term. By leveraging dictionaries, lemmatization distinguishes between words with different meanings based on their POS. Once more, using an example from Maalej et al., [40, 42], *"lemmatization recognizes 'good' as the lemma of 'better' and stemmer will reduce 'goods' and 'good' to the same term"*.

Both lemmatization and stemming contribute to unifying keywords with similar meanings but different language forms, ultimately increasing their count and aiding classifiers. However, lemmatization's emphasis on linguistic context makes it particularly valuable in applications where understanding the nuanced meaning of words is crucial, as is often the case in the context of user reviews [47].

One widely adopted approach for lemmatization involves using WordNet to identify and map words to their lemmas [5, 18].

WordNet comprises three databases, one for nouns and verbs and a third dedicated to adjectives and adverbs. Each database has a collection of lemmas, each annotated with a set of meanings. Words sharing similar meanings are systematically grouped [31].

3.3.3.3 Stop Word Removal

A contentious point arises concerning the treatment of stop words in the preprocessing pipeline. While most analyzed papers choose to remove stop words, two articles deviate from this trend [7, 35]. Interestingly, some studies that opted for stop word removal observed adverse effects stemming from this decision [40, 74, 4].

Papers choosing to retain stop words present a similar rationale, contending that these words often carry valuable information about user intentions. For instance, words like “will” and “would” might signal a desire for new features, while terms like “isn’t,” “did,” and “could” could indicate the representation of an issue.

A notable study by Maalej et al. [40] contributed intriguing insights by experimenting with both options and comparing outcomes. They discovered that removing stop words could reduce classification accuracy while potentially increasing precision and recall. This dual impact arises because eliminating stop words reduces noise, amplifying the influence of informative terms such as “bug” or “add.” This, in turn, may enhance the accuracy of document classifiers.

In essence, removing or retaining stop words presents a double-edged sword. Both choices bring benefits and drawbacks, necessitating a meticulous analysis to determine the most suitable path for the dataset.

3.3.3.4 Remove Infrequent Words

Stop words are not the only words that might get removed from the dataset.

Some authors, such as Williams, Mahmoud, and Ali et al. [74], focused on removing words that infrequently appear in the dataset. The rationale behind this approach is that words with low frequency are unlikely to carry generalized information and may instead introduce noise.

By filtering out words that lack significant representation in the dataset, researchers aim to highlight more essential keywords, focusing on terms that contribute meaningfully to the analysis. This additional step in word removal reflects a strategic effort to enhance the relevance and informativeness of the dataset for subsequent analysis.

3.3.3.5 Negation Removal

In the pursuit of refining the preprocessing pipeline, some authors, such as Villarroel et al. [70], recognize the potential impact of negations on classification models and take proactive steps to address this concern. Removing negations aims to mitigate potential misclassifications due to negated terms in user reviews.

For instance, consider the following review using these authors' example: "I love it. It runs smoothly, and there are no lags or glitches." While the words "lags" and "glitches" might typically indicate a *bug report*, the presence of the negation "no" changes the statement's sentiment. Neglecting to account for this negation could lead to a misclassification. To circumvent this challenge, these authors employed tools like the Stanford parser to identify and remove negated terms.

After this preprocessing step, the example review transforms into: "I love it, it runs smooth," ensuring a more accurate representation of the user's sentiment and reducing the likelihood of misclassifications.

3.3.3.6 Synonym Unifying

Villarroel et al. [70] incorporate synonym merging/unifying as a strategic step in their data preprocessing approach. This technique involves mapping different synonyms to a standardized representation, simplifying the data, and rendering it more suitable for machine learning algorithms.

Recognizing the inadequacy of an ordinary thesaurus in the context of user reviews, the authors discern that many words function as synonyms within this specific domain but not necessarily in a broader context. For example, terms like "freeze" and "crash" may be considered synonyms in the context of user reviews, reflecting issues with the software's performance. However, these words may not be synonymous in a regular context.

Villarroel et al. take a meticulous approach to address this domain-specific synonym challenge. They construct a customized dictionary by manually analyzing 1000 reviews from previous work. This tailored lexicon enables them to accurately capture and unify synonymous terms unique to the domain of user reviews, contributing to a more nuanced and contextually aware preprocessing procedure.

3.3.3.7 Verb Tense Detection

The types of verbs within a sentence can correlate to its overall classification. For instance, using past tense verbs often signifies reporting a previous experience, such as encountering a past issue with the software. Conversely, verbs in the future tense might be employed to describe a hypothetical situation, such as proposing a new feature.

Tizard et al. [69] use the NLTK (Natural Language Toolkit) part-of-speech (POS) tagger, which plays a pivotal role in categorizing the types of verbs in each sentence. During classifier training, each verb tag's simple presence or absence served as a crucial text feature.

Maalej et al. [40] took a more nuanced approach by calculating the ratio of each verb tense (past, present, future) within a review. Recognizing that a single review can encompass multiple tenses, this approach offers a more granular understanding of the temporal characteristics of the text. The extraction of verb tense can be viewed as a NLP technique, as it is often identified through POS tagging, a feature commonly provided in NLP libraries. Moreover, this information can be stored and utilized as metadata alongside the text, offering additional insights for analysis beyond the simple presence or absence of verb tags.

3.3.3.8 Tokenization

Tokenization is a fundamental process involving segmenting paragraphs into sentences or sentences into words. This crucial step was performed in several papers using Stanford Core NLP.

Tokenization serves as a prerequisite for various subsequent tasks, including n-gram extraction, the creation of bag-of-words representations, and text summarization. By breaking down text into its constituent units, tokenization facilitates the extraction of meaningful patterns and structures, enabling more sophisticated analyses and feature extractions in the subsequent stages of natural language processing [45, 57, 53].

3.3.3.9 Part-of-Speech Tagging

In the context of NLP, POS tagging enhances the understanding of the syntactic structure of text, providing valuable information about the roles and relationships of words in a sentence. When used as features, this information contributes to the effectiveness of various language processing tasks, including sentiment analysis, information extraction, and other classification tasks [5, 45, 69].

3.3.4 Sentiment

Sentiment analysis is a valuable tool for categorizing feedback and prioritizing requirements based on user sentiments. In automatic sentiment analysis, researchers employed unsupervised and manual approaches, leveraging various algorithms to assess sentiments expressed in user-generated content.

It utilized algorithms like SentiStrength, which are designed specifically for short informal texts such as social media posts, comments, or reviews. SentiStrength evaluated sentiments on a scale ranging from -5 to +5 [27, 40, 42]. Another approach is the Stanford CoreNLP for sentiment analysis [16, 45, 43].

In addition to automated methods, numerous studies adopted manual evaluation using a five-level Likert scale, ranging from “very positive” (+2) to “very negative” (-2) [1, 2, 22, 24, 25, 33, 47, 51, 57, 70].

Generally, the sentiment in the analyzed feedback tended toward neutrality, with a score of zero [22].

Researchers often observed positive sentiment polarities in posts expressing satisfaction (praise) and negative sentiment polarities in posts expressing dissatisfaction (*bug reports*, complaints). Notably, exceptions were found in categories such as *feature request*, where the expression of a user’s need for the system coexisted with a positive sentiment polarity.

Traditional sentiment analysis typically leans on English opinion lexicons and emotion-evoking words like “love,” “hate,” and “like” to discern sentiments in text. Yet, social media posts about software often feature technical terms such as “brick,” “fix,” and “crash,” each carrying complex layers of positive and negative human sentiments. Generic sentiment analysis methods frequently overlook these nuances [73].

To address this challenge, Williams and Mahmoud [73] adopted a supervised approach, training prediction models (NB and SVM) on manually labeled tweets for sentiment predictions - the algorithms had a binary classification to worry about positive or negative polarity. These algorithms achieved respectable levels of accuracy, recognizing many emotion-evoking software-specific words missed by SentiStrength. However, they acknowledged the necessity of large-scale datasets manually annotated data for training and achieving acceptable accuracy.

Innovative approaches included a deep learning-based method, as demonstrated by Jeong et al. [29]. Here, a deep neural network model analyzed users' sentiments in posts, focusing on keywords rather than entire sentences and creating a sentiment matrix.

As highlighted earlier, sentiment analysis plays a pivotal role in prioritizing requirements. The magnitude of the negative sentiment score correlates with the extent of user discontentment, emphasizing the urgency to address and rectify the situation promptly for user satisfaction.

3.3.5 Multi-Labeling versus Single

While most analyzed papers adopted a single-label approach, mapping posts to a sole label, a noteworthy percentage opted for a multi-labeling system. This alternative approach involves automatically classifying documents into one or more labels. The binary relevance method emerged as a popular choice among the methodologies employed in multi-label classification.

The binary relevance method entails training a binary classifier for each label, and the union operator is applied to the predictions from these independent classifiers, forming the final classification result. This method, while straightforward, assumes label independence, which poses challenges when there is a high correlation between categories, leading to information loss [44]. As an algorithm, NB consistently outperformed others in multi-label classification scenarios.

In specific studies [40, 42], binary classifiers demonstrated higher accuracy for predicting post types than multiclass classifiers. The explanation lies in each binary classifier utilizing two training sets: one where the corresponding type is observed and one where it is not, reinforcing its effectiveness.

Guzman et al. [22] highlighted an interrelationship between tweet relevance for non-technical stakeholders and the general public. The authors compared the performance of the binary relevance method against a multi-label classification method that considers each relevance category combination as a single class, producing comparable results.

McIlroy et al. [44] experimented with various multi-labeling approaches, including binary relevance, classifier chains (CC), and pruned sets with threshold extension. Classifier chains extend binary relevance by serially building models, with each k^{th} model taking into account the prediction of the $(k - 1)^{\text{th}}$ model, thus avoiding label independence assumptions. The third method treats each possible multi-label combination as a value in a single label, leveraging correlations between labels. Pruned sets with threshold extension exhibited superior accuracy and exact match results, outperforming other approaches.

3.3.6 Feature Engineering

Feature engineering constitutes selecting, manipulating, and transforming raw data into features suitable for supervised learning. This crucial step is prevalent in most analyzed papers, with researchers employing diverse techniques to enhance the effectiveness of their models.

Feature engineering serves as a critical aspect of the machine learning pipeline, contributing to the development of informative and relevant features that can significantly impact the performance of supervised learning algorithms. The application of various techniques underscores the adaptability and creativity researchers employ to extract meaningful patterns and representations from the data, fostering more accurate and robust models.

3.3.6.1 N-Gram Extraction

N-grams are a commonly employed text feature that entails grouping a consecutive sequence of N tokens (words or numbers) within a sentence [38, 41, 70]. Extracting n-grams introduces a parameter, denoted as n, which signifies the number of words in a sequence. This parameter can take any positive integer greater than one. However, the selection of n impacts the quantity of extracted n-grams, where higher values of n result in fewer extracted n-grams and, consequently, a reduced likelihood of successfully matching tweets to n-grams [41].

Rahman et al. and Maalej et al. [57] and [40] set the value of n to 2, focusing on bigrams. Using Maalej et al.'s examples: in the sentence "The app crashes often," the bigrams include "The, app," "app, crashes," and "crashes, often." [40] Bigrams capture contextual information in the review when utilized by document classifiers instead of single terms. The bigrams "crashes always" and "never crashes" convey distinct meanings and may reveal different types of reviews. In contrast, the three single terms "crashes," "never," and "always" might only indicate a general *bug report*, potentially overlooking nuances in sentiment or user experience.

The papers analyzed used the `nltk` Python library to generate n-grams.

3.3.6.2 Bag of Words & Term Frequency-Inverse Document Frequency

Document classification is a prominent technique in information science that involves assigning a document to a specific class based on its content. A fundamental approach to document classification is the Bag of Words (BoW) model [42].

In the BoW model, the classifier constructs a dictionary containing all the words in the corpus. For each document, the model calculates the presence or absence of each word in the dictionary, creating a representation that disregards the word order [15, 69, 74].

TF-IDF (Term Frequency-Inverse Document Frequency) builds on the principles of the BoW model but enhances this representation. It considers the frequency of terms (words) in a document and how common or rare those terms are across all documents in the corpus. This helps highlight important terms that are more unique and informative.

Ali et al. [4] introduced a variation of BoW known as Distributed BoW, which aims to capture semantic meaning and context, in contrast to the original version.

The analyzed papers commonly utilized the BoW model with the CountVectorizer. They also used a single TF-IDF model with the TfidfVectorizer. In two different studies, Khan et al. [7, 35] classified user comments using CountVectorizer, which assigns equal weight to all words in the corpus. They also assessed machine learning algorithms using TfidfVectorizer, experimenting with different parameters. The studies implemented an n-gram tokenizer with (1,3) as the minimum and maximum length of words, with “*Requirements knowledge acquisition from online user forums*” [7] additionally using an n-gram tokenizer with (2,3). This approach aimed to capture patterns of terms appearing together, providing indicators for intentional elements or issues in user comments.

Furthermore, these studies employed n-grams with both “word” and “char” analyzers to evaluate algorithm performance with words and character n-grams. This comprehensive approach allowed for a nuanced understanding of the content and context of user comments, enhancing the effectiveness of document classification models.

3.3.6.3 Length

The length of user reviews, measured by the number of words, emerges as a valuable classifier feature, offering insights into the informativeness and nature of the content. Maalej’s findings suggest a correlation between review length and relevance [40, 42, 62]. Specifically, they observed that most irrelevant reviews are short. In contrast, lengthy reviews often provide more comprehensive information, potentially indicating a detailed report on user experience or a specific software bug.

Leveraging the length of user reviews as a feature provides classifiers with additional context, enabling them to distinguish between concise, potentially less informative reviews and longer, more detailed narratives that may contain valuable insights or bug reports. This approach contributes to the nuanced analysis of user-generated content, enhancing the effectiveness of classification models in discerning the relevance and significance of reviews.

3.3.6.4 Frequency Based Features

Researchers commonly employed the technique of converting text into a vector space model to facilitate the understanding of user review text by different classifiers. The vector space model represents the text in a numerical format that machine learning algorithms can process. Among the various weighting schemes used for this purpose, the term frequency-inverse document frequency (TF-IDF) emerged as a prevalent choice in most analyzed papers [22, 25, 28, 62, 57].

TF-IDF assigns weights to words based on their frequency in a single user review and overall occurrence across multiple user reviews. The goal is to amplify the importance of unique or infrequent words within a single review while de-emphasizing words that occur frequently across many reviews. For instance, common words like “app” would receive lower weights, while less common words like “omission” would be assigned higher weights [44].

However, Panichella et al. [52] opted for a nuanced approach, utilizing only the term frequency (TF) instead of TF-IDF indexing. Their rationale was to avoid excessive penalization of terms that appeared frequently across reviews. They were particularly interested in retaining terms like

“fix,” “problem,” or “feature,” which, despite their frequent occurrence, could constitute valuable features guiding machine learning techniques in classifying sentences containing helpful feedback from a software maintenance and evolution perspective.

3.3.6.5 Word and Sentence Embedding

The analyzed papers explore word and sentence embedding techniques to capture semantic relationships and contextual information. Word embedding focuses on individual words, while sentence embedding extends the concept to entire text passages. Each word or sentence is mapped to a high-dimensional vector in both cases, and their positions in the vector space reflect semantic similarity.

Two prominent algorithms are used for sentence embedding: Sentence-Bidirectional Encoder Representations from Transformers (SBERT) and Universal Sentence Encoder (USE). SBERT utilizes the BERT (Bidirectional Encoder Representations from Transformers) network and employs a Siamese Network Architecture to generate high-quality sentence embeddings. On the other hand, USE, developed by Google Research, offers another robust approach to sentence embedding [57].

In word embedding, two prevalent algorithms emerge: Word2Vec and FastText.

Word2Vec creates word embeddings by adjusting weights based on forward and backward propagation, leveraging error gradients to refine the embeddings [57].

FastText, an unsupervised approach, learns high-dimensional vector representations for words from a substantial training corpus. Notably, FastText composes word embeddings from subword embeddings, differentiating it from Word2Vec, which learns embeddings for entire words. Stanik et al. [62] opted for FastText over Word2Vec due to its ability to handle subword embeddings. This distinction allows the model from their study to recognize words not present in the training corpus and capture spelling mistakes, enhancing its adaptability and performance.

3.3.7 Multilingual

While most of the papers analyzed predominantly focus on datasets comprising text in English, two notable papers delve into the intricacies of handling multilingual datasets [62, 27]. Both papers involve datasets consisting of English app reviews and tweets alongside Italian tweets. The English and Italian texts undergo the same preprocessing techniques with no language-specific adjustments. Notably, Stanik et al. [62] manage to extract verb tenses more effectively for Italian tweets than for English ones.

Stanik et al. [62] employed the `fastText` library for word embedding. Despite the availability of pre-trained models for several languages, including Italian, the researchers opted to train domain-specific models using parts of the dataset (English app reviews, English tweets, and Italian tweets). They also trained separate models for each component of the dataset. Significantly, Italian tweets demonstrated superior performance compared to their English counterparts.

In contrast, Henao et al. [27] explored monolingual and multilingual prediction approaches. The former involves training a separate model for each language and using it for classification,

while the latter trains a model capable of handling multiple languages. It is worth noting that multilingual prediction poses a more challenging task, as the model must generalize across vastly different languages. The researchers trained a Multilingual BERT model (M-BERT) on the English portion of the dataset, evaluating it on Italian data and vice versa. The model was pre-trained on the concatenation of Wikipedia corpora from 104 languages. Additionally, two monolingual prediction models were trained: one for English, pre-trained on English text from BookCorpus and Wikipedia, and one for Italian, pre-trained on Italian text from OPUS corpora and Wikipedia. The Italian model outperformed its English counterpart, like the findings in [62]. When contrasting monolingual and multilingual approaches, the former outperformed the latter. Intriguingly, M-BERT demonstrated superior performance when trained on English data and applied to Italian tweets.

Oehri and Guzman [51] also had non-English feedback, but they merely translated it into English.

3.3.8 Data Imbalance

Several studies encountered a common challenge in supervised machine learning, known as imbalanced data. This challenge arises when the dataset disproportionately represents one class over others, leading classifiers to skew towards the majority class and potentially resulting in biased and inaccurate results.

Most studies adopted the Synthetic Minority Over-sampling Technique (SMOTE) to mitigate the impact of imbalanced data. SMOTE addresses imbalanced datasets by applying oversampling to the minority classes, generating synthetic examples to achieve a more balanced distribution [7, 28, 35, 57].

However, Henao et al. [27] opted for a different approach to address imbalanced data. Faced with a skewed distribution where the majority class was labeled “irrelevant,” they employed the Random Undersampling technique. This technique involves randomly selecting data points from the majority class and excluding them from the dataset until a balanced distribution is achieved. While undersampling may discard potentially helpful information, it was deemed acceptable in this study because the majority class was labeled as “irrelevant,” thus, removing these instances did not compromise the study’s objectives.

3.3.9 Grouping

A crucial step in the analyzed studies involves grouping tasks to understand the dataset better. Unsupervised techniques, specifically clustering and topic modeling, are commonly applied to achieve this goal.

3.3.9.1 Clustering

While clustering assumes that each user review instance belongs to precisely one cluster (group), this assumption may not hold for instances that exhibit multiple topics to varying degrees, particularly at the sentence level [12].

To address this challenge and identify groups of related reviews, clustering is performed on reviews belonging to the same category, such as those categorized as *bug report* or *suggestions for new features*. This clustering process serves two crucial purposes:

- Developers, faced with a large volume of reviews in a specific category, may experience information overload. Clustering helps manage this overload, preserving the advantages provided by review classification;
- Knowing the popularity of a specific problem (bug) or a particular feature is valuable information that indicates the urgency of addressing bugs or implementing features [70, 71].

As highlighted by Villarroel et al. [70], it is unnecessary to cluster all user reviews; instead, the focus is on clustering reviews classified as *bug report* or suggestions for new features (*feature request*)—categories crucial for software development and evolution.

The Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm is applied for review clustering. DBSCAN identifies clusters as areas of high element density, allowing for the detection of both dense and sparse regions. Elements in low-density regions are assigned to singleton clusters, which consist of a single element. This approach efficiently captures the grouping of user reviews based on shared characteristics, facilitating a more focused and actionable analysis for developers.

3.3.9.2 Topic Modeling

After the classification of the dataset, topic modeling is employed to extract common themes, providing a concise summary of topics present in the entire document [12, 14, 48]. This technique is widely used in text mining to unveil hidden semantic relations within a text document, particularly when the document pertains to a specific domain, such as extracting requirements for software development from social media posts [4].

Unlike clustering, topic modeling assigns multiple topics to each review instance. This flexibility makes it suitable for capturing diverse themes within individual reviews. Using Chen et al.'s example [12], “the review “*Just add emojis and more themes.*” is modeled as a distribution over two topics (50% “*emoji*”, 50% “*theme*”).”.

Latent Dirichlet Allocation (LDA) is the most popular algorithm for topic modeling in this context. LDA is well-established and known to perform effectively with short-form text content, including tweets and posts from various social media platforms [12, 14, 47, 48].

One approach to applying LDA is using tools like jsLDA, which leverages JavaScript to run topic modeling conveniently within a web browser. Ali et al. [4] utilized this tool to model related topics within the smartwatch domain, as illustrated in [Figure 3.2](#).

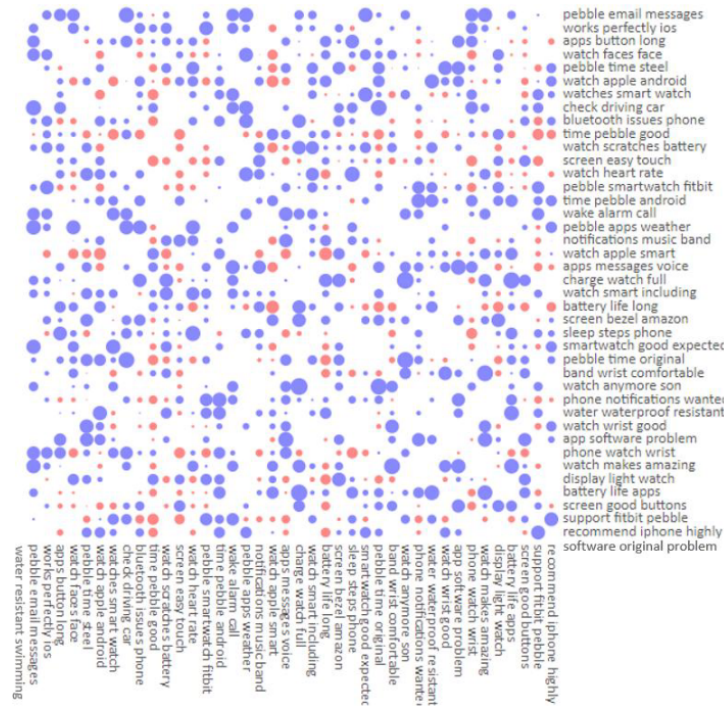


Figure 3.2: Topic modeling for smartwatch domain [4].

The visualization depicts topic combinations, with blue circles indicating higher-than-expected frequency and red circles representing lower-than-expected frequency. This information helps software developers understand user scenarios and considerations, contributing significantly to eliciting user requirements for the following product release.

While LDA is inherently an unsupervised learning model, a supervised variant called Labelled Latent Dirichlet Allocation (LLDA) exists. McIlroy et al. [44] utilized LLDA to probe the reasons behind the underperformance of specific labels. Supervised approaches like LLDA construct models that capture correlations between training examples and their corresponding labels, facilitating predictions for future labels associated with new examples

Even though LDA is widely adopted, some authors, such as Guzman et al. [24], and Noaen and Far [50], opt for Bitern Topic Modeling (BTM). This choice is motivated by BTM's specialization in handling short text and its demonstrated outperformance, particularly in grouping short and long text compared to LDA. This flexibility ensures that the chosen topic modeling technique aligns with the data's characteristics, enhancing its effectiveness in extracting meaningful insights.

3.3.10 Summarization

Summarization's objective is to distill the data to its essential points, facilitating better visualization for stakeholders.

While not extensively detailed in most papers, the summarization process is a vital component for effectively presenting key insights. Rahman et al. [57] used summarization immediately after

mining content from social media to eliminate irrelevant parts and retain high-quality data. The study employed three text summarization algorithms—SumBasic, TextRank, and LexRank. Their performance was rigorously evaluated using Recall-Oriented Understudy for Gisting Evaluation (ROUGE-N) metrics, specifically ROUGE-1, ROUGE-2, ROUGE-3, and ROUGE-4. The analysis revealed that TextRank was the most suitable for summarizing their dataset, demonstrating its effectiveness in extracting essential information.

Williams and Mahmoud [74] employed frequency-based extractive summarization techniques. The study evaluated three methods known for their efficacy in micro-blogging data on social media: hybrid TF, and hybrid TF.IDF, and SumBasic. Performance assessment, measured using recall, indicated that, on average, SumBasic outperformed hybrid TF.IDF and TF in the summarization task. This highlights the importance of selecting appropriate summarization techniques tailored to the characteristics of the data being analyzed to enhance the clarity and impact of the visualized insights.

3.3.11 Training and Evaluation

The papers under analysis adhere to established practices in the machine learning domain for training and evaluating classifiers.

K-fold cross-validation, a widely employed technique, is prevalent in these studies. The value of k , representing the number of folds, is typically set at either five or ten, striking a balance between variance and bias by the variance-bias trade-off. Rahman et al. [57] exemplified this choice by constructing the GridSearchCV model to determine the optimal estimator parameters and n -fold value, ultimately settling on a 10 for k .

Some studies opt for the holdout method, randomly splitting the truth set into training (70%) and testing (30%) sets. While this approach is utilized in specific papers [74], it is regarded as less precise due to its potential for increased variance compared to k -fold cross-validation [40, 42].

Maalej et al. [38] introduced a variant of cross-validation known as Monte Carlo cross-validation. This method involves ten runs with a random 70:30 split ratio for each run. In each iteration, 70% of the truth set is randomly selected for training, and the remaining 30% is designated as the test set.

Traditional machine learning metrics—precision, recall, and f -measure—are employed across most studies to gauge classifier performance. Some studies accurately augment these metrics, representing the percentage of correctly classified records in the test set.

However, an ongoing debate surrounds the preference between precision and recall, fueled by considerations of misclassification—False Negatives or False Positives—and their associated costs. Some studies advocate prioritizing recall over precision, recognizing the challenges of manually identifying False Negatives within extensive user comments [27]. Emphasizing recall aims to minimize the risk of overlooking crucial information, ensuring a comprehensive approach to handling user feedback within the domain of requirements engineering.

Despite the merits of precision and recall, imbalanced label distributions pose challenges. In cases where one label significantly outweighs others, accuracy may exhibit bias towards the more

prevalent label. Addressing this issue, Ebrahimi and Barforoush [18] adopted a weighted f1-measure metric for label distribution, offering a more accurate portrayal of classifier performance.

Maalej and Nabil [42] conducted pairwise statistical tests (Paired T-tests) to assess the statistical significance of differences in precisions and recalls. Additionally, they quantify the informativeness of classification features.

Tizard et al. [69] incorporated resampling techniques, generating Receiver Operating Characteristic (ROC) curves to identify the optimal resampling technique and determine the ideal number of iterations. ROC curves plot the percentage of True Positives (TP) against the percentage of False Positives (FP) for each classifier, providing insights into their performance.

3.4 Visualization of Candidate Requirements from Social Media

In the ever-evolving software development landscape, extracting valuable insights from user feedback scattered across social media platforms has become a crucial aspect of the requirements engineering process.

Panichella et al. [53] introduced an all-in-one tool, ARdoc (App Reviews Development Oriented Classifier), designed to automatically classify pertinent sentences within user reviews, particularly focusing on software maintenance and evolution perspectives. ARdoc presents users with a user-friendly Graphic User Interface, as hyperref[fig:ardoc]Chapter 3.3 illustrates.

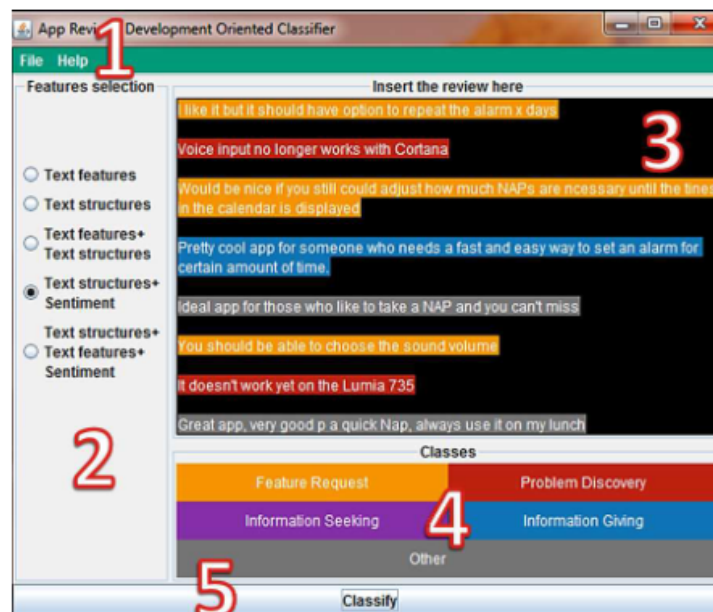


Figure 3.3: Ardoc's GUI [53].

The tool's interface is structured into distinct sections:

- Menu Bar (Point 1): Allows users to create new windows, load text for classification, import reviews, and export classified data;

- Features Selection Panel (Point 2): Allows users to choose features for review classification;
- Input Text Area (Point 3): Enables users to input or paste reviews for classification and visualizes the results;
- Panel with Captions (Point 4): Displays categories and their associated colors;
- Classify Button (Point 5): Initiates the classification process and generates results.

For the classification of multiple reviews, users can separate them using blank lines, as demonstrated in the accompanying image. The tool displays the classification results in real-time, categorizing reviews into types such as *feature request*; two of them were classified as *problem discovery*; two were classified as *other* and finally, there was a single one that was *information giving*.

While ARdoc prioritizes simplicity and intuitiveness, it lacks advanced features such as filtering, review navigation, or saving analyzed reviews. However, it provides users with an interactive and immediate classification experience.

Stanik and Maalej [63] introduced *OpenReq Analytics*, a comprehensive software requirements intelligence service designed to collect, process, analyze, and visualize user feedback from app stores and **Twitter**. Implemented through ten microservices, the service offers a unified web-tool interface that accesses necessary data from the Datastorage microservice. The main view is the dashboard shown in hyperref[fig:openreq1]Figure 3.4. The main view, illustrated in hyperref[fig:openreq2]Figure 3.5, comprises three rows with distinct functionalities:

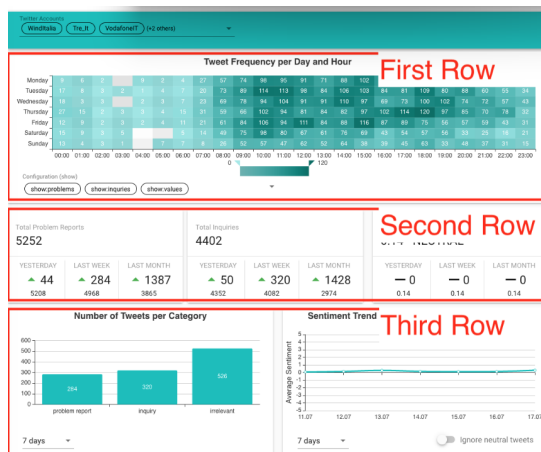


Figure 3.4: OpenReq's Dashboard [63].

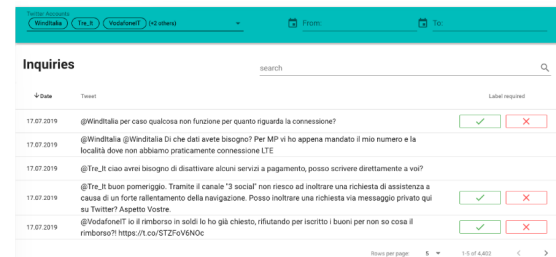


Figure 3.5: OpenReq's Focus Views [63].

- The first row presents a heat map offering insights into when users provide feedback, indicating issue occurrences or new functionality considerations. OpenReq Analytics allows filtering for relevant entries, focusing on problem reports, feature requests, or inquiries;
- The second row features three trend reports, providing information within predefined time windows. These reports offer overviews of user problems, inquiry rates (including feature requests, and overall user sentiment when providing feedback;

- The third row empowers users to analyze specific periods, such as software releases. The sentiment chart aids in understanding when user sentiment returns to normal after peaks in problem reports, offering insights into the effectiveness of customer support strategies.

OpenReq Analytics comprises two primary focus views (refer to [hyperref\[fig:openreq2\]](#) Figure 3.5), each consolidating information related to either problem reports or inquiries. These focus views are designed to aggregate feedback, empowering development teams to delve into specific user insights. Users have the capability to search for keywords of interest, filter feedback sources and languages, and define time frames. The separation into problem reports and inquiries is executed by a machine learning model, allowing development teams to validate or contest the classification. In instances of disagreement, the system suggests alternative categories, contributing to the refinement of the model’s accuracy. Importantly, feedback is only labeled when the classification model expresses uncertainty, ensuring targeted improvements and preventing development teams from being inundated with labeling tasks.

Villarroel et al. [70] introduced CLAP (Crowd Listener for releAse Planning), which categorizes user reviews based on their information, cluster related reviews, and automatically prioritize review clusters for implementation during app release planning.

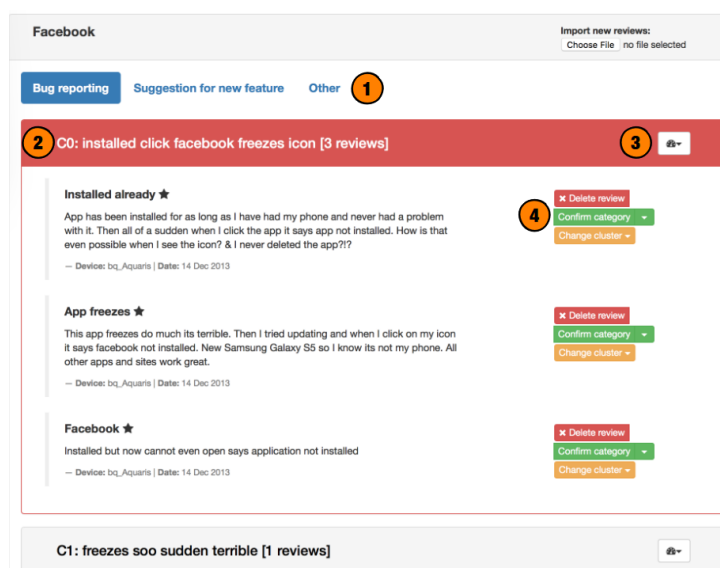


Figure 3.6: CLAP’s GUI [70].

[hyperref\[fig:clap\]](#) Figure 3.6 provides an excerpt of CLAP’s user interface, illustrating its functionality. Users can import user reviews from **Facebook** into CLAP, resulting in categorized reviews such as “Bug reporting” (*bug report*), “Suggestion for new feature” (*feature request*), and “Other” at the top of the image. The *bug report* category can be expanded to reveal review clusters (element 2), each labeled with an identifier (C0 and C1) and the five most frequent terms in the reviews. Red clusters (C0) are deemed as “high priority,” while grey clusters (C1) are labeled “low priority.” The tool includes a feedback mechanism (element 3) allowing developers to indicate

their intention to implement the reviews in a cluster, aiding in manual feedback for prioritization refinement based on implemented features/bugs. Expanding a cluster reveals the reviews it contains, providing detailed insights. In the example, cluster C0's three reviews report a similar problem related to **Facebook**, and a feedback mechanism (element 4) allows users to change the review category or assign it to a different cluster.

Di Sorbo et al. [16] present SURF (Summarizer of User Reviews Feedback), a tool designed for analyzing, classifying, and systematically summarizing information within app reviews.

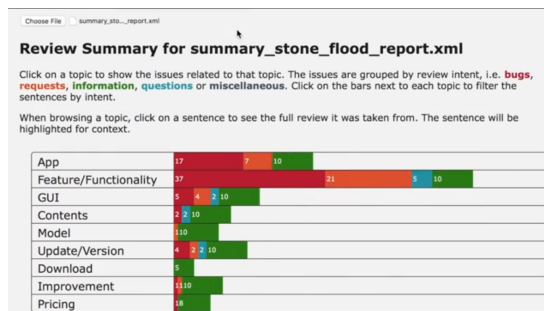


Figure 3.7: Generated Summary XML file¹



Figure 3.9: Most Relevant Review Sentences.¹

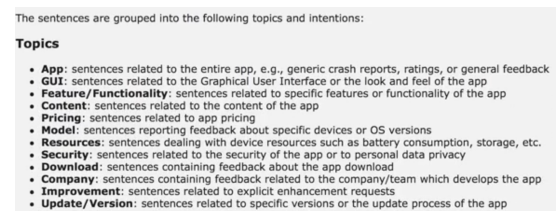


Figure 3.8: Topics in the Generated Summary.¹

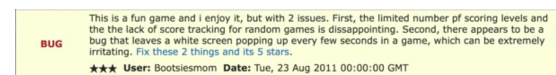


Figure 3.10: Complete Review.¹

SURF is implemented as a command-line tool requiring users to specify the XML file path containing user reviews for summarization and the output file path for the generated summary. The resulting summary is provided as a single XML file.

Post-summarization, users can analyze the output report using a web-based browsing utility in the SURF package (report-viewer.html). The viewer categorizes reviews by topic and presents an integrated bar chart indicating the distribution of review intentions for each topic (hyperref[fig:surf1]Figure 3.7). This visual representation lets developers quickly grasp reviews' distribution across topics and intentions. By clicking on a topic, users can access all related reviews while selecting a bar-chart segment for specific intent filter reviews for the chosen topic. The most relevant sentence is displayed for each review (hyperref[fig:surf3]Figure 3.9), and clicking on a sentence expands the review text, revealing additional metadata to provide context within the original review (hyperref[fig:surf4]Figure 3.10).

¹<https://www.youtube.com/watch?v=Yf-U5ylJXvo>

In a research conducted by Maalej et al. [40], an investigation into processes for the automatic classification of app reviews from various app stores led to the development of a review analytics tool. The prototype of the review analytics tool is depicted in the Figures 3.11, 3.12, 3.13, showcasing its features and functionalities.

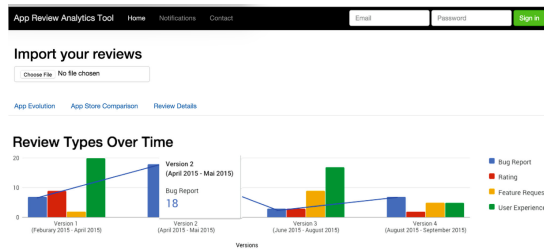


Figure 3.11: App Reviews Evolution View [40].



Figure 3.12: App Store Comparison View [40].

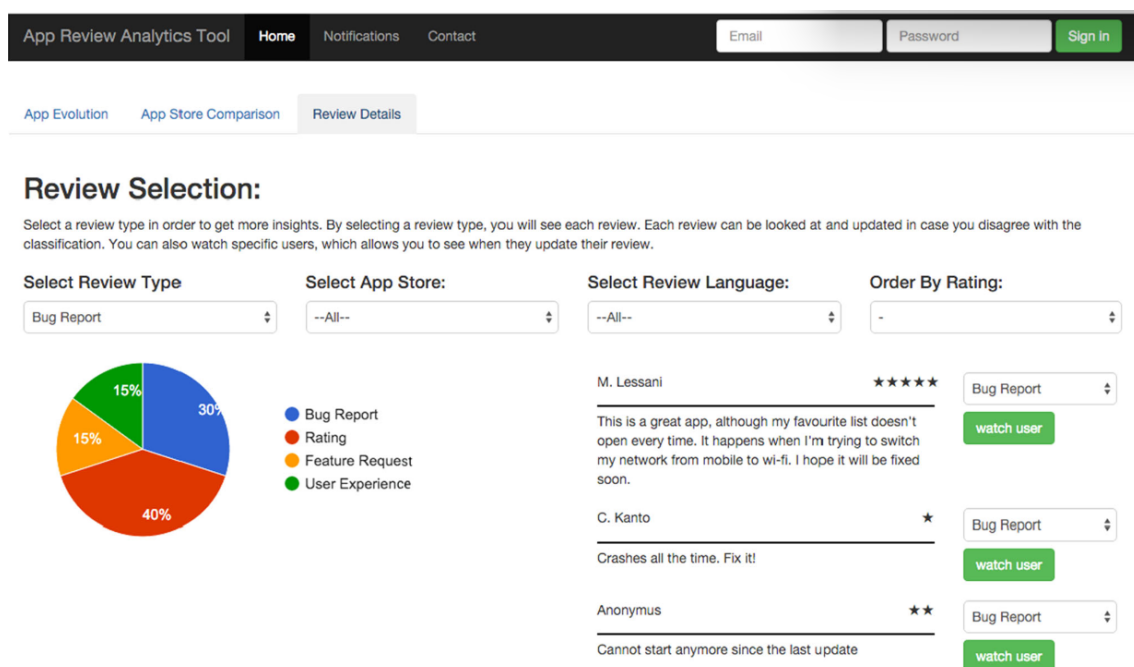


Figure 3.13: Review Details View [40].

The navigation bar at the top of each figure provides users with options to log in, access support, submit feedback, view notifications (such as reviews from monitored app users), or navigate to the home screen to explore the dashboard. Users can import reviews from a file (bellow “Import your reviews” in the hyperref[fig:automatic1]Figure 3.11) or employ a crawler to automatically retrieve app reviews based on a link to the app page in the respective store.

Once reviews are imported, users gain insights into current trends. hyperref[fig:automatic1]Figure 3.11 illustrates a grouped bar chart showcasing the number and ratios of classified reviews for each category (*bug report*, *feature request*, *user experience*, and *rating*) and each app version. This overview provides information about what app users have reported within specified time frames.

Users can switch to the App Store Comparison tab for apps distributed across multiple stores, as seen in [Figure 3.12](#). Pie charts represent each store's overall distribution of review types, offering a comparative view of app performance. Project managers can leverage this information to set development priorities.

The third tab, Review Details, provides detailed information about each review type, as seen in [Figure 3.13](#). Developers seeking in-depth insights into the current version or specific review types can retrieve automatically extracted information in this view. Users can narrow the selection of displayed reviews using drop-down menus, filtering by review type, source (app stores), language, and star rating. The filtered data is visualized below the menus, with a pie chart on the left for additional filtering and a scrollable list of reviews on the right. For each review, users can correct misclassifications and “watch” reviews to track the impact of updates on user feedback. The correction can help improve the classification as it extends the training set. Developers can use the “watch” feature to track if an update helped the user and if it had any effect on the review.

3.5 Conclusion

Many papers have delved into extracting requirements from social networks, examining both manual and automated approaches. The first group of papers analyzed the feasibility of extracting requirements from social networks and expert opinions in the field and emphasized the importance of this idea.

In automatically classifying extracted requirements from social media, most studies adhered to a well-established methodology: data extraction via a scraper/API, preprocessing, categorization using diverse algorithms, and subsequent evaluation of each algorithm's performance. Numerous research endeavors incorporated additional steps such as sentiment analysis, grouping, summarization, feature engineering, and addressing data imbalance issues.

Embracing the established methodology, we aimed to enhance it with additional steps while maintaining a vigilant eye on the model's performance. A thorough analysis of existing literature highlighted considerable variation in the effectiveness of machine learning classifiers for requirement classification, contingent upon the dataset and processing techniques employed. Techniques such as stemming, lemmatization, and stop-word removal exhibited diverse impacts across classifiers and datasets. Notably, the optimal algorithm and preprocessing techniques were found to be dataset-dependent, emphasizing the need to tailor approaches to the unique characteristics of the data.

Given these findings, our approach involved experimenting with various classifiers and diverse preprocessing techniques identified during our literature review. We were particularly committed to implementing summarization techniques in our dataset, recognizing their potential to significantly aid our objective of developing a platform with a robust visual representation of candidate requirements. Condensing the extracted feedback into a concise format through summarization

aligned with our goal of providing stakeholders with an accessible overview rather than requiring them to sift through numerous social media posts.

Empirical studies underscored the critical value of user feedback garnered from app stores. Yet, they also reveal that user insights manifest more swiftly on alternative social media platforms, such as **Twitter**. In optimizing developers' responsiveness to bug-related challenges, prioritizing the analysis of these dynamic social networks became paramount over app store data alone.

A unique aspect of our research lay in its distinct context: working closely with a specific company and one of its products, in this case, MOG and Vizzi. Unlike traditional end-user scenarios, where feedback is openly shared, Vizzi's end-users are corporate entities less inclined to discuss their experiences on social media or app stores. In light of this, our data collection strategy diverged from conventional methods, avoiding reliance on interactions with official accounts or searching for posts that include the product name for which we wish to gather feedback. Vizzi, a versatile platform tailored for entertainment companies, allows customization to meet brand requirements and facilitates content sharing with customers. While end-users may not know they are using that product, they interact with its features. Our approach involved examining user interactions with official accounts of Vizzi's clients or platforms powered by Vizzi. Additionally, we employed keyword searches to identify social media posts referencing these platforms. By filtering posts mentioning specific Vizzi features and extracting valuable insights or advice from clients, we aimed to construct a nuanced understanding of possible requirements for Vizzi.

The studies we examined were exclusively conducted in academic environments, lacking direct involvement with industry or companies, even though some focused on specific software applications. The absence of stakeholder engagement contributed to the limited emphasis on visualization in these papers, particularly concerning the representation of candidate requirements. However, five studies delved into visualization, which was discussed in depth in hyper-ref[sec:visualization]Section 3.4. We intended to draw inspiration from these studies regarding features and the overall user experience as we developed our platform, leveraging valuable insights from their successes and learning from their limitations. ARdoc from Panichella et al. [53] allows users to input or paste reviews for classification but falls short in features like filtering, review navigation, and saving analyzed reviews. It provides users with an immediate classification, however. We aimed to offer a more refined experience, extracting reviews and incorporating advanced features like filtering that ARdoc lacks. OpenReq Analytics [63] extracts reviews from app stores and tweets, referencing official accounts of the products they are commenting on. We also aimed to do this; however, as described above, our process of finding posts must be somewhat different. OpenReq Analytics includes a valuable feedback mechanism where users can validate or contest classifications. This mechanism, essential for refining the model's accuracy, is a feature we planned to implement on our platform to enhance user interaction and the overall RE process. Villarroel et al.'s CLAP [70] requires users to manually import reviews, similar to ARdoc. However, it distinguishes itself with a feedback mechanism and clustering. The automatic prioritization of clusters based on user reviews is a practical feature that contributes to the prioritization phase of RE. This feature aligned with our goals, and we intended to implement a similar clustering mech-

anism on our platform. SURF’s main feature [16] is the systematic summarization of information within reviews, presenting it in a single XML file. Despite not being a full-fledged platform, its summarization features are compelling. We recognized the utility of such features for users, as they provide a quick understanding of requirements without extensive reading. While we aimed for a more comprehensive platform, we intended to incorporate efficient summarization features inspired by SURF. The tool developed by Maalej et al. [40] allows users to import reviews from a file or extract the reviews from an app page. It classifies reviews and produces a bar chart with categories. Its feedback mechanism and the “watch” feature, tracking the impact of updates on reviews, are intriguing. However, implementing such features on open social media platforms poses challenges. It would require tracking users’ posts to identify product mentions over time. In contrast, Maalej et al.’s tool operates within app stores, where reviews are confined to the application’s page, simplifying the implementation of this feature. While each tool has its strengths and limitations, we aimed to incorporate the most compelling features, refining and enhancing the user experience in the context of requirements engineering.

It is crucial to note that our dataset may not be entirely in English since we dealt with an extensive range of streaming platforms with users worldwide. Therefore, in data extraction and analysis, we recognized the need to account for the multilingual nature of user feedback. Anticipating a multilingual dataset, we planned to follow a monolingual approach, creating different models for each language, drawing inspiration from Henao et al. [27] and Stanik et al. [62]. However, if the results were not satisfying, we were prepared to adopt a naive translation approach, translating every text to English for easier processing, like Oehri and Guzman [51].

Chapter 4

Analysis of Social Media Feedback

Our solution consists of two fundamental phases: first, the development of a social media feedback processing workflow, and second, the development of an intuitive visualization platform to empower stakeholders to act on the requirements gathered from social media. In this chapter, we will go into detail on the first phase.

We elaborated a workflow diagram to illustrate our work that has the following phases:



Figure 4.1: Social Media Feedback Processing Workflow Diagram.

4.1 Data Extraction

We meticulously selected a range of prominent streaming platforms, some of which use Vizzi and others that do not, and obtained data from diverse social media sources. The selected streaming platforms include:

- Netflix;
- Hulu;
- (HBO) Max;
- Disney+;
- Peacock;
- Prime Video;
- RTP Play;
- ACL Player;
- Itaú Cultural Play;
- Gas Digital Network.

We chose these streaming services due to their popularity and our familiarity with them. Additionally, the last four are MOG's customers, with the last three being Vizzi users.

We extracted data from various social media sources to gather insights into user opinions and sentiments regarding these platforms. Our data collection spanned multiple platforms, including **iTunes**, **Google Play**, **Twitter**, and **Reddit**. These platforms were chosen to capture a comprehensive view of user feedback and discussions surrounding the streaming services.

We encountered several challenges during this phase, primarily related to accessing reviews from specific platforms. Notably, ACL Player is exclusively available for download from its website, while Gas Digital Network is inaccessible on **Google Play**. As a result, we were unable to retrieve reviews for these platforms from **iTunes** and **Google Play** for ACL Player and just from **Google Play** for Gas Digital Network.

We narrowed our scope to posts from the past year to streamline our data collection efforts and focus on recent insights. This allowed us to gather a manageable yet representative sample of user-generated content for analysis.

4.1.1 Mobile Stores

We leveraged open-source libraries from **iTunes** and **Google Play** to extract data. For **iTunes**, we employed `itunes_app_scraper` developed by Digital Methods Initiative¹, and `app_store_scraper` developed by Facundo Olano². Similarly, for **Google Play**, we utilized two versions of `google_play_scraper`: one by Digital Methods Initiative³ and the other by JoMingyu et al.⁴. While the libraries from Digital Methods Initiative provided the streaming platform IDs, we relied on the libraries by Facundo Olano and JoMingyu et al. to obtain reviews from **iTunes** and **Google Play**, respectively. This was due to the former only retrieving application details, lacking review information, and the latter requiring the application ID for a successful search.

In the case of Itaú Cultural Play, we encountered difficulties extracting reviews from **Google Play**. Despite existing information, the Digital Methods Initiative's library failed to find the correct ID for the application. After multiple attempts with different search terms, we developed a program to scrape reviews from **Google Play**, explicitly targeting Itaú Cultural Play.

Limiting the time frame for data extraction from **iTunes** was feasible. However, due to limitations in **Google Play**, we extracted many reviews and filtered out those not from the past year.

4.1.2 Reddit

The literature introduces two open-source tools for data extraction from *Reddit*: *Pushshift* and *Python Reddit API Wrapper (PRAW)*. Unfortunately, the former was not feasible due to restrictions imposed by its creators after **Reddit**'s API limits announcement. Access to Pushshift was limited to **Reddit** moderators, and our project did not meet the qualifications⁵. Consequently, we

¹<https://github.com/digitalmethodsinitiative/itunes-app-scraper/tree/master>

²<https://github.com/facundooolano/app-store-scraper>

³<https://github.com/digitalmethodsinitiative/google-play-scraper>

⁴<https://github.com/JoMingyu/google-play-scraper/tree/master>

⁵https://www.reddit.com/r/pushshift/comments/14ei799/pushshift_live_again_and_how_moderators_can/

had to forgo utilizing `Pushshift`. However, we successfully employed `PRAW`, which required us to create credentials with **Reddit**'s API.

Our initial search strategy involved conducting exact searches across **Reddit** for the names of streaming services. The Python code snippet below illustrates this approach using `PRAW`:

```
1 reddit.subreddit('all').search('disney plus', time_filter='year', sort='new')
```

It is worth noting that this method has a post limit of 100, whereas `PRAW` allows the extraction of up to 1000 posts. Unfortunately, this extended limit is not achievable using the search method alone.

In addition to posts, we explored the extraction of comments associated with each post.

4.1.3 Twitter

Many analyzed studies used the open-source tool, `tweepy`. This was impossible for us due to the **Twitter** API restrictions; currently, `tweepy` only retrieves one-week-old tweets.

We researched other tools, such as `snsrape`, `twint`, `tweeterpy`, and `twikit`. `snsrape` was rejected since it is not operable anymore. Due to the new update to **Twitter** policies in 2023, it is necessary to authenticate to scrape information from **Twitter**, which is something the developers said that they were not going to implement⁶. We also rejected `twint`, even though it authenticated the user because the developer archived the project before the changes to **Twitter**'s policies, so the tool does not work anymore. `tweeterpy` did not work as well because it was impossible to establish a connection even after multiple attempts. We ended up using `twikit`. With it, we were able to search tweets and extract their information. However, it was not possible to extract their replies.

As for our search strategy, we implemented something similar to our **Reddit** search strategy: exact searches across **Twitter** for the names of streaming services. The Python code snippet below illustrates this approach using `twikit`:

```
1 client.search_tweet('"disney plus" since:2023-01-01', 'Top', count=20)
```

The `search_tweet` method has a 20-tweet limit. However, gathering more tweets with the next method is possible. Nonetheless, it is not possible to extract more than 100 tweets. `twikit` also has that as a rate limit; however, the value resets after 15 minutes⁷.

To extract tweets' replies, we had to develop our scraper. We tried using the Python library, `beautifulsoup` for this endeavor but were unsuccessful due to **Twitter**'s restrictions against bots. This library allows developers to extract data from static HTML pages and XML documents. With the restrictions, it is necessary to interact with **Twitter** due to the necessity of accepting

⁶<https://github.com/JustAnotherArchivist/snsrape/issues/996>

⁷<https://github.com/d60/twikit/blob/main/ratelimits.md>

cookies. To circumvent this issue, we used another Python library, `selenium`, allowing user interaction with webpages when scraping. Due to the nature of the HTML code, it was not possible to extract engagement levels of the tweets (number of replies, retweets, and likes), only its text and ID. Due to `selenium`'s 300000 ms timeout, the extraction process took time.

We also noted that when conducting searches with a significant time frame, such as a year, the search results only included tweets published within the seven days preceding the search.

The [Table 4.1](#) depicts the number of posts about each streaming service and the social media they were extracted from:

	iTunes	Google Play	Twitter	Reddit	Total
ACL Player	NA	NA	32	487	519
Gas Digital Network	52	NA	54	109	215
Itaú Cultural Play	5	3000	164	7	3176
RTP Play	15	199	196	183	593
Netflix	1013	3000	604	50	4667
(HBO) Max	302	3000	631	418	4351
Hulu	1004	3000	1021	58	5083
Prime Video	1008	3000	1116	80	5204
Peacock	1008	3000	179	655	4842
Disney Plus	1008	3000	751	183	4942
Total	5415	24299	4748	2230	36702

Table 4.1: Number of Extracted Posts.

4.1.4 Statistical sampling

To assess the relevance of our datasets, we conducted statistical sampling. We randomly selected 1% of each dataset and categorized the sampled posts as either relevant or irrelevant.

Specifically, we labeled 54 posts from the **iTunes** dataset, with 54% of them deemed relevant. Similarly, we labeled 142 posts from the **Google Play** dataset, with 68% of them found to be relevant. However, we found none relevant after analyzing 35 posts from **Twitter** and 20 posts from **Reddit**. Consequently, we determined that a change in strategy was necessary for collecting data from these platforms.

4.1.5 New Strategy regarding Reddit

In response to the need for more relevant posts, we implemented a new strategy for data collection on **Reddit**. The key changes made are outlined below:

1. **Selection of SubReddits:** Instead of conducting a global search, we focused our search efforts on specific forums (*SubReddits*) related to the streaming platforms of interest. These included:

- r/ACL_cornhole, r/Cornhole (regarding ACL Player);

- r/peacock (about Peacock Tv);
- r/DisneyPlus (Disney+);
- r/netflix (Netflix);
- r/hbo, r/HBOMAX ((HBO) Max);
- r/Hulu, r/HuluLive (Hulu);
- r/amazonprime, r/AmazonPrimeVideo (Prime Video);
- r/SamsungTV, r/appletv, r/samsung, r/VisionPro, r/AppleVisionPro, r/playstation, r/ShieldAndroidTV, r/Chromecast, r/GalaxyTab, r/LGOLED, r/OLED, r/Roku, r/firetvstick, r/IPhoneApps, r/apple, r/Android, r/androidapps, r/AndroidTV, r/samsung, r/fireTV, r/techsupport (devices).

2. **Refined Search Queries:** In *SubReddits* directly related to streaming services, we refined our search queries by using keywords such as “feature,” “option,” “fix,” or “issue” to target posts discussing specific features or problems. For other forums, we conducted exact searches.

The [Table 4.2](#) summarizes the number of posts extracted for each streaming platform:

	Reddit
ACL Player	428
Gas Digital Network	99
Itaú Cultural Play	4
RTP Play	6157
Netflix	5883
(HBO) Max	5286
Hulu	2901
Prime Video	3161
Peacock	3293
Disney Plus	4554
Total	31766

Table 4.2: Number of posts regarding each streaming service, using a new search strategy for **Reddit**.

Furthermore, we conducted a random sampling of 1% of the extracted posts, resulting in a larger sample size of 317 posts. We then categorized these posts as relevant or not, with the relevance rate improving from 0% to 34%.

4.1.6 New version of Twitter

We also adjusted the search query for **Twitter** to mirror the approach taken for **Reddit**. Since **Twitter** does not have dedicated forums for each streaming service, our search query included terms such as “feature,” “option,” “issue” and “fix” along with the name of the streaming service.

For example, the search query for Peacock would be “‘peacock’ (feature OR fix* OR issue).” We utilized the wildcard feature by appending “*” after “fix” to capture variations such as “fixing” and “fixed.”

To overcome the limitation of search results being restricted to the past seven days, we conducted multiple searches with date filters spanning month-to-month intervals instead of searching for posts over an entire year. Subsequently, we combined the results from these searches, resulting in a notable increase in the number of obtained posts for each streaming service. The following table illustrates the number of extracted posts:

	Twitter
ACL Player	69
Gas Digital Network	64
Itaú Cultural Play	300
RTP Play	1029
Netflix	1010
(HBO) Max	1170
Hulu	1349
Prime Video	1170
Peacock	1264
Disney Plus	988
Total	8413

Table 4.3: Number of posts regarding each streaming service, using a new search strategy for **Twitter**.

Finally, we performed a random sampling of 1% of the dataset, comprising 84 posts. These posts were then labeled as relevant or irrelevant. Notably, we observed an improvement in relevance from 0% to 25% following this adjustment.

4.1.7 Search for “bugs”

To further improve the search for posts on **Twitter** and **Reddit**, we incorporated the term “bug” into our search queries. We anticipated that this addition would yield more posts that could be classified as *bug report*. As a result, this adjustment increased the number of posts by 3053 for **Reddit** and 3300 for **Twitter**, so x posts in total for **Reddit** and 11762 for **Twitter**.

Subsequently, we randomly sampled 1% of the datasets and classified them as relevant or not. For **Reddit**, this comprised 30 posts, with 23% deemed relevant. Meanwhile, for **Twitter**, 33 posts were sampled, with 33% classified as relevant.

These additions had a notable impact on the overall relevance of the datasets. Specifically, the relevance of **Reddit**’s dataset decreased from 34% to 27%, while **Twitter**’s relevance increased from 25% to 26.4%.

Overall, the total number of posts we labeled manually and their relevant percentage can be seen in Table 4.4.

Source	Number of Labeled Posts	Percentage of Relevant Posts
iTunes	54	54%
Google Play	142	68%
Reddit	348	27%
Twitter	117	26.4%
Total	660	-

Table 4.4: Random Sampling of 1% of the Dataset.

4.2 Preprocessing

Upon extracting the data from various social media platforms, we standardized the data formats and ensured uniformity across datasets. We initiated the preprocessing phase with the following steps:

1. **Date Formatting:** We standardized the date format across all datasets to “month day year time” (e.g., “May 01 2023 04:43:01”);
2. **Column Alignment:** We aimed to harmonize the columns across datasets as much as possible. The initial columns for each dataset were as follows:
 - **Google Play:** *reviewId, userName, userImage, content, score, thumbsUpCount, reviewCreatedVersion, at, replyContent, repliedAt, appVersion*;
 - **iTunes:** *date, review, rating, isEdited, title, userName, developerResponse*;
 - **Reddit:** *content, title, commentId, parentId, upVotes, date, link*;
 - **Twitter:** *link, date, text, retweet_count, favorite_count, reply_count, commentId, parentId*;

To ensure consistency and relevance, we performed the following modifications:

3. **Column Renaming:** We renamed the *text* and *review* columns to *content* and the approval-related columns to *approveCount* across all datasets, except for the **iTunes** dataset, which lacks an approval column;
4. **Column Removal:** We removed columns such as *userName, userImage, reviewCreatedVersion, replyContent, repliedAt, appVersion, isEdited, developerResponse* to maintain user anonymity and focus solely on user-generated content. We retained the *userName* column in the **iTunes** dataset for post-identification purposes since its posts did not have an ID like **Google Play**’s or a link, like **Reddit** and **Twitter**;
5. **Column Standardization:** The *at* column in the **Google Play** dataset was renamed to *date* for consistency and clarity. The *score* column was also renamed to *rating* to represent its content better. The columns *retweet_count, favorite_count, reply_count* from **Twitter** were not removed because they show each post’s engagement levels;

6. **Data Concatenation:** We concatenated the titles and feedback text into a single string for the **iTunes** and **Reddit** datasets, following a similar approach as Oehri and Guzman [51].

We ended up with the following columns for each dataset:

Source	commentId	date	content	approveCount	rating	userName	link	parentId	retweetCount	replyCount
Google Play	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No
iTunes	No	Yes	Yes	No	Yes	Yes	No	No	No	No
Reddit	Yes	Yes	Yes	Yes	No	No	Yes	Yes	No	No
Twitter	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes

Table 4.5: Summary Matrix of Dataset Columns.

Furthermore, we performed the following data-cleaning operations:

1. **Null and Duplicate Removal:** We removed null content, duplicate entries, and bot-generated content from all datasets;
2. **Keyword Filtering:** We filtered out rows containing keywords related to “giveaway” and removed rows with extra whitespace or containing GIF keywords;
3. **Content Filtering:** We removed markdown keywords and specific keywords representing deleted or removed content (e.g., [deleted], [removed]) from the **Reddit** dataset;
4. **Official Account Removal:** We excluded posts made by official streaming service accounts, particularly from the **Twitter** dataset, to focus solely on user-generated content;
5. **Connect Tweets with Their Replies:** We established connections within the dataset to analyze the relationship between original tweets and their corresponding replies.

To standardize text and prepare it for classification models (see more in next section), we performed the following text processing steps:

1. **Text Normalization:** We converted all text to lowercase and removed special characters;
2. **URL Replacement:** Following Guzman et al. [22], we replaced all URLs with the marker `_link_`;
3. **Word Expansion:** Abbreviated words such as “couldn’t” were expanded to their full forms (e.g., “could not”) using a list of common English contractions⁸, similar to the work done by McIlroy et al. [44];
4. **Minimum Word Requirement:** We ensured that all posts contained at least three words to retain meaningful content.

We wanted to create models with two different objectives: to predict if a post is relevant or not and to categorize posts into three distinct categories: *bug report*, *feature request*, and *other*.

⁸<https://www.yourdictionary.com/articles/contractions-correctly>

However, before proceeding with this task, we planned to extract more information from the posts to facilitate categorization.

Our data analysis observed numerous abbreviations commonly used in the posts. We replaced these abbreviations with their fully spelled-out meanings to improve readability and understanding. For example, “op” will be replaced by “original poster.” We took precautions to ensure that these replacements occur only where appropriate, avoiding instances where abbreviations are part of more significant words.

Furthermore, we identified and removed the replacement character that appeared frequently during our dataset analysis. The presence of this symbol could potentially introduce noise or inconsistencies in our models, so removing it aims to enhance the overall quality of our data.

Drawing inspiration from the works of Tizard et al. [69] and Maalej et al. [40], we embarked on identifying the verb tense of each post using the Python library `nltk`. Our approach involved tokenizing the sentences within the posts and utilizing a POS tagger to identify verbs and their respective tenses.

During this process, we encountered limitations with the POS tagger, particularly in recognizing composite verbs in the future tense and irregular verbs. To address this, we manually implemented additional conditions to account for these cases. Additionally, we acknowledged, similar to the findings of Maalej et al. [40], that a post may contain multiple verb tenses due to comprising multiple sentences. We calculated the ratio of three primary verb tenses to address this complexity: past, present, and future.

It is worth noting that implementing this process required a certain level of grammatical awareness to identify different verb tenses and irregularities. Furthermore, since the NLTK’s POS tagger is unavailable for all languages in the dataset, we needed to translate all posts into English using the Python library `googletrans`, which interfaces with the Google Translate API, before detecting their verb tenses.

In line with existing literature, we opted to conduct sentiment analysis to uncover potential variations in sentiment across different categories of posts. It is commonly observed that posts with positive sentiment polarities often express satisfaction or praise, while those with negative sentiment polarities may convey dissatisfaction, such as bug reports or complaints.

To perform sentiment analysis, we utilized the Python library `PySentiStr`, which implements the SentiStrength algorithm. Similar to our approach for verb tense detection, we acknowledged that each social media post could exhibit multiple sentiments. Therefore, we adopted a strategy to calculate the average sentiment of each post’s author. Before applying this algorithm, we systematically removed negations from the posts to address the potential misclassifications resulting from negated terms, taking inspiration from Villarroel et al.’s work [70]. This was achieved by creating a comprehensive list of negations, ensuring that negated terms did not skew the sentiment analysis or classification process.

Social media posts often contain typos, which can impact the accuracy and clarity of the text. To address this issue, we developed a function to correct typos using the Python library `SpellChecker`. However, it is essential to note that this library may not support all languages in

our dataset. Therefore, to ensure effective typo correction, we verify that the posts are in English before applying the `SpellChecker` function.

It is crucial to recognize that while this function can significantly improve text accuracy, it is not flawless. Many typos are context-dependent, meaning they may not be detected or corrected accurately due to the complexity of language and meaning. For example, consider the sentence “You might loose it.” While all the words are correctly spelled, the intended meaning is likely, “You might lose it.” This discrepancy underscores the limitations of automated typo correction tools in capturing contextually-dependent errors.

Following these preprocessing steps, we obtained refined datasets that were ready for further analysis. The tables below depict the number of posts extracted for each streaming service from each social media platform before and after preprocessing.

	iTunes	Google Play	Twitter	Reddit	Total
ACL Player	NA	NA	68	428	496
Gas Digital Network	50	NA	63	99	212
Itaú Cultural Play	5	452	298	4	759
RTP Play	15	25	983	6154	7177
Netflix	1013	2999	2222	6636	12870
(HBO) Max	302	760	1838	5432	8332
Hulu	1004	2463	1845	3032	8344
Prime Video	1008	3000	1492	4440	9940
Peacock	1008	2385	1788	3343	8524
Disney Plus	1008	2146	1109	5251	9514
Total	5413	14230	11706	34819	66168

Table 4.6: Number of Posts After Preprocessing.

4.3 Training and Evaluating Classification Models

4.3.1 Predicting Relevance

With the data extraction and analysis phase completed, we progressed to the next stage of our project: training and testing classification models. The primary objective was to employ supervised learning techniques to predict the relevance of posts, with “relevant” as the target variable. Leveraging the statistical sampling conducted earlier, we used the manually labeled random sampling dataset to evaluate the models’ performance.

We explored multiple variations of models to assess their effectiveness with our datasets comprehensively. The algorithms employed included DT, RF, SVM, NB, and Multinomial NB. We chose these algorithms based on our experience with them and their popularity in the literature. We experimented with techniques such as stopwords removal, stemming, and lemmatization in preprocessing. All model variations involved the tokenization of the dataset.

We employed the `punkt` tokenizer from the Python library `nltk` for tokenization.

Due to its prevalence in the literature, we implemented the Snowball Stemmer algorithm using the `nltk` library.

Additionally, we utilized WordNet for word lemmatization, leveraging the functionality provided by the `nltk` library.

In our standard dataset, we retained stopwords, motivated by arguments suggesting their potential significance, such as words like “will” and “would” indicating desires for new features, which could aid classification efforts. However, recognizing the prevalent practice in NLP projects, we also compared model performance with and without stopwords. We obtained a list of stopwords using the `nltk` library for identification and removal.

We employed various techniques to enhance our models’ effectiveness. Initially, we utilized `TfidfVectorizer` and `CountVectorizer` from the `sklearn` Python library. Additionally, we incorporated the length of the posts as a classifying feature, drawing inspiration from Maalej’s findings [40, 42, 62].

Furthermore, we explored word and sentence embedding techniques, including `Word2Vec`, `FastText`, `SBERT`, and `USE`. For `Word2Vec` and `FastText`, we relied on the `gensim` Python library, while `SBERT` was implemented using `sentence_transformers` and `USE` with `tensorflow_hub`.

In addition to these techniques, we experimented with N-gram extraction, utilizing `n=2` and `n=3`, with the assistance of the `nltk` Python library. We also combined N-grams with `CountVectorizer` and `TfidfVectorizer` to further enhance feature representation.

We evaluated the model’s performance under different scenarios to address data imbalance. Specifically, we tested the following configurations:

- Normal testing without any resampling techniques;
- Testing with `SMOTE` applied;
- Testing with `SMOTE` and `GridSearchCV` for hyperparameter tuning;
- Testing with `SMOTE` and `RandomSearch` for hyperparameter tuning.

Each configuration assessed the model’s performance under varying data balance and parameter optimization conditions.

We assessed the models’ performance using standard data science metrics, including precision, recall, F1-measure, accuracy, and AUC. To mitigate bias and ensure robust evaluation, we adopted the holdout method, splitting the dataset into 80% for training and 20% for testing. Additionally, we employed cross-validation techniques, utilizing both `GridSearchCV` and `RandomSearch`, to optimize model hyperparameters and validate performance across different subsets of the data.

The original scraped data, detailed variations and corresponding results can be found in [GitHub](#). Interestingly, excluding stopwords yielded results comparable to when stopwords were included. `GridSearchCV` and `RandomSearch` produced similar outcomes, leading us to discontinue testing these variations (removal of stopwords and `RandomSearch`) after analyzing a few datasets.

However, it is essential to note some limitations. For instance, we found that Multinomial NB does not perform well with word and sentence embeddings, as it may result in classes having negative values, which the model cannot appropriately handle. This is why this model is absent in some tables and marked as “NA” in others.

We selected the best model for each social media platform by comprehensively analyzing the values of F1-measure, accuracy, and AUC. Based on these metrics, we identified the model and technique combination that produced the best results. It is essential to acknowledge that our results might be subject to biases, such as overfitting, and that real-world performance could be lower than the training results due to these factors. Table 4.7 showcases the models and techniques chosen to predict the relevance of posts on each social media platform.

Social Media	Technique	P	R	F	A	AUC
Google Play	RF + CV ⁹ + SMOTE	0.90	0.90	0.90	0.86	0.90
iTunes	SVM + CV + SMOTE	0.80	1.00	0.89	0.91	0.96
Reddit	NB + USE	0.65	0.57	0.60	0.76	0.81
Twitter	NB + CV + SMOTE	0.60	0.67	0.63	0.78	0.81

Table 4.7: Best Models and Techniques for Each Social Media Platform.

We explain the evaluation metrics in the (Appendix B.1.4), but here is a more focused explanation:

- Precision indicates the proportion of relevant posts among those classified as relevant by the model;
- Recall measures the model’s ability to identify all relevant posts;
- Accuracy reflects the ratio of correctly predicted instances (both relevant and non-relevant) to the total instances;
- AUC assesses the model’s capability to distinguish between relevant and non-relevant posts across various thresholds.

Analyzing the results model by model, we observe that **Google Play**’s model achieves balanced precision and recall with high AUC, indicating reliable performance in identifying relevant posts with minimal false predictions. Conversely, **iTunes**’s model excels in recall with slightly lower precision, capturing nearly all relevant posts but with a few false positives (80% precision rate). **Reddit**’s and **Twitter**’s models exhibit similar performances, showing moderate precision and recall alongside good AUC. However, they may miss some relevant posts while including some false positives.

For convenience, we investigated whether a single model and technique could work across all social media platforms. Upon analyzing the data, we observed that three social media platforms utilized the same technique (CountVectorizer) coupled with SMOTE. However, we found

⁹CountVectorizer

no single model that performed well across all platforms using this technique. Table 4.8 shows the evaluation metrics for different models and techniques across various social media platforms, confirming our findings. We highlighted in bold the best values for each column.

	Google					iTunes					Twitter					Reddit				
	P	R	F	A	U ¹⁰	P	R	F	A	U	P	R	F	A	U	P	R	F	A	U
DT	.74	.86	.79	.7	.61	.43	.6	.5	.45	.47	.2	.33	.25	.5	.44	.38	.48	.42	.57	.55
RF	.9	.9	.9	.86	.9	.6	.38	.46	.59	.68	.4	.44	.42	.69	.65	.39	.52	.44	.57	.55
SVM	.74	.86	.79	.69	.31	.8	1	.89	.91	.96	0	0	0	.75	.57	.29	.22	.25	.57	.44
NB	.67	1	.81	.67	.5	.5	.6	.55	.55	.55	.6	.67	.63	.78	.81	.45	.39	.42	.64	.58
MNB	.75	.86	.8	.69	.77	.71	.62	.67	.71	.66	.33	.17	.22	.71	.53	.49	.74	.59	.66	.76

Table 4.8: Evaluation Metrics for Different Algorithms Across All Social Media Platforms.

Additionally, the Table 4.9 summarizes the number and percentage of relevant posts predicted by our models from our complete dataset.

	Total Posts	Number of Relevant Posts	Percentage of Relevant Posts
iTunes	5413	2591	47.9%
Google Play	14230	8623	60.6%
Reddit	34819	8745	25.1%
Twitter	11706	2888	24.7%
Total	66168	22847	-

Table 4.9: Relevant Posts Overview.

We can estimate the overall precision and recall of our dataset predicted as “relevant” using the values in Table 4.7.

Firstly, we calculate the TP because precision and recall are directly computed based on it (Appendix B.1.4).

TP Calculation:

• iTunes:

$$TP_{iTunes} = \text{Total Predicted Relevant}_{iTunes}^{11} \times \text{Precision}_{iTunes}$$

Given:

$$\text{Total Predicted Relevant}_{iTunes} = 2591 \quad \text{and} \quad \text{Precision}_{iTunes} = 0.80$$

$$TP_{iTunes} = 2591 \times 0.80 = 2073$$

• Google Play:

$$TP_{Google} = 8623 \times 0.90 = 7761$$

• Reddit:

$$TP_{Reddit} = 8745 \times 0.65 = 5684$$

¹⁰AUC

- **Twitter:**

$$TP_{\text{Twitter}} = 2888 \times 0.60 = 1733$$

To calculate recall values, besides TP, we need the value of Actual Relevant posts (TP + FN):

- **iTunes:**

$$\text{Actual Relevant}_{\text{iTunes}} = \frac{2073}{1.00} = 2073$$

- **Google Play:**

$$\text{Actual Relevant}_{\text{Google}} = \frac{7761}{0.90} = 8623$$

- **Reddit:**

$$\text{Actual Relevant}_{\text{Reddit}} = \frac{5684}{0.57} = 9972$$

- **Twitter:**

$$\text{Actual Relevant}_{\text{Twitter}} = \frac{1733}{0.67} = 2587$$

Combining these estimates, we calculate the overall precision and recall:

$$\text{Total True Positives} = 2073 + 7761 + 5684 + 1733 = 17251$$

$$\text{Total Predicted Relevant} = 2591 + 8623 + 8745 + 2888 = 22847$$

$$\text{Total Actual Relevant} = 2073 + 8623 + 9972 + 2587 = 23255$$

Therefore, the overall precision and recall are approximately:

$$\text{Overall Precision} = \frac{17251}{22847} \approx 0.755(75.5\%)$$

$$\text{Overall Recall} = \frac{17251}{23255} \approx 0.742(74.2\%)$$

Overall Precision represents the proportion of correctly predicted relevant posts among all posts predicted as relevant. Overall Recall measures the proportion of correctly predicted relevant posts among all actual relevant posts in the dataset.

4.3.2 Categorizing Posts

We meticulously labeled our random sampling datasets before training and testing our models. However, we encountered a challenge: our dataset did not neatly fit into the three categories we initially envisioned — *bug report*, *feature request*, and *other*.

While most irrelevant posts were easily categorized as *other*, the relevant ones posed a more nuanced classification dilemma. Many were appropriately labeled as *bug report*, but a subset that defied easy classification remained. These posts ranged from users expressing dissatisfaction

¹¹Total Predicted Relevant = TP + FP

with streaming service behavior without specific bug reports to those offering praise or seeking information or assistance from the community, particularly on **Reddit**.

Inspired by insights from the literature, we expanded our classification scheme to include seven classes: *complaint*, *praise*, *information giving*, *information seeking*, and *user experience*. However, we were wary of the potential impact of increasing the number of classes on model performance.

We explored various approaches to address the multiclass nature of the problem. Initially, we evaluated all classes simultaneously, but the results were unsatisfactory. Subsequently, we experimented with binary classification for each class, treating other classes as simply “not that class.” This approach also yielded poor performance.

In response, we streamlined our classification scheme by merging *praise* and *complaint* into a single class labeled *user experience* and combining *information giving* and *information seeking* into *information*. Although this consolidation led to some improvement, the gains were marginal. Ultimately, we opted to remove the “information” class altogether, resulting in a slight enhancement in performance. Additionally, we removed the *other* class by only categorizing posts already established as relevant by the models created in the previous section. Of course, this introduced some uncertainties since the models predicting the relevance of posts are not perfect.

Our main issue was that the dataset was deeply imbalanced, with a disproportionate number of posts in the *bug report* category compared to the other two categories. The smaller the dataset, the worse the issue was; for example, the randomly sampled **iTunes** dataset only had four posts that were a *feature request* and eight that were a *user experience*; the rest were a *bug report*. There are ways to deal with dataset imbalance, such as using SMOTE, but this did not work due to the small number of posts in each category. This did not work; it only worked for the dataset from **Google Play** and was still unsuccessful. We tried to manufacture a balanced dataset (an equal number of posts for each category) and had good results. However, we did not pursue this strategy because we recognized those results were unrealistic since the dataset could not appear balanced in a real-life scenario.

We also tried oversampling using `RandomOverSampler`¹². However, the results were still not favorable. Once again, we theorize that this is due to the overwhelming amount of *bug report* posts. We would need a more balanced dataset to get good results for this model. To achieve that, we would have to adjust our data extraction strategy by using fewer keywords associated with *bug report* or more keywords associated with *feature request*. However, our current strategy uses an equal number of keywords for each category. Moreover, keywords associated with *feature request* often involve verbs, which could introduce additional complexity in data extraction. We believe there are more posts from the *bug report* category because people primarily use the internet to present their complaints. Furthermore, many posts in this category also include feature requests, as users often complain about an issue and suggest improvements. These suggestions range from fixing a broken feature to creating a new one.

¹²https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.RandomOverSampler.html

4.4 Summarization

Once we assembled the dataset with the predicted values, we removed the posts that were classified as irrelevant. Our goal in summarizing posts is to provide users with a more straightforward visualization of relevant posts on our platform. Since we will only show relevant posts, there is no use in wasting resources summarizing irrelevant ones.

We found out that there are two types of summarization algorithms. They are either **abstractive** or **extractive**. **Abstractive** algorithms summarize text by generating new sentences from the extracted information, while **extractive** algorithms form a summary by taking small parts of the text.

We tested four **extractive** summarization algorithms: *TextRank*, *LexRank*, *SumBasic* and *SPACY* (the first three were mentioned in the literature), and six **abstractive** summarization algorithms: *BART*¹³, *T5 model*¹⁴, *GPT2*¹⁵, *XL-Net*¹⁶, *Pegasus*¹⁷ and *Txtai*¹⁸.

4.4.1 Evaluation

To evaluate and compare the performances of all these algorithms, we followed Rahman et al.'s example [57] and used ROUGE metrics, specifically, **ROUGE-N**, ROUGE-L, and ROUGE-Lsum. ROUGE checks if an algorithm's short summaries are similar to what a person would write. To make this comparison, we manually summarized a series of posts, submitted them to each algorithm, and then compared the machine-generated summaries to the human-made ones. The L in ROUGE-L stands for the Longest Common Subsequence (LCS), which is a method that identifies the longest sequence of words that appear in the same order in both the reference and machine-generated summaries. The ROUGE-Lsum is related to the ROUGE-L metric but applies a slightly different calculation method. It applies the ROUGE-L calculation method at the sentence level and then aggregates all the results for the final score¹⁹.

The **Table 4.10** showcases our results, with each ROUGE metric divided into three familiar metrics: precision, recall, and f-measure.

¹³<https://huggingface.co/facebook/bart-large-cnn>

¹⁴https://huggingface.co/docs/transformers/model_doc/t5

¹⁵<https://huggingface.co/openai-community/gpt2>

¹⁶<https://huggingface.co/xlnet/xlnet-base-cased>

¹⁷https://huggingface.co/docs/transformers/v4.41.3/en/model_doc/pegasus#overview

¹⁸<https://neuml.github.io/txtai/pipeline/text/summary/>

¹⁹<https://github.com/google-research/google-research/tree/master/rouge>

	ROUGE-1			ROUGE-2			ROUGE-L			ROUGE-Lsum		
	P	R	F ²⁰	P	R	F	P	R	F	P	R	F
Spacy	0.24	0.40	0.28	0.14	0.22	0.16	0.21	0.35	0.24	0.21	0.35	0.24
Pegasus	0.26	0.34	0.28	0.13	0.18	0.14	0.24	0.30	0.25	0.24	0.30	0.25
XLNet	0.37	0.37	0.35	0.20	0.21	0.19	0.35	0.34	0.32	0.35	0.34	0.32
GPT2	0.37	0.37	0.35	0.20	0.21	0.19	0.35	0.34	0.32	0.35	0.34	0.32
Txtai	0.40	0.42	0.38	0.23	0.23	0.22	0.36	0.37	0.34	0.36	0.37	0.34
SumBasic	0.32	0.59	0.39	0.16	0.32	0.21	0.27	0.51	0.34	0.27	0.51	0.34
TextRank	0.31	0.66	0.40	0.18	0.38	0.23	0.27	0.57	0.35	0.27	0.57	0.35
LexRank	0.34	0.67	0.43	0.20	0.39	0.26	0.30	0.58	0.38	0.30	0.58	0.38
BART	0.37	0.59	0.44	0.22	0.35	0.26	0.33	0.53	0.39	0.33	0.53	0.39
T5	0.39	0.58	0.45	0.24	0.35	0.27	0.35	0.51	0.40	0.35	0.51	0.40
GPT-3.5 Turbo	0.81	0.84	0.81	0.76	0.80	0.77	0.80	0.83	0.80	0.80	0.83	0.80

Table 4.10: Summarization Algorithms Evaluation.

Let us focus on the f-measure values since this metric is a harmonic blend of precision and recall. The values were low, with 0.45 being the highest f-measure value, achieved only for ROUGE-1. Based on these results, *T5* was the best algorithm, but the values are not very strong.

We tried several methods to improve the results. We experimented with different tokenizers and models for the **abstractive** summarization algorithms, but they all produced similar results. We also attempted fine-tuning algorithms using `Trainer` from `Transformers`, which did not improve the results either.

We decided to try another approach, Large-Language Models (LLMs), specifically ChatGPT. We used the ChatGPT API with the model **gpt-3.5-turbo-0125** and the following prompt (as one can guess, the variable `text` is the text to summarize):

```

1 prompt = f"Concisely summarize this feedback, capturing its main idea: '{text}'
    Maintain the original language style; avoid third-person. Keep it under 20
    words"

```

We observed that the results were nearly twice as good as the best results of the previous algorithms. Based on this, we decided to continue with this approach. One concern with this method, and the reason we did not immediately adopt it, was the cost associated with the ChatGPT API, whereas the other algorithms are free and open-source. The API cost depends on the number of input and output tokens, with the price per output token being higher than the price per input token.

In our prompt, we instructed ChatGPT to keep the output under 20 words, which helps control the output size and keep costs manageable. The input size varies depending on the length of the text to be summarized, as our prompt size remains constant. On average, we spent 8 to 10 cents per 1000 posts, which seems affordable.

²⁰F-measure

4.5 Grouping

We aimed to employ clustering or topic modeling techniques to reveal critical themes and topics within the user feedback and to group similar posts to improve visualization.

First, we created word clouds for each dataset to visualize the most frequently used words in the posts. Prior to this, we removed stopwords. Some of the prominent words included: “chrome-cast,” “ad,” “glitchy,” “subtitle,” “brightness,” “mirroring,” “login,” “audio,” among others.

Initially, we explored topic modeling using LDA, as suggested in the literature, to group similar posts based on underlying topics. However, this method did not produce the desired results. The primary goal was to cluster posts to combine similar content, but LDA failed to achieve this effectively. The generated topics were too broad and did not align well with our need for more cohesive and specific groupings of posts.

Following the unsatisfactory results with LDA, we turned to clustering methods, starting with DBSCAN. The literature recommended this method for its ability to form clusters based on density, which we hypothesized would better group similar posts. We used TF-IDF and Count Vectorization techniques to transform the text data into a format suitable for clustering. Unfortunately, DBSCAN also did not yield optimal results. It often paired posts with little in common, resulting in clusters lacking meaningful coherence.

Despite these challenges, DBSCAN showed some promise as it came closer to what we aimed to achieve than LDA. This encouraged us to continue exploring clustering methods.

Next, we experimented with K-Means clustering²¹, incorporating Principal Component Analysis (PCA)²² to reduce dimensionality and improve cluster formation. Additionally, we explored the use of Jaccard similarity²³ to measure the similarity between sets of posts. While these methods offered some improvements, they did not fully meet our expectations regarding producing highly coherent clusters.

Inspired by ROUGE, we implemented two custom clustering algorithms to group posts based on sentence similarity²⁴. These algorithms aimed to create clusters that better captured the nuanced meanings of posts, ensuring that similar content was grouped effectively.

The first algorithm focuses on creating initial clusters by iterating through each post and assigning it to the most similar existing cluster. If no suitable cluster is found, a new cluster is created. The steps are as follows:

- **Initialization:** A dictionary is initialized to store the clusters, and a unique cluster ID is assigned to each new cluster;

²¹<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>

²²<https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>

²³https://scikit-learn.org/stable/modules/generated/sklearn.metrics.jaccard_score.html

²⁴https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html

- **Similarity Calculation:** For each post, the algorithm calculates its similarity to existing clusters using cosine similarity on sentence embeddings;
- **Similarity Calculation:** For each post, the algorithm calculates its similarity to existing clusters using cosine similarity on sentence embeddings. Specifically, the algorithm computes embeddings for the new post and each post already in the cluster. It then calculates the cosine similarity between the embedding of the new post and the embeddings of each post in the cluster;
- **Similarity Aggregation:** The algorithm aggregates these similarities by taking the maximum similarity value among all the calculated similarities. This means that the similarity of a post to a cluster is represented by the highest similarity score between the post and any post within that cluster;
- **Cluster Assignment:** If the maximum similarity to an existing cluster exceeds a specified threshold (0.70), the post is added to that cluster. A new cluster is created for the post if no suitable cluster is found.

This process ensures that each post is assigned to the most semantically similar cluster, allowing for the formation of initial groupings that reflect the content's underlying themes.

The second algorithm iteratively refines cluster assignments to enhance the initial clustering quality, aiming to resolve any suboptimal groupings identified during the initial phase:

- **Iterative Reassignment:** This phase involves iterating over each cluster and every post within it. The algorithm evaluates whether a post could belong to a different cluster that better aligns with its semantic content;
- **Simultaneous Similarity Assessment:** During each iteration, the algorithm computes the similarity between the post and all other clusters, excluding its current cluster. This comparison seeks to identify the cluster where the post could potentially contribute more effectively;
- **Selective Reassignment:** If the algorithm identifies a cluster with a sufficiently high similarity threshold (above 0.50), the post is reassigned to that cluster. This process ensures that posts are grouped into clusters that more accurately capture their semantic relevance.

Addressing concerns about cascading effects, the algorithm employs several safeguards to maintain stability and convergence:

- **Threshold-driven Reassignment:** Posts are reassigned based on substantial similarities, preventing minor matches from triggering unnecessary reassignments;
- **Finite Iteration Scope:** The iterative process operates within a finite number of clusters and posts. As iterations progress, clusters become increasingly cohesive, minimizing the need for further adjustments;

- **Sequential Modification:** By working with a copy of current cluster reviews and updating clusters only after potential reassignments are evaluated, the algorithm avoids immediate cascading effects within a single iteration. This methodical approach promotes stability and fosters more coherent cluster formations over successive iterations.

These measures collectively ensure that the algorithm effectively refines initial cluster assignments without perpetually reassigning posts, thereby enhancing the overall clustering quality based on semantic similarities.

Both algorithms rely on sentence embeddings to represent the posts. These embeddings capture the semantic content of the posts, allowing the algorithms to measure similarity more accurately than simple keyword matching. The cosine similarity measure evaluates how closely related the posts are, guiding the clustering and reassignment processes.

To further refine our clustering, we experimented with clustering the complete posts, human-generated summaries, and machine-generated summaries. We found that clustering using the summaries yielded the best results. Summarizing posts before clustering helped reduce noise and focused on the most relevant information, making it easier to form meaningful clusters. We ended up with 1184 clusters for our entire dataset of 22847 posts.

To enhance visualization, we realized that having some sort of title was necessary to help users identify each cluster and see if they were interested in the posts contained within it. Initially, we followed the literature and tried to make titles using the five most common words in the posts within a cluster. However, the titles ended up slightly off because there was no connection between the words. We used the ChatGPT API to generate more coherent and meaningful titles. We crafted a prompt to summarize all post summaries within each cluster to form a title:

```
1 prompt = f"Suggest a suitable title for this cluster of user complaints or  
feedback:{text}"
```

We spent 0.52€ making titles for all 1184 clusters, which seems affordable.

4.6 Conclusion

In conclusion, we have developed a robust social media feedback processing workflow inspired by insights from existing literature. While each phase of our workflow has been successfully applied to datasets within the specific domain of streaming services, all processes were encapsulated in separate Python scripts and Jupyter Notebooks. These processes can all be seamlessly executed using a single shell script, as detailed in [Appendix F](#).

This modular approach ensures versatility, enabling its application to datasets across diverse domains. Moving forward, this adaptable framework is poised to support comprehensive analysis and insights generation from social media feedback, facilitating informed decision-making and strategic planning across various domains.

Chapter 5

Platform Development

This chapter details the development of the visualization platform, entitled “Watching the Watchers.”

5.1 User Stories

Before beginning the platform’s development, our first task was to extract the requirements with the help of both dissertation supervisors and the company. These requirements, presented in a user story format, can be found in [Appendix C](#).

5.2 Architecture

Our platform aims to provide users easy access to valuable data extracted and processed according to the workflow described in the [previous chapter](#). To achieve this goal, we adopted a web-based application approach consisting of three main components: a database for storing processed data, a frontend client application for user interaction, and a backend server application to facilitate communication between the database and client.

We deliberately chose familiar technologies to develop this web application because our primary focus was delivering the best possible product. Our priority was to optimize development time and resources, and we recognized that leveraging technologies we were already proficient in would allow us to achieve this goal more efficiently. By selecting familiar tools, we minimized the need for additional learning curves, enabling us to dedicate more time and energy to actual development tasks rather than onboarding with new technologies. This strategic approach ensured that our efforts were squarely focused on enhancing the platform’s functionality and user experience without being encumbered by the complexities of mastering unfamiliar technologies.

For the frontend development, we leveraged Vue.js¹, a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS, and JavaScript. It provides a declarative

¹<https://vuejs.org/guide/introduction.html>

and component-based programming model that helps develop user interfaces, whether simple or complex.

On the backend side, we opted for FastAPI², a modern and high-performance web framework for creating APIs with Python 3.7+. Leveraging standard Python-type hints facilitates the creation of endpoints accessible to the frontend. Additionally, FastAPI's compatibility with SQLAlchemy³ allowed us to integrate PostgreSQL as our chosen database seamlessly. The Database Structure is available in [Appendix D](#).

5.3 Features and User Interface

This section presents a storyboard walkthrough of the application, providing a comprehensive overview of its functionality, features, and user flow. The storyboard format guides readers through the application step-by-step, detailing each page's purpose, layout, and interactions. This presentation approach aims to clearly understand the application's design and functionality, facilitating an in-depth user experience exploration. This narrative journey through the application aims to explore its various pages and functionalities, offering insights into its seamless user flow and intuitive design.

5.3.1 Landing Page

The application begins at the **Landing** Page, where users are greeted by its logo. The logo features two people watching TV, with an eye inside the screen that moves as if watching them, a playful nod to the app's name. To proceed to the next page, the user needs to click on the area inside the TV screen ([Figure 5.1](#)).

²<https://fastapi.tiangolo.com/>

³<https://fastapi.tiangolo.com/tutorial/sql-databases/>

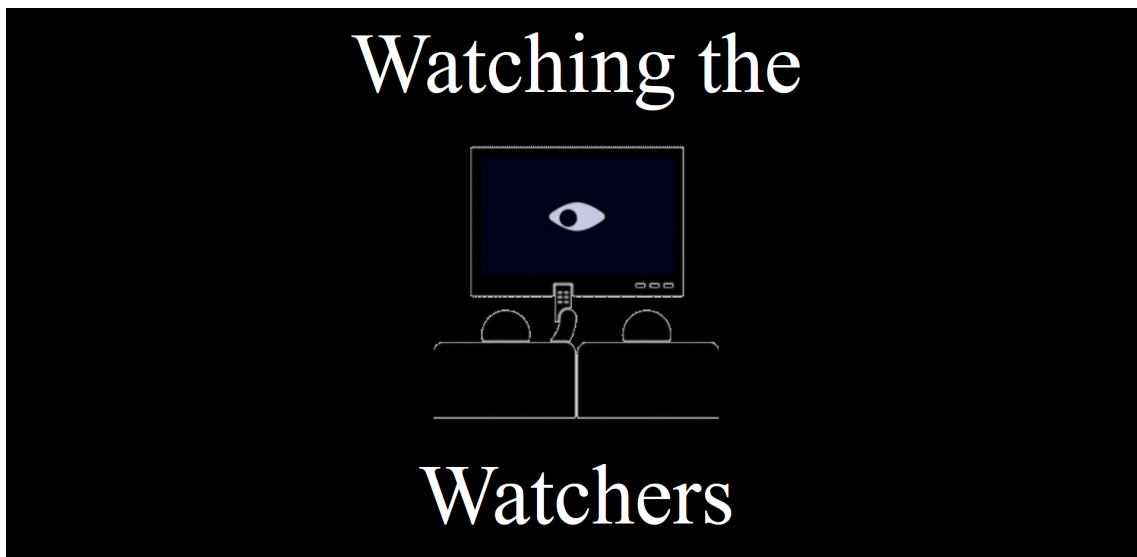
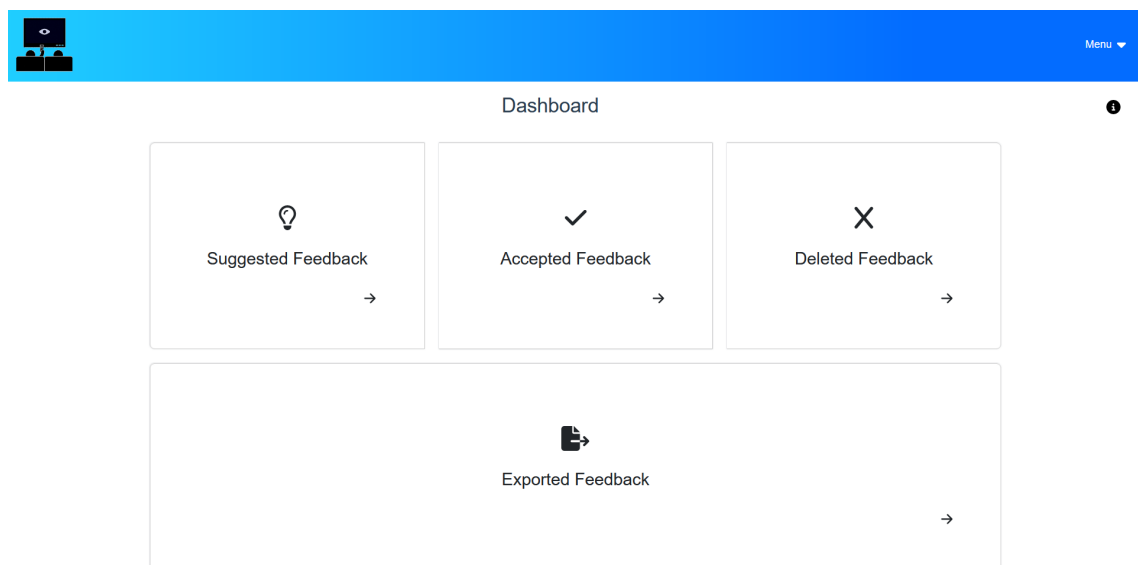


Figure 5.1: Landing Page.

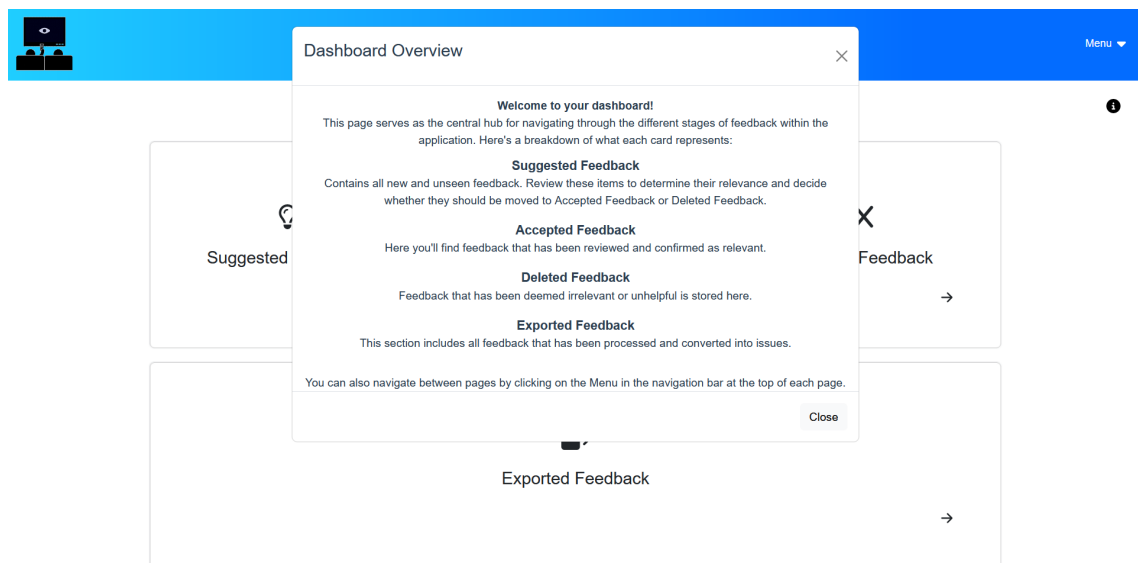
5.3.2 Dashboard Page

The user will arrive at the **Dashboard Page** (Figure 5.2), which serves as the central hub for navigating through the different stages of feedback within the application (US1). Each card on this page represents a different category (US9):

- **Suggested Feedback:** Contains all new and unseen feedback. Users can review these posts to determine their relevance and decide whether they should be moved to **Accepted Feedback** or **Deleted Feedback**;
- **Accepted Feedback:** Posts reviewed and confirmed as relevant. Users can create issues from these posts or reclassify them as irrelevant, moving them to **Deleted Feedback**;
- **Deleted Feedback:** Posts reviewed and deemed irrelevant. Users can reconsider their decisions, classify these posts as relevant, and move them to **Accepted Feedback**;
- **Exported Feedback:** Includes all posts processed and converted into issues.

Figure 5.2: **Dashboard.**

Users can access this information by clicking on the “i” button in the upper right corner of the page. Every page includes this button, which reveals a modal with instructions to help users navigate the page (Figure 5.3).

Figure 5.3: Instructions for **Dashboard.**

The **Dashboard** is not the only way users can navigate between pages. They can also use the *Menu* in the navigation bar at the top of each page (Figure 5.4). The current page will be highlighted in green.

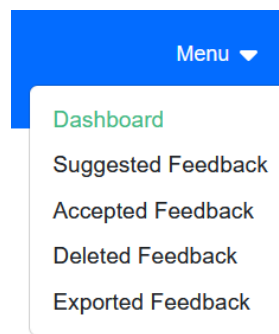


Figure 5.4: Menu.

5.3.3 Suggested Feedback Page

Upon arriving at the **Suggested Feedback** Page (Figure 5.5), the user is greeted with unseen feedback presented in two different viewing options: *View All*, which displays all posts together, and *Clustered View*, which groups the posts into clusters.

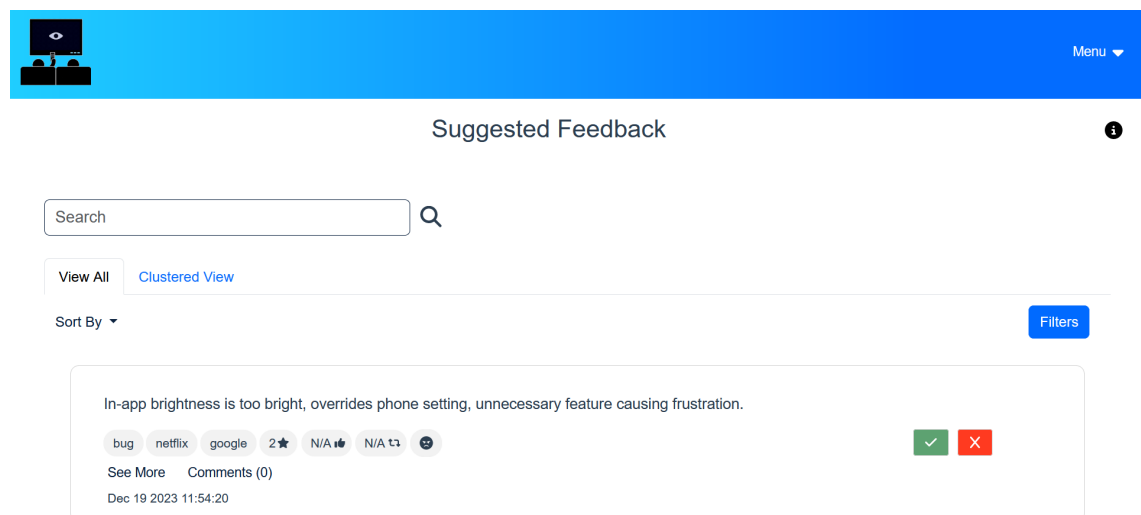


Figure 5.5: Suggested Feedback Page.

Once again, the user can see the instructions for the page if any doubt arises (Figure 5.6).

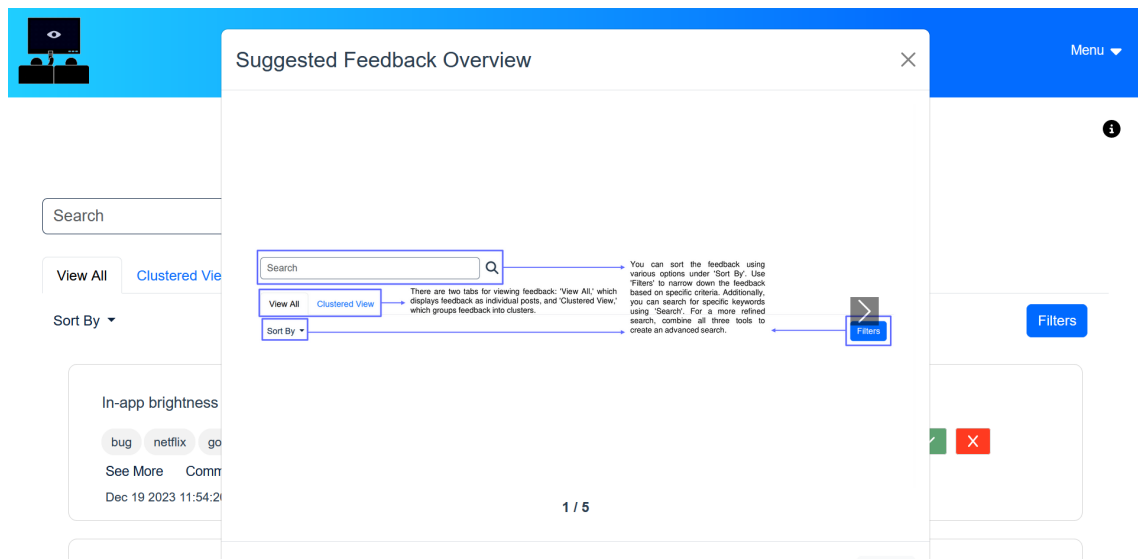


Figure 5.6: Instructions for **Suggested Feedback** Page.

Every post is showcased in a card containing detailed information about the post, as shown in [Figure 5.7](#). The first thing the user sees is the post’s summary. Below the summary, additional information is displayed using a tag-like system inspired by GitHub, including:

1. **Category:** A post can be classified into one of three categories: *bug*, *feature*, or *user experience*. In this example, the category is *bug*. Currently, due to the lackluster results from the models created to predict categories ([Chapter 4.3.2](#)), every post is categorized as *bug* ([US2.6](#));
2. **Streaming Service:** The streaming service related to the post. This information is derived from the search terms used to extract the data. In this example, the streaming service is *Disney +*. To maintain brevity, we shortened the names of most services for display purposes ([US2.8](#));
3. **Social Media:** The social media platform from which the post was extracted. In this example, it is **Twitter** ([US2.7](#));
4. **Rating:** **Google Play** and **iTunes** allow users to rate apps using a star system, and we display this rating in our application. For posts not extracted from these platforms, we display *N/A* (nonapplicable) ([US2.3](#));
5. **Likes:** The number of likes a post received on its original platform. For posts from **Reddit**, this value is derived from the upvote/downvote system. In this example, the number of likes is 0 ([US2.9](#));
6. **Reposts:** **Twitter** allows users to republish posts, and we display that number here. We display *N/A* for posts not extracted from **Twitter** ([US2.9](#));

7. **Sentiment:** Sentiment analysis was conducted for each post during the Preprocessing phase. The sentiment score could be positive or negative. Instead of displaying a numeric score, we use emojis to represent the sentiment: a *smiley face* for scores above 0.50, an *angry face* for scores below -0.50, and a *neutral face* for scores between -0.50 and 0.50. In this case, the sentiment is represented by an *angry face* (US2.5);
8. **Full Text:** Users can view the full text of a post by clicking “See More,” which opens a modal displaying the complete text (Figure 5.9) (US2.2);
9. **Link:** For posts extracted from **Twitter** or **Reddit**, users can view the original post by clicking “Original,” which opens the post in a new tab. Note that the post might have been changed or deleted since it was fetched, and the application includes a tooltip warning to manage expectations. Posts from **Google Play** or **iTunes** do not have links, as these platforms do not support sharing reviews (US2.4);
10. **Comments:** Comments made on the post can be viewed by clicking the “Comments” button, which opens a modal displaying the comments, as seen in Figure 5.8 (US2.10);
11. **Date:** Each post includes a timestamp and all dates are formatted consistently (US2.11).

Most of this additional information, such as **Rating**, **Likes**, **Reposts**, **Sentiment**, and **Comments**, contributes to helping users identify which posts are more important or relevant among all the relevant posts.



Figure 5.7: Post.



Figure 5.8: View Post's Comments.

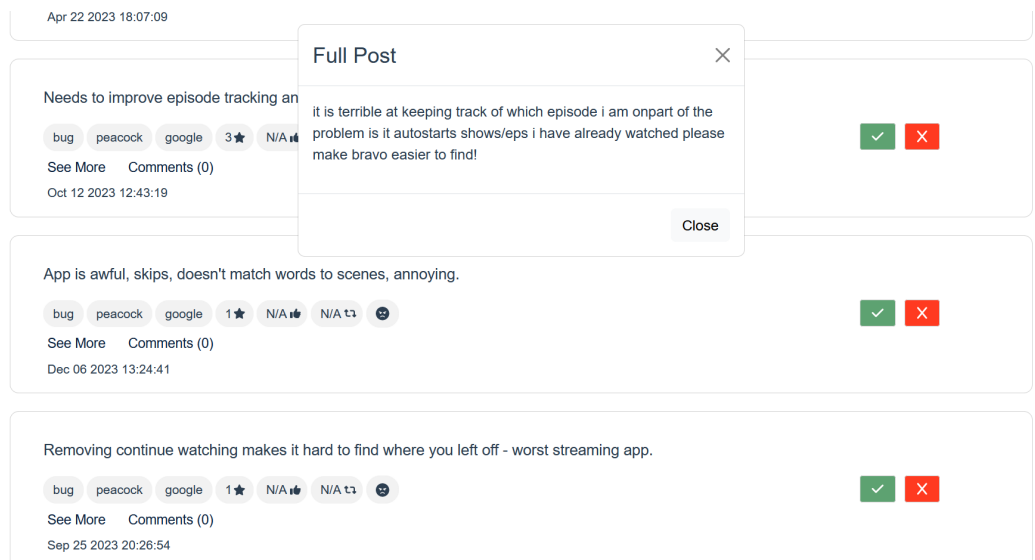


Figure 5.9: View Post's Full Text.

Besides viewing post information, users can classify each post as relevant or not by interacting with the two buttons at the right edge of the post (Figure 5.7) (US7). This feature was inspired by Stanik and Maalej's OpenReq [63]. The green check button is used to mark a post as relevant. Clicking this button opens a modal that asks the user to confirm the category (Figure 5.10) (US8), a feature inspired by Villarroel et al.'s CLAP [70]. This is a setup for a future feature aimed at collecting more data for our classification models to implement reinforcement learning. We can train our models with new data by obtaining user input to label more data, ensuring that future

posts are classified more accurately. The post is moved to **Accepted Feedback**, and the user gets a warning through a toast of this (Figure 5.11).

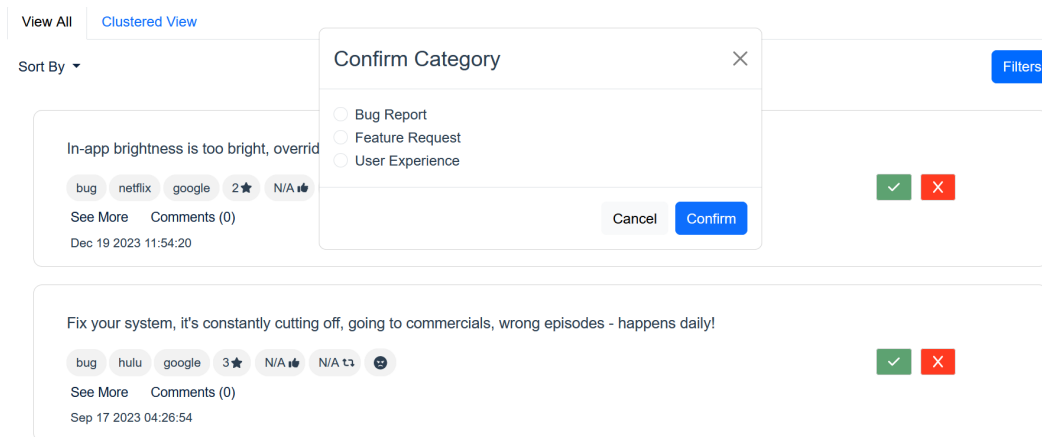


Figure 5.10: Confirm Category Modal.

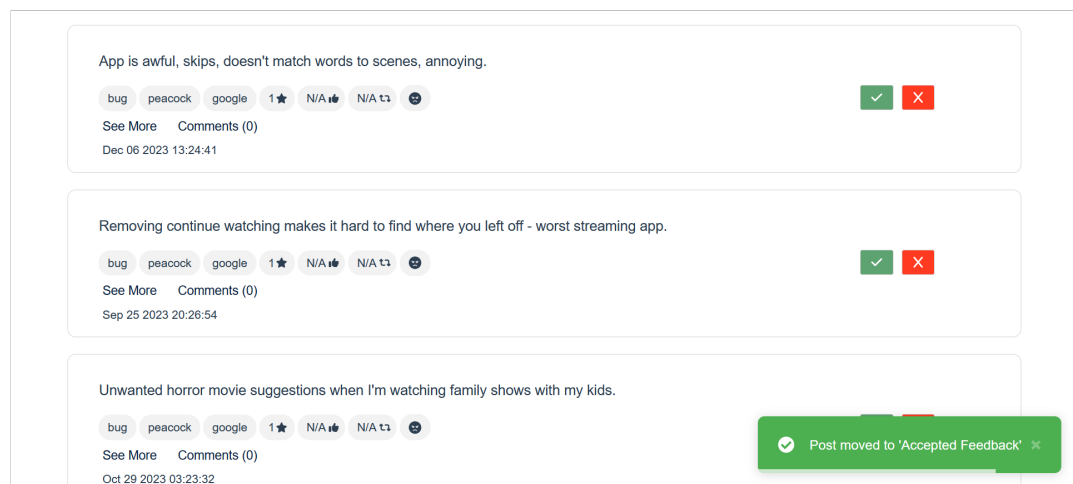


Figure 5.11: Accepted Feedback Toast.

The red cross button is used to mark a post as irrelevant. Clicking this button opens a modal that asks the user to explain why they consider the post irrelevant (Figure 5.12). This feedback is crucial for gathering more labeled data for the models, particularly the one classifying the relevance of posts. We offer users three different reasons for marking a post as irrelevant. The first two reasons indicate that the post itself is relevant. However, the user considers it irrelevant due to external factors (e.g., they have seen similar posts before or solved the issue). For our labeling needs, these posts are classified as relevant. The last reason indicates that the post is genuinely irrelevant. Once again, the user sees a toast informing them that the post gets moved to another

page, similar to what is seen in [Figure 5.11](#) but instead of moving to **Accepted Feedback**, this time the post goes to **Deleted Feedback**.

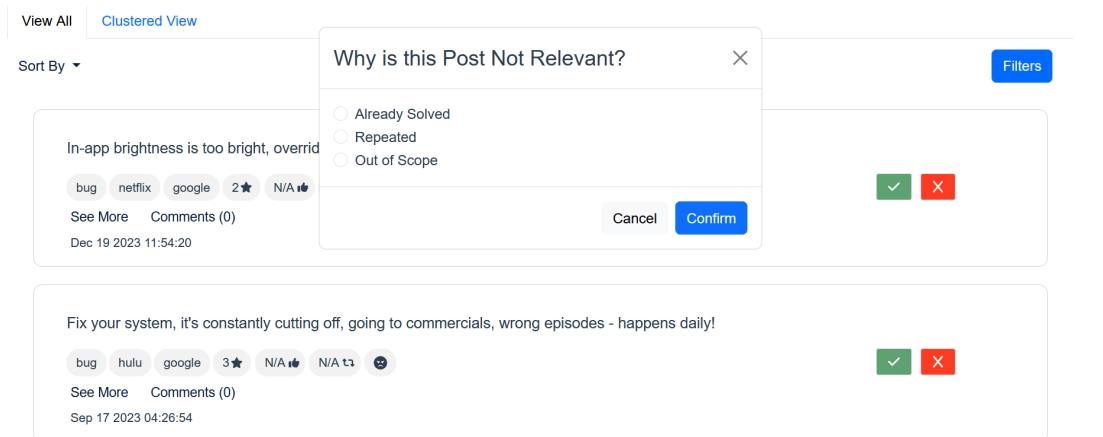


Figure 5.12: Post not Relevant Modal.

While scrolling, when users reach 75% of the page, additional posts will automatically load.

To enhance feedback accessibility, we implemented a comprehensive search and sorting system. Users can search for posts using the search bar located below the navigation bar ([US6](#)). The keywords the user enters are searched through the text of all posts, and matching posts are displayed. Additionally, users can apply filters by clicking the “Filters” button ([Figure 5.13](#)) ([US5](#)). This reveals options to specify a date range and select desired categories, sentiments, social media from which the post was extracted, and streaming services. Users simply check the relevant checkboxes to set their preferences. Once satisfied, they can click “Apply filters” to see the filtered results.

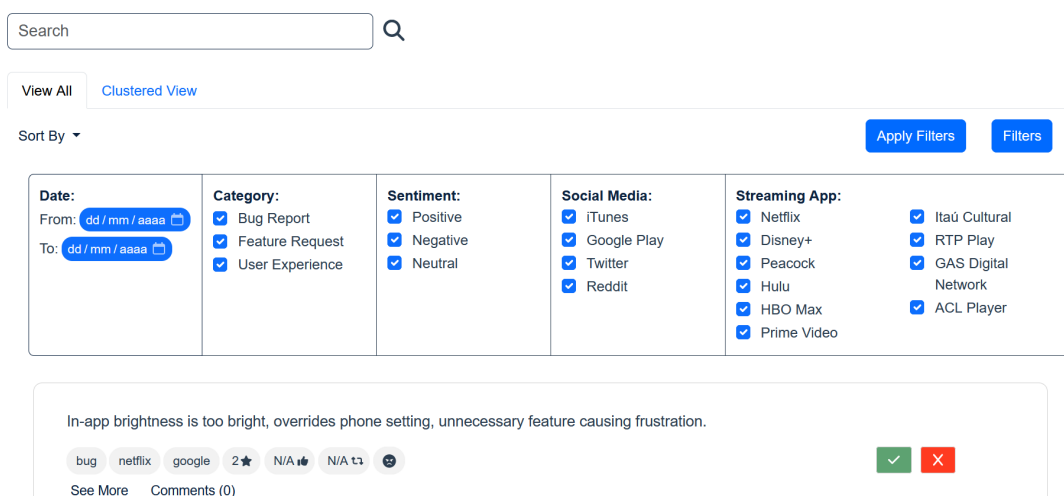


Figure 5.13: Filters.

Users can sort posts based on various attributes. To see all the sorting options, users can click on “Sort By” and be greeted by a dropdown menu, as shown in [Figure 5.14 \(US4\)](#). Users can choose to order posts by date (“Newest/Oldest Date First”), sentiment (“Positive/Negative Feelings First”), rating (“Rating Ascending/Descending”), likes (“Number of Likes Ascending/Descending”), or reposts (“Number of Reposts Ascending/Descending”). Alternatively, they can select the “Default Sort,” which, as the name suggests, is the one that is selected if the user does not pick something else - it aims to maximize the number of relevant posts at the top by prioritizing posts with the worst sentiment score. The selected sorting option will be highlighted in green.

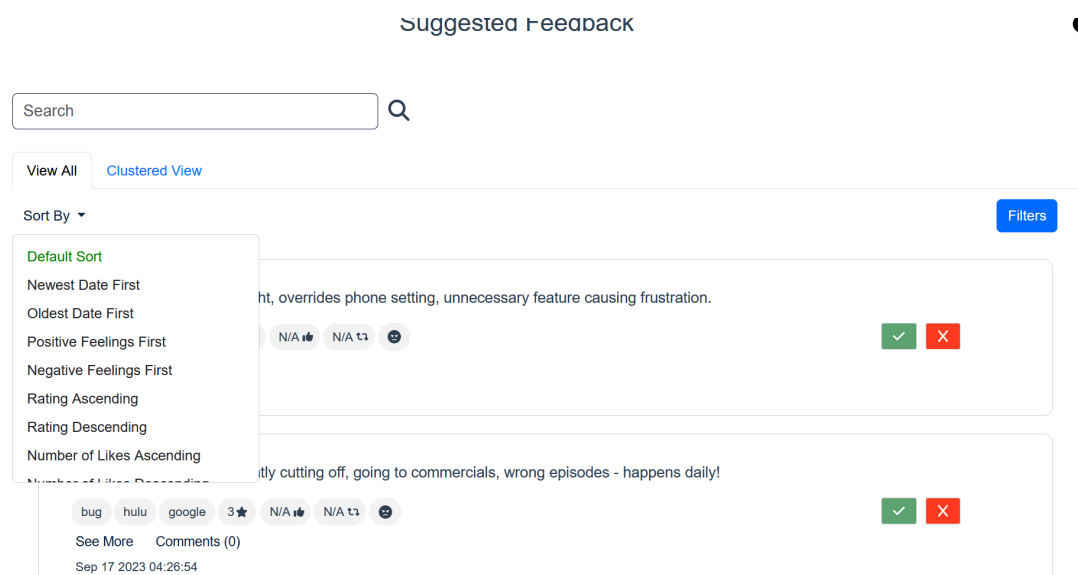


Figure 5.14: Sorting Options.

The search bar, filters, and sorting options combine to provide users with the best possible experience.

In the other visualization option, *Clustered View*, users can switch to it by simply changing the tab ([US3](#)). They will see a series of clusters, displaying their titles and the number of posts they contain ([Figure 5.15](#)).

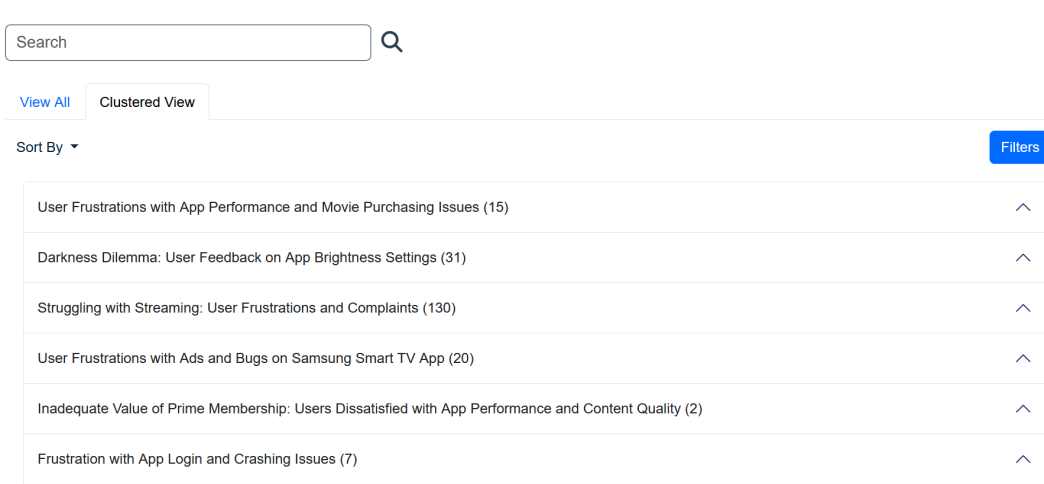


Figure 5.15: *Clustered View*.

To expand a cluster, users just need to click on it, revealing the cluster posts (Figure 5.16).

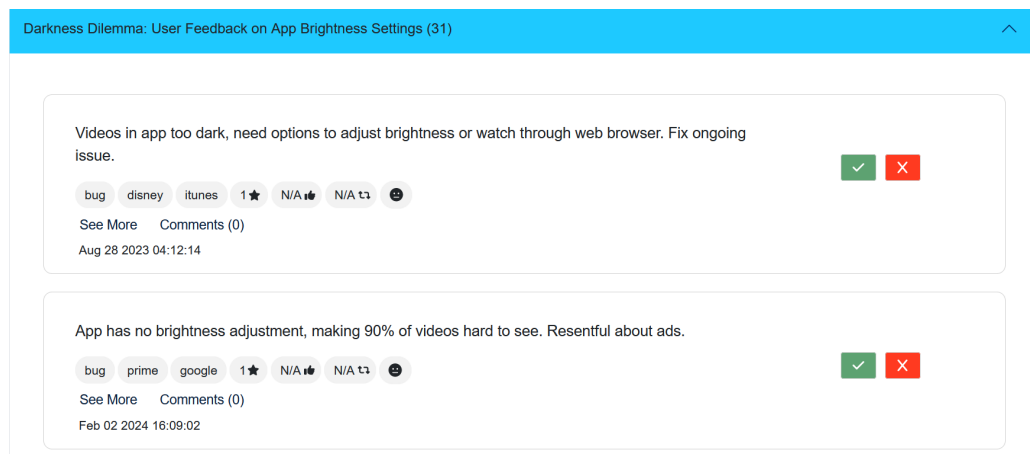


Figure 5.16: Expanded Cluster.

All posts may not be displayed at once, depending on the cluster size. To optimize loading times, the backend sends 50 posts per cluster initially. Clusters take more time to load than individual posts due to the additional API calls required. For clusters with more than 50 posts, users can click the “Load More” button to fetch additional posts (Figure 5.17).

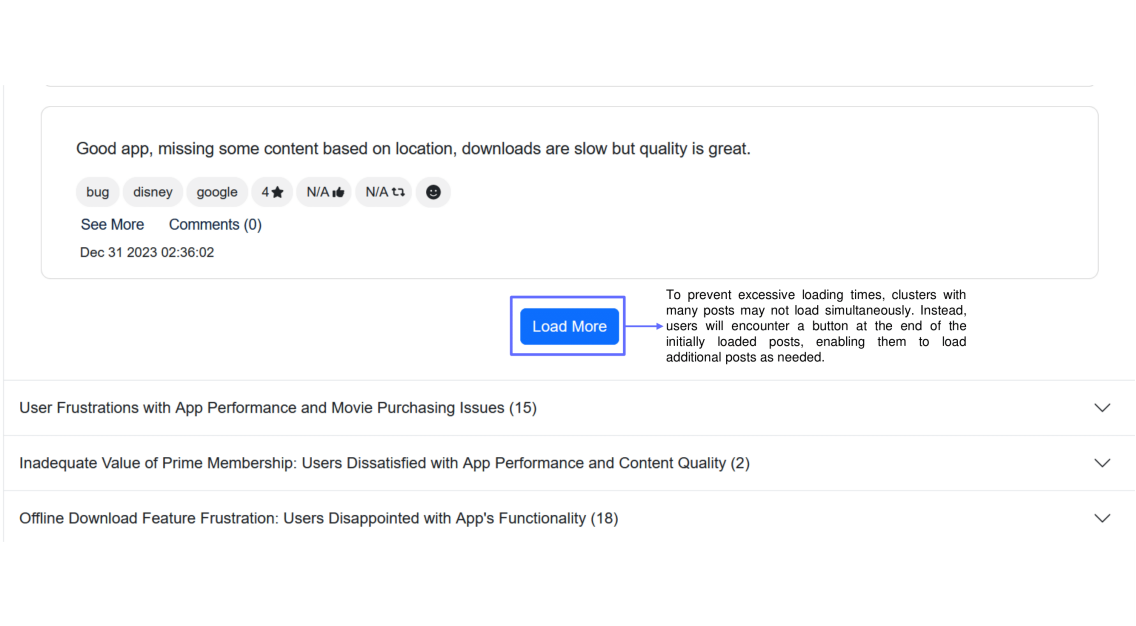


Figure 5.17: “Load More” Button.

The search, filter, and sorting functionalities work for clusters as for individual posts. Additionally, there are two sorting options specific to clusters: “Cluster Size Ascending” and “Cluster Size Descending.”

Furthermore, we implemented a loading symbol to indicate when posts and/or clusters are loading (Figure 5.18).



Figure 5.18: Loading Symbol.

5.3.4 Deleted Feedback Page

The **Deleted Feedback Page** (Figure 5.19) offers a more straightforward interface compared to the **Suggested Feedback Page**. There is only one view, all posts are displayed together, and users can move posts only to the **Accepted Feedback Page** if they reconsider their relevance (US10).

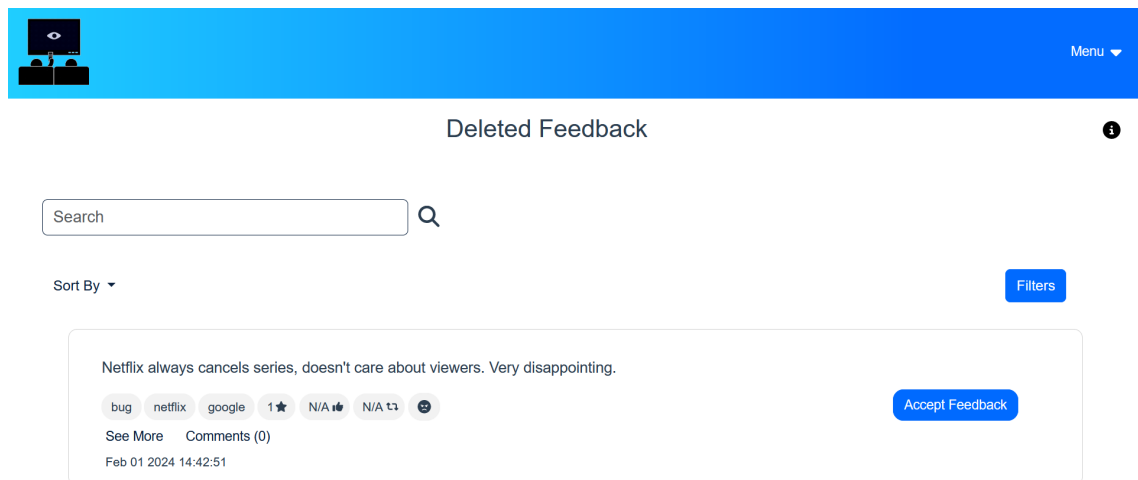


Figure 5.19: Deleted Feedback Page.

As with other pages, users can access the instructions for this page to assist with navigation and functionality (Figure 5.20).

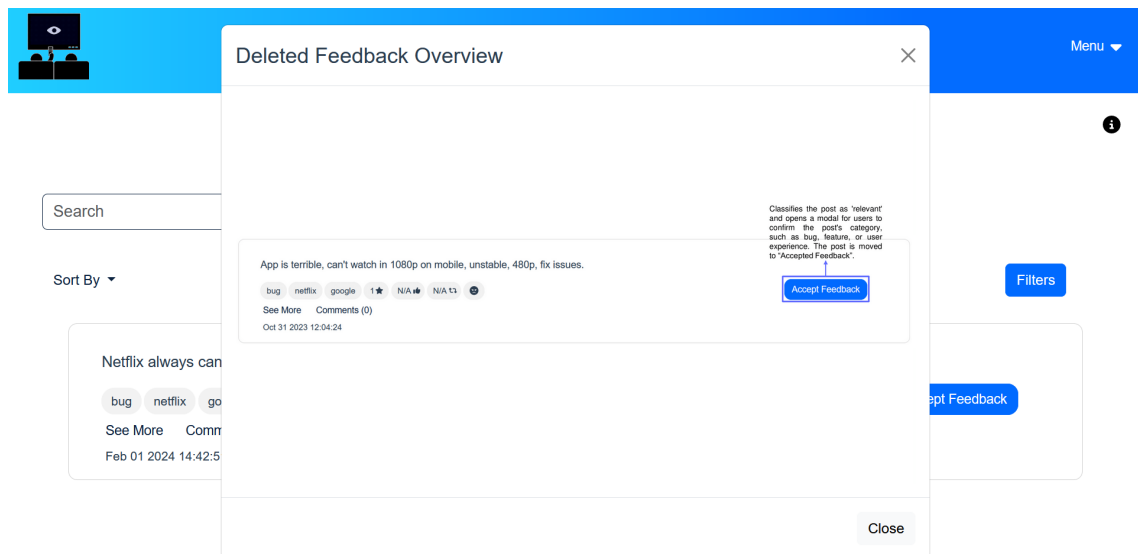


Figure 5.20: Instructions for Deleted Feedback Page.

5.3.5 Accepted Feedback Page

The **Accepted Feedback Page** (Figure 5.21) reintroduces the two visualization options seen on the **Suggested Feedback Page**. However, there are some differences. For instance, the posts do not have action buttons on their cards, and three additional buttons are next to the search bar, which were not present on the previous page.

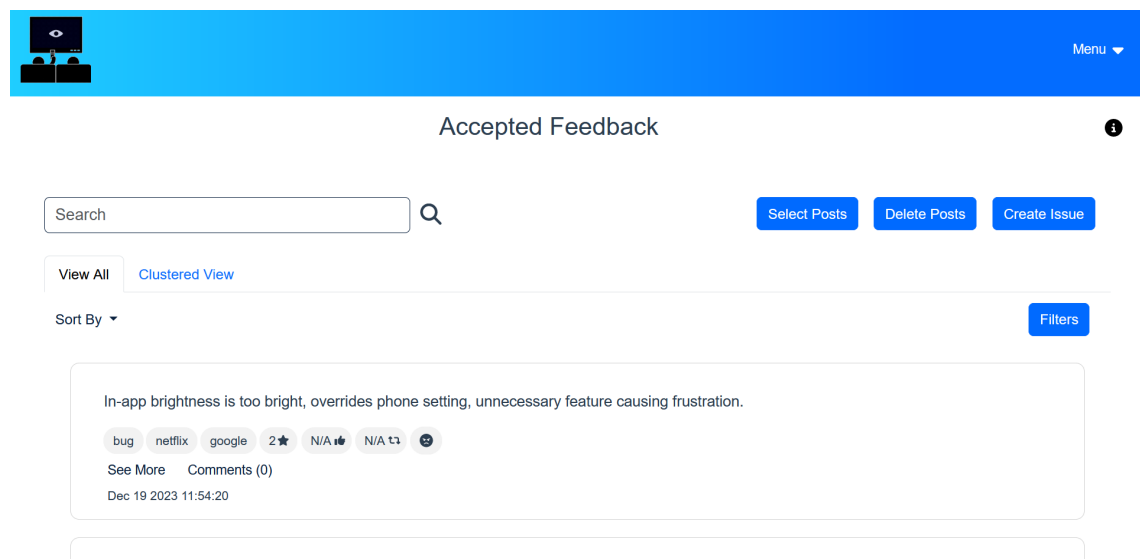


Figure 5.21: Accepted Feedback Page.

As with other pages, users can view instructions specific to this page. Figure 5.22 shows the first instruction image, explaining the functions of the new buttons.

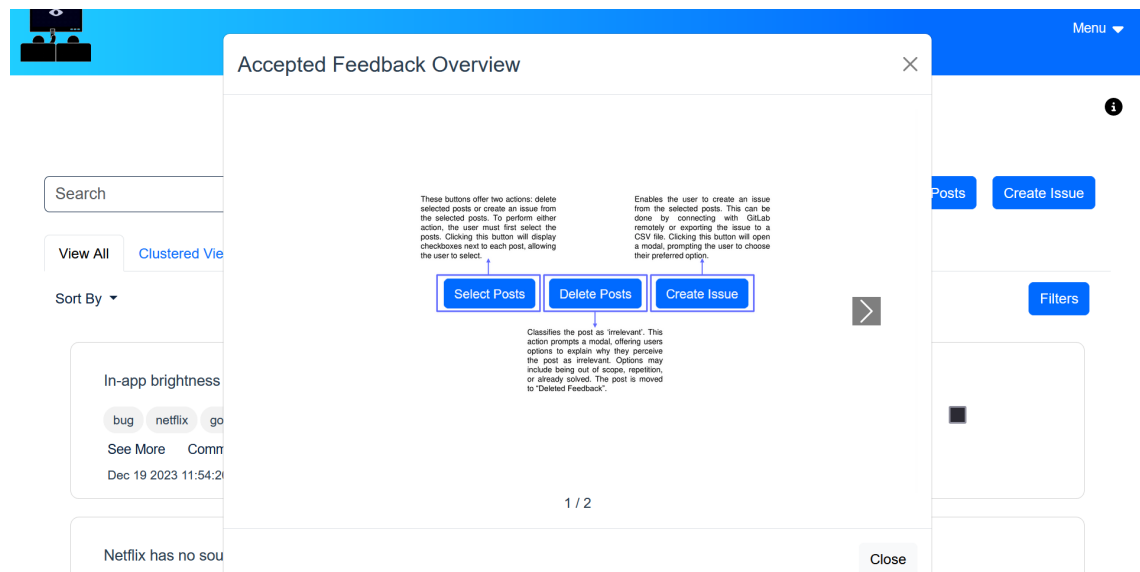


Figure 5.22: Instructions for Accepted Feedback Page.

This page allows users to perform two main actions: move feedback to the **Deleted Feedback** Page if they decide it is irrelevant (**US10**) or create issues based on the feedback (**US11**). Users may want to create a single issue from multiple posts (**US11.2**), especially those within clusters. To enable this, we created the “Select Posts” button (Figure 5.23). Clicking this button makes each post’s checkbox visible.

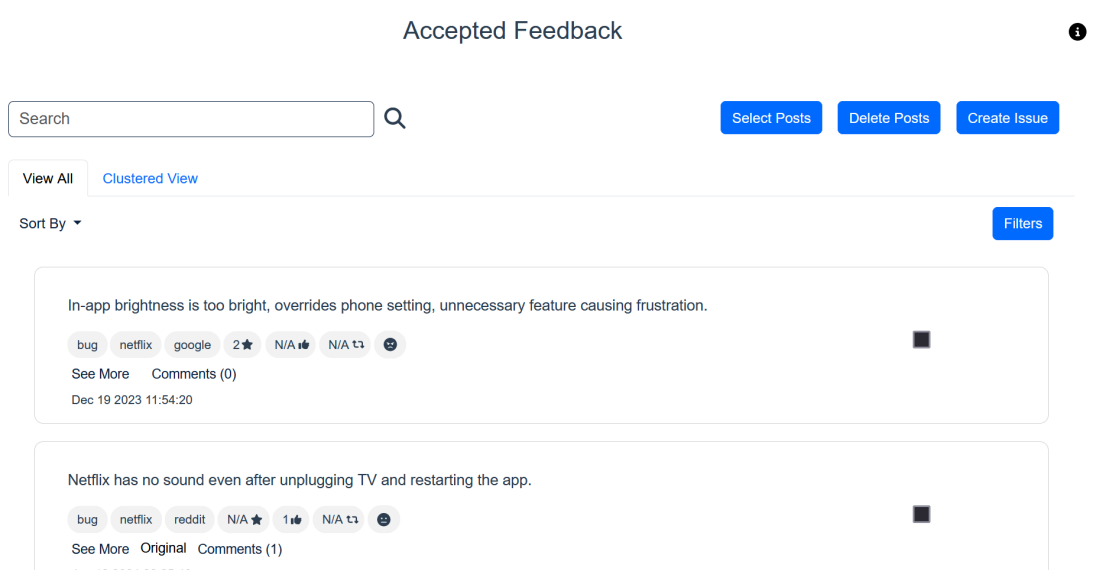


Figure 5.23: Selected Posts.

Before using the “Delete Posts” or “Create Issue” buttons, users must select at least one post. If no posts are selected, a warning will appear via a toast notification (Figure 5.24).

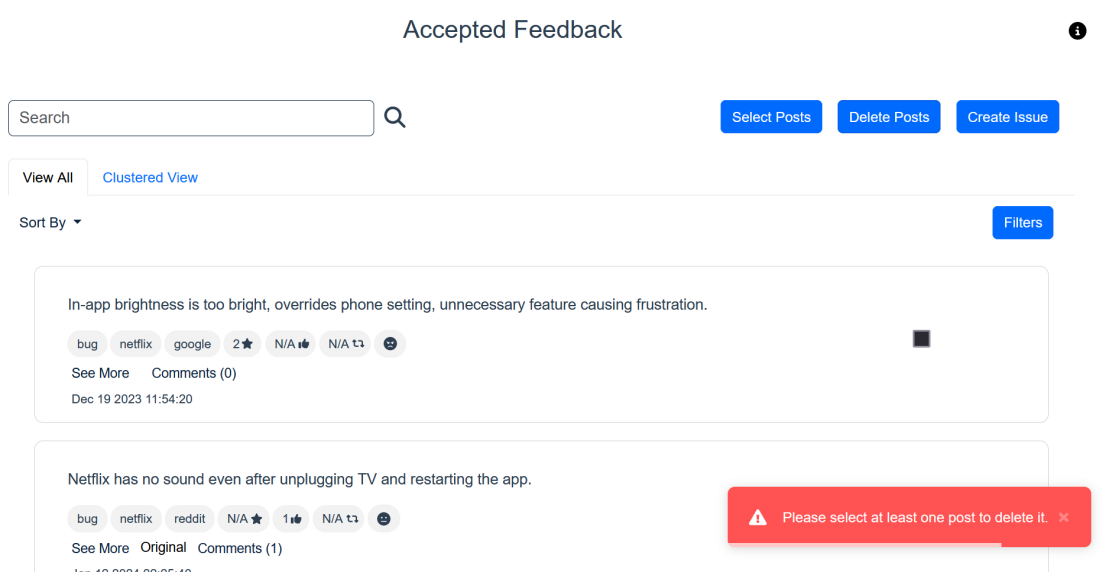


Figure 5.24: Select Post Warning.

When “Select Posts” is active in the *Clustering View*, users can either select individual posts within a cluster or select all posts in a cluster by clicking the checkbox next to the cluster’s title and size (Figure 5.25) (US11.1).

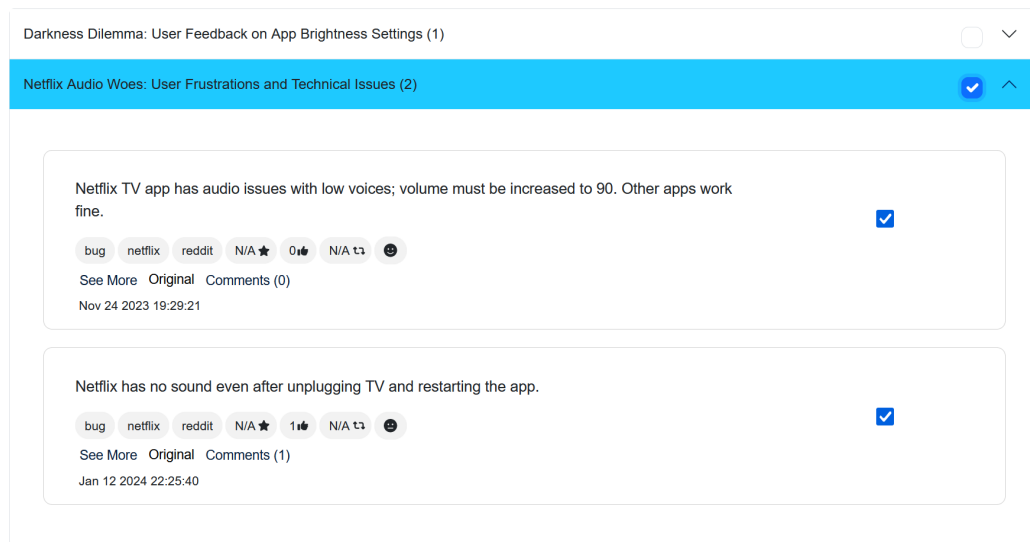


Figure 5.25: Accepted Feedback Clustered View.

Once at least one post is selected, users can either move the post to the **Deleted Feedback** Page or create an issue. To delete posts, users click the “Delete Posts” button. They are presented with a modal asking for the reason the post is considered irrelevant, similar to the process on the **Suggested Feedback** Page (Figure 5.12).

To create an issue, users click the “Create Issue” button, triggering a modal (Figure 5.26) that asks whether they want to connect to an issue-tracking system directly or export the issue to a CSV file (US11.4). We chose to integrate with GitLab because MOG uses it. GitLab allows users to create issues directly on their platform or via CSV file.

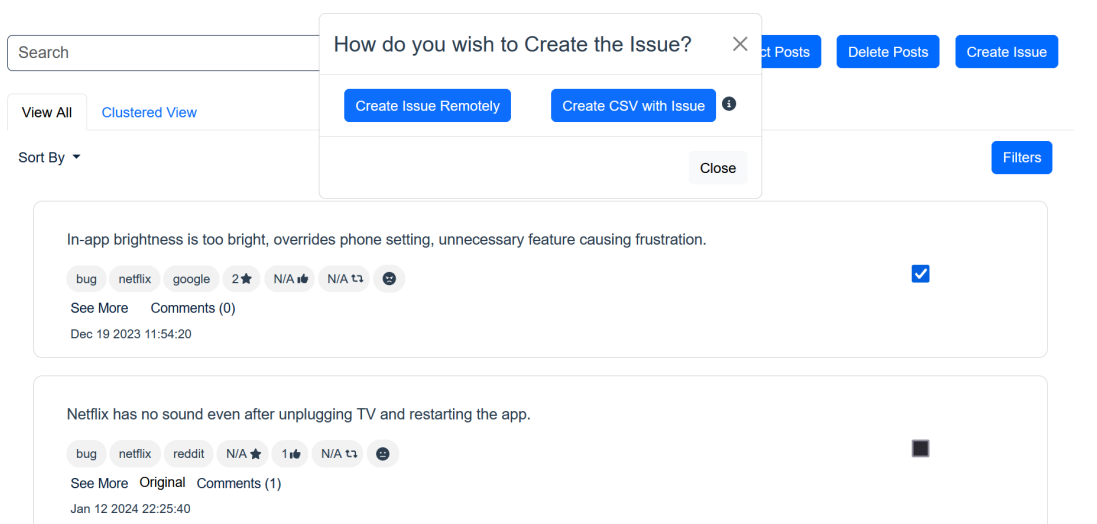


Figure 5.26: Create Issue Modal.

Since creating issues via CSV file is less common in other issue-tracking systems, we included quick instructions (Figure 5.27). Users can access these instructions by clicking the “i” button next to the “Create CSV with issue” option in the modal.

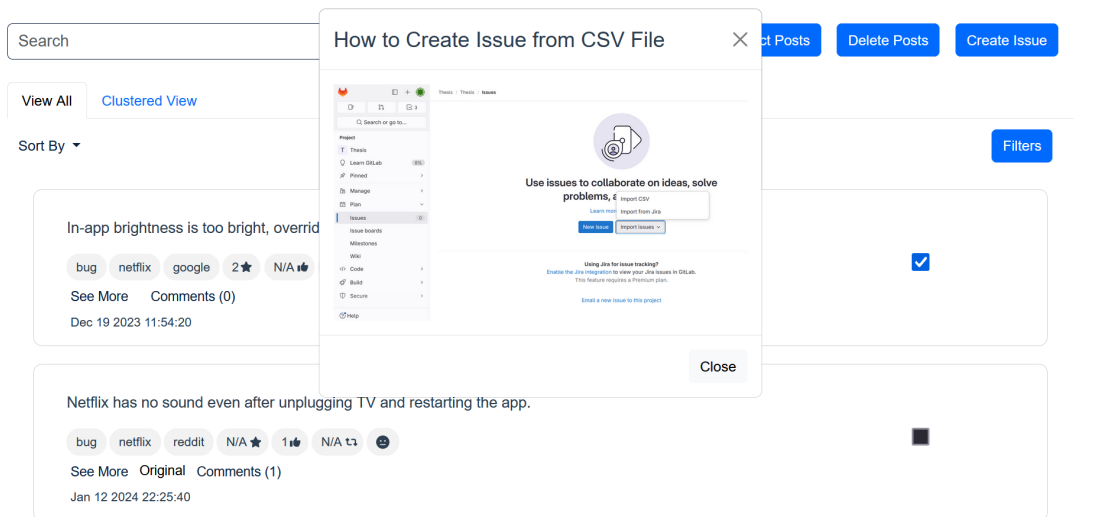


Figure 5.27: Import CSV with Issues to GitLab.

When users choose the CSV file option, they are presented with a form to create the issue (Figure 5.28). If they create an issue remotely, they see a different, more complex modal.

This modal contains two fields: `Title` and `Description`. The application generates suggested values for these fields, but the user can edit them (US11.3).

If a single post is selected, the `Title` suggestion is the post’s title. The cluster’s title is suggested if multiple posts from the same cluster are selected. If multiple posts from different clusters are selected, the summaries of the posts are suggested as the title.

The `Description` field uses one of two Markdown templates⁴, detailed in [Issue Templates](#). The templates differ based on the category of the posts: one for *bug* and another for *feature* or *user experience* posts. *Bug* templates explain the bug and the expected behavior, while *feature* templates include user stories, tasks, and acceptance tests. The full text of the selected post(s) is included in both templates to provide additional context for developers. A tooltip is available for this field; users can view it by hovering over the “i” symbol, which advises them to use Markdown notation when writing in this field.

Once the user clicks “Export,” a CSV file is created and downloaded, and the selected post(s) is/are moved to the **Exported Feedback Page**.

⁴<https://playbook.sparkfabrik.com/tools-and-policies/gitlab-issue-templates>

Figure 5.28: Issue Template.

A more detailed form appears if the user creates an issue remotely (Figure 5.29). Besides the `Title` and `Description` fields, users must fill in `Private Token` and `Project Name`, which are mandatory for GitLab integration. These fields are marked with an asterisk (“*”) to indicate their obligatory nature. A toast notification will show an error message if these fields are incomplete. The connection to GitLab is made through our backend using the GitLab API⁵.

The `Private Token` is needed to access the user’s GitLab account. Instructions for creating this token are available by clicking the “i” button next to the `Private Token` field (Figure 5.30). Since tokens are sensible information, they are encrypted using CryptoJS with the Advanced Encryption Standard cipher algorithm⁶ before being sent to our backend.

The `Project Name` tells GitLab where to create the issue. A tooltip explains the format of this field.

Creating issues remotely allows us to provide additional information, such as `Labels` and `Due Date`. The suggested label is the post(s) category that inspired the issue.

Figure 5.29: Remotely Create Issue for GitLab.

⁵<https://python-gitlab.readthedocs.io/en/stable/>

⁶AdvancedEncryptionStandard

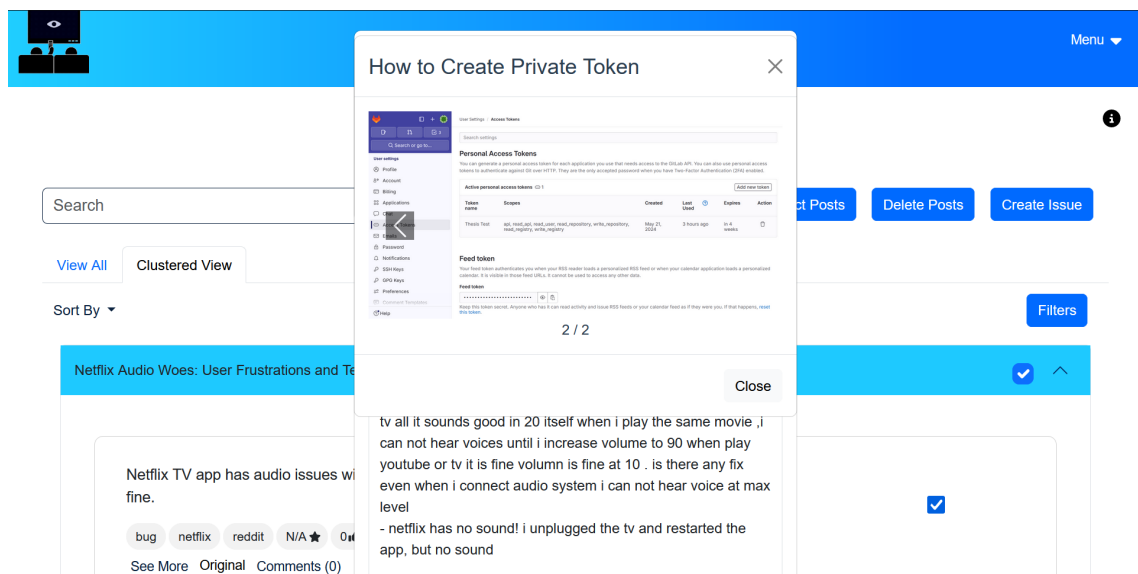


Figure 5.30: Instructions to Create Private Token.

If the user provides the correct access information and fills in all required fields, and if there are no issues with the GitLab API, the issue is created successfully. The application then displays a success message with a link to the created issue (Figure 5.31).

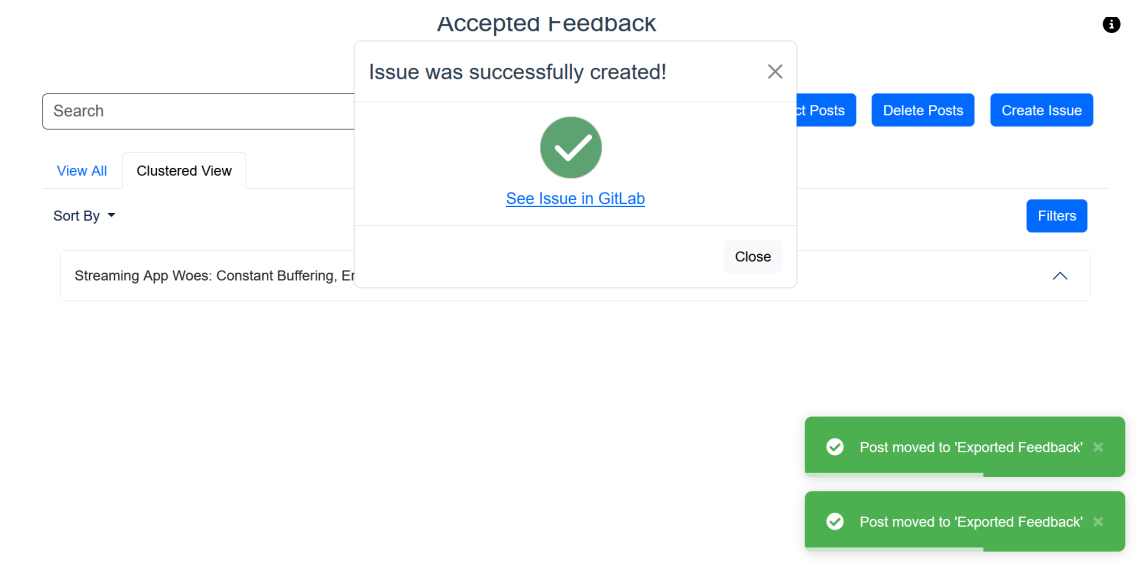


Figure 5.31: Created Issue in GitLab.

5.3.6 Exported Feedback Page

The **Exported Feedback Page** (Figure 5.32) offers a straightforward interface similar to the **Deleted Feedback Page**. There is only one view where all posts are displayed together, and users cannot move these posts to other pages. This page functions as an archive, allowing users to

see posts that have been converted into issues (US12). Additionally, it provides information about the nature of each issue, indicating whether it was exported as a CSV file or resides in GitLab. If the latter is true, a link to the associated GitLab issue is provided (US13).

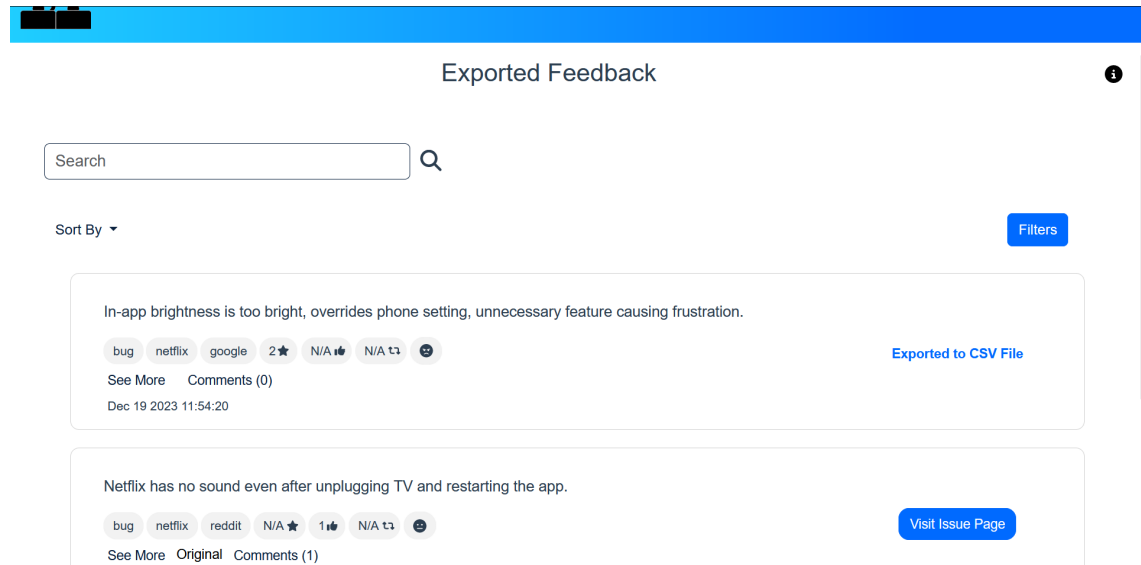


Figure 5.32: Exported Feedback Page.

As with other pages, users can access the instructions for this page (Figure 5.33).

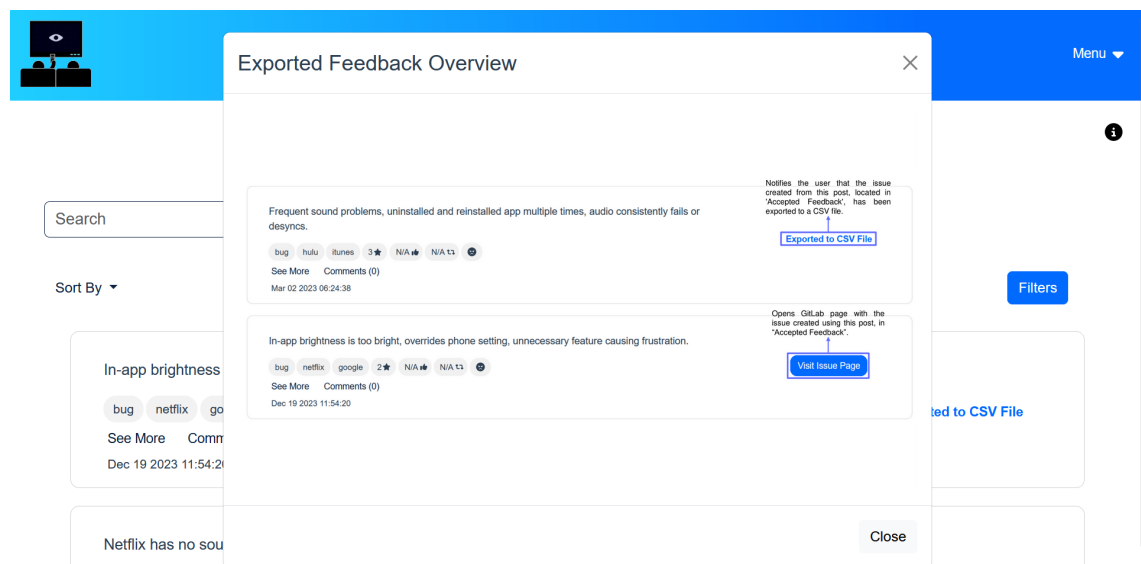


Figure 5.33: Instructions for Exported Feedback Page.

5.4 Conclusion

Overall, “Watching the Watchers” is designed to streamline feedback management, from initial feedback collection to issue creation and review. Each page has intuitive features and user-friendly navigation tools, ensuring users can efficiently manage feedback, leading to better decision-making and improved user satisfaction.

Chapter 6

Validation

This chapter presents two validation methods for the “Watching the Watchers” application.

First, we introduced the application to 13 members of MOG and asked them to complete a quantitative questionnaire with their feedback about the platform. Additionally, we encouraged questions and comments during and after the presentation to gather more in-depth feedback.

Second, we monitored the active usage of the platform by a user, tracking the number of posts deemed relevant and the number of posts leading to issue creation.

6.1 User Feedback Evaluation

6.1.1 Questionnaire Design

Our questionnaire consists of four sections designed to gather information on different aspects:

- **User Demographics and Background:** This section collects relevant information about the participants, including their position in the company, the duration of their tenure in that position, and the frequency with which they handle user feedback. Understanding the background of participants, especially those who regularly engage with user feedback, is crucial as their insights can provide valuable perspectives on the effectiveness and usability of the application;
- **Usability:** This section includes questions inspired by the System Usability Scale (SUS), a widely used tool to evaluate the usability of products and services developed by John Brooke [10]. These questions are rated on a 5-point Likert scale ranging from “Strongly Disagree” to “Strongly Agree.” The SUS scoring method will be explained in detail in the following subsection;
- **Relevance:** Inspired by the SUS, this section includes statements regarding the relevance of posts and the effectiveness of the platform’s actions. Users rate these statements on a 5-point Likert scale.

6.1.2 Results

6.1.2.1 User Demographics and Background

We received responses from 6 out of 13 participants:

- One Chief Marketing Officer with one year of experience;
- Two Software Developers, one with two years and the other with one year and three months of experience;
- One Communication Specialist with two years of experience;
- One Designer with three years of experience;
- One VP Digital Strategy with over two years of experience.

Regarding the frequency of interacting with user feedback, we used a 5-point Likert scale, where 1 means “Never” and 5 means “Very Often”. The responses were as follows: two participants rated it as 3, two rated it as 4, and two rated it as 5.

6.1.2.2 Usability

To evaluate usability, we adapted eight of the ten standard SUS statements and added one custom statement: “I found the platform’s design intuitive.”

Since we adapted 80% of the SUS statements, we will calculate their score following the SUS methodology, as explained by Brooke [10]. Each statement’s score contribution ranges from 0 to 4 (“Strongly Disagree” to “Strongly Agree”). For odd-numbered statements (positive statements), the score contribution is the user’s response minus 1. For even-numbered statements (negative statements), the contribution is five minus the user’s response. To calculate the total score, we sum the score contributions of all statements and multiply by 2.5 to obtain a score between 0 and 100.

Given our modification of having nine statements instead of 10, we adjusted the final multiplication factor. Since we have nine statements, we multiplied the total score by approximately 2.78 (25/9) instead of 2.5. Additionally, we needed to determine how to calculate the score for our custom statement. We observed that all even-numbered statements were negative, and all odd-numbered statements were positive. Since our custom statement was positive, we applied the calculation method for odd-numbered statements. In [Figure 6.1](#), we listed all the questions and answers.

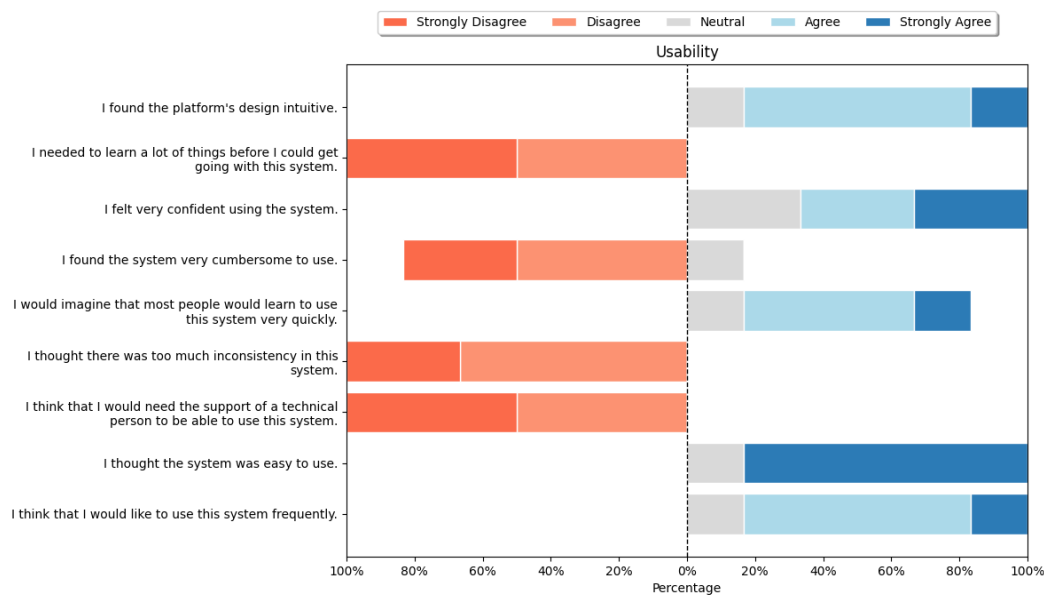


Figure 6.1: Usability Survey Section Answers.

Finally, we averaged the scores from all responses to determine the overall SUS score and used Bangor et al.'s work [8] to translate this score into an adjective rating. Our average SUS score was 78.7, placing "Watching the Watchers" solidly in the "Good" category, which ranges from approximately 74 to 85.

6.1.2.3 Relevance

In this section, we follow the same questionnaire style as SUS by giving participants multiple statements and asking them to rate them from "Strongly Disagree" to "Strongly Agree." We did not adapt the calculations from SUS since we only made positive statements, so instead, we calculated a simple average for each item using a scale from 0 to 4.

Overall, each statement received an average score of 3 or higher ("Agree"), indicating general agreement among participants. The last two statements, which evaluated the platform's helpfulness in creating issues and whether participants would recommend it, received exceptionally high scores, averaging 3.5. In Figure 6.2, we listed all the questions and answers.

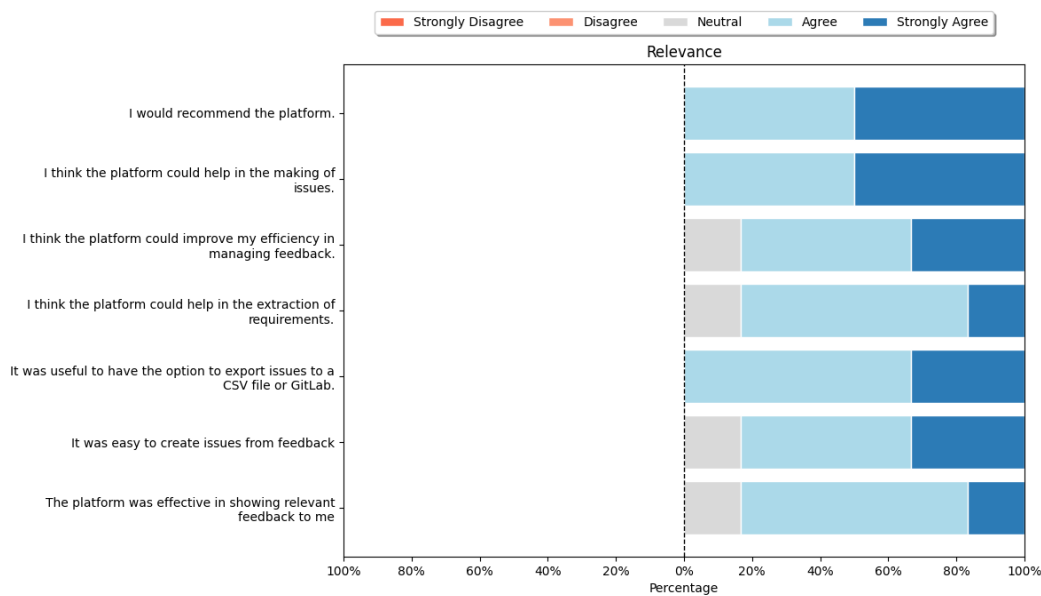


Figure 6.2: Relevance Survey Section Answers.

These results suggest that participants found the platform valuable and effective in its intended functions. The consistently high scores across different aspects of relevance indicate intense user satisfaction and highlight the platform’s potential to improve feedback management and issue creation processes.

6.1.3 Participants’ Feedback

We also gathered additional qualitative feedback from all participants to understand the specific areas of strength and potential improvement. The feedback highlighted that the platform effectively aggregates user feedback into short, filtered results supported by a competent user interface and a feature-rich workset. Participants emphasized the platform’s role in automating feedback management and giving the company a strategic edge at its current stage of development. Here are some examples of their statements:

“Watching the Watchers came from the idea that user feedback aggregation could be done at a large scale into short filtered results. It accomplishes this goal with a very competent user interface and a feature-rich workout.”

“Extremely impressive at its current stage where it already gives our company a clear edge in strategy.”

Participants also raised questions about the potential repurposing of the platform for other types of data, which was affirmed as feasible. Additionally, they expressed interest in seeing all feedback aggregated on a single platform, achievable with our filter system, and reaffirmed its importance.

Several suggestions for future work were provided, highlighting the interest in “Watching the Watchers,” such as:

- Expanding the platform to aggregate comments and capture the actual opinions of a specific audience;
- Developing capabilities to correlate feedback with changes made on the platform, for instance, tracking user reactions to new features introduced by the streaming app developers.

This qualitative feedback provides valuable insights into the practical application of the platform in real-world scenarios and highlights user experiences that could guide future enhancements. Incorporating these suggestions and addressing the identified areas for improvement can further enhance the platform’s effectiveness and user satisfaction.

6.2 Active Usage Monitoring

6.2.1 Experiments and Results

We conducted three distinct experiments with a user to evaluate the relevance of posts, identify potential issues with relevant posts, and gather user feedback. The user was tasked with reviewing posts in **Suggested Feedback** Page (Figure 5.5) and classifying them as relevant or not. For posts deemed relevant, they were then instructed to create issues using the **Accepted Feedback** Page (Figure 5.21). They were advised to create issues using the *Clustered View* (Figure 5.25) to evaluate the effectiveness of the clustering effort.

Each experiment differed in the types of posts the user analyzed:

1. In the first experiment, the user reviewed all existent posts regarding streaming apps created using Vizzi;
2. In the second experiment, the user analyzed the first 100 posts displayed by the platform using the “Default Sort” method (most negative feelings and descending number of likes);
3. In the final experiment, the user assessed 100 posts randomly selected from the database.

The results of these experiments can be summarized in the following table:

	#Posts	#Relevant Posts	Precision	Total #Issues	#Single-Post Issues	#Multi-Post Issues	Duration
Experiment 1	55	47	85.4%	24	13	11	~35 min
Experiment 2	100	89	89%	47	32	15	~1h 20 min
Experiment 3	100	68	68%	42	32	10	~1h

Table 6.1: Summary of Results of Active Usage Monitoring.

Two of the issues created are detailed below.

Title

Login Failure - User Not Recognized and Password Reset Unresponsive

Posts Describing the Needs

- it is hard to find an app this bad. logging in is hell, it keeps giving me errors. i give up
- nothing works haha! i can not long in on the app or the web site. i reach out to the studio and get no response and no help. what a waste of time. love the shows though.
- the initiative is great, but it has a lot of errors, you can not log in and you can not recover your passwords
- I registered, I received an e-mail confirming my registration, when I try to log in it says that the user does not exist. I tried to change my password but it does not work
- app and site down unable to log on account is on auto-pay but site and app now say my user does not exist. tried resetting password but same issue.

Expected behavior

- Users should be able to log in without encountering errors.
- The login system should recognize registered users and allow access with the correct credentials.
- Password reset functionality should work reliably, allowing users to reset their passwords and regain access to their accounts.
- The platform should maintain a stable and reliable status, minimizing downtime and ensuring accessibility for users at all times.

Title

Implementation of Back Navigation Feature

User Stories

For: Users who navigate through the app or website.

How: By providing a back navigation feature that allows users to return to the previous screen or menu.

I want: To easily navigate back from a video or any other page to the main menu or the previous page without having to scroll to the bottom or exit the app.

Posts Describing the Needs

- "Good start, needs back feature instead of scrolling to the bottom. Also needs access to all Legion of Skanks episodes; that is the reason I subscribed, not to just go back to 2022."
- "Good initial launch. Please implement a way to exit out of the videos. I could not get back to the main menu after the video ended. Drove me nuts!"

Acceptance Criteria (done when...)

- A back button is visible on all screens and allows users to return to the previous screen.
- The back button has a consistent style across the application, with a label 'Back' or an icon representing back navigation.
- When the back button is pressed, the user is taken to the previous screen without losing their current state.
- The back button is easily accessible and does not interfere with other UI elements.
- Users can return to the main menu from a video screen using the back button.

Validation Scenarios

Scenario: User uses the back button to return from a video screen to the main menu

Given The user is on a video screen

Given When the user presses the back button

Given Then the user is taken back to the main menu

Scenario: User navigates back from a sub-menu to the main menu

Given The user is on a sub-menu

Given When the user presses the back button

Given Then the user is taken back to the main menu

Tasks

- Design the back button UI for consistency across the application.
- Implement the back button functionality on all screens.
- Test the back button to ensure it navigates to the previous screen without losing the current state.
- Conduct user testing to verify the ease of use and functionality of the back button.
- Gather feedback and make necessary adjustments to the back navigation feature.

6.2.2 Discussion

The first experiment demonstrated that “Watching the Watchers” can effectively identify many relevant posts about platforms using Vizzi quickly. While this capability is incredibly positive for MOG, they also value gaining a comprehensive understanding of the streaming industry, which includes insights from other streaming services.

The second and third experiments highlighted the effectiveness of the platform’s sorting mechanism. While both experiments involved analyzing 100 posts, the sorting mechanism in the second experiment yielded 21% more relevant posts than the third experiment, all due to how the posts were sorted.

Furthermore, we aim to compare the precision values observed in each experiment with the overall precision of the dataset, which we calculated in [Chapter 4.3.1](#) to be 75.5%. We cannot calculate the values of recall observed in each experiment because it would require knowing the number of False Negatives (FN), which represent actual relevant posts that were not identified as

relevant by the system. Since we lack the total count of these missed posts by the models, we cannot accurately calculate recall from the provided data. These FN values are unavailable for the experiments because the relevant posts were missed during the classification process before the posts entered the platform.

Precision, however, measures the proportion of correctly identified relevant posts among all posts classified as relevant by the system. It is calculated using $\frac{TP}{TP+FP}$, where TP is represented by the *#Relevant Posts* column, and TP + FP by the *#Posts* column in [Table 6.1](#). The precision values for each experiment can be seen in the *Precision* column of the same table. Experiment 3 achieved 68%, slightly lower than the overall dataset average but within an acceptable range. Experiments 2 and 3 demonstrated higher precision than the overall database average, further highlighting the effectiveness of the sorting mechanism.

Regarding issue creation, the user utilized the function to create issues for entire clusters and employed the post-selection feature to combine posts that did not belong to any cluster. These posts addressed similar subjects; however, our clustering algorithm did not group them together because the authors used different terminology to refer to the same subjects. An example of this is the following posts:

- Needs back feature and access to all episodes. I subscribed for full Legion of Skanks experience.
- Good start, need exit option for videos to return to main menu. Frustrating to navigate.

Our clustering algorithm uses sentence similarity to create clusters, assuming posts are similar if they use similar terminology. When this does not occur, no cluster is created. In this case, the user and we realized that the posts were related due to context awareness, which our algorithm currently lacks. Addressing this limitation may require the use of LLMs.

Overall, the user found the application intuitive and straightforward to use. They noted that the application is particularly helpful in creating issues and providing a clear overview of all the feedback.

Chapter 7

Conclusions and Future Work

This dissertation aimed to leverage social media to gather user feedback and develop an intuitive visualization platform that enables stakeholders to interpret and act upon this feedback. The overarching objectives were comprehensively addressed, yielding significant advancements in theory and practice.

The platform effectively manages feedback, streamlining the process for stakeholders and enabling more efficient decision-making. It was validated by MOG employees, who agreed that it was intuitive and effective in helping them manage user feedback, providing a strategic edge to MOG at its current stage of development.

Our work introduces several innovative aspects:

- **Industry Validation:** Unlike purely academic studies, our work integrates practical validation from industry experts. This collaboration ensures that our platform not only meets academic rigor but also fulfills real-world industry needs, bridging the gap between theory and application effectively;
- **Automated Data Extraction and Social Media Feedback Processing Workflow:** We automated the data extraction process using customized scripts for multiple social media platforms. This automation enhances efficiency and scalability, eliminating manual data collection and ensuring comprehensive coverage across different platforms and languages. In processing user feedback, we employed innovative techniques such as leveraging LLMs for summarization and implementing a custom clustering algorithm;
- **Integrated Feedback Visualization and Issue Creation:** Our platform integrates data aggregation and visualization, providing stakeholders with actionable insights through a user-friendly interface. This integration empowers users to identify and create issues to fix bugs or create requested features. Beyond the theoretical exploration of social media's role in RE, the challenge persists in translating insights into actionable outcomes. The existing literature in this domain primarily serves academic purposes, where researchers leverage

popular applications to obtain data and test hypotheses. However, a critical gap exists in the interaction of stakeholders involved in developing those products with the requirements derived from these studies;

- **Indirect Feedback Acquisition:** By focusing on MOG's Vizzi platform, we captured user feedback indirectly from various streaming services. This unique approach allowed us to glean insights without direct user interaction with MOG or Vizzi, pioneering a novel method for feedback aggregation. This unique scenario introduces a novel layer of filtering that is unexplored in the existing literature.

We developed a roadmap for future work to enhance our application further:

- **Enhancing Model Performance:** Utilize analyzed posts to train the models with additional data, thereby improving their performance. As mentioned in [Chapter 5](#), user feedback is crucial. We ask users to confirm the relevance of posts and specify reasons when they mark a post as irrelevant. This helps us determine whether irrelevance is due to context or content and ensures accurate categorization;
- **Recurrent Automated Post Extraction:** Establish a weekly background task where the application fetches new posts and processes them through the pipeline. While the extraction and pipeline process is already implemented, setting up the automated background task is pending;
- **Post Refresh Option:** Include an option for users to refresh current posts to get updated likes, reposts, and comments, thereby providing more accurate information;
- **User Notifications:** Implement a notification system to alert users when new posts are available for analysis.

Additionally, two suggestions for future work were identified during the Validation phase ([Chapter 6.1.3](#)):

- **Aggregating Posts:** Enhance the platform to aggregate related posts and capture the collective sentiment and opinions of users on various topics. Utilizing LLMs, we will analyze and summarize aggregated posts effectively. This approach simplifies user interaction by presenting summarized insights rather than individual posts. Currently, the platform aggregates posts into clusters and provides a summary based on these clusters. This enhancement aims to focus on presenting the general opinion related to a topic without displaying all individual posts;
- **Correlating Feedback with Platform Changes:** Develop capabilities to correlate user feedback with changes made on the platform. For example, tracking user reactions to new features introduced by the streaming app developers. This would require elaborating on the connection between the application and GitLab, enabling searches through existing issues in

the repository. Keywords from descriptions or titles could be used to search among already extracted posts or initiate new searches on social media.

Additionally, while the platform is currently applied within the streaming industry, its design is modular, allowing it to be adapted to various domains. The only modifications needed would be in the data extraction scripts to align with the keywords and context of the new domain, showcasing the platform's versatility and broad applicability.

In conclusion, this dissertation has successfully developed a powerful tool that leverages social media insights for requirements engineering. The platform not only meets the current needs of MOG but also sets the stage for future advancements, ensuring ongoing relevance and utility in a dynamic development environment. The contributions made here pave the way for further innovation and application in academic and industry contexts.

Appendix A

Comprehensive Analysis of Research Papers

Categories	SubCategories	References
Social Media	Mobile Stores (Apple Store and Google Play)	[14, 23, 40, 42, 44, 47, 48, 51, 52, 70, 71, 30, 53, 17, 12]
	Twitter	[2, 4, 5, 14, 18, 22, 24, 25, 41, 47, 48, 50, 51, 62, 73, 74]
	Reddit	[7, 28, 29, 33, 35, 57]
	Online Forums	[35, 38, 45, 69, 33]
	Facebook	[2, 4, 5, 51]
Classification Algorithms	(Multinomial) Naive Bayes	[57, 28, 7, 35, 18, 14, 24, 74, 17, 69, 42, 40, 51, 62, 38, 15, 47, 48, 52, 23, 12, 73, 4, 22, 44]
	SVM	[57, 28, 7, 35, 14, 74, 25, 17, 62, 38, 15, 47, 52, 23, 73, 4, 22, 44]
	RF	[57, 28, 7, 35, 18, 24, 17, 62, 70, 4, 22]
	DT	[28, 18, 25, 17, 42, 40, 62, 52, 22, 44]
	bug report, feature request	[28, 27, 35, 18, 74, 51, 47, 14, 24, 42, 40, 15, 25, 22, 62, 70, 16, 53, 52, 23, 4]
Requirement Categories	other	[18, 74, 51, 47, 24, 70, 16, 53, 52, 25, 22]
	user experience	[42, 40, 15, 23, 25, 22, 4]
	irrelevant	[28, 27, 62, 4]
	feature strength, feature shortcomings, praise, complaint	[23, 25, 22]
	rating	[42, 40, 15]
	information giving, information seeking	[16, 53, 52]
	Hardware constraint, Software constraint, Advertisement, Dissuasion, Question, How to, Feature information, Software price, Compliance issue, Software extension, Other product, Service, Social interaction, Content related, Job advertisement, Noise, Unclear, Unrelated	[25, 22]
	claims	[35]
Preprocessing	Tokenization	[27, 35, 24, 25, 30, 53, 71, 12, 45, 4, 22, 57, 28, 7, 18, 15, 16]
	Lemmatization	[7, 2, 18, 14, 42, 40, 62, 15, 47, 48, 71, 5, 45]
	Stemming	[28, 18, 14, 24, 74, 40, 16, 53, 70, 52, 23, 71, 12, 4, 44]
	POS	[2, 69, 30, 71, 5]
Sentiment	SentiStrength	[24, 74, 42, 40, 51, 62, 73, 22]
	StanfordCoreNLP	[16, 45]
Feature Engineering	Frequency Based Features	[28, 24, 25, 17, 40, 51, 62, 38, 52, 23, 71, 22]
	Bag of Words	[7, 35, 74, 17, 69, 42, 40, 15, 4]
	N-Gram Extraction	[2, 24, 69, 40, 38, 15, 70, 71, 4, 41]
Multilingual Feedback	Word and Sentence Embedding	[57, 50, 62]
	English Only Feedback	[40, 28, 68, 47, 74, 34, 57, 71, 29, 25, 42, 38, 44, 14, 18, 73, 22, 35, 33, 4, 12, 53, 5, 70, 69, 45, 15, 50, 7, 16, 2, 23, 17, 30, 52, 41, 48]
Data Imbalance	English and non-English Feedback	[27, 62, 51]
	SMOTE	[57, 28, 7, 35, 17]
Grouping	Under-sampling	[27, 17]
	Topic Modeling	[50, 14, 24, 29, 47, 48, 12, 45, 4, 41, 44]
Summarization	Clustering	[51, 70, 71, 12]
	SumBasic	[57, 74]
	TextRank, LexRank	[57]
	Hybrid TF, Hybrid TF-IDF	[74]
Training and Evaluation	F-Measure	[7, 35, 18, 14, 24, 74, 25, 17, 69, 42, 40, 62, 38, 15, 47, 30, 53, 52, 23, 12, 33, 4, 22, 44, 57, 28, 27]
	Recall	[7, 35, 14, 24, 74, 25, 69, 42, 40, 62, 38, 15, 47, 30, 53, 70, 52, 23, 12, 33, 4, 22, 44, 57, 28, 27]
	Precision	[27, 7, 35, 14, 24, 74, 25, 69, 42, 40, 62, 38, 15, 47, 30, 53, 70, 52, 12, 33, 4, 22, 41, 44, 57, 28]
	Cross-Validation	[57, 28, 7, 35, 18, 24, 74, 25, 17, 69, 42, 62, 38, 47, 48, 70, 52, 23, 73, 4, 44]
	Accuracy	[28, 27, 38, 30, 70, 71, 44]
	Hold-out Method	[74, 40, 52, 23]
	ROC Curve	[69, 62, 70]

Appendix B

Machine Learning and NLP Techniques

In this chapter, we provide an overview of the Machine Learning and NLP techniques relevant to the papers analyzed in [Chapter 3](#). Understanding these methodologies is essential for comprehending the technical approaches and advancements discussed throughout this dissertation.

B.1 Machine Learning

B.1.1 Classification Algorithms

In text analysis and data mining, the choice of classification algorithms plays a crucial role in extracting meaningful insights from diverse datasets. In this section, we explore several prominent classification algorithms, each with its unique characteristics and applications. These algorithms, including SVM, NB, DT, and RF, are fundamental tools in supervised machine learning.

SVM is a supervised machine learning algorithm utilized for classification and regression analysis within multidimensional data spaces. Its primary objective is to identify optimal hyperplanes for linearly separable patterns in the data, with an emphasis on maximizing the margins around these hyperplanes. Support vectors, crucial instances within the training set, play a pivotal role in determining the position of the dividing hyperplane. SVM achieves data classification by mapping input vectors into an N-dimensional space and determining which side of the defined hyperplane a data instance resides on. SVMs exhibit effectiveness in sparse and highly dimensional data, largely attributed to their tendency to avoid overfitting, a characteristic inherent in the margin maximization process [\[74\]](#).

SVM has demonstrated success in text classification, especially for handling large feature sets, making it particularly suitable for text data analysis [\[4\]](#).

It has also been heavily used in **Twitter** data and has shown interchangeably good performance (predominantly linear kernel of SVM) [\[73, 74\]](#).

NB is an efficient linear probabilistic classifier grounded in Bayes' theorem. Its underlying assumption revolves around the conditional independence of data attributes, indicating that classification features are considered independent given the class [\[74\]](#). Taking an example from Nayebe

et al. [47] to explain further, “*this would mean that if the tweet contains the word ‘tracker,’ this gives us no further information about ‘calorie’*”.

In text classification, features are often individual words represented using the BoW approach. This approach typically represents the data by a 2-dimensional word x document matrix. The Bernoulli NB model uses binary values to indicate word presence (i.e., 0,1), while the Multinomial NB uses normalized word frequencies to construct the word x document matrix. It is shown to be consistently outperforming the binary feature model [74].

Despite the unrealistic conditional independence assumption, NB tends to be robust to noise, making it effective in **Twitter** data classification [4]. It is characterized by its simplicity, few parameters, and ease of interpretation, making it one of the easiest algorithms to use, especially when dealing with noisy datasets [23, 40, 42].

DT are a popular classification algorithm that assumes finite discrete domains for all features, with a single target feature representing classification. They are straightforward to explain, making it an accessible choice for various applications [42].

RF is a supervised machine learning classifier that builds a collection of decision trees and introduces randomness. It applies to regression and classification problems, demonstrating promising results in text classification. RF’s ability to mitigate overfitting and handle diverse predictor variables makes a valuable classification algorithm [4, 70].

B.1.2 Data Imbalance

Class distribution, defined as the number of examples of each class, plays a crucial role in classification. An uneven class distribution, known as imbalanced data, poses a challenge in machine learning. The imbalance can result in algorithms ignoring the minority class (with fewer examples), leading to poor performance. One strategy to address this issue is oversampling the minority classes.

One prominent oversampling technique is SMOTE, which generates artificial minority samples based on randomly selected ones and their k-nearest neighbors. The SVM SMOTE is an extension of SMOTE designed to balance datasets. The SVM classifier is trained on the original training set in SVM SMOTE. New examples are randomly generated by connecting support vectors from all minority classes. The critical distinction between SMOTE and SVM SMOTE lies in the algorithm used to create synthetic examples: k-nearest neighbors in SMOTE and support vectors in SVM SMOTE [28].

B.1.3 Clustering

Review clustering is achieved by applying DBSCAN, a clustering algorithm that identifies clusters based on areas of high element density, assigning elements in low-density regions to singleton clusters (clusters composed only of a single element) [19].

Unlike some clustering methods, DBSCAN does not require the a-priori definition of the number of clusters to extract. However, it necessitates the specification of two parameters:

- minPts: The minimum threshold of points needed to constitute a dense region;
- Maximum distance (Eps): The maximum distance between two points for them to be deemed part of the same dense region (cluster).

Typically, these parameters must be known for each cluster, but it is challenging to get them in advance. DBSCAN uses a heuristic to estimate these parameters globally.

The algorithm starts with an arbitrary point and finds all points within a certain distance Eps from it. This set of points forms a cluster if the central point is a core point (has enough neighbors). If the central point is a border point (not enough neighbors), the algorithm moves to the next point in the database. Since global values for Eps and minPts are used, two clusters might merge if they are close to each other. This merging is based on the distance between the two clusters. If the distance is more significant than Eps, they remain separate. Recursive calls of DBSCAN may be necessary for clusters detected with a higher value for minPts. This recursive application enhances the algorithm's efficiency.

A heuristic is used to determine Eps and minPts for the “thinnest” cluster in the database. The idea is based on the observation that the distance of a point to its k^{th} nearest neighbor gives insights into the density distribution. The algorithm creates a sorted k-dist graph to help visualize the density distribution. The user can interactively choose a threshold point on this graph, and the corresponding k-dist value becomes the Eps value for DBSCAN.

DBSCAN offers a simple yet effective way to find clusters and noise in a spatial database. It uses a density-based approach, making it more flexible and robust. The algorithm's average run time complexity is $O(n * \log_n)$, making it efficient for practical use.

Similar to N-gram extraction, certain preprocessing steps should not be performed with clustering. For instance, synonym unifying should be avoided. Conducting this process before clustering might be counterproductive since, taking Villarroel et al.'s example [70], “*a review containing “freezes” and a review containing “crash” could indicate two different bugs.*”

B.1.4 Training and Evaluation

In the dynamic realm of text classification, robust evaluation methods and comprehensive training procedures are pivotal for ensuring the effectiveness of classifiers.

K-fold cross-validation is a cornerstone in this process, dividing the data into K subsets to iteratively train and evaluate the classifier's performance. The first group is selected as the testing data, and the other K-1 groups are selected as the training data. The process is repeated K times with a different group chosen as the testing data for each repetition [44].

The triad of evaluation metrics—Precision, Recall, and F-measure—serves as the compass guiding the classifier's performance assessment.

Recall, a measure of coverage, gauges the ratio of correctly classified instances to the total instances belonging to a specific label. Precision, representing accuracy, calculates the ratio of correctly classified instances to the total classified instances. The F-measure, a harmonic blend of Precision and Recall, encapsulates the balance between these two crucial metrics [74].

They can be calculated through the following formulas [12]:

$$R = \frac{TP}{TP + FN} \quad (B.1) \quad P = \frac{TP}{TP + FP} \quad (B.2)$$

$$F - measure = \frac{2 * P + R}{P + R} \quad (B.3)$$

Formulas such as those for Recall (R), Precision (P), and F-measure guide the quantitative evaluation process, relying on metrics derived from True Positives (TP), False Positives (FP), and False Negatives (FN). These metrics offer a nuanced understanding of how well the classifier distinguishes between different types, providing insights into potential misclassifications and their consequences.

Giving the example of the classification of user reviews [42]:

- TP is the number of reviews classified as a type i and are that type i;
- True Negative (TN) is the number of reviews classified as not being type i and are not that type i;
- FP is the number of reviews classified as a type i but actually from a different type, j;
- FN is the number of reviews classified as a type j but actually from type i.

The ROC curve is a graphical tool for assessing the performance of binary diagnostic classification methods. It illustrates the trade-off between “1 – specificity (FP rate)” on the x-axis and “sensitivity (TP rate)” on the y-axis across different cut-off values.

The curve’s shape is influenced by adjusting the criterion for positivity, impacting the balance between TP and FP rates. An implicit “separator” variable represents this decision criterion, often called a “latent” or unobservable variable [46].

The ROC curve has notable advantages, offering a comprehensive visualization for distinguishing normal from abnormal results across the entire spectrum of test outcomes. It eliminates the need for data grouping and remains unaffected by disease prevalence. Sensitivity, also known as the TP rate, measures the proportion of actual positives correctly identified by the diagnostic test.

However, drawbacks include the absence of a direct display for the cut-off value and sample count. The area under the curve (AUC) is employed to address these limitations. The AUC represents the entire range of ROC curve performance. It serves as a combined measure of sensitivity and specificity, providing a single value to assess the inherent validity of diagnostic tests [26]. An AUC of 1 indicates perfect discrimination, while 0.5 suggests no discriminative ability beyond chance. An example of a ROC curve can be seen in Figure B.1.

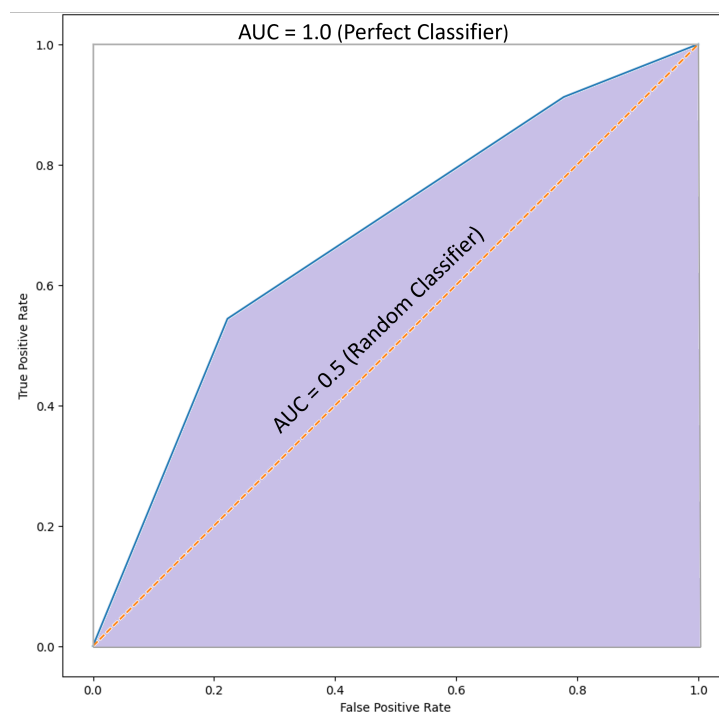


Figure B.1: ROC curve example, showcasing different AUCs.

B.2 NLP Techniques

B.2.1 Stemming

Stemming algorithms are pivotal in NLP and information retrieval by simplifying words and grouping terms with similar meanings. Among the widely used algorithms, the Porter Stemmer, developed by Martin Porter [55], stands out for its effectiveness and simplicity.

The Porter Stemmer operates through a series of rules applied to words, focusing on removing common suffixes. The algorithm progresses through several steps:

- Step 1a: Suffix Removal
 - Remove “sses” and replace with “ss” (e.g., “caresses” becomes “caress”);
 - Remove “ies” and replace with “i” (e.g., “ponies” becomes “poni”);
 - Remove “ss” (e.g., “caress” remains “caress”);
 - Remove “s” (e.g., “cats” becomes “cat”).
- Step 1b: Further Suffix Removal
 - Remove “eed” and replace with “ee” if the word has a measure greater than 0 (e.g., “agreed” becomes “agree”);
 - Remove “ed” or “ing” if the word contains a vowel:

- * If ending in “ed,” replace with nothing (e.g., “plastered” becomes “plaster”);
 - * If ending in “ing,” replace with nothing (e.g., “motoring” becomes “motor”);
 - * For specific cases (ending in “at,” “bl,” or “iz”), add “e” (e.g., “conflate” becomes “conflate”).
- Step 1c: Suffix Removal
 - Replace “y” with “i” if the word contains a vowel (e.g., “happy” becomes “happi”).
 - Step 2-5: Further Suffix Removal
 - Apply various rules to remove specific suffixes based on the word’s measure.

These rules prioritize the removal of suffixes based on the longest match, maintaining the integrity of stems while reducing words to their essential forms.

Developed later by Martin Porter [56], the Snowball Stemmer extends the Porter Stemmer to address the needs of languages beyond English. It introduces a formal language for expressing stemming algorithms, enhancing implementation in various programming languages like C or Java.

The Snowball Stemmer minimizes stemming errors, including understemming, over-stemming, and mis-stemming. Despite the potential inferiority to dictionary-based stemmers, its speed and resource efficiency advantages make it valuable for large-scale information retrieval systems.

It provides algorithmic stemmers, which offer speed and resource efficiency advantages despite being potentially inferior to well-constructed dictionary-based stemmers. The simplicity and effectiveness of algorithmic stemmers make them valuable for large-scale information retrieval systems. Stemming is an integral part of the information retrieval system’s process, involving extracting words from text for indexing. Handling irregular grammatical forms and stopwords is crucial for an effective stemming process. This process involves addressing irregular grammatical forms and handling stopwords. While dictionary-based stemmers may offer completeness, algorithmic stemmers like Snowball prioritize efficiency, occasionally accepting the presence of rare forms without undue concern.

B.2.2 Sentiment

Sentiment analysis, a crucial task in natural language processing, involves determining the overall sentiment polarity of a text and its strength, often expressed as a numerical rating [66]. One of the most prominent algorithms for conducting sentiment analysis is SentiStrength [27, 40, 42].

SentiStrength, tailored for social web texts, employs a lexical approach to simultaneously predict the strength of positive and negative sentiments. It assigns a positive sentiment strength ranging from 1 (indicating no positive sentiment) to 5 (representing powerful positive sentiment) and a negative sentiment strength ranging from -1 (indicating no negative sentiment) to -5 (representing powerful negative sentiment) to each text (zeros are not used) [67].

The algorithm operates based on a lexicon of 2,608 terms and term stems. Human judges assigned each term a positive or negative score, reflecting its typical sentiment in social web text [13].

For instance, “good” may score +3 on the positive scale, while “scare” scores -4 (matching terms like “scared”). Consequently, a sentence like “I was scared, but it was good” might receive a +3 on the positive scale and a -4 on the negative scale. SentiStrength handles cases with multiple sentiment words by selecting the most decisive sentiment word. If no sentiment words are present, a neutral sentiment is assumed. Special rules address negations, questions, booster words (e.g., “very”), emoticons, and other exceptional cases. The positive number can be added to the negative number to obtain a single overall number for sentiment strength, resulting in a score of -4 to 4. SentiStrength operates in two modes: supervised and unsupervised. In unsupervised mode, it utilizes pre-defined sentiment strength weights for the lexicon. In supervised mode, a training set of data adjusts the lexicon term weights to enhance accuracy. Both modes have demonstrated similar levels of accuracy across various social web texts [67].

Despite its strengths, SentiStrength has limitations. It may be misled by figurative language such as sarcasm and irony, words with context-dependent meanings (e.g., “wicked”), and minority meanings of sentiment words (e.g., “fat” used as an insult) [13].

B.2.3 N-Gram Extraction

Including N-grams as a fundamental text feature in the analysis process provides a powerful mechanism for capturing the contextual information embedded in sentences. N-grams, in essence, group together a continuous sequence of N tokens, where tokens can be words or numbers. This widely employed technique proves invaluable in understanding the nuanced structures and patterns within textual data.

However, it is imperative to exercise caution in preprocessing N-grams. Specific techniques, such as stop word removal, stemming, unifying synonyms, and negation removal, may not be suitable for N-grams. Implementing these steps risks the inadvertent loss of vital information encapsulated in these grouped sequences. Moreover, some preprocessing steps, like negation removal, may not be necessary when dealing with N-grams, potentially leading to enhanced performance by omitting unnecessary transformations.

Consider the example given by Villarroel et al. [70], “*n-grams extracted from a review like “I love it, it runs smooth no lags or glitches” will include no lags, no lags or glitches, etc.*” Retaining these variations without preprocessing ensures a more accurate representation of the original sentiment. Additionally, refraining from merging synonyms with N-grams is a deliberate choice to preserve their nuanced meaning and avoid unintended alterations.

B.2.4 BoW

The concept of “Bag-of-Words (BoW)” encapsulates the idea of representing text as an unordered collection of words [74]. In the BoW approach, a classifier constructs a dictionary that encom-

passes all terms present in the entire corpus of reviews. It then evaluates the presence and frequency of each term in reviews of a specific type. Through supervised machine learning, algorithms can be trained using a set of reviews (training set) to distinguish review types based on the occurrence and frequency of terms. Common terms like “app,” “version,” or “use” may appear more frequently across all reviews.

An advantageous feature of BoW is its independence from manually curated keyword lists for each review type. Furthermore, the classifier can exploit patterns of keyword co-occurrences to predict review types. Beyond the review text, this technique can be extended by incorporating other machine learning features [42].

A widely used BoW model in natural language processing is CountVectorizer. It is a feature extraction method in scikit-learn that transforms text documents into a matrix of token counts. Each document is represented as a vector, where each element corresponds to the count of a specific word in the document. It converts a collection of text documents into a matrix of word counts, providing a foundation for various machine-learning tasks [54].

B.2.5 TF-IDF

TF-IDF uses the BoW model as a foundation and enhances it by adding a weighting mechanism to account for the distribution of terms across the corpus. The weight is computed as the product of two terms. The first term calculates the normalized TF, representing how often a word appears in a document. The second term is the Inverse Document Frequency (IDF), assessing the importance of a term. Due to variations in document lengths, a term might appear more frequently in a longer document. To address this, TF is often divided by the total number of words in the document for normalization. While computing TF, all terms are initially considered equally important. However, common terms like “is,” “of,” and “the” may appear frequently but carry little significance. Therefore, the approach involves down-weighting frequent terms while amplifying rare ones to enhance the representation’s informativeness [15].

TfidfVectorizer is a popular application of this model. It is a feature extraction method in scikit-learn, seamlessly integrates the TF-IDF and vectorization processes. Handling a collection of text documents as numerical vectors with TF-IDF scores transforms them into a matrix of TF-IDF features. This transformer is fundamental for converting raw document collections into matrices of TF-IDF features.

B.2.6 Word and Sentence Embedding

Two prominent algorithms are widely employed for sentence embedding: SBERT and USE [57].

SBERT is an adaptation of the BERT network¹. SBERT incorporates siamese and triplet networks during fine-tuning. Siamese networks enable the comparison of sentence pairs, optimizing

¹a pre-trained transformer network known for achieving state-of-the-art results in various NLP tasks, including question answering, sentence classification, and sentence-pair regression

weights to make embeddings semantically meaningful. Triplet networks extend this idea by considering a reference sentence and a positive and negative example, further refining the embeddings. A pooling operation is added to the BERT output to obtain a fixed-sized sentence embedding. The resulting SBERT model captures nuanced semantic relationships between sentences, making it valuable for information retrieval and document similarity analysis tasks. The specific network structure adapts based on the characteristics of the available training data [58].

The USE is a type of pre-trained NLP model developed by Google. It is designed to generate fixed-size sentence embeddings that capture semantic information about the input sentences. Sentence embeddings are vector representations of sentences in a semantic space, where the distance between vectors reflects the semantic similarity between sentences. The model is pre-trained on a large corpus of text data, allowing it to learn contextualized representations of sentences without the need for task-specific training data [11].

In word embedding, two prevalent algorithms emerge: Word2Vec and FastText.

Word2Vec, a model proposed by Google, is a shallow neural network² that operates on text data. It consists of two learning models: Continuous Bag of Words (CBOW) and Skip-gram.

CBOW predicts a word given its context, while Skip-gram predicts the context given a word. By feeding the text corpus into either learning model, Word2Vec generates word vectors. This involves building a vocabulary from the training text corpus and learning vector representations for each word. Additionally, Word2Vec can calculate cosine distances between words. This feature enables the clustering of similar words based on their distances, projecting the original feature dimension to a new, lower dimension [39].

`fastText` is a library developed by **Facebook** AI Research for constructing text representations and performing text classification. Unlike traditional word embeddings, `fastText` represents words as sets of n -gram characters. Each word is augmented with symbols `<` and `>` at its beginning and end, respectively, to differentiate prefixes and suffixes from other characters. Take into consideration the example given by Wibowo and Musdholifah [72], “*given the word “virus,” with $n=3$, then the n -gram character set of the word is: `<vi, vir, iru, rus, us>` and a special sub word is `<virus>`.*”

An innovative aspect of `fastText` is its consideration of subwords. The vector representation for an individual word is obtained by summing the vectors of all its subwords. This allows `fastText` to generate vectors for complex words, even those absent from the dictionary, by leveraging the subword structure of related words. Similar to Word2vec, `fastText` offers two architectures: skip-gram and CBoW [72].

The distinction lies in `fastText`’s use of the entire text as features to predict the category of the text [39].

Crucially, before building word embeddings with `fastText`, one must specify the architecture type, minimum count (minimum frequency of occurrence), and vector dimension [72].

²Unlike deep neural networks, which have multiple hidden layers, shallow neural networks only have one or two hidden layers between the input and output [65].

B.2.7 Topic Modeling

Topic modeling is a statistical machine learning model utilized to uncover abstract topics within a collection of documents, playing a pivotal role in text mining by revealing hidden semantic relations in text documents related to a specific domain [4].

LDA is a well-established method for identifying common topics in natural language text documents. Topics are formed through combinations of words that frequently co-occur in the corpus's documents. LDA assumes a fixed number of K topics and assigns a probability distribution over topics to each document. Determining the optimal number of topics, K is guided by perplexity (indicating the log-likelihood of a held-out test set) and the intuitive meaningfulness of results. Each topic comprises a set or cluster of “w” words describing the topic [47]. Using Nayebi et al.'s example [48], “[...] if we decided to describe each topic by three words ($w = 3$), then LDA would group “gas,” “traffic,” and “map” together under one topic, and “accommodation,” “room,” and “hotel” under another topic.”

A generative process is applied to generate topic and term distributions, employing methods like Gibbs Sampling or Collapsed Variation Bayes. A limitation of LDA is the necessity to pre-define the number of topics, as LDA makes no assumptions about the topic count per document [44].

B.2.8 Summarization

In the realm of vast textual data, summarization emerges as a pivotal technique to distill essential information and unveil insights.

Several approaches are employed in the art of summarization, each tailored to address specific challenges presented by diverse types of text. One of these approaches is SumBasic, which employs the average TF of words in tweets to assess their value. However, the weight of individual words is adjusted after selecting a tweet to reduce redundancy. The process involves:

- Calculating the probability of a word w_i in the input corpus of size N words as $\rho(w_i) = \frac{f(w_i)}{N}$, where $f(w_i)$ is the frequency of the word in the entire corpus;
- Determining the weight of a tweet with n words as the average probability of its words: $\frac{1}{n} \sum_{i=1}^n \rho(w_i)$;
- Selecting the highest-scoring tweet. For each word in the chosen tweet, its probability is reduced by $\rho(w_i)_{new} = \rho(w_i) \times \rho(w_i)$ to control redundancy or minimize the chances of selecting tweets describing the same topic with high-frequency words;
- Repeating the process from the second point until the required summary length is achieved [74].

Two other prominent algorithms, TextRank and LexRank, draw inspiration from Google's PageRank algorithm, measuring web page importance for search engine result ranking. These algorithms build a graph out of webpages, assigning directed edges between vertices x and y if x

has a link to y . Edge weights are determined by the importance of the source vertex x , with links from more important webpages carrying more weight. Initial importance is standardized across webpages and is iteratively updated to reflect the true importance given the existing links. At equilibrium, webpages are ranked based on the weight directed towards them [61].

TextRank directly applies PageRank to textual content, creating a sentence graph where each vertex represents a sentence. Edges are weighted by the normalized size of the sentence overlap, allowing for a fully connected graph with varied edge weights. Utilizing PageRank on this graph helps identify the top-ranked sentences. An advantage of TextRank is its applicability without labeled data, enhancing transferability and reproducibility [36].

However, TextRank has notable weaknesses. It overlooks extraneous words and lacks consideration for word rarity, treating all words equally [61].

LexRank improves upon TextRank, offering refined definitions for word and sentence similarity, introducing a minimum similarity threshold for graph edge creation. LexRank uses TF-IDF for data encoding, so the fewer documents a term appears in, the rarer it is. This addresses TextRank's word rarity issue. Nevertheless, LexRank requires a collection of domain-specific documents to build IDF counts. Furthermore, since word frequencies change across domains, this collection must match the domain as the summarized documents [61].

The critical difference lies in the similarity metric, with LexRank using cosine similarity instead of TextRank's simple matching count. This adjustment normalizes sentence length differences, offering a more nuanced similarity measure. The graph generation represents each sentence as a vertex, creating undirected edges with weights equal to the similarity score [61].

A challenge with these techniques is their limited consideration of topic coverage. They often prioritize "representative" sentences without ensuring comprehensive coverage of various topics within the summary [36].

A practical method for assessing the performance of these algorithms involves leveraging ROUGE³, a comprehensive toolkit comprising metrics and software widely employed to evaluate automatic summarization and machine translation in NLP. This toolkit facilitates the comparison of automatically generated summaries or translations with their human-produced counterparts. A notable metric within the ROUGE toolkit is ROUGE-N, which specifically focuses on assessing the matching N-grams in the comparative analysis. It calculates conventional machine learning metrics, including recall, precision, and f-measure.

³<https://huggingface.co/spaces/evaluate-metric/rouge>

Appendix C

User Stories

- **User Story (US) 1** As a user, I want to navigate the platform very easily and use it efficiently without frustration.
- **US 2** As a user, I want only to see relevant posts and view their information in a digestible format.
 - **US 2.1** As a user, I want to read a summary of every post to understand its content quickly.
 - **US 2.2** As a user, I want to read the full text of a post to get all the details.
 - **US 2.3** As a user, I want to see the rating of posts to identify the most valuable ones.
 - **US 2.4** As a user, I want to see the original post on the social media platform from which it was extracted to verify its context.
 - **US 2.5** As a user, I want to see the sentiment analysis of each post so that I can understand the general tone.
 - **US 2.6** As a user, I want to receive suggestions about which category each post belongs to (Bug, Feature, User Experience) to classify them accurately.
 - **US 2.7** As a user, I want to know from which social media each post comes to understand its source.
 - **US 2.8** As a user, I want to know which streaming service each post refers to relate it to my interests.
 - **US 2.9** As a user, I want to know how many people agreed with each post by either liking it or reposting it.
 - **US 2.10** As a user, I want to read posts that are direct replies to another post, to get further information.
 - **US 2.11** As a user, I want to know when a post was originally published to know the age of an issue or desire of a user.

- **US 3** As a user, I want to see all posts organized in clusters and view them together to choose my preferred view.
- **US 4** As a user, I want to be able to order posts by a specific criterion so that I can prioritize my review.
- **US 5** As a user, I want to filter the posts I see to focus on those that most matter to me.
- **US 6** As a user, I want to be able to search for specific posts using keywords.
- **US 7** As a user, I want to be able to either approve or reject feedback due to its relevance and help the platform learn from my feedback.
- **US 8** As a user, I want to be able to give feedback about the classification of a post as *Bug*, *Feature*, or *User Experience* to help improve the categorization.
- **US 9** As a user, I want to see posts divided into different pages based on their stage to manage them more effectively.
- **US 10** As a user, I want to be able to reconsider and change the classifications I have given to posts (relevant or not) so that I can correct any mistakes.
- **US 11** As a user, I want to create issues using the posts on the "Acceptance Feedback" page to address feedback efficiently.
 - **US 11.1** As a user, I want to create issues using all the posts in a cluster to handle related feedback together.
 - **US 11.2** As a user, I want to create issues using more than one post, even if they do not belong to the same cluster, to address broader issues.
 - **US 11.3** As a user, I want the platform to suggest what should be in each field of the issue form on GitLab so that I can create issues more efficiently.
 - **US 11.4** As a user, I want to choose between creating an issue remotely with GitLab or exporting the issue to a CSV file so I can manage my workflow flexibly.
- **US 12** As a user, I want to see an archive of posts already turned into issues to track past actions.
- **US 13** As a user, I want to be able to go directly to the issue page on GitLab if the issue was created using the application so that I can follow up easily.

Appendix D

Database Structure

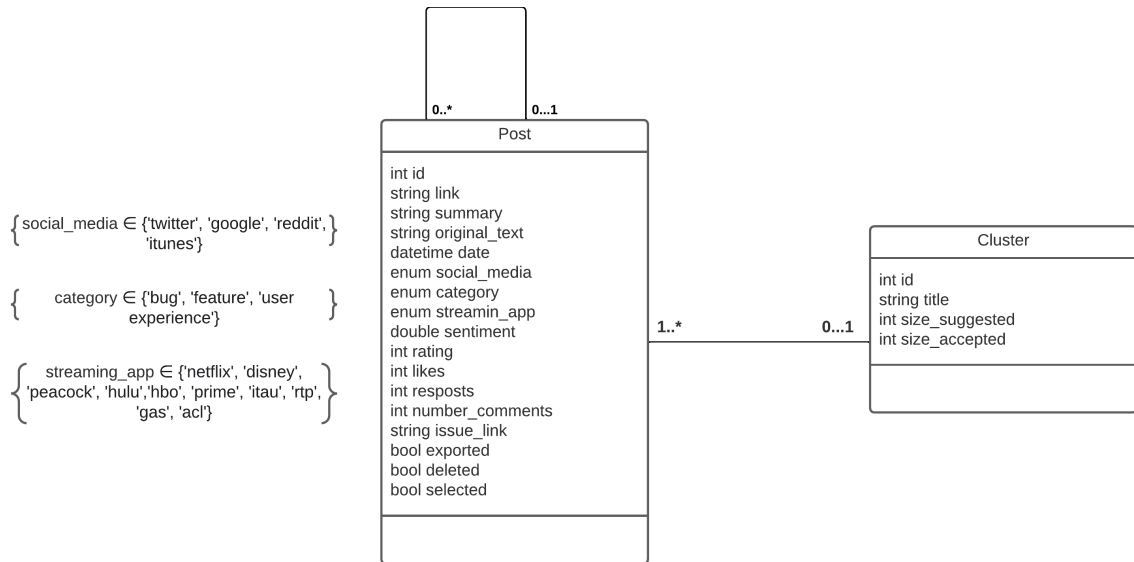


Figure D.1: Database Schema in UML.

The database consists of two tables: Post and Cluster.

The Post table includes all the information about each post. It contains boolean attributes that determine which page the posts should appear on `exported` (**Exported Feedback Page**), `selected` (**Selected Feedback Page**), and `deleted` (**Deleted Feedback Page**). Additionally, there is an attribute for an issue link (`issue_link`) in case the post was exported and used to create an issue on GitLab. Posts can be connected to other posts as comments and associated with a cluster. Each post can belong to only one cluster, but a cluster can contain one or many posts. A cluster must have at least one post to exist.

The Cluster table has attributes, including a title, the number of posts in the Accepted Feedback page, and the number of posts in the Suggested Feedback page. These attributes help manage the size of the clusters on different pages and check if more posts need to be loaded from the database.

Appendix E

Issue Templates

E.1 Bug Report Template

Title

Title of the Issue

Posts Describing the Needs

Post or List of Posts

Expected behavior

A clear and concise description of what you expect to happen.

E.2 Feature Request/User Experience Template

Title

Title of the Issue

User Stories

For:

How:

I want:

Posts Describing the Needs

Post or List of Posts

Acceptance Criteria (done when...)

Tip: Write a **checkboxed** list of sentences in the present tense describing the desired state of the system.

GOOD (clear how to verify, it reports design and interaction decisions agreed):

- The page shows a button only to logged in users
- The button has the style XXX and label 'Logout'
- When pressed, opens a confirmation modal with 'OK' and 'Cancel' buttons
- ...

BAD (cannot be activated during verification, too arbitrary):

- Logout button
- Modal for confirmation

Validation Scenarios

Hint: Use Gherkin syntax to lay them out. It is complete and explicit, human-readable but easy to convert into self-executable tests.

OK: ‘Administrator logs into back-office‘

BAD: ‘Backoffice login‘

Tasks

This section is populated by the developer and reports the expected practical activities to meet the requirements. Useful for estimating and for day-by-day monitoring. Better if it reports checkboxes to track completion.

- Task 1
- Task 2
- ...

Appendix F

Automated Execution Script

The entire processing workflow can be automated and executed sequentially using a shell script, `run_workflow.sh` (Listing F.1). This script orchestrates the following sequence of tasks:

1. **Data Extraction:** Executes Python scripts (`google_play.py`, `itunes.py`, `reddit.py`, `twitter.py`, and `twitter_replies.py`) to gather user feedback from social media platforms;
2. **Preprocessing:** Runs the preprocessing steps as detailed in `preprocessing.ipynb`;
3. **Training and Evaluating Classification Models:** Uses pre-trained classification models (as detailed in Figure 4.7) to classify user feedback data, controlled by `classification.ipynb`;
4. **Summarization:** Utilizes the ChatGPT API through `summarization.py` to generate summaries of user posts;
5. **Clustering:** Applies the clustering algorithm (discussed in Chapter 4.5) to group posts and generate cluster titles using the ChatGPT API through `clustering.py`.

This script ensures the entire workflow is executed in a consistent and automated manner, validating the automated nature of our social media feedback processing workflow.

```
1 # Define paths to Python scripts and Jupyter notebooks
2 DATA_EXTRACTION_SCRIPTS=("organized/data_extraction/google_play.py" "organized/
   data_extraction/itunes.py" "organized/data_extraction/reddit.py" "organized/
   data_extraction/twitter.py" "organized/data_extraction/twitter_replies.py")
3 PREPROCESSING_NOTEBOOK="organized/data_processing/preprocessing.ipynb"
4 CLASSIFICATION_NOTEBOOK="organized/classification/classification.ipynb"
5 SUMMARIZATION_SCRIPT="organized/summarization/summarization.py"
6 CLUSTERING_SCRIPT="organized/clustering/clustering.py"
7
8 # Step 1: Data Extraction
9 echo "Step 1: Running data extraction scripts..."
```

```
10 for script in "${DATA_EXTRACTION_SCRIPTS[@]}"
11 do
12     echo "Executing $script..."
13     python "$script"
14     if [ $? -ne 0 ]; then
15         echo "Error: Failed to execute $script. Exiting."
16         exit 1
17     fi
18 done
19 echo "Data extraction completed."
20
21 # Step 2: Preprocessing
22 echo "Step 2: Running preprocessing notebook..."
23 jupyter nbconvert --execute "$PREPROCESSING_NOTEBOOK"
24 if [ $? -ne 0 ]; then
25     echo "Error: Failed to execute preprocessing notebook. Exiting."
26     exit 1
27 fi
28 echo "Preprocessing completed."
29
30 # Step 3: Classification
31 echo "Step 3: Running classification notebook..."
32 jupyter nbconvert --execute "$CLASSIFICATION_NOTEBOOK"
33 if [ $? -ne 0 ]; then
34     echo "Error: Failed to execute classification notebook. Exiting."
35     exit 1
36 fi
37 echo "Classification completed."
38
39 # Step 4: Summarization
40 echo "Step 4: Running summarization script..."
41 python "$SUMMARIZATION_SCRIPT"
42 if [ $? -ne 0 ]; then
43     echo "Error: Failed to execute summarization script. Exiting."
44     exit 1
45 fi
46 echo "Summarization completed."
47
48 # Step 5: Clustering
49 echo "Step 5: Running clustering script..."
50 python "$CLUSTERING_SCRIPT"
51 if [ $? -ne 0 ]; then
52     echo "Error: Failed to execute clustering script. Exiting."
53     exit 1
54 fi
55 echo "Clustering completed."
56
57 echo "Workflow execution finished."
```

Listing F.1: Shell script for automated workflow (run_workflow.sh)

References

- [1] A. Al-Hunaiyyan, A. Alzayed, R. Alhajri, and A. Khalfan. Using social networking sites for requirements elicitation: Perspectives and challenges. *International Journal of Intelligent Systems and Applications in Engineering*, 11(4s):357–368, 2023.
- [2] Ali and Hong. Value-oriented requirements: Eliciting domain requirements from social network services to evolve software product lines. *Applied Sciences*, 9(19), 2019.
- [3] N. Ali and J. E. Hong. A bird’s eye view on social network sites and requirements engineering. In *ICSOF 2019 - Proceedings of the 14th International Conference on Software Technologies*, pages 347–354.
- [4] N. Ali, S. Hwang, and J. E. Hong. Your opinions let us know: Mining social network sites to evolve software product lines. *KSII Transactions on Internet and Information Systems*, 13(8):4191–4211, 2019.
- [5] N. Ali, S. Kim, and J. E. Hong. Listen closely, respond quickly: Enhancing conformity of spl domain requirements through sns. In *2016 International Conference on Information Science and Communications Technologies, ICISCT 2016*.
- [6] Nazakat Ali, Jang-Eui Hong, and Lawrence Chung. Social network sites and requirements engineering: A systematic literature review. *Journal of Software: Evolution and Process*, 33(4), 2020.
- [7] Javed Ali Khan, Lin Liu, and Lijie Wen. Requirements knowledge acquisition from online user forums. *IET Software*, 14(3):242–253, 2020.
- [8] Aaron Bangor, Philip T. Kortum, and James T. Miller. Determining what individual sus scores mean: adding an adjective rating scale. *Journal of Usability Studies archive*, 4:114–123, 2009.
- [9] P. Bourque and R. Fairley. *Guide to the Software Engineering Body of Knowledge, Version 3.0 (SWEBOK Guide V3.0)*. IEEE Computer Society, 2014.
- [10] J. B. Brooke. Sus: A ‘quick and dirty’ usability scale. 1996.
- [11] D. Cer, Y. Yang, S. Kong, N. Hua, N. Limtiaco, S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, Y. Sung, B. Strope, and R. Kurzweil. Universal sentence encoder, 2018.
- [12] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang. Ar-miner: Mining informative reviews for developers from mobile app marketplace. In *Proceedings - International Conference on Software Engineering*, pages 767–778.

- [13] J. Culpeper, A. Findlay, B. Cortese, and M. Thelwall. Measuring emotional temperatures in shakespeare's drama. *English Text Construction*, 11(1):10–37, 2018.
- [14] G. Deshpande and J. Rokne. User feedback from tweets vs app store reviews: An exploratory study of frequency, timing and content. In *Proceedings - 2018 5th International Workshop on Artificial Intelligence for Requirements Engineering, AIRE 2018*, pages 15–21.
- [15] V. T. Dhinakaran, R. Pulle, N. Ajmeri, and P. K. Murukannaiah. App review analysis via active learning: Reducing supervision effort without compromising classification accuracy. In *Proceedings - 2018 IEEE 26th International Requirements Engineering Conference, RE 2018*, pages 170–181.
- [16] A. Di Sorbo, S. Panichella, C. V. Alexandru, C. A. Visaggio, and G. Canfora. Surf: Summarizer of user reviews feedback. In *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017*, pages 55–58.
- [17] R. dos Santos, K. Villela, D. T. Avila, and L. H. Thom. A practical user feedback classifier for software quality characteristics. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, volume 2021-July, pages 340–345.
- [18] A. M. Ebrahimi and A. A. Barforoush. Preprocessing role in analyzing tweets towards requirement engineering. In *ICEE 2019 - 27th Iranian Conference on Electrical Engineering*, pages 1905–1911.
- [19] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings - 2nd International Conference on Knowledge Discovery and Data Mining, KDD 1996*, pages 226–231.
- [20] M. Glinz. A glossary of requirements engineering terminology, 2018.
- [21] Standish Group. The standish group report. Technical report, 2014.
- [22] E. Guzman, R. Alkadhi, and N. Seyff. An exploratory study of twitter messages about software applications. *Requirements Engineering*, 22(3):387–412, 2017.
- [23] E. Guzman, M. El-Haliby, and B. Bruegge. Ensemble methods for app review classification: An approach for software evolution. In *Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015*, pages 771–776.
- [24] E. Guzman, M. Ibrahim, and M. Glinz. A little bird told me: Mining tweets for requirements and software evolution. In *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017*, pages 11–20.
- [25] Emitza Guzman, Rana Alkadhi, and Norbert Seyff. A needle in a haystack: What do twitter users say about software? 2016.
- [26] K. Hajian-Tilaki. Receiver operating characteristic (roc) curve analysis for medical diagnostic test evaluation. *Caspian Journal of Internal Medicine*, 4(2):627–635, 2013.
- [27] P. R. Henao, J. Fischbach, D. Spies, J. Frattini, and A. Vogelsang. Transfer learning for mining feature requests and bug reports from tweets and app store reviews. In *Proceedings of the IEEE International Conference on Requirements Engineering*, volume 2021-September, pages 80–86.

- [28] T. Iqbal, M. Khan, K. Taveter, and N. Seyff. Mining reddit as a new source for software requirements. In *Proceedings of the IEEE International Conference on Requirements Engineering*, pages 128–138.
- [29] B. Jeong, J. Yoon, and J. M. Lee. Social media mining for product planning: A product opportunity mining approach based on topic modeling and sentiment analysis. *International Journal of Information Management*, 48:280–290, 2019.
- [30] T. Johann, C. Stanik, A. M. B. Alizadeh, and W. Maalej. Safe: A simple approach for feature extraction from app descriptions and app reviews. In *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017*, pages 21–30.
- [31] Daniel Jurafsky and James H. Martin. Appendix c: Wordnet: Word relations, senses, and disambiguation. In *Speech and Language Processing*. Prentice-Hall, 2018. Draft of September 19, 2018.
- [32] G. M. Kanchev and A. K. Chopra. Social media through the requirements lens: A case study of google maps. In *1st International Workshop on Crowd-Based Requirements Engineering, CrowdRE 2015 - Proceedings*, pages 7–12.
- [33] G. M. Kanchev, P. K. Murukannaiah, A. K. Chopra, and P. Sawyer. Canary: Extracting requirements-related information from online discussions. In *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017*, pages 31–40.
- [34] J. A. Khan. Mining requirements arguments from user forums. In *Proceedings of the IEEE International Conference on Requirements Engineering*, volume 2019-September, pages 440–445.
- [35] J. A. Khan, Y. Xie, L. Liu, and L. Wen. Analysis of requirements-related arguments in user forums. In *Proceedings of the IEEE International Conference on Requirements Engineering*, volume 2019-September, pages 63–74.
- [36] N. Khosla, A. Raghuvarshi, and S. Sax. Saltier: Summarization of articles using lexrank for texts via intermediate embedded representations, 2016.
- [37] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering – a systematic literature review. *Information and Software Technology*, 51(1):7–15, 2009. Special Section - Most Cited Articles in 2002 and Regular Research Papers.
- [38] C. Li, L. Huang, J. Ge, B. Luo, and V. Ng. Automatically classifying user requests in crowd-sourcing requirements engineering. *Journal of Systems and Software*, 138:108–123, 2018.
- [39] Long Ma and Yanqing Zhang. Using word2vec to process big text data. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2895–2897, 2015.
- [40] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik. On the automatic classification of app reviews. *Requirements Engineering*, 21(3):311–331, 2016.
- [41] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik. Are tweets useful in the bug fixing process? an empirical study on firefox and chrome. *Empirical Software Engineering*, 23:1704–1742, 2017.

- [42] W. Maalej and H. Nabil. Bug report feature request or simply praise on automatically classifying app reviews. 2015.
- [43] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, volume 2014-June, pages 55–60.
- [44] S. McIlroy, N. Ali, H. Khalid, and A. E. Hassan. Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. *Empirical Software Engineering*, 21(3):1067–1106, 2016.
- [45] I. Morales-Ramirez, D. Munante, F. Kifetew, A. Perini, A. Susi, and A. Siena. Exploiting user feedback in tool-supported multi-criteria requirements prioritization. In *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017*, pages 424–429.
- [46] F. S. Nahm. Receiver operating characteristic curve: overview and practical use for clinicians. *Korean Journal of Anesthesiology*, 75(1):25–36, 2022.
- [47] M. Nayebi, H. Cho, and G. Ruhe. App store mining is not enough for app improvement. *Empirical Software Engineering*, 23(5):2764–2794, 2018.
- [48] M. Nayebi, R. Quapp, G. Ruhe, M. Marbouti, and F. Maurer. Crowdsourced exploration of mobile app features: A case study of the fort mcmurray wildfire. In *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Society Track, ICSE-SEIS 2017*, pages 57–66.
- [49] A. Nazakat and J. Hong. Using social network service to determine the initial user requirements for small software businesses. *International Journal of Applied Business and Economic Research*, 2017.
- [50] M. Noaen and B. H. Far. The efficacy of using social media data for designing traffic management systems. In *Proceedings - 4th International Workshop on Crowd-Based Requirements Engineering, CrowdRE 2020*, pages 11–17.
- [51] E. Oehri and E. Guzman. Same same but different: Finding similar user feedback across multiple platforms and languages. In *Proceedings of the IEEE International Conference on Requirements Engineering*, volume 2020-August, pages 44–54.
- [52] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. 2015.
- [53] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. Gall. Ardoc: App reviews development oriented classifier. In *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, volume 13-18-November-2016, pages 1023–1027.
- [54] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [55] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
- [56] M. F. Porter. Snowball: A language for stemming algorithms, 2001.

- [57] S. Rahman, F. Ahmed, and M. Nayebi. Mining reddit data to elicit students' requirements during covid-19 pandemic.
- [58] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference*, pages 3982–3992. Export Date: 18 December 2023; Cited By: 3035.
- [59] J. Scanlan, K. de Salas, D. Lim, and E. Roehrer. Using social media to support requirements gathering when users are not available. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, pages 4227–4236.
- [60] Norbert Seyff, Irina Todoran, Kevin Caluser, Leif Singer, and Martin Glinz. Using popular social network sites to support requirements elicitation, prioritization and negotiation. *Journal of Internet Services and Applications*, 6(1), 2015.
- [61] Steven Shearing, Abigail Gertner, Benjamin Wellner, and Liz Merkhofer. Automated text summarization: A review and recommendations. Technical Report MITRE Technical Report, The MITRE Corporation, November 2020. Approved for Public Release; Distribution Unlimited. Public Release Case Number 20-3129.
- [62] C. Stanik, M. Haering, and W. Maalej. Classifying multilingual user feedback using traditional machine learning and deep learning. In *Proceedings - 2019 IEEE 27th International Requirements Engineering Conference Workshops, REW 2019*, pages 220–226.
- [63] C. Stanik and W. Maalej. Requirements intelligence with openreq analytics. In *Proceedings of the IEEE International Conference on Requirements Engineering*, volume 2019-September, pages 482–483.
- [64] Statista. Internet and social media users in the world 2023. Available at <https://www.statista.com/statistics/617136/digital-population-worldwide/>, 2023.
- [65] Guoying Sun, Yanan Cheng, Zhaoxin Zhang, Xiaojun Tong, and Tingting Chai. Text classification with improved word embedding and adaptive segmentation. *Expert Systems with Applications*, 238:121852, 2024.
- [66] M. Thelwall. Gender bias in sentiment analysis. *Online Information Review*, 42(1):45–57, 2018.
- [67] M. Thelwall and K. Buckley. Topic-based sentiment analysis for the social web: The role of mood and issue-related words. *Journal of the American Society for Information Science and Technology*, 64(8):1608–1617, 2013.
- [68] J. Tizard. Requirement mining in software product forums. In *Proceedings of the IEEE International Conference on Requirements Engineering*, volume 2019-September, pages 428–433.
- [69] James Tizard, Hechen Wang, Lydia Yohannes, and Kelly Blincoe. Can a conversation paint a picture? mining requirements in software forums. 2019.

- [70] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Release planning of mobile apps based on user reviews. In *Proceedings - International Conference on Software Engineering*, volume 14-22-May-2016, pages 14–24.
- [71] P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen. Mining user opinions in mobile app reviews: A keyword-based approach. In *Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015*, pages 749–759.
- [72] Dikih Arif Wibowo and Aina Musdholifah. Sentiments analysis of indonesian tweet about covid-19 vaccine using support vector machine and fasttext embedding. In *2021 4th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, pages 184–188, 2021.
- [73] G. Williams and A. Mahmoud. Analyzing, classifying, and interpreting emotions in software users’ tweets. In *Proceedings - 2017 IEEE/ACM 2nd International Workshop on Emotion Awareness in Software Engineering, SEmotion 2017*, pages 2–7.
- [74] G. Williams and A. Mahmoud. Mining twitter feeds for software user requirements. In *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017*, pages 1–10.