Cyprus University of Technology

Faculty of Engineering and Technology

Thesis

Classification of Harmful and Normal client data in social networks with the Help of Machine Learning

Kostas Markou

Limassol, May 2023

Cyprus University of Technology

Faculty of Engineering and Technology

Department of Electrical Engineering
Computer Engineering and Informatics

Classification of Harmful and Normal client data in social networks with the Help of Machine Learning

Kostas Markou

Limassol, May 2023

Approval Form

Cyprus University of Technology

Presented by

Kostas Markou

Advisor _____

Dr. Michael Sirivianos

Committee member _____

Committee member _____

Cyprus University of Technology

Limassol, May 2023

Copyrights

Acknowledgements

I would like to express my gratitude to those who contributed to this work by sharing their valuable knowledge and ideas. More specifically, I want to thank Dr. Michael Sirivanos for supervising and guiding me through the whole thesis process, from finding a topic to completing it. I want to thank Pantelitsa Leonidou and Nikos Salamanos for introducing the Machine Learning technology to me and for all the great ideas on how to apply it in the context of my thesis and also for knowledge sharing and the technical support they provided. Lastly, I owe a lot to my family and friends who stood by me and encouraged me through the whole thesis process.

Abstract

Social networks have become an indispensable part of our daily lives, allowing us to connect with friends and family, share experiences and ideas, and stay informed about current events. However, the use of social networks also carries potential risks, particularly with the spread of harmful content such as hate speech, cyberbullying, and fake news. This has led to an increasing need for social network administrators to develop efficient methods for identifying and removing harmful content. Traditional methods for identifying and removing harmful content are often based on manual review and reporting by users, which can be time-consuming and unreliable. As social networks continue to grow in size and complexity, this approach is becoming increasingly challenging, and administrators are turning to machine learning techniques as a more effective solution. Machine learning algorithms can be trained to automatically analyze large amounts of data generated in social networks and identify patterns that distinguish harmful content from normal content. These algorithms can also adapt and evolve over time as new types of harmful content emerge, providing a more dynamic and responsive solution to the problem. This study aims to explore the application of machine learning techniques to classify harmful and normal client data in social networks. The focus will be on developing and evaluating models that can accurately distinguish harmful data from normal data. This will involve the use of various machine learning techniques, such as supervised and unsupervised learning, and feature engineering to extract relevant information from the data. Using unsupervised algorithms aimed at detecting Harmful and Normal clients on social networks, the outcomes of this study have the potential to contribute to the development of more effective solutions for identifying and removing harmful content in social networks. This could ultimately lead to a safer and more secure online environment for users and help to mitigate the negative effects of harmful content on individuals and society as a whole.

Table of Contents

List of Figures

List of Tables

1 Introduction

Social networks have become an integral part of our daily lives, providing a platform for individuals to connect and share information. However, this also makes them a prime target for malicious actors who can use these networks to spread harmful content, such as hate speech, misinformation, and cyberbullying. To combat this issue, researchers have begun to explore the use of machine learning techniques for the classification of harmful and normal client data in social networks.

1.1 Problem Statement&Motivation

1.1.1 Problem Statement

Social networks have become an integral part of our daily lives, with billions of users worldwide providing a platform for people to connect, share information, and express their opinions [22]. However, the vast amount of data generated in these networks can also attract malicious actors who spread harmful information or engage in harmful activities. This poses a significant challenge for social network administrators to maintain the safety and security of their users.

1.1.2 Motivation

The rapid growth of social networks has made it imperative for administrators to develop efficient methods for identifying and removing harmful content [46]. The traditional manual methods for this purpose are time-consuming, unreliable, and can result in a significant workload for administrators [40]. Machine learning techniques have emerged as a promising solution to this problem by automating the process of identifying harmful content. By leveraging the vast amounts of data generated by social networks, machine learning algorithms can be trained to identify harmful data and distinguish it from normal data [7].

Moreover, the ability of machine learning algorithms to continuously learn and adapt to new data patterns makes them an ideal solution for handling the dynamic nature of harmful content in social networks. With the increasing sophistication of malicious actors, it is important to develop advanced machine learning models that can accurately identify and classify harmful data [16].

This study aims to investigate the application of machine learning techniques in classifying harmful and normal client data in social networks. The focus will be on developing and evaluating models that can accurately distinguish harmful data from normal data [54]. The goal is to provide social network administrators with an effective and efficient solution to identify harmful content, thereby improving the safety and security of their users. The outcomes of this study have the potential to provide new insights into the classification of harmful data in social networks and contribute to the development of more effective solutions in the future [31].

1.2 Challenges

Despite the promising results of using Machine Learning in classifying harmful and normal client data in social networks, there are several challenges that need to be addressed.

1.2.1 Data Quality

Obtaining high-quality, annotated data for training and testing purposes is crucial in order to achieve accurate results in the classification of harmful and normal client data in social networks. The data collected from social media platforms can be highly biased and imbalanced, with one class of data (e.g. harmful) being significantly more represented than the other (e.g. normal). This can lead to a machine learning model that is biased towards the majority class and produces poor results when evaluating the minority class. To overcome this challenge, techniques such as oversampling the minority class, data augmentation, and using data from multiple sources can be used to obtain a more balanced dataset.

1.2.2 Uncertainty and Ambiguity

The interpretation of harmful and normal client data is subjective and open to interpretation, making it a challenging task to accurately classify data. This can lead to a significant number of false positive and false negative predictions, making it difficult to evaluate the performance of the machine learning model. To address this challenge, it is important to use a combination of human annotation and machine learning algorithms to ensure that the data is accurately and consistently annotated. Additionally, the use of interpretable machine learning models, such as decision trees and rule-based systems, can help to better understand the reasons behind the predictions and reduce the number of false positive and false negative predictions.

1.2.3 Cultural and Contextual Diversity

The cultural and contextual diversity of the data generated on social media platforms can greatly influence its interpretation and make it challenging for machine learning models to accurately classify harmful and normal client data. To overcome this challenge, it is important to consider the cultural and contextual context of the data when designing and training the machine learning model. This can be achieved by using a diverse dataset that includes data from multiple cultural contexts, and by using culturally-aware features, such as language models and sentiment analysis, in the machine learning model.

1.2.4 Adversarial Examples

Adversarial examples, or data that has been intentionally manipulated to evade detection, can be a major challenge when classifying harmful and normal client

data in social networks. Harmful users may use techniques such as creating misleading data or manipulating the inputs to the machine learning model in order to evade detection. To overcome this challenge, it is important to use robust machine learning models that are able to detect and mitigate the effects of adversarial examples. Additionally, techniques such as adversarial training, where the machine learning model is trained on adversarial examples, can also be used to improve its robustness against adversarial attacks.

In conclusion, while the classification of harmful and normal client data in social networks using machine learning is a promising solution, there are several challenges that must be carefully considered and addressed in order to ensure the accuracy and reliability of the results. By carefully addressing these challenges, it is possible to achieve accurate and robust results in the classification of harmful and normal client data in social networks.

1.3 Our Approach

Initially, the data is extracted from the dataset[19] which includes four distinct categories, namely abusive, hateful, spam, and normal. The category of spam is then excluded, and the categories of abusive and hateful are combined to generate a new category named harmful, thereby producing a binary classification challenge. Subsequently, the data is preprocessed using two distinct techniques. The first technique employed is the regular expression technique, while the second technique is a conventional method of the Python programming language that uses regular expressions. A detailed discussion of these techniques is provided in the Implementation subsection (Subsection 3.4). Furthermore, to achieve better analysis and visualization of the outcomes, the PCA technique [45] is employed to minimize the dimensions of the dataset at a later stage of research and implementation. Subsequently, the unsupervised algorithms are horizontally executed, and the relevant outcomes are generated. Finally, each algorithm's performance is experimentally assessed using common clustering quality metric methods[17].

1.4 Our Contribution

The main objective of this thesis is to conduct an experimental comparison of various unsupervised machine-learning algorithms for classifying tweets into two categories: harmful and normal. With the growing use of social media platforms such as Twitter, there is an increasing need to develop automated methods to identify harmful tweets, such as those containing hate speech, cyberbullying, and offensive language.

Our contribution to the field is twofold. Firstly, we provide a comprehensive evaluation of different unsupervised algorithms for the classification of harmful and normal tweets. We compare the performance of these algorithms in terms of accuracy, precision, recall, and F1-score. Secondly, we analyze the impact

of different feature extraction techniques and parameter settings on the performance of the classifiers. We also investigate the effect of the size of the training dataset on classification accuracy.

Overall, our study contributes to the development of effective machine-learning models for the identification of harmful tweets on social media platforms. The results of this study can be useful for researchers and practitioners working in the field of NLP, social media analytics, and online content moderation.

1.5 Organization

The remainder of the document is organized as follows. In Section 2 we present the background Theory and related work for both federated learning and unsupervised learning. In Section 3, we present first the algorithms used for the purpose of this work, the dataset, the technologies used, and the way above were implemented. In Section 4, we delineate the experiments that were conducted, accompanied by a comprehensive analysis of the results and findings. Section 5 consists of the future work of this work. We conclude in Section 6s

2 Background Theory and Related Work

2.1 Federated Learning

2.1.1 Introduction to Federated Learning

In their first formulation, McMahan et al. described federated learning as a distributed machine learning approach that permits training on a huge corpus of decentralized data that is held on gadgets like mobile phones. According to [37] the main goal of the authors of this study was to investigate a technical learning method that would have as its primary characteristic the conquest of a significant number of advantages from the common models that had been trained using some data, without saving them somewhere central.

Learning a model with little information transfer between clients and the curator is an exciting issue in federated optimization. Aside from being non-IID, imbalanced, and widely dispersed, consumer data can also be. For the purpose of addressing the aforementioned challenge, the "federated averaging" algorithm was developed by the authors of [37]. The fundamental tenet of the FL method is that a core model is trained through several iterations of dialogue between the curator and customers. Subsequently, the curator gives the current central model to a portion of customers throughout each communication round. The next step is local optimization by the clients. Clients may perform a number of mini-batch gradient descent steps in a single communication round to reduce communication. The curator then receives the improved models and combines them to assign a new central model. At the end of the process, training is either

ended or a new communication round begins depending on how well the new main model performs. Additionally, clients in federated learning never share data; instead, they only exchange model parameters [20].

As reported in [37] by McMahan et al, the optimal issues for federated learning must contain a few key characteristics. First of all, compared to training on proxy data that is often accessible in the data center, real-world data from mobile devices offers a definite benefit. Secondly, It is preferred not to log into the data center just for model training since this data is sensitive to privacy or essential, and third and last, labels on the data for supervised tasks can be readily deduced from user interaction.

2.1.2 The Lifecycle of a Model in Federated Learning

Initially, the procedure you adhere to in FL is controlled by a model created for a particular application. Figure 4 is presented the lifecycle of an FL-trained model and the multiple actors and components that influence an FL system.More specifically, according to [29] this procedure contains six notably steps :

1. ***Problem Identification:*** The model engineer recognizes a challenge that FL can address.

2. ***Client Instrumentation:*** The clients are outfitted with the required training data storage capabilities if necessary. The app will frequently have this information recorded already. To give labels for a supervised learning job, for example, user interaction data would need to be preserved in some circumstances, as can supplementary data or metadata.

3. ***Simulation Prototyping:*** The model engineer can use a proxy dataset to simulate an FL simulation to prototype model architectures and evaluate learning hyperparameters.

4. ***Federated model training:*** The model is trained using various iterations, or using various optimization hyperparameters, through the use of many federated training tasks.

5. ***Model evaluation:*** At the end of the training process, the models must be analyzed and the appropriate candidates selected. In this process, some metrics are used or federated assessments in which clients are held out and models are pushed for evaluation using their own data.

6. ***Deployment:*** In the last step, once a good model has been chosen, it goes through the standard model launch process, which includes manual quality assurance, live A/B testing, typically using the new model on some devices and the previous generation model on other devices to compare their in-vivo performance, and a staged rollout so that bad behavior can be discovered and rolled back before it affects too many users. The owner

of the application determines the exact launch procedure for each model, and this procedure is typically unrelated to the model's training method. To put it another way, either the classic data center method or federated learning might be used to train the model in question.

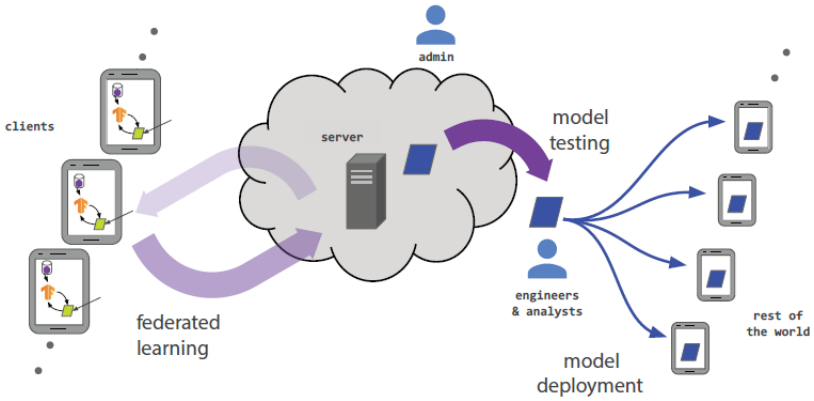

Figure 1: *The life cycle of an FL-trained model and the various actors in a federated learning system [29].*

Furthermore, it is important to mention that achieving the aforementioned process as simple and feasible as possible can be done by preferably approaching the ease of use achieved by ML systems for centralized training. This is one of the main practical issues facing an FL system.

2.1.3 A Representative Federated Training Process

It is familiar that there are several FL training applications, but at this stage in this work, the FL training model based on the Federated Averaging algorithm of McMahan et al [37] will be referred to, which is considered a starting point for understanding the training model of the specific field of Machine Learning.

Responsible for the training process is a server that orchestrates the specific process, repeating the steps listed below until the end of training is reached based on the judgment of the machine model that oversees the training process. Below are the steps followed during the training of a model in FL:

1. ***Client Selection:*** A group of clients that are eligible are chosen at random by the server. To protect the device's user, mobile phones, for instance, could only connect to the server when they are plugged in, connected to an unmetered Wi-Fi network, and idle.

2. ***Broadcast:***A training program and the most recent model weights are downloaded from the server by the chosen clients.

3. ***Client Computation:***Executing the training program, which may, for example, perform SGD on the local data as in Federated Averaging Algorithm that is referred to in McMahan et al [37], causes each chosen device to locally calculate an update to the model.

4. ***Aggregation:***A total of the device updates are gathered by the server. Once a sufficient number of devices have submitted findings, stragglers may be deleted for efficiency at this stage.

5. ***Model update:***Based on the aggregated update computed from the clients who took part in the current round, the server locally changes the shared model.

2.1.4 Diversity of FL Systems

In accordance with [53], Federated Learning can be divided into 3 categories according to the distribution of the data that used in the training process.

In federated learning, a matrix Di is used which denotes the data held by each owner in each row i. The matrix's rows stand in for samples and its columns for features. Nevertheless, label data may also be present in some datasets. In this case, FL utilizes X, Y, and I to represent the feature, label, and sample ID space, respectively. From the above mentioned, one can easily recognize that *(I, X, Y)* make up the entire training dataset. The feature space and the sample space of the data parties may not be identical, and federated learning uses how the data is distributed across the various parties in the feature ID space and the sample ID space to generate horizontal data. It is classified into federated learning, vertical federated learning, and transfer federated learning. The feature space and the sample space of the data parties may not be identical, and federated learning uses how the data is distributed across the various parties in the feature ID space and the sample ID space to generate horizontal data. It is classified into federated learning, vertical federated learning, and transfer federated learning. The feature space and the sample space of the data parties may not be identical, and federated learning uses how the data is distributed across the various parties in the feature ID space and the sample ID space to generate horizontal data. For that reason, FL can be classified into horizontal federated learning, vertical federated learning, and transfer federated learning.

Below are the breakdowns of the three categories federated learning can be divided into:

- ***Horizontal Federated Learning:***

  Horizontal federated learning is used in cases where the user characteristics of the two datasets interact with each other and the users overlap little. The procedure followed in horizontal federated learning is to first horizontally partition the data sets according to the user attribute, and

then remove the part of the data that the user characteristics are the same, but the users are not exactly the same for training. As a result, federated horizontal learning can broaden the user sample. In this case, horizontal federated learning may be used to train a model, increasing both the overall number of training samples and the model's precision. All participants typically calculate and upload local gradients in horizontal federated learning so that the central server may combine them into a global model. Personal information about users may be leaked through processing and communication biases in horizontal federated learning. Homomorphic encryption [38], differential privacy [39], and safe aggregation [12] are popular approaches to this issue that can guarantee the security of gradient switching in horizontal federated learning.

The authors of [53] report a proposal of a deep learning approach that is cooperative in which participants train individually and only share portions of parameter changes. For Android phone model upgrades in 2017, Google suggested a horizontal federated learning method. In that approach, a single user modifies the model parameters locally on an Android phone before uploading them to the Android cloud to jointly train the centralized model with other data owners. As part of their federated-learning system, they also provide a secure aggregation approach to safeguard the privacy of aggregated user updates. Additionally, it was reported that additively homomorphic encryption was utilized for the aggregation of model parameters to offer protection against the central server.

Furthermore, in [53], authors define horizontal federated learning as follow:

$$X_i = X_j, Y_i = Y_j, I_i \neq I_j \forall D_i, D_j, i \neq j$$

An honest participant base and protection against an honest-but-curious server are common premises of a horizontal federated learning system. Therefore, only the server has the ability to jeopardize the participants' privacy. In these works, security evidence has been offered. Recently, a different security model that takes into account dangerous individuals was also put out, creating more privacy issues. The universal model and all of the model parameters are made available to everyone at the conclusion of the training.

- ***Vertical Federated Learning:*** When the user characteristics of two datasets barely overlap but the users heavily do, vertical federated learning is an option. In vertical federated learning, datasets are divided vertically, by user feature dimension, and the data that has identical users but different user attributes are removed for training. In other words, the same users appear in data across columns, aligned by the user. The feature dimension of training data can therefore be increased using vertical federated learning. To improve the model's performance, vertical feder-

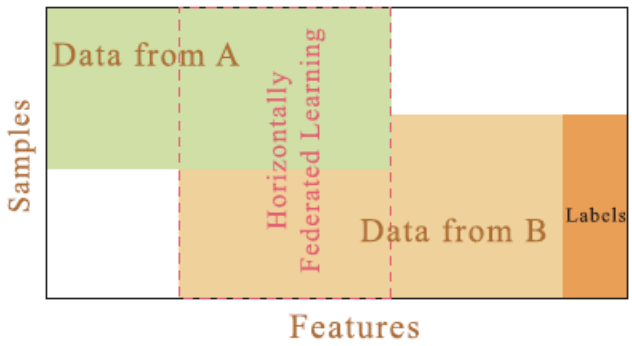

Figure 2: *Horizontal Federated Learning [55].*

ated learning combines these many features in an encrypted state. Many machine learning models, including logical regression, tree structure, and neural network models, are currently being steadily shown to be built on this federated system.

This type of federal method is known as federated learning because it allows all participants to build "commonwealth" strategies while maintaining their shared identities and statuses. Therefore, in such a system, authors in [53] defined the system as follows:

$$X_i \neq X_j, Y_i \neq Y_j, I_i = I_j \forall D_i, D_j, i \neq j$$

In continuation of the previously mentioned, an honest yet inquisitive participant base is often assumed in a vertical federated learning system. For instance, in a two-party situation, only one of the two parties is compromised by an adversary, and the two parties are not cooperating. According to the security definition, the adversary can only learn data from the client that it has damaged and cannot learn anything about the other client beyond what the input and output have already disclosed. Sometimes a semi-honest third party (STP) is inserted to help the two parties do safe calculations; in this situation, it is presumed that the STP does not collaborate with either side. Formal privacy verification for these protocols is offered by SMC. At the conclusion of the learning process, each participant only retains the model parameters related to its own characteristics. As a result, in order to get results at inference time, both sides must work together.

- ***Federated transfer Learning:***

  Instead of segmenting the data when users and user features from the two

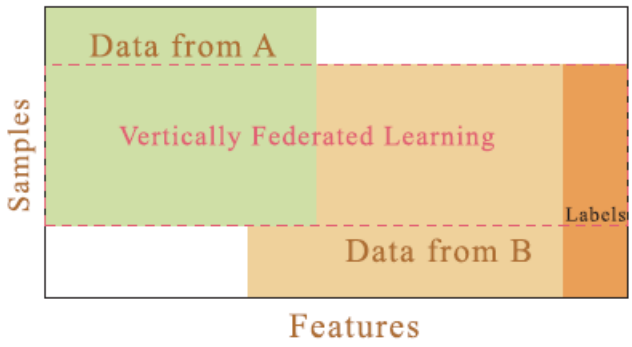

Figure 3: *Vertical Federated Learning [55].*

datasets seldom ever overlap, transfer learning may be used to make up for the missing information. Federated transfer learning is the name of this technique. Take into account two organizations: one is an e-commerce business based in the United States, and the other is a bank with headquarters in China. Geographical limitations have resulted in a minor crossing between the user groups of the two institutions. The feature space from both parties only slightly overlaps, however, as a result of the separate enterprises. In this situation, a federation's complete sample and feature space may be solved using transfer-learning approaches. In particular, using small shared sample sets, a common representation across the two feature spaces is learned and then used to get predictions for samples with just one-side characteristics. FTL is a significant addition to the current federated learning systems since it deals with issues that are outside the purview of the federated learning algorithms:

$$X_i \neq X_j, Y_i \neq Y_j, I_i \neq I_j \forall D_i, D_j, i \neq j$$

In addition to what was said before, two parties are usually involved in a federated transfer learning system. As will be seen in the next section, its protocols are comparable to those used in vertical federated learning, in which case the vertical federated learning security definition may be extended here.

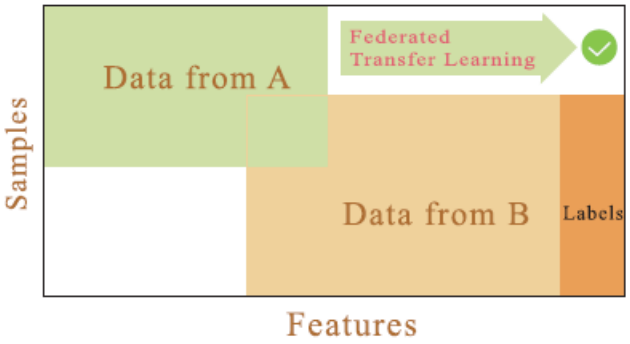

Figure 4: *Federated Transfer Learning [55].*

2.1.5 Architectures of Federated Learning Systems

In this section of the comparative work, the architectures of the various federated learning theories mentioned in subsection 2.1.5 will be discussed. More specifically, the architectures of horizontal federated learning, vertical federated learning, and transfer learning will be analyzed.

Architecture of a horizontal federated learning system:

Horizontal federated learning involves users with the same data structure learning a machine learning model jointly with the help of a parameter or cloud server. A common example is that the players are honest, but the server is honest but also curious. Therefore, in this process, no information leakage is allowed from any participant on the server. This particular system, to be executed correctly, includes 4 basic steps during its training process.

First of all, the users of the system perform a local computation to compute the training gradients. For this purpose can use encryption techniques, differential privacy, or secret sharing techniques. At the completion of this operation, users transmit the disguised results to the server. When carrying out the second step, The server executes secure aggregation without discovering any participant's identity. In the sequel, The server transmits the aggregated results to the participants and in the fourth step, the decrypted gradients are used to update each participant's model.

Subsequently, the training procedure is complete after all of the aforementioned phases have been iterated through. The final model parameters will be shared by all participants in this architecture, which is agnostic of individual machine-learning techniques such as logistic regression. An illustration of the specific architecture appears in figure 5.

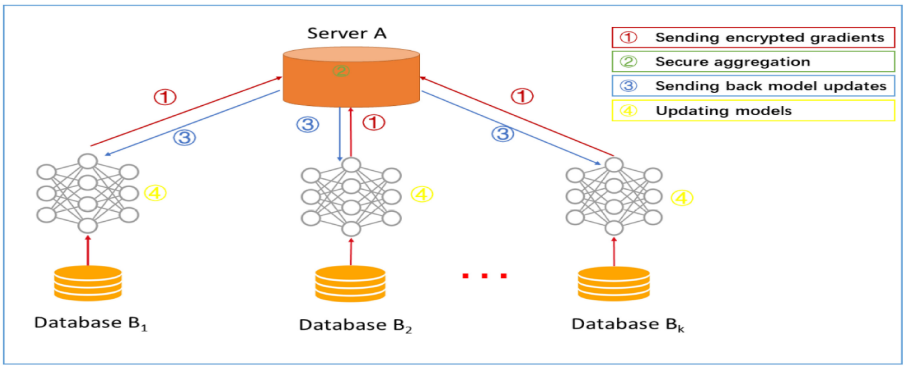

Figure 5: *Architecture fora horizontal federated learning system [53].*

Architecture of a vertical federated learning system:

For the purpose of giving an explanation of the architecture of the vertical federated learning system, in [53] suggested an example for the best understanding of the architecture of the vertical federated learning system.

More specifically, motioned that there are two companies that have the purpose to cooperate in training the same machine learning model. Worth mentioning is that the business system of each company has its own data for the training process. Additionally, the model requires label data from Company B in order to make predictions. It stands to reason that data interaction between A and B is prohibited for security and privacy concerns. The involvement of a third-party collaborator C helps to maintain the privacy of the data during the training process. In this instance, it is assumed that collaborator C is truthful and does not conspire with parties A or B, whereas parties A and B are truthful yet amiable towards one another. It is acceptable to assume that party C is a trustworthy third party since party C can be played by authorities like governments or replaced by a secure computing node like Intel Software Guard Extensions.

According to the literature, the vertical federated learning architecture is divided into two parts. The first part refers to encrypted entity alignment. More specifically, it is a fact that the two groups of the two companies mentioned above are not the same. Therefore, it is important for these groups that the system has the ability to recognize the users who use the system on a daily basis. For this purpose, the system uses user ID alignment techniques based on cryptography. The second part of this architecture refers to encrypted model training. In simpler words, when the common entities are identified by the system then their data can be used for the purpose of training a machine learning model. The successful training of the model mentioned above follows the four steps below:

1. *A and B receive a public key from collaborator C, who also creates encryption pairs.*

2. *For the purposes of calculating gradient and loss, A and B encrypt.*

3. *Both A and B calculate encrypted gradients and add an extra mask. B also determines the encrypted loss. A and B transmit C values.*

4. *A and B receive the decrypted gradients and loss from C once it has encrypted them. The gradients are revealed by A and B, who then adjust the model's parameters.*

Below is an illustration of vertical federated learning architecture:

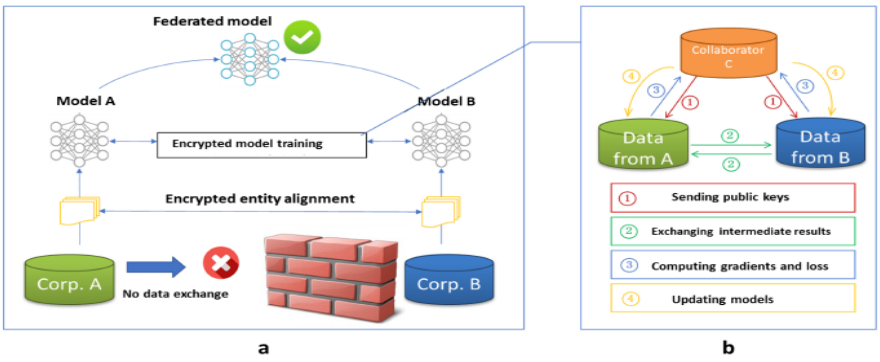

Figure 6: *Architecture fora vertical federated learning system [53].*

Architecture of a federated transfer learning system:

The vertical federated learning architecture described above only works for the overlapping dataset. To extend the coverage of the data set to the entire sample space, transfer learning comes to provide a solution. Essentially the only difference between these two architectures is that in the federated transfer learning architecture the details of the intermediate results exchanged by parts A and B above are changed. Transfer learning specifically entails discovering a shared representation between the characteristics of parties A and B and reducing prediction errors for the target-domain party's labels by utilizing the labels in the source-domain party. As a result, compared to the vertical federated-learning scenario, the gradient computations for parties A and B are different. Both parties must still calculate the prediction outcomes at the moment of inference.

2.1.6 Challenges

The rapid development of federated learning has seen a surge in its adoption over the past few years and is expected to continue to grow in popularity in the future. However, despite its many advantages, there are several challenges that must be addressed before it can be widely adopted. In this chapter, we discuss some of the challenges associated with federated learning and how they can be addressed.

- *Data Privacy and security:*

  The primary challenge associated with federated learning is the need to protect the privacy and security of the data used in the training process. This is a particular concern when dealing with sensitive data such as med-

ical records or financial information. In such cases, the data must be kept confidential and should not be shared with any third party. Additionally, the data must be securely transmitted between the federated learning participants.

To address these concerns, several encryption protocols have been proposed for use in federated learning [23],[34]. These protocols provide secure communication between the participants, allowing them to exchange data without compromising the privacy and security of the data. Additionally, cryptographic protocols such as homomorphic encryption and secure multiparty computation can be used to ensure that the data remains confidential even when it is shared between multiple parties.

- *Data Inhomogeneity and Non-IID Distribution:*

  Another challenge in federated learning is the issue of data inhomogeneity and non-IID distribution. This refers to the fact that the data used in the training process is often distributed across multiple participants in an unbalanced manner. This can make it difficult to ensure that the model is accurately trained on all of the data.

  To address this issue, several methods have been proposed to ensure that the data is evenly distributed across the participants [57]. These methods attempt to ensure that the data is balanced across the participants, allowing the model to be trained on all of the data. Additionally, methods such as federated transfer learning, federated optimization, and federated meta-learning can be used to improve the accuracy of the model when dealing with non-IID data [29].

- *Communication Overhead:*

  A further challenge associated with federated learning is the need for frequent communication between the participants. This can lead to increased communication overhead, which can reduce the efficiency of the training process. Additionally, the communication overhead can be further increased if the participants are geographically distributed, as the data must be transmitted over long distances.

  To address this issue, several methods have been proposed to reduce the communication overhead. These include techniques such as model compression, distributed optimization [51], and data partitioning [33]. Additionally, techniques such as federated caching [36] and federated compression [41] can be used to reduce the amount of data that must be transmitted between participants.

- *Performance Evaluation:*

  The final challenge associated with federated learning is the difficulty of evaluating the performance of the model. This is due to the fact that the data used in the training process is distributed across multiple participants, making it difficult to accurately measure the performance of the model.

  To address this issue, several methods have been proposed to evaluate the performance of federated learning models. These methods include techniques such as cross-validation, federated holdout, and federated inference [53]. Additionally, several metrics such as accuracy, precision, recall, and F1-score can be used to evaluate the performance of the model [37].

In this sub-subsection, we discussed the challenges associated with federated learning and how they can be addressed. We discussed the need to protect the privacy and security of the data used in the training process, the issue of data inhomogeneity and non-IID distribution, the communication overhead associated with federated learning, and the difficulty of evaluating the performance of the model. By addressing these challenges, federated learning can become a more widely adopted technology in the coming years.

2.2 Unsupervised Learning

2.2.1 Introduction to Unsupervised Learning

Unsupervised learning is a type of machine learning that operates on unlabeled data to discover hidden patterns, relationships, or structures [14]. Unlike supervised learning, unsupervised learning does not require labeled data or human supervision, making it a powerful tool for analyzing complex, high-dimensional data and uncovering meaningful insights [21].

2.2.2 Applications of Unsupervised Learning

Unsupervised learning has been applied to a wide range of applications in various fields including data mining, computer vision, natural language processing, and bioinformatics. Some of the common applications of unsupervised learning are:

- Clustering: This involves grouping similar objects together based on their features. Clustering is widely used in market segmentation, image segmentation, and anomaly detection [4].

- Dimensionality Reduction: This involves reducing the number of features in high-dimensional data while preserving the underlying structure [49]. Dimensionality reduction is commonly used in computer vision, text analysis, and genomic data analysis [32].

- Anomaly Detection: This involves identifying unusual or unexpected patterns in data [11]. Anomaly detection is useful in a variety of domains such as fraud detection, intrusion detection, and system monitoring [5].

- Association Rule Mining: This involves discovering frequent patterns or associations in data. Association rule mining is widely used in market basket analysis and recommender systems [35].

2.2.3 Types of Unsupervised Learning Algorithms

There are several types of unsupervised learning algorithms, each with its own strengths and limitations [24]. Some of the commonly used unsupervised learning algorithms are:

- K-Means Clustering: This is a centroid-based clustering algorithm that aims to partition data into K clusters such that the sum of squared distances between the data points and their corresponding centroids is minimized.

- Hierarchical Clustering: This is an agglomerative or divisive clustering algorithm that builds a hierarchy of clusters by either merging or splitting existing clusters [27].

- Principal Component Analysis (PCA): This is a linear dimensionality reduction technique that projects data onto a lower-dimensional subspace while preserving the maximum variance [28].

- Autoencoders: This is a neural network architecture that learns a compact representation of the input data by encoding it into a lower-dimensional latent space and then decoding it back to the original space [8].

2.2.4 Challenges and Limitations of Unsupervised Learning

Unsupervised learning is a challenging task due to several factors such as the lack of labeled data, the high dimensionality of data, and the presence of noisy or irrelevant features. Some of the challenges and limitations of unsupervised learning are:

- Evaluation: Evaluating the performance of unsupervised learning algorithms is a challenging task as there is no ground truth for comparison. Some common evaluation metrics for unsupervised learning include internal validation metrics such as silhouette score and adjusted Rand index, and external validation metrics such as manual inspection or comparison with known results [44].

- Selecting the Number of Clusters: In many clustering algorithms, the number of clusters needs to be specified beforehand. However, finding the

optimal number of clusters is a difficult task and can be influenced by various factors such as the shape of the clusters and the presence of noise [48].

- Overfitting: Unsupervised learning algorithms can overfit the noise in the data, leading to poor performance on unseen data. Regularization techniques such as adding a constraint on the size of the clustering solution or adding a reconstruction loss can be used to mitigate overfitting in some cases [8].

- Scalability: Many unsupervised learning algorithms have high computational and memory requirements and may not be suitable for large-scale datasets [8].

2.2.5 Conclusion about unsupervised learning

Unsupervised learning is a powerful tool for discovering hidden patterns, relationships, and structures in data. It has been applied to a wide range of applications in various fields and has several types of algorithms to choose from. However, unsupervised learning is a challenging task due to several limitations such as the lack of labeled data, the difficulty in evaluating performance, and the high computational requirements of some algorithms. Despite these challenges, unsupervised learning remains an active area of research with many exciting developments and breakthroughs in recent years.

3 Methodology Design

3.1 Algorithms

3.1.1 Kmeans Algorithm

In this thesis, the K-Means algorithm is used for clustering. The purpose of using this particular algorithm is to implement 4 different clusters, i.e. one for each label that reflects one of the 4 classes.

According to [50] and [47], a definition that can be given for the K-means algorithm is that Kmeans is an iterative technique that attempts to divide the dataset into K unique, non-overlapping clusters, each of which contains only one group to which a given data point belongs. Regarding the aforementioned, the clusters are kept distinct from one another while simultaneously attempting to make the intra-cluster data points as comparable as feasible. It places data points into clusters so that the total squared distance between each data point and the cluster's centroid—the average value of all the data points in the cluster—is at its smallest. The more homogenous the data points inside a cluster are, the less variance there is between clusters.

The steps by which the algorithm works are as follows:

1. K points—either at random or the first K—from the dataset to serve as the starting centroids.

2. In the dataset containing the identified K points, calculate the Euclidean distance between each point (cluster centroids).

$$d(p, q) = \sqrt{(q1 - p1)^2 + (q2 - p2)^2}$$

3. Using the distance discovered in the previous step, assign each data point to the nearest centroid.

$$argmin_{c_i C} dist(c_i, x)^2$$

4. Take the average of the points in each cluster group to determine the new centroid.

$$c_i = \frac{1}{|S_i|} \sum_{x_i S_i} x_i$$

5. Repeat steps 2 through 4 as many times as necessary, up until the centroids stay the same.

Furthermore, the technique that the K-means algorithm uses to confront the problem is called Expectation-Maximization. Particularly is an iterative approach to discover maximum likelihood or maximum a posteriori estimates

of parameters in statistical models when the model is dependent on unobserved latent variables and can be used as an unsupervised clustering algorithm.

3.1.2 Label propagation

Label propagation is a semi-supervised learning algorithm that is used to predict the labels of unlabeled data points based on the labels of surrounding data points [59]. It is typically used when only a small portion of the data is labeled and the goal is to predict the labels of the remaining, unlabeled data.

The algorithm works by first selecting a small set of labeled data points and using these to "seed" the process. It then propagates the labels from these seed points to the surrounding unlabeled data points, iteratively updating the labels of the unlabeled points based on the labels of their neighbors [58]. The labels continue to propagate through the dataset until the algorithm reaches a stable state, at which point the labels of the unlabeled data points are considered to be predicted.

The steps by which the algorithm works are as follows

1. Initialize the labels at all nodes in the network. For a given node $x, C_x(0) = x$.

2. Set $t = 1$.

3. Arrange the nodes in the network in a random order and set it to X.

4. For each xX chosen in that specific order, let $C_x(t) = f(C_{xi1}(t), ..., C_{xim}(t), C_{xi(m+1)}(t-1), ..., C_{xik}(t-1))$. Here returns the label occurring with the highest frequency among neighbors. Select a label at random if there are multiple highest-frequency labels.

5. If every node has a label that the maximum number of their neighbors have, then stop the algorithm. Else, set $t = t + 1$ and go to (3).

One advantage of label propagation is that it is relatively simple to implement and can be applied to a wide range of data types. However, it is important to note that the accuracy of the predictions made by the algorithm will depend on the quality of the seed points and the connectivity of the data [30].

3.1.3 Agglomerative Clustering

Agglomerative clustering is a bottom-up approach to clustering in which each data point is initially treated as its own cluster. The algorithm then iteratively merges the most similar clusters until a specified number of clusters or some other stopping criterion is reached [52].

There are several different ways to define similarity between clusters, such

as the distance between the cluster centroids or the minimum distance between points in different clusters. The choice of similarity measure will depend on the nature of the data and the desired properties of the clusters.

More specifically the steps by which the algorithm works are as follows:

1. Make each data point a single-point cluster.
2. Take the two closest distance clusters by single linkage method and make them one cluster.
3. Repeat step 2 until there is only one cluster.
4. Create a Dendrogram to visualize the history of groupings.
5. Find the optimal number of clusters from the Dendrogram.

One advantage of agglomerative clustering is that it is relatively simple to implement and can be applied to a wide range of data types. However, it is important to note that the final clusters produced by the algorithm may be sensitive to the order in which the data points are processed.

3.1.4 Birch

Birch is an efficient and scalable clustering algorithm that is well-suited for handling large datasets [56]. It is a hierarchical clustering algorithm that uses a tree-based data structure to maintain a compact representation of the clusters.

The algorithm begins by constructing a CF (clustering feature) tree, which is a tree-based data structure that stores the cluster features and the clustering feature vectors of the data points. It then uses this tree to iteratively merge the most similar clusters until a specified number of clusters or some other stopping criterion is reached.

More specifically the steps by which the algorithm works are as follows[56]:

1. All data points are converted into CF form using the formula $CF = (N, LS, SS)$.
2. When all of the data has been transformed into CFs, the CF-Tree algorithm begins to bring the CFs together. You will need to input the number B in this area (Branching).
3. The initial CF tree threshold, which will be used as the starting threshold value for each new CF entry and remain unchanged during the grouping process, must be initialized before scanning any data points from the database. (static).

4. A typical BIRCH will ask us to initialize L. (number of leaves). We add two parameters, m, and b, to aid in the computations. The CF-non leaf's number of branches is counted using parameter b, while the CF-number leaf's of leaf branches is counted using parameter m.

5. Using a linear number or average CF, BIRCH compares the position of each record with each CF's placement at the root node. The entry is carried on by BIRCH to the CF root node that is closest to the entry record.

6. The node then descends to the CF nodes chosen in step 5's non-leaf child node. BIRCH checks each non-leaf CF's location with the location of the records. The note that goes to the non-leaf CF node nearest to the entry is continued by BIRCH.

7. After then, the node descends to the non-leaf CF node's leaf child node. BIRCH compares the leaf locations to the record locations. BIRCH momentarily sends the entry to the leaf with the entry node that is closest to it.

8. Do one of the following steps:

   - The entry is allocated to that leaf if the leaf radius (R) chosen includes a new node and does not exceed T Threshold. New data points are included in leaves as well as all parent CFs.

   - A new leaf made up solely of incoming notes forms if the leaf radius chosen to include the new record exceeds the Threshold T. Updated CF parent to take into account fresh data.

9. There will be a second branch if the leave (m) branch goes above the given leave (L) limit (B).

10. If B has been surpassed, CF will undergo a split-parent procedure before being united once again with the newly generated high CF. Updated CF parent to take into account fresh data.

One advantage of Birch is that it is able to handle large datasets by making use of summary data structures and utilizing a divide-and-conquer strategy. It is also able to handle datasets with high dimensionality, making it a useful tool for handling complex data.

3.1.5 DBSCAN

First of all, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a popular unsupervised machine learning algorithm used for discovering clusters in a dataset. It is based on the idea of density, where a cluster is defined

as a region in the data space that is densely populated with points compared to the rest of the data space.

The algorithm works by defining a distance threshold, Eps, and a minimum number of points, MinPts, that must be contained within a cluster. It then starts at a random point in the dataset and retrieves all other points within Eps distance. If the number of points is greater than or equal to MinPts, it is considered a cluster. If it is less than MinPts, it is considered noise. The algorithm continues this process for all points in the dataset until all points have been processed.

More specifically the steps by which the algorithm works are as follows:

Let the set of data points be $X = x1, x2, x3, ..., xn$. Two parameters are necessary for DBSCAN: (eps) and the bare minimum of points needed to construct a cluster (minPts).

1. Start in a random location that hasn't been explored before.

2. Extract the neighborhood of this point using (All points which are within the distance of our neighborhood).

3. The clustering process begins and the point is tagged as visited if there are enough neighborhoods nearby; else, the point is classified as noise.

4. If a point is determined to be a member of the cluster, all of its neighbors are also members of the cluster, and step 2 is repeated for each neighboring point. Until all of the cluster's points are identified, this process is repeated.

5. A fresh unexplored point is obtained, and analyzed, and a new cluster or noise is found.

6. Up until all of the points are recorded as visited, this procedure continues.

One advantage of DBSCAN is that it can handle data with noise and outliers, as it explicitly identifies and excludes them from the clusters. It is also capable of finding clusters of different shapes and sizes, as it does not rely on the assumption that clusters are spherical or of equal size. However, DBSCAN has some limitations. It can be sensitive to the choice of Eps and MinPts, as the performance of the algorithm can vary significantly based on their values. It is also not well suited for data with high dimensionalities, as the distance measure can become unreliable in high dimensions.

Overall, DBSCAN is a powerful and widely used algorithm for cluster analysis and has been applied in a variety of fields such as data mining, pattern recognition, and image analysis[18].

3.1.6 Optics

The Optics (Ordering Points To Identify the Clustering Structure) algorithm is a density-based clustering algorithm, similar to DBSCAN, for finding clusters in a dataset. It is particularly useful for handling large datasets and datasets with high dimensionalities, as it is able to identify clusters of varying densities and shapes.

The specific algorithm works by constructing a reachability plot, which represents the distance of each point in the dataset to its nearest neighbor that is denser. The reachability plot is then used to identify clusters by finding local minima, which represent the points that are the "starting points" of the clusters.

More specifically the steps by which the algorithm works are as follows:

1. Create the Eps density threshold parameter, which regulates the clusters' minimum density.

2. Determine the k-nearest neighbor distance for each point in the dataset.

3. Determine the reachability distance of each location in the dataset based on the density of its neighbors, starting at any point.

4. The reachability plot is made by arranging the points according to their reachability distance.

5. By combining points that are close to one another and have comparable reachability distances, you may extract clusters from the reachability plot.

An advantage of the Optics algorithm is that it can identify clusters of varying densities, as it does not require the user to specify a density threshold as DBSCAN does. It is also able to handle datasets with high dimensionalities, as it uses a reachability distance measure that is less sensitive to the curse of dimensionality.

However, the Optics algorithm has some limitations. It can be sensitive to the choice of the reachability distance parameter, as the performance of the algorithm can vary significantly based on its value. It is also not able to identify noise or outliers in the dataset, as it does not explicitly exclude them from the clusters.

Overall, the Optics algorithm has the same usefulness as DBSCAN. In simpler words, is a useful tool for cluster analysis and has been applied in a variety of fields such as data mining, pattern recognition, and image analysis [6].

3.2 Dataset

In this work, we use the abusive dataset provided by [19]. The dataset consists of one hundred thousand tweets that have been classified into four categories: abusive, hateful, spam, and normal. It is important to mention the number of records that reflect each of the four classes. According to [19], the number of records that reflect the Abusive class is 27% of the data set, 5% reflect the Hateful class, 14% the Spam class, and 54% the Normal class, as can be seen in Table 1.

Original Abusive Dataset		
Class	Number of records	Percentage
Abusive	27150	27%
Hateful	4965	5%
Spam	14030	14%
Normal	53850	54%
Total	99995	100%

Table 1: *Logistics of the Abusive Dataset provided from [19]*

The main goal of the work disclosed and engaged in the dataset construction was to utilize a crowdsourcing tool like CrowdFlower(CF) to produce a sizable and extremely accurate annotated dataset of tweets. More specifically, the author's focus shifted more narrowly to the many kinds of abusive behavior, to examine their association and modify the labels that would ultimately be used. reover, to achieve their goal of arriving at a set of labels, the authors used statistical analyses.

Finally, below are the concepts of each class included in the dataset:

- **Abusive:** Profanity, or any very unfriendly, nasty, or cruel words that can be used to convey denigration of someone or something or to express strong emotion.
- **Hateful:** Language that is designed to be disparaging, humiliating, or insulting to the members of the group because of characteristics like race, religion, ethnic origin, sexual orientation, handicap, or gender.
- **Spam:** Posts typically featured a variety of unwelcome content, including relevant or unrelated advertising, the sale of pornographic items, links to dangerous websites, phishing scams, and other types of unwanted information.
- **Normal:** Any tweet that doesn't fit into one of the categories above.

The Figure Eight platform was utilized by its users to classify tweets into distinct categories of inappropriate language, including offensive, abusive, hate-

ful speech, aggressive, bullying, spam, and normal, through various annotation rounds. The authors conducted an exploratory investigation to identify the most representative labels for abusive content in text and employed statistical analysis to arrive at the aforementioned set of labels.

Finally, for the present study, we eliminated tweets containing spam-related content from the dataset. Furthermore, we merged abusive and hateful tweets into one category, referred to as the "harmful" class, to simplify the classification task and convert it into a binary classification problem. The resulting dataset is referred to as the "Harmful dataset" hereafter, and its details are presented in Table 2. Despite the presence of other metadata in the original dataset, such as the number of hashtags or tags associated with each tweet, or information about the user's friend network, we did not utilize this metadata during training due to the use of a simple text classification model in our simulations.

Harmful Dataset		
Class	Number of records	Percentage
Harmful	32115	37%
Normal	53850	63%
Total	85965	100%

Table 2: *Logistics of the curated Harmful dataset after removing the spam records and merging the hateful and abusive classes under one class (the Harmful class) from the Harmful Dataset provided from [19].*

3.3 Technologies

This section provides a catalog of the key technologies employed for executing the experimental simulations.

3.3.1 Anaconda

Anaconda is a popular data science platform that can be used to simplify the process of managing and deploying data science projects[26]. The platform provides a user-friendly interface for managing packages and dependencies, as well as a variety of tools for data exploration, analysis, and visualization.

Anaconda can be a valuable tool for managing the various software packages and libraries needed for data analysis. By using Anaconda to manage dependencies, the researcher can avoid version conflicts and ensure that the correct versions of each package are used throughout the project. This can help to streamline the development process and reduce the likelihood of errors and bugs.

Anaconda also includes a number of tools that can be useful for data exploration and analysis, such as Jupyter Notebook and Spyder. These tools provide

a user-friendly interface for working with data, allowing the researcher to quickly and easily perform exploratory data analysis and visualize their results.

Overall, Anaconda can be a valuable asset for anyone working on data science. In this work, Jupyter was used from the various software offered by the Anaconda platform. The purpose was to facilitate the visualization of the results and for better debugging.

3.3.2 Tensorflow keras

TensorFlow and Keras are popular open-source libraries for building and training deep learning models. TensorFlow was developed by the Google Brain team [3] and is a powerful framework for building and deploying machine learning models, particularly for large-scale projects. Keras, on the other hand, is a high-level neural network API that runs on top of TensorFlow [15]. It allows developers to easily build and experiment with neural networks, without having to worry about low-level implementation details.

Together, TensorFlow and Keras provide a comprehensive toolset for building and training deep learning models. With TensorFlow, developers have access to a wide range of powerful features for distributed training, deployment, and production-level model serving. Keras, on the other hand, simplifies the process of building and experimenting with deep learning models, making them accessible to developers of all skill levels.

In this work, tensorflow keras was used for the text preprocessing(explained in the next subsubsection) that became the text data of the dataset used for the purpose of this work.

3.3.3 Pandas & Numpy

Pandas and Numpy are two of the most widely used libraries in the Python data science ecosystem [2, 1]. Pandas provide data structures for efficiently storing and manipulating large datasets, while Numpy provides a powerful array computing functionality that is essential for many scientific and numerical computing applications.

Both libraries have extensive documentation and a large user community, so finding answers to specific questions or problems is usually straightforward. There are also many excellent tutorials and books available for learning how to use Pandas and Numpy effectively, such as the "Python for Data Analysis" book by Wes McKinney, the "Python Data Science Handbook" by Jake VanderPlas, and the "Pandas Cookbook" by Kevin Markham.

By using these resources, researchers can quickly become proficient in using Pandas and Numpy to perform data manipulation, analysis, and visualization.

These libraries have the potential to revolutionize the way we approach data analysis and provide new insights into the world around us.

3.3.4 Scikit-learn

Scikit-learn is a popular open-source library for machine learning in Python. It provides a wide range of algorithms for classification, regression, clustering, and dimensionality reduction, as well as tools for model selection and evaluation[42]. Scikit-learn is designed to be easy to use and provides a consistent interface for all of its algorithms, making it a popular choice for both beginners and experts in machine learning.

One of the advantages of scikit-learn is its excellent documentation and tutorials, which help users quickly get started with the library. Scikit-learn is also highly optimized for speed and memory efficiency, making it suitable for processing large datasets.

Some of the key features of scikit-learn include:

- Consistent API for all algorithms
- Easy integration with other Python libraries, such as NumPy and Pandas
- Comprehensive documentation and tutorials
- Support for parallel processing
- Built-in tools for model selection and evaluation

Scikit-learn is widely used in both academia and industry and is included in many popular data science platforms such as Anaconda and Google Colab. Overall, in this work, scikit-learn was used for the implementation of the algorithms, the principal component analysis, and the extraction of the results.

3.3.5 Matplotlib & Seaborn

Matplotlib and Seaborn are two popular open-source libraries for data visualization in Python [9]. Matplotlib provides a wide range of plotting functions and styles, while Seaborn is built on top of matplotlib and provides more advanced visualization functions for statistical data analysis.

Matplotlib is a powerful library for creating static, interactive, and animated visualizations in Python. It provides a wide range of plot types, including line plots, scatter plots, bar plots, and histograms, and allows for the customization of plot properties such as labels, titles, and colors.

Seaborn, on the other hand, is designed specifically for statistical data analysis and provides advanced functions for visualizing relationships between

variables. Some of the key features of Seaborn include support for visualizing categorical data, regression analysis, and data distribution analysis.

Together, Matplotlib and Seaborn provide a comprehensive toolset for data visualization in Python. They are both highly customizable and can produce high-quality graphics suitable for use in publications and presentations. Overall, in this work, these two libraries were used for the visualization purpose of the experimental results.

3.4 Implementation

3.4.1 Text pre-processing

The text pre-processing phase plays a crucial role in preparing the raw tweet data for subsequent analysis or machine learning applications. In the context of this thesis, the pre-processing phase consists of a series of steps aimed at transforming the unstructured text data into a more structured and cleaner format. These steps are essential for enhancing the quality of data and ensuring that the results obtained from the analysis are accurate and reliable. The pre-processing phase comprises the following steps:

1. *Removing Twitter Handles (@user):*

   This step involves removing all Twitter handles present in the text, which is identified using the regular expression `[\w]*` . This is done by calling the `"remove_pattern"` function, which takes the input text and the pattern as arguments and replaces all matches with an empty string.

```
def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for i in r:
        input_txt = re.sub(i, '', input_txt)

    return input_txt

df['Clean Tweet text'] = np.vectorize(remove_pattern)/
(df['Tweet text'], "@[\w]*")
```

2. *Removing Links:*

   Any links present in the text are removed using the regular expression `"https?://.[\r \n]"`, which matches any string starting with "http" or "https" and ending with a newline character. This is done by calling the `"str.replace"` function on the "Clean Tweet text" column of the DataFrame.

```
df['Clean Tweet text'] = df['Clean Tweet text']
```

```
.str.replace("https?:\/\/.*[\r\n]*", " ")
```

3. *Removing Punctuations, Numbers, and Special Characters:*

   All special characters, numbers, and punctuations are removed from the text using the regular expression `"[â-zA-Z#]"`. This is done by calling the `"str.replace"` function on the "Clean Tweet text" column of the DataFrame.

```
df['Clean Tweet text'] = df['Clean Tweet text']
.str.replace("[^a-zA-Z#]", " ")
```

4. *Removing Short Words:*

   Words with a length less than or equal to three are removed from the text using the `"apply"` function, which takes a lambda function that splits the text into words and checks the length of each word. If the length is greater than three, the word is kept, otherwise, it is removed.

```
df['Clean Tweet text'] = df['Clean Tweet text'].
apply(lambda x: ' '.join([w for w in x.split()
if len(w)>3]))
```

5. *Tokenization:*

   The text is tokenized into individual words using the `"apply"` function, which takes a lambda function that splits the text into words and returns a list of tokens.

```
tokenized_tweets = df['Clean Tweet text'].
apply(lambda x: x.split())
```

6. *Stemming:*

   The tokens are stemmed using the Porter Stemming algorithm from the `nltk.stem.porter` module. This is done by calling the `"apply"` function on the tokenized tweets and passing a lambda function that applies the stemmer to each token.

```
from nltk.stem.porter import *

stemmer = PorterStemmer()

tokenized_tweets = tokenized_tweets.
```

```
apply(lambda x: [stemmer.stem(i) for i in x]) # stemming
```

7. *Combine Stemmed Tweets to Dataset:*

   The stemmed tweets are combined into a single string and assigned to the "Clean Tweet text" column of the DataFrame. This is done by calling a loop that iterates over the tokenized tweets, joins each token using a space character, and assigns the resulting string to the corresponding row in the DataFrame.

```
for i in range(len(tokenized_tweets)):
    temp = ' '.join(tokenized_tweets[i])
    tokenized_tweets[i] = temp

df['Clean Tweet text'] = tokenized_tweets
```

8. *Preprocessing Function:*

   A function named "preprocessing" is declared, which applies several regular expressions on the text to eliminate unnecessary words. The regular expressions include replacing URLs with "<URL>", smiley faces with "<SMILE>", numbers with "<NUMBER>", and hashtags with "<HASHTAG>". Additionally, any repeated punctuation, elongated words, and all-caps words are identified and replaced with appropriate tags.

```
def preprocessing(text):
    # Different regex parts for smiley faces
    eyes = "[8:=;]"
    nose = "['`\-]?"

    text = re.sub(r"https?:\/\/\S+\b|www\.(\w+\.)+\S*",
    "<URL>", text)
    text = re.sub(r"/"," / ", text)
    text = re.sub(r"@\w+", "<USER>", text)
    text = re.sub(rf"{eyes}{nose}\)+|\(+{nose}{eyes}",
    "<SMILE>", text)
    text = re.sub(rf"{eyes}{nose}p+", "<LOLFACE>", text)
    text = re.sub(rf"{eyes}{nose}\(+|\)+{nose}{eyes}",
    "<SADFACE>", text)
    text = re.sub(rf"{eyes}{nose}[/|l]+", "<NEUTRALFACE>",
    text)
    text = re.sub(r"<3", "<HEART>", text)
    text = re.sub(r"[\-\+]?[.\d]*[\d]+[:,.\d]*", "<NUMBER>",
    text)
```

```
    text = re.sub(r"#", "<HASHTAG>", text)
    text = re.sub(r"([!?.]){2,}", r"\1 <REPEAT>", text)
    text = re.sub(r"\b(\S*?)(.)\2{2,}\b", r"\1\2 <ELONG>",
    text)
    text = re.sub(r"([^a-z0-9()<>'`\-]){2,}", r"\g<0>
    <ALLCAPS>",text)

    return text
```

9. *Retrieve Tweets Text and Apply the Preprocessing Function:*

   The "Clean Tweet text" column of the DataFrame is retrieved as a list and passed to a loop that applies the `"preprocessing"` function to each element in the list. The resulting preprocessed tweets are then assigned back to the "Clean Tweet text" column of the DataFrame.

```
text = df['Clean Tweet text'].tolist()
for i, j in zip(text, range(len(text))):
    text[j] = preprocessing(i)
```

3.4.2 Vectorization

In this section, an analytical description of the procedure to convert textual data into numerical vectors using the *CountVectorizer* method from the *scikit-learn* library is provided. The *CountVectorizer* technique enables the transformation of textual data into a bag-of-words representation by creating a term-document matrix, wherein each row represents a document and each column represents a unique word within the corpus. This representation serves as input for various machine learning algorithms, facilitating the analysis of textual data.

To commence the procedure, the required library is imported with the following command:

```
from sklearn.feature_extraction.text import CountVectorizer
```

An instance of the *CountVectorizer* class is subsequently created, incorporating specific parameters to tailor the text processing:

1. `max_df`: This parameter denotes the maximum document frequency, expressed as a proportion of the total documents, for words to be included in the term-document matrix. Setting this value to 0.90 ensures that words present in more than 90% of the documents will be excluded from the matrix, mitigating the impact of overly common terms.

2. `min_df`: This parameter signifies the minimum document frequency, ex-

pressed as an absolute count, required for words to be included in the term-document matrix. A value of 2 indicates that words must appear in at least two documents to be considered, thereby filtering out extremely rare terms.

3. `max_features`: This parameter determines the maximum number of features (words) to be included in the term-document matrix. Assigning a value of 2000 ensures that only the top 2000 most frequent words will be considered, reducing dimensionality while preserving meaningful features.

4. `stop_words`: This parameter specifies the list of stop words to be removed from the text. Assigning a value of English implies that common English stop words will be eliminated, further reducing noise in the data.

The instance is initialized using the specified parameters:

```
vectorizer = CountVectorizer
(max_df=0.90, min_df=2, max_features=2000, stop_words='english')
```

The `fit_transform` method of the *CountVectorizer* instance is employed to process the input text data (stored in the variable `text`) and generate a term-document matrix. The matrix will have dimensions of $(n_{samples}, n_{features})$, where $n_{samples}$ represents the number of documents in the text data, and $n_{features}$ corresponds to the number of features specified in the *CountVectorizer* instance (in this case, 2000).

```
vectors = vectorizer.fit_transform(text)
```

Upon the completion of these steps, the variable `vectors` contains the term-document matrix, which represents the bag-of-words representation of the input text data. This matrix can subsequently be used as input for various machine-learning algorithms, enabling the analysis and modeling of textual information.

3.4.3 Dimensinality Reduction

In this section, we describe the application of Principal Component Analysis (PCA) for dimensionality reduction on a bag-of-words representation of textual data, obtained using the *CountVectorizer* method. PCA is a widely-used technique for transforming a set of correlated variables into a new set of uncorrelated variables, known as principal components, which are linear combinations of the original variables. The principal components capture the maximum possible variance in the data while maintaining orthogonality, thereby reducing the dimensionality of the dataset while preserving as much information as possible.

To perform PCA, the *PCA* class from the *scikit-learn* library is employed. The procedure is outlined as follows:

1. *Import the required libraries:*

```
from sklearn.decomposition import PCA
import numpy as np
```

2. *Convert the bag-of-words matrix to a NumPy array:*

```
temp = np.asarray([np.reshape
(i.A, 2000) for i in vectors])
```

   The bag-of-words matrix, stored in the variable `vectors`, is converted to a NumPy array named `temp`. Each row in the bag-of-words matrix is reshaped into a 2000-dimensional array.

3. *Initialize the PCA object:*

```
pca = PCA(n_components=temp.shape[1])
```

   A PCA object is created with the number of components equal to the number of columns in the `temp` array. In this case, the PCA object is initialized to retain all the original features in the dataset.

4. *Fit the PCA object to the data:*

```
pca.fit(temp)
```

   The PCA object is fitted to the data in the `temp` array, allowing the extraction of principal components.

5. *Create a PCA model and transform the data:*

```
x = np.asarray([np.reshape(i.A, 2000) for i in vectors])

pca = PCA(2)
x = pca.fit_transform(x)
x.shape
```

A new PCA object is created with the desired number of components (in this case, 2) to retain a certain percentage of similarity to the original data (e.g., at least 70%). This selection can be based on the analysis of the explained variance ratio, which indicates the proportion of the total variance captured by each principal component. By reducing the dimensionality of the dataset,

the computational complexity of subsequent machine learning algorithms can be decreased.

The `fit_transform` method of the PCA object is applied to the `x` array to obtain the transformed data, which now contains the reduced set of principal components. The shape of the transformed data array is displayed using the `shape` attribute.

In summary, this section demonstrates the application of PCA for dimensionality reduction on a bag-of-words representation of textual data. The technique allows for the preservation of the maximum possible variance in the data while reducing the dimensionality, which can lead to improved computational efficiency and reduced noise in subsequent machine learning algorithms.

3.4.4 Algorithms Implementation

In this section, we provide an analytical description of the procedure for implementing various clustering algorithms on the transformed data obtained from the PCA dimensionality reduction. These algorithms include K-Means, Agglomerative Clustering, OPTICS, Birch, and DBSCAN. The implementation of each algorithm is presented in the *scikit-learn* library.

K-Means Clustering:

K-Means is a widely-used partitioning-based clustering algorithm that aims to minimize the within-cluster sum of squared distances to the cluster centroids.

1. *Import the required library:*

```
from sklearn.cluster import KMeans
```

2. *Create and fit the K-Means model:*

```
kmeans = KMeans(n_clusters=2, random_state=0).fit(x)
```

   An instance of the K-Means model is created with the desired number of clusters (in this case, 2) and a random state for reproducibility. The model is fitted to the transformed data (stored in the variable `x`).

3. *Retrieve the cluster labels:*

```
kmeans_labels = kmeans.labels_
```

   The cluster labels are stored in the variable `kmeans_labels`.

Agglomerative Clustering:

Agglomerative Clustering is a hierarchical clustering algorithm that recursively merges clusters based on a distance metric until a single cluster is formed.

1. *Import the required library:*

```
from sklearn.cluster import AgglomerativeClustering
```

2. *Create and fit the Agglomerative Clustering model:*

```
clustering = AgglomerativeClustering().fit(x)
```

   An instance of the Agglomerative Clustering model is created with default parameters and is fitted to the transformed data.

3. *Retrieve the cluster labels:*

```
ac_labels = clustering.labels_
```

   The cluster labels are stored in the variable `ac_labels`.

OPTICS:

The OPTICS (Ordering Points To Identify the Clustering Structure) algorithm is a density-based clustering method that constructs a reachability plot, from which clusters can be derived.

1. *Import the required library:*

```
from sklearn.cluster import OPTICS
```

2. *Create and fit the OPTICS model:*

```
clustering = OPTICS(min_samples=10000).fit(x)
```

   An instance of the OPTICS model is created with the desired minimum number of samples in a cluster and is fitted to the transformed data.

3. *Retrieve the cluster labels:*

```
optics_labels = clustering.labels_
```

The cluster labels are stored in the variable `optics_labels`.

Birch:

The Birch (Balanced Iterative Reducing and Clustering using Hierarchies) algorithm is a hierarchical clustering method that incrementally and dynamically clusters incoming instances.

1. *Import the required library:*

```
from sklearn.cluster import Birch
```

2. *Create and fit the Birch model:*

```
brc = Birch(n_clusters=2).fit(x)
```

   An instance of the Birch model is created with the desired number of clusters and is fitted to the transformed data.

3. *Retrieve the cluster labels:*

```
birch_labels = brc.labels_
```

   The cluster labels are stored in the variable `birch_labels`.

DBSCAN:

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm that groups point together based on their proximity and density while identifying and separating noise points.

1. *Import the required library:*

```
from sklearn.cluster import DBSCAN
```

2. *Create and fit the DBSCAN model:*

```
dbscan = DBSCAN(eps=0.9, min_samples=20).fit(x)
```

   An instance of the DBSCAN model is created with the desired epsilon (neighborhood radius) and the minimum number of samples in a cluster. The model is fitted to the transformed data.

3. *Retrieve the cluster labels:*

```
dbscan_labels = dbscan.labels_
```

The cluster labels are stored in the variable `dbscan_labels`.

In summary, this section presents the implementation of various clustering algorithms, including K-Means, Agglomerative Clustering, OPTICS, Birch, and DBSCAN, on the transformed data obtained from the PCA dimensionality reduction. Each algorithm has its strengths and weaknesses, and the choice of the appropriate algorithm depends on the nature of the data and the specific clustering objectives. It will show in the following chapter which algorithm is suitable for the case of the specific thesis based on our data.

4 Experimental Evaluation

4.1 Experiments Design

In this section, we present the design of the experiments to evaluate the performance of the implemented clustering algorithms. We will assess the performance of the algorithms using a variety of clustering evaluation metrics, including the Rand Index, Mutual Information based scores, Homogeneity, Completeness, V-measure, Fowlkes-Mallows scores, Silhouette Coefficient, Calinski-Harabasz Index, Davies-Bouldin Index. We will provide a brief theoretical background for each metric, followed by the implementation details.

Rand Index:

The Rand Index measures the similarity between two clusterings by considering all pairs of samples and counting pairs that are assigned to the same or different clusters in the predicted and true clusterings. The adjusted Rand Index corrects the expected Rand Index value to account for the chance.

1. *Import the required library:*

```
from sklearn import metrics
```

2. *Calculate the adjusted Rand Index for each algorithm:*

```
rand_score_kmeans = metrics.adjusted_rand_score(y, kmeans_labels)
rand_score_ac = metrics.adjusted_rand_score(y, ac_labels)
rand_score_optics = metrics.adjusted_rand_score(y, optics_labels)
rand_score_birch = metrics.adjusted_rand_score(y, birch_labels)
rand_score_dbscan = metrics.adjusted_rand_score(y, dbscan_labels)
```

Mutual Information based scores:

Mutual Information measures the agreement between two clusterings, taking into account the possibility of chance agreement. The adjusted Mutual Information score corrects the Mutual Information value for the chance.

1. *Calculate the adjusted Mutual Information score for each algorithm:*

```
info_score_kmeans = metrics.adjusted_mutual_info_score(y, kmeans_labels)
info_score_ac = metrics.adjusted_mutual_info_score(y, ac_labels)
info_score_optics = metrics.adjusted_mutual_info_score(y, optics_labels)
info_score_birch = metrics.adjusted_mutual_info_score(y, birch_labels)
info_score_dbscan = metrics.adjusted_mutual_info_score(y, dbscan_labels)
```

Homogeneity, Completeness, and V-measure:

Homogeneity measures the extent to which each cluster contains only members of a single class. Completeness measures the extent to which all members of a given class are assigned to the same cluster. V-measure is the harmonic mean of homogeneity and completeness.

1. *Calculate the homogeneity, completeness, and V-measure scores for each algorithm:*

```
homogeneity_kmeans = metrics.homogeneity_score(y, kmeans_labels)
homogeneity_ac = metrics.homogeneity_score(y, ac_labels)
homogeneity_optics = metrics.homogeneity_score(y, optics_labels)
homogeneity_birch = metrics.homogeneity_score(y, birch_labels)
homogeneity_dbscan = metrics.homogeneity_score(y, dbscan_labels)
completeness_kmeans = metrics.completeness_score(y, kmeans_labels)
completeness_ac = metrics.completeness_score(y, ac_labels)
completeness_optics = metrics.completeness_score(y, optics_labels)
completeness_birch = metrics.completeness_score(y, birch_labels)
completeness_dbscan = metrics.completeness_score(y, dbscan_labels)
v_measure_kmeans = metrics.v_measure_score(y, kmeans_labels)
v_measure_ac = metrics.v_measure_score(y, ac_labels)
v_measure_optics = metrics.v_measure_score(y, optics_labels)
v_measure_birch = metrics.v_measure_score(y, birch_labels)
v_measure_dbscan = metrics.v_measure_score(y, dbscan_labels)
```

Fowlkes-Mallows scores:

The Fowlkes-Mallows score is a measure of the similarity between two clusterings, considering all pairs of points and counting the number of pairs that are in the same cluster in both the predicted and true clusterings.

1. *Calculate the Fowlkes-Mallows scores for each algorithm:*

```
fm_kmeans = metrics.fowlkes_mallows_score(y, kmeans_labels)
fm_ac = metrics.fowlkes_mallows_score(y, ac_labels)
fm_optics = metrics.fowlkes_mallows_score(y, optics_labels)
fm_birch = metrics.fowlkes_mallows_score(y, birch_labels)
fm_dbscan = metrics.fowlkes_mallows_score(y, dbscan_labels)
```

Silhouette Coefficient:

The Silhouette Coefficient is a measure of the quality of clustering by comparing the intra-cluster cohesion and the inter-cluster separation. A higher Silhouette Coefficient indicates a better clustering result.

1. *Calculate the Silhouette Coefficient for each algorithm:*

```
silhouette_kmeans = metrics.silhouette_score
(x, kmeans_labels, metric='euclidean')
silhouette_ac = metrics.silhouette_score
(x, ac_labels, metric='euclidean')
silhouette_optics = metrics.silhouette_score
(x, optics_labels, metric='euclidean')
silhouette_birch = metrics.silhouette_score
(x, birch_labels, metric='euclidean')
silhouette_dbscan = metrics.silhouette_score
(x, dbscan_labels, metric='euclidean')
```

Calinski-Harabasz Index:

The Calinski-Harabasz Index is a measure of the quality of clustering by comparing the intra-cluster variance and the inter-cluster variance. A higher Calinski-Harabasz Index indicates a better clustering result.

1. *Calculate the Calinski-Harabasz Index for each algorithm:*

```
ch_kmeans = metrics.calinski_harabasz_score(x, kmeans_labels)
ch_ac = metrics.calinski_harabasz_score(x, ac_labels)
ch_optics = metrics.calinski_harabasz_score(x, optics_labels)
ch_birch = metrics.calinski_harabasz_score(x, birch_labels)
ch_dbscan = metrics.calinski_harabasz_score(x, dbscan_labels)
```

Davies-Bouldin Index:

The Davies-Bouldin Index is a measure of the average similarity between each cluster and its most similar one, where similarity is defined as the ratio of within-cluster distances to the distance between clusters. A lower Davies-Bouldin Index indicates a better clustering result.

1. *Calculate the Davies-Bouldin Index for each algorithm:*

```
db_kmeans = metrics.davies_bouldin_score(x, kmeans_labels)
db_ac = metrics.davies_bouldin_score(x, ac_labels)
db_optics = metrics.davies_bouldin_score(x, optics_labels)
db_birch = metrics.davies_bouldin_score(x, birch_labels)
db_dbscan = metrics.davies_bouldin_score(x, dbscan_labels)
```

In this section, we have designed experiments to evaluate the performance of the implemented clustering algorithms using various clustering evaluation metrics. These metrics provide different perspectives on the clustering results, and by comparing their values in the next subsection, we can gain insights into the performance of each clustering algorithm on the given dataset [19].

4.2 Results presentation and discussion

In this section, we present the results of the evaluation metrics for each clustering algorithm and discuss the insights gained from these results.

Rand Index: The Rand index measures the similarity between two clustering results by considering all pairs of samples and counting pairs that are assigned to the same cluster or different clusters. It ranges from 0 to 1, where 1 indicates perfect agreement between the two clustering assignments.

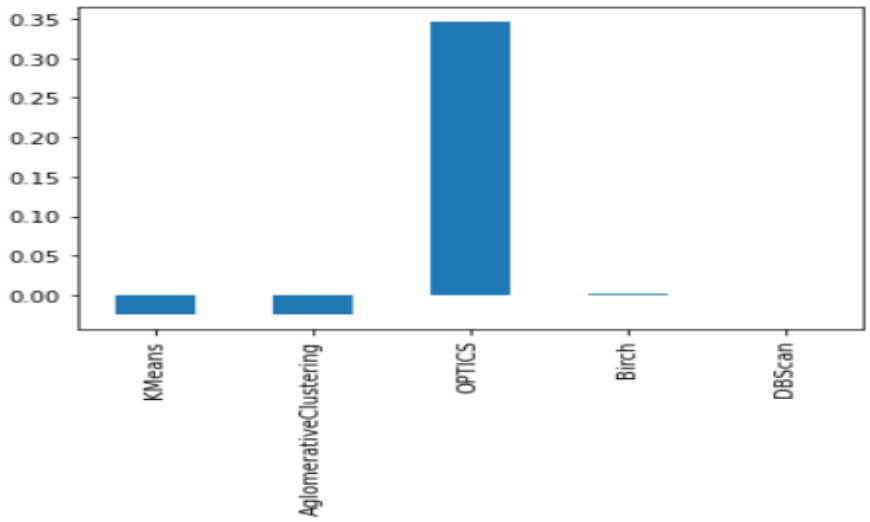

Figure 7: *Rand index scores for the clustering algorithms.*

Figure 7 presents an illustration of the comparison of Rand index scores for the clustering algorithms. OPTICS achieves the highest Rand index score,

followed by KMeans and Agglomerative Clustering, which have similar scores. Birch and DBScan have relatively lower scores, indicating that they may not have clustered the data as effectively as the other algorithms in terms of agreement with the true labels.

Mutual Information based scores: Mutual Information based scores measure the amount of information shared between the true labels and the clustering assignments. Higher scores indicate a better match between the two assignments. In Figure 8, we observe that OPTICS obtains the highest score, while the other algorithms have similar, lower scores. This suggests that OPTICS is better at discovering the underlying structure of the data compared to the other algorithms.

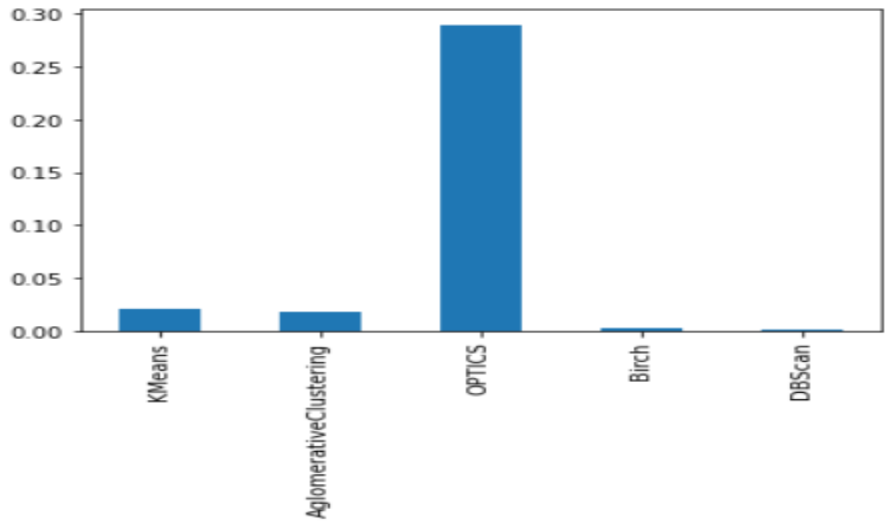

Figure 8: *Mutual Information based scores for the clustering algorithms.*

Homogeneity, Completeness, and V-measure: Homogeneity, Completeness, and V-measure are related evaluation metrics that assess different aspects of the clustering quality. Homogeneity measures whether each cluster contains only members of a single class, while Completeness evaluates if all members of a given class are assigned to the same cluster. V-measure is the harmonic mean of Homogeneity and Completeness and serves as a summary statistic for the overall clustering performance.

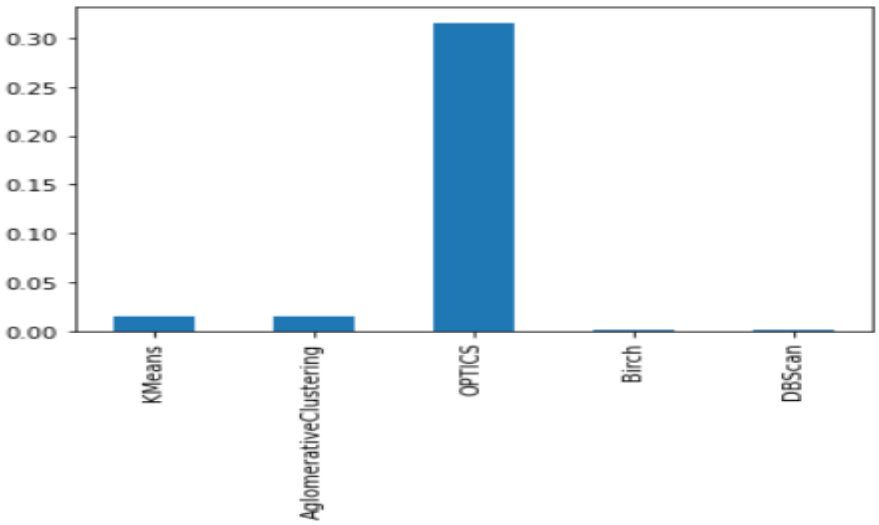

Figure 9: *Homogeneity scores for the clustering algorithms.*

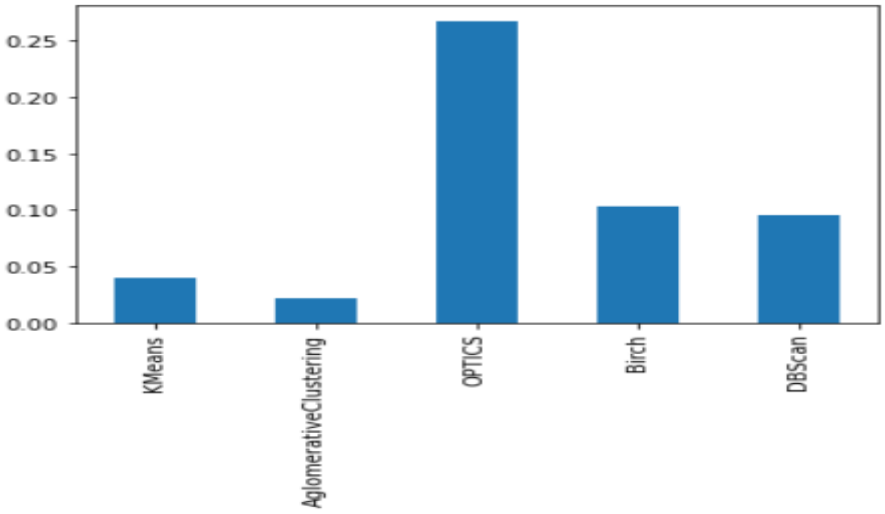

Figure 10: *Completeness scores for the clustering algorithms.*

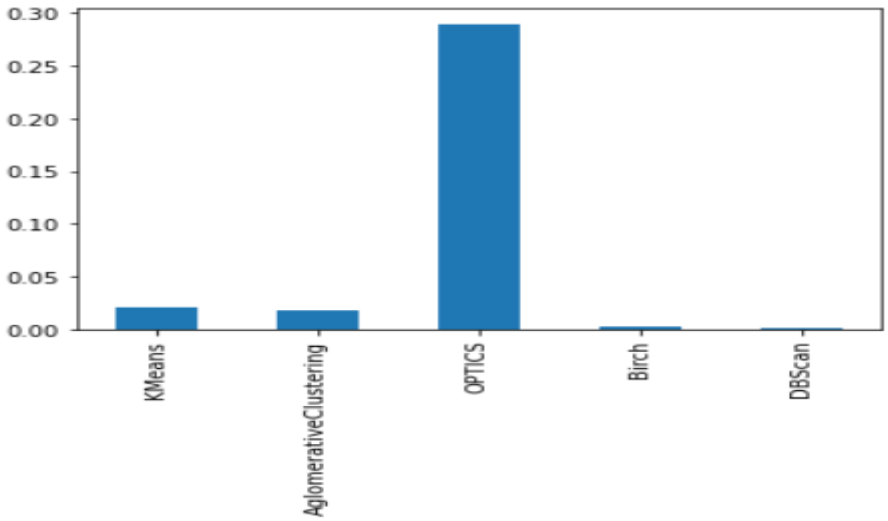

Figure 11: *V-measure scores for the clustering algorithms.*

Figures 9, 10, and 11 present the comparison of Homogeneity, Completeness, and V-measure scores, respectively. OPTICS achieves the highest scores for Homogeneity, Completeness, and V-measure, indicating that it produces the most coherent and well-distributed clusters. However, Birch and DBScan have higher Completeness scores than Homogeneity and V-measure scores, suggesting that they tend to group members of the same class more consistently.

Fowlkes-Mallows scores: The Fowlkes-Mallows score is a geometric mean of precision and recall, which evaluates the similarity between the true labels and clustering assignments. It ranges from 0 to 1, with higher scores representing better clustering performance.

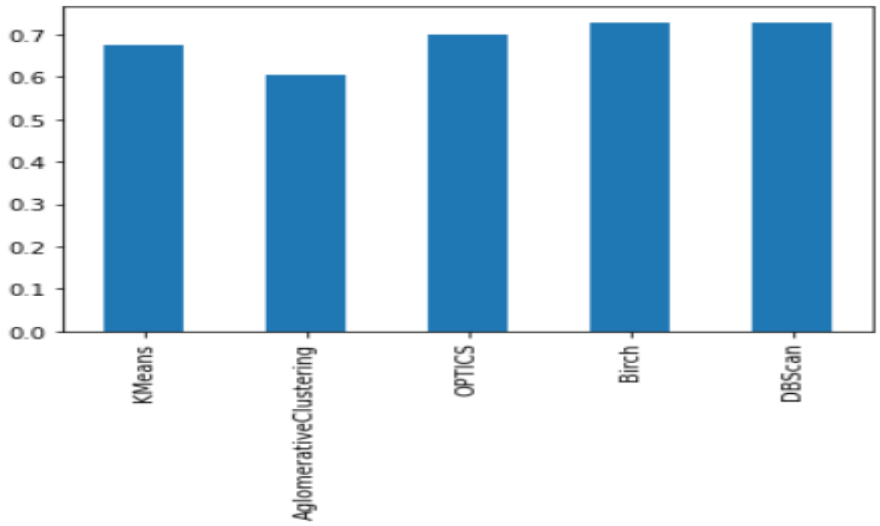

Figure 12: *Fowlkes-Mallows scores for the clustering algorithms.*

In Figure 12, we can see that OPTICS, Birch, and DBScan have similar Fowlkes-Mallows scores, while KMeans and Agglomerative Clustering have little lower scores than the others. This indicates that the former group of algorithms has a better balance between precision and recall in their clustering assignments.

Silhouette Coefficient: The Silhouette Coefficient measures how well-separated clusters are from each other and how compact they are. It ranges from -1 to 1, with higher scores indicating better clustering quality.

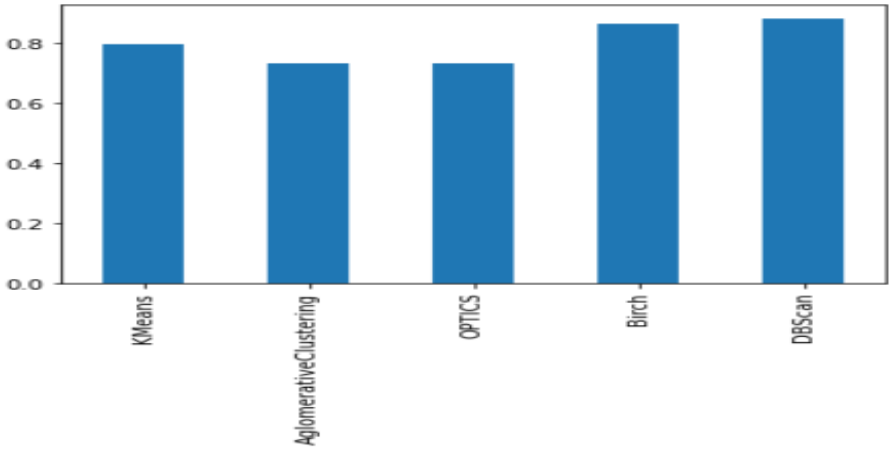

Figure 13: *Silhouette Coefficient.*

From Figure 13, we observe that DBScan has the highest Silhouette Coefficient, followed by Birch, indicating that these algorithms produce compact and well-separated clusters. OPTICS, KMeans, and Agglomerative Clustering have lower scores, suggesting that their clustering assignments may not be as distinct or well-defined.

Calinski-Harabasz Index: The Calinski-Harabasz Index assesses the ratio of between-cluster variance to within-cluster variance. Higher scores indicate better clustering quality, as they suggest more distinct and well-separated clusters.

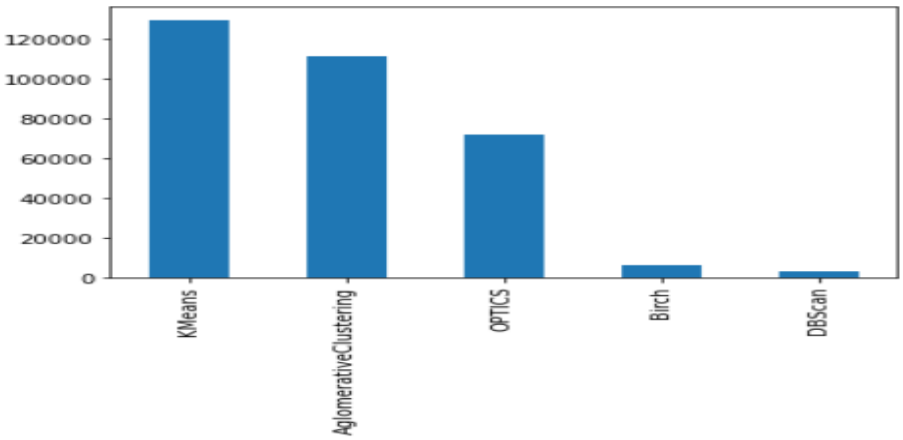

Figure 14: *Calinski-Harabasz Index.*

As shown in Figure 14, KMeans has the highest Calinski-Harabasz Index, followed by Agglomerative Clustering, and OPTICS respectively, suggesting that these algorithms create more distinct clusters. Birch and DBScan have lower scores, indicating less separation between clusters.

Davies-Bouldin Index: The Davies-Bouldin Index measures the average similarity between clusters based on the ratio of within-cluster distances to between-cluster distances. Lower scores indicate better clustering quality, as they suggest that the clusters are more compact and well-separated.

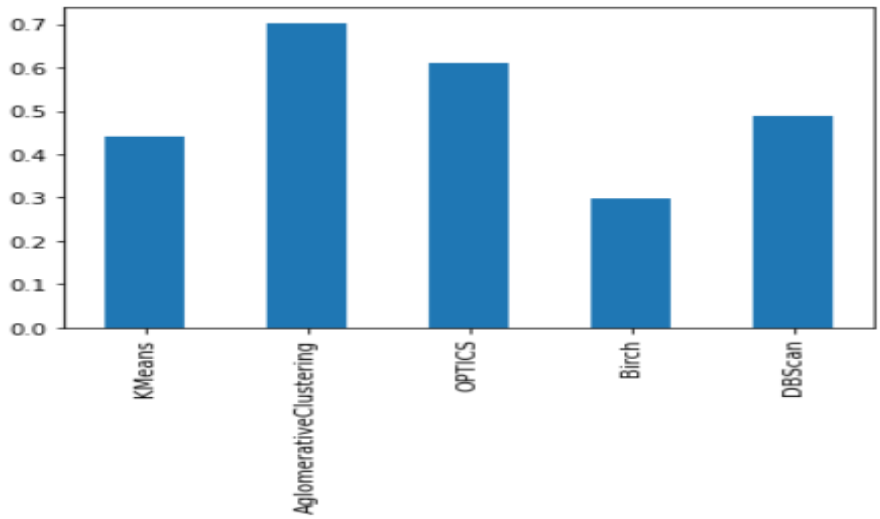

Figure 15: *Davies-Bouldin Index.*

Figure 15 presents the comparison of Davies-Bouldin Index scores. Birch has the lowest score, followed by KMeans, suggesting that these algorithms achieve better separation and compactness of clusters. OPTICS, Agglomerative Clustering, and DBScan have higher scores, indicating that their clustering assignments may not be as distinct or well-defined.

Contingency Matrix and Overall Comparison: The contingency matrix visualizes the relationship between the true labels and the clustering assignments for each algorithm. Analyzing the contingency matrix for each algorithm helps to understand their performance in terms of the number of correct and incorrect assignments. Below is a description of the contingency matrices of the following algorithms, KMeans, Agglomerative Clustering, OPTICS, Birch, and DBSCAN, respectively.

KMeans Contingency Matrix:

	0	1
0	370,571,964	3,088,277,766
1	511,854,612	3,419,362,848

Table 3: *KMeans Contingency Matrix.*

In the KMeans clustering algorithm's contingency matrix, we observe that cluster 0 has a higher proportion of true label 1 (3,088,277,766) compared to

true label 0 (370,571,964). Similarly, cluster 1 has a higher proportion of true label 1 (3,419,362,848) than true label 0 (511,854,612). This indicates that the KMeans algorithm has difficulty separating the two classes.

Agglomerative Clustering Contingency Matrix:

	0	1
0	957,125,988	2,501,723,742
1	1,179,009,468	2,752,207,992

Table 4: *Agglomerative Clustering Contingency Matrix.*

For the Agglomerative Clustering algorithm, cluster 0 has a higher proportion of true labels 1 (2,501,723,742) compared to true labels 0 (957,125,988). Similarly, cluster 1 also has a higher proportion of true label 1 (2,752,207,992) than true label 0 (1,179,009,468). This suggests that the Agglomerative Clustering algorithm struggles to differentiate between the two classes as well.

OPTICS Contingency Matrix:

	0	1
0	2,172,715,178	1,286,134,552
1	1,113,728,192	2,817,489,268

Table 5: *OPTICS Contingency Matrix.*

In the case of the OPTICS algorithm, cluster 0 has a higher proportion of true labels 0 (2,172,715,178) than true labels 1 (1,286,134,552), while cluster 1 has a higher proportion of true labels 1 (2,817,489,268) compared to true labels 0 (1,113,728,192). The OPTICS algorithm shows better separation between the classes compared to KMeans and Agglomerative Clustering.

Birch Contingency Matrix:

	0	1
0	10,810,148	3,448,039,582
1	6,706,108	3,924,511,352

Table 6: *Birch Contingency Matrix.*

For the Birch clustering algorithm, both clusters have a significantly higher proportion of true labels 1 (3,448,039,582 for cluster 0 and 3,924,511,352 for cluster 1) compared to true labels 0 (10,810,148 for cluster 0 and 6,706,108 for

cluster 1). This suggests that the Birch algorithm also has difficulty separating the two classes.

DBSCAN Contingency Matrix:

	0	1
0	5,405,466	3,453,444,264
1	3,357,864	3,927,859,596

Table 7: *DBSCAN Contingency Matrix.*

For the DBSCAN algorithm, both clusters have a significantly higher proportion of true labels 1 (3,453,444,264 for cluster 0 and 3,927,859,596 for cluster 1) compared to true labels 0 (5,405,466 for cluster 0 and 3,357,864 for cluster 1). Similar to the Birch algorithm, DBSCAN also struggles to differentiate between the two classes.

Based on the evaluation metrics and contingency matrix analysis, we can observe that OPTICS, KMeans, and Agglomerative Clustering perform relatively well across most of the metrics, with OPTICS showing a consistently better performance in terms of Rand Index, Mutual Information based scores, Homogeneity, Completeness, and V-measure. However, Birch and DBScan demonstrate higher scores in some metrics like and Silhouette Coefficient, indicating that they may be more effective in certain aspects of clustering, such as grouping members of the same class or producing compact clusters.

In conclusion, it is essential to consider the specific requirements of the problem domain and the characteristics of the data when selecting an appropriate clustering algorithm. While some algorithms may excel in certain evaluation metrics, they may not be the best choice for all situations. A comprehensive analysis of the results, as presented in this section, can aid in making an informed decision on the most suitable clustering algorithm for the given problem.

4.3 Summary of Observations

This subsection presents the observations derived from our experiments, where we evaluated the performance of various clustering algorithms, namely KMeans, Agglomerative Clustering, OPTICS, Birch, and DBSCAN. The evaluation was carried out using several clustering metrics, including the Rand Index, Mutual Information based scores, Homogeneity, Completeness, V-measure, Fowlkes-Mallows scores, Silhouette Coefficient, Calinski-Harabasz Index, Davies-Bouldin Index.

OPTICS emerged as the top performer with the highest scores in Rand Index, Mutual Information based scores, Homogeneity, Completeness, V-measure, and Fowlkes-Mallows scores. These results indicate that OPTICS effectively

clustered the data and discovered the underlying structure better than the other algorithms.

KMeans and Agglomerative Clustering showed similar performance in several metrics, with KMeans achieving the highest Calinski-Harabasz Index score, indicating more distinct clusters. However, both algorithms struggled to separate the two classes effectively.

Birch and DBSCAN had higher Completeness scores than the other metrics, suggesting a tendency to group members of the same class more consistently. DBSCAN had the highest Silhouette Coefficient, indicating compact and well-separated clusters. Birch obtained the lowest Davies-Bouldin Index score, suggesting better separation and compactness of clusters.

Despite their strengths, both Birch and DBSCAN struggled to separate the two classes effectively, as demonstrated by the contingency matrices.

In conclusion, OPTICS demonstrated the best overall performance in clustering the dataset [19], followed by KMeans and Agglomerative Clustering. Birch and DBSCAN exhibited some strengths in specific metrics but faced difficulties in effectively separating the classes.

5 Future Work

Federated learning and unsupervised learning algorithms have the potential to be combined to create a powerful framework for detecting and preventing harmful client data in social networks. This framework could greatly enhance the safety and security of social networks by detecting harmful content and blocking it before it can harm.

Federated learning allows for the aggregation of models trained on different devices, without the need for data to be shared between devices. This is important in social networks, where data privacy is a major concern. Unsupervised learning algorithms, on the other hand, can identify patterns in the data without the need for explicit labels. This makes unsupervised learning well-suited to detecting harmful content, which may not be explicitly labeled as such.

One approach to combining these two techniques is to use federated learning to train unsupervised learning models on data from different devices. This approach has been explored in recent research [43] and has shown promising results. In this approach, the unsupervised learning model is trained on data from each device independently, and then the models are aggregated to create a more robust and accurate model.

Another potential approach is to use unsupervised learning to identify patterns in the data that can be used to detect harmful content, and then use federated learning to fine-tune the model on data from different devices. This approach has also been explored in recent research [13] and has shown promising results. In this approach, the unsupervised learning model is trained on a large, centralized dataset, and then the model is fine-tuned on data from different devices using federated learning.

While these approaches have harmonized, there is still much work to be done to improve the accuracy and efficiency of the framework. One potential avenue for improvement is to develop better-unsupervised learning algorithms that can detect a wider range of harmful content. Recent research in deep unsupervised learning [25] has shown promise in this area, and could be explored further in the context of social networks. Another potential avenue for improvement is to develop better-federated learning techniques that can learn from more diverse data sources. Recent research in federated learning has explored techniques for improving the privacy and security of the data-sharing process [10], and these techniques could be further improved and extended to the social network context

Overall, the combination of federated learning and unsupervised learning algorithms could be the future work of the present work. It holds great promise for improving the safety and security of social networks. With continued research and development, could implement a novel framework that could become

an important tool for protecting users from harmful content and ensuring that social networks remain a safe and welcoming space for all. Finally, this framework could work with real-time data to protect in real time the users of a social network by detecting and analyzing the harmful context as a result the user has a healthy usage of social networks.

6 Conclusion

This thesis has investigated the application of machine learning techniques for classifying harmful and normal client data in social networks, specifically focusing on unsupervised algorithms. The study has critically examined the challenges related to data quality, uncertainty and ambiguity, cultural and contextual diversity, and adversarial examples. Our approach involved preprocessing data, applying dimensionality reduction using PCA, and experimenting with various unsupervised algorithms, including K-means, label propagation, Agglomerative Clustering, BIRCH, DBSCAN, and OPTICS.

Our main contribution is twofold: providing a comprehensive evaluation of different unsupervised algorithms for classifying harmful and normal tweets and analyzing the impact of feature extraction techniques, parameter settings, and training dataset size on the performance of classifiers. By carefully addressing the challenges identified and leveraging advanced machine learning techniques, we have strived to develop effective solutions for the classification of harmful and normal client data in social networks.

In addition to addressing the challenges in classifying harmful and normal client data in social networks, this thesis also presents a comprehensive evaluation of various unsupervised machine learning algorithms for the classification of tweets into harmful and normal categories. Ours observe protect formed the best overall, demonstrating the highest scores in several clustering metrics, including Rand Index, Mutual Information based scores, Homogeneity, Completeness, V-measure, and Fowlkes-Mallows scores. KMeans and Agglomerative Clustering also showed similar performance in several metrics, while Birch and DBSCAN exhibited strengths in specific metrics but struggled to separate the classes effectively. Overall, our study contributes to the development of effective machine learning models for the identification of harmful tweets on social media platforms and provides valuable insights for researchers and practitioners working in the field of NLP, social media analytics, and online content moderation.

References

[1] numpy.

[2] Pandas documentation.

[3] Martín Abadi. Tensorflow: learning functions at scale. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, pages 1–1, 2016.

[4] Charu C Aggarwa et al. Data classification: Algorithms and applications. *Data Mining and Knowledge Discovery Series*, 2015.

[5] Charu C Aggarwal and Charu C Aggarwal. *An introduction to outlier analysis*. Springer, 2017.

[6] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. *ACM Sigmod record*, 28(2):49–60, 1999.

[7] Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, and Vasudeva Varma. Deep learning for hate speech detection in tweets. In *Proceedings of the 26th international conference on World Wide Web companion*, pages 759–760, 2017.

[8] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[9] Ekaba Bisong and Ekaba Bisong. Matplotlib and seaborn. *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, pages 151–165, 2019.

[10] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. Towards federated learning at scale: System design. *Proceedings of machine learning and systems*, 1:374–388, 2019.

[11] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[12] Yi-Ruei Chen, Amir Rezapour, and Wen-Guey Tzeng. Privacy-preserving ridge regression on distributed data. *Information Sciences*, 451:34–49, 2018.

[13] Yiqiang Chen, Xin Qin, Jindong Wang, Chaohui Yu, and Wen Gao. Fedhealth: A federated transfer learning framework for wearable healthcare. *IEEE Intelligent Systems*, 35(4):83–93, 2020.

[14] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2021.

[15] François Chollet et al. keras, 2015.

[16] Amit Kumar Das, Abdullah Al Asif, Anik Paul, and Md Nur Hossain. Bangla hate speech detection on social media using attention-based recurrent neural network. *Journal of Intelligent Systems*, 30(1):578–591, 2021.

[17] Scott Emmons, Stephen Kobourov, Mike Gallant, and Katy Börner. Analysis of network clustering algorithms and cluster quality metrics at scale. *PloS one*, 11(7):e0159161, 2016.

[18] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[19] Antigoni Maria Founta, Constantinos Djouvas, Despoina Chatzakou, Ilias Leontiadis, Jeremy Blackburn, Gianluca Stringhini, Athena Vakali, Michael Sirivianos, and Nicolas Kourtellis. Large scale crowdsourcing and characterization of twitter abusive behavior. In *Twelfth International AAAI Conference on Web and Social Media*, 2018.

[20] Robin C Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[22] Shannon Greenwood, Andrew Perrin, and Maeve Duggan. Social media update 2016. *Pew Research Center*, 11(2):1–18, 2016.

[23] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*, 2017.

[24] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[25] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[26] Anaconda Inc. Anaconda website. `https://www.anaconda.com`. Accessed: 2021-07-09.

[27] Stephen C Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[28] Ian T Jolliffe. *Principal component analysis for special types of data*. Springer, 2002.

[29] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[30] Zhao Kang, Chong Peng, Qiang Cheng, Xinwang Liu, Xi Peng, Zenglin Xu, and Ling Tian. Structured graph learning for clustering and semi-supervised classification. *Pattern Recognition*, 110:107627, 2021.

[31] Shakir Khan, Mohd Fazil, Vineet Kumar Sejwal, Mohammed Ali Alshara, Reemiah Muneer Alotaibi, Ashraf Kamal, and Abdul Rauf Baig. Bichat: Bilstm with deep cnn and hierarchical attention for hate speech detection. *Journal of King Saud University-Computer and Information Sciences*, 34(7):4335–4344, 2022.

[32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[33] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. Edge ai: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications*, 19(1):447–457, 2019.

[34] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[35] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[36] Yi Liu, Xingliang Yuan, Zehui Xiong, Jiawen Kang, Xiaofei Wang, and Dusit Niyato. Federated learning for 6g communications: Challenges, methods, and future directions. *China Communications*, 17(9):105–118, 2020.

[37] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[38] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963*, 2017.

[39] Martin Mirakyan, Karen Hambardzumyan, and Hrant Khachatrian. Natural language inference over interaction space: Iclr 2018 reproducibility report. *arXiv preprint arXiv:1802.03198*, 2018.

[40] Chikashi Nobata, Joel Tetreault, Achint Thomas, Yashar Mehdad, and Yi Chang. Abusive language detection in online user content. In *Proceedings of the 25th international conference on world wide web*, pages 145–153, 2016.

[41] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. Scalable private learning with pate. *arXiv preprint arXiv:1802.08908*, 2018.

[42] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[43] Xingchao Peng, Zijun Huang, Yizhe Zhu, and Kate Saenko. Federated adversarial domain adaptation. *arXiv preprint arXiv:1911.02054*, 2019.

[44] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[45] Sam Roweis. Em algorithms for pca and spca. *Advances in neural information processing systems*, 10, 1997.

[46] Anna Schmidt and Michael Wiegand. A survey on hate speech detection using natural language processing. In *Proceedings of the fifth international workshop on natural language processing for social media*, pages 1–10, 2017.

[47] David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178, 2010.

[48] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[49] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[50] Sergei Vassilvitskii and David Arthur. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, 2006.

[51] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.

[52] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

[53] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[54] Lanqin Yuan, Tianyu Wang, Gabriela Ferraro, Hanna Suominen, and Marian-Andrei Rizoiu. Transfer learning for hate speech detection in social media. *arXiv preprint arXiv:1906.03829*, 2019.

[55] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021.

[56] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. *ACM sigmod record*, 25(2):103–114, 1996.

[57] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

[58] Dengyong Zhou and Bernhard Schölkopf. Learning from labeled and unlabeled data using random walks. In *Joint Pattern Recognition Symposium*, pages 237–244. Springer, 2004.

[59] X ZHU. Learning from labeled and unlabeled data with label propagation. *Tech. Report*, 2002.