NEWCASTLE UNIVERSITY

BACHELOR'S THESIS

Privacy-Preserving Support Vector Machine Based Approach for Ensuring Agricultural Sensor Network Integrity

Author:
Justina METRIKYTE

Supervisor:
Dr. Shishir NAGARAJA

A Thesis Submitted for the Completion of
Requirements for the Degree of
Bachelor of Science with Honours In Computer Science
(Security and Resilience)

May 2023
(14,683 words)

Declaration of Authorship

I, Justina METRIKYTE, declare that this thesis titled, "Privacy-Preserving Support Vector Machine Based Approach for Ensuring Agricultural Sensor Network Integrity" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a degree at this University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- All the figures, tables, and equations in this report are original and have not been taken from any other person's work, except where otherwise stated.

Abstract

Privacy-Preserving Support Vector Machine Based Approach for Ensuring Agricultural Sensor Network Integrity

This study investigates the application of Support Vector Machines (SVMs) and differentially private machine learning in addressing challenges in agricultural sensor networks. The key focus areas are anomaly detection and the safeguarding of the farmers' data used to train machine learning models, which help to ensure the integrity of agricultural sensor networks.

Illuminance data was gathered using a custom agricultural sensor box, enabling the creation of a simulated sensor network. This simulation reflects real-world interactions between sensor nodes, servers, and cloud-based machine learning systems. The study then established a simple SVM model to detect anomalies within this simulated agricultural sensor network.

Subsequently, privacy-preserving methods were incorporated. The first step involved introducing a method to add noise to the training data. This was followed by the implementation of a novel differentially private SVM, based on the work of Chaudhuri, Monteleoni, and Sarwate [24]. Next, membership inference attacks were conducted to evaluate potential privacy compromises under varying privacy budget values and provide essential information to users, enabling them to select suitable privacy parameters in order to balance their desired level of data privacy with the effectiveness of anomaly detection.

The research outcome is a simulated system that can potentially be applied to real agricultural sensor networks. This system allows farmers to define their privacy preferences and offers anomaly alerts. While this study focused on illuminance data, the methodology is extendable to broader datasets, highlighting the promising application of privacy preserving SVMs to safeguard both data integrity and privacy in agricultural sensor networks.

Acknowledgements

To begin with, I want to express my gratitude towards my project supervisor, Dr Shishir Nagaraja, for granting me the opportunity to partake in this project, helping me with the organisation of the project, and consistently providing valuable guidance.

I would also like to extend my appreciation to Dr Carl Dickinson for teaching me the fundamental electrical principles and soldering, as well as providing me with many helpful academic resources and tips.

Contents

List of Figures

List of Abbreviations

DP	**D**ifferential **P**rivacy
SVM	**S**upport **V**ector **M**achine
ASN	**A**gricultural **S**ensor **N**etwork
IDE	**I**ntegrated **D**evelopment **E**nviroment
DPSVM	**D**ifferentially **P**rivate **S**upport **V**ector **M**achine

List of Symbols

Support Vector Machines (SVMs)

w	normal vector
b	bias term
C	error term
λ	regularisation parameter

Differential Privacy (DP)

δ	failure probability
ϵ	privacy budget

Differentially Private SVM

z	residual
h	outlier detection threshold
α	normalising value
λ	regularisation parameter
$\mathbf{b}$	noise vector
T	vector transpose

Chapter 1

Introduction

1.1 Context and Motivation

In the past few years, there has been a significant increase in the use of sensor networks in agriculture due to the growing popularity of Smart and Precision Agriculture [1, 67], which rely on data-driven decision making to optimise crop yields and reduce input costs. These sensor networks often collect multifaceted environmental data, from weather patterns to soil conditions, informing automated decisions like irrigation and fertilisation.

However, the integrity of such systems is vulnerable to sensor data manipulation and malfunction. A compromised sensor network can mislead farmers or disrupt automated systems, such as watering or lighting, adversely affecting crop health.

Support Vector Machines (SVMs) are a type of machine learning algorithm that have proven effective in a diverse range of applications. They are particularly useful for anomaly detection within computer [34, 66, 73], medical [54], and smart grid networks [59, 16]. Given this proven efficacy, SVMs hold a significant promise for promoting integrity in agricultural sensor networks through reliable anomaly detection.

However, if attackers manage to infer the data that trained these models, it could precipitate a two-fold issue. Firstly, it could potentially undermine the trust of those whose data was used for training. Secondly, and perhaps more alarmingly, this acquired knowledge could enable malicious actors to bypass these anomaly detection systems, thereby rendering them ineffective.

Due to that, various privacy-preserving machine learning techniques have been proposed to address this. Homomorphic encryption, for example, performs computations on encrypted data and reveals predictions only through a decryption key, thereby securing original training data. However, in Smart Agriculture, where sensors generate vast amounts of real-time data [70], the computational overhead of this method often makes it impractical.

An alternate method, Federated Learning with Secure Aggregation, decentralises algorithm training across devices. These devices encrypt their model updates for server aggregation, protecting sensitive update information. However, this method demands considerable local data processing and device communication [42, 68], posing challenges for resource and memory limited ASNs [55].

Because of this, this paper aims to propose a solution to these problems by exploring a Support Vector Machine based approach to detect anomalous Illuminance data

in an agricultural sensor network. This approach involves using a differentially private technique, which introduces a certain amount of random noise into the machine learning process. This technique serves two purposes: preserving the overall system utility and making it more challenging for attackers to discern specifics about the individual data points used for algorithm training

SVMs are chosen for their proven efficacy in a variety of applications, including agricultural contexts [43, 35], and anomaly detection [34, 66, 73, 54, 59, 16]. Additionally, their resilience to noise [38] — a pertinent feature given the minor perturbations introduced by differential privacy techniques and the environmental fluctuations that agricultural sensor data is subject to. Concurrently, illuminance data is targeted due to its widespread use in Smart Agriculture farms' automated systems, such as irrigation and climate control, making its accuracy vital to plant health and growth.

1.2 Project Aim and Objectives

1.2.1 High Level Aim

The overall aim of this project is to create a prototype privacy preserving Support Vector Machine based solution for ensuring agricultural sensor network data integrity.

1.2.2 Objectives

The main objectives for completing this aim are:

Objective	**Explanation**
1. Analyse, summarise, and compare two published examples of how privacy preserving SVMs have been implemented	To implement a privacy preserving SVM solution, I will need to gain a better understanding about the theory behind it. To do this, I will analyse two papers on this topic by summarising the mentioned approaches and then comparing them. This will help me to find the most suitable approach for implementing the privacy preserving SVM that will be used to ensure sensor network data integrity. The objective will be complete once I compare (in the Background section of my thesis) two published examples of how a privacy preserving SVM can be created. This objective should be completed by the end of the research phase in my project.

Objective	**Explanation**
2. Develop an agricultural network simulation	To demonstrate the practical application and effectiveness of the proposed solution, it is important to simulate its operation within a real-world ASN. As such, I plan to develop a network simulation that mirrors the key components of these systems. This will include creating the client node/s to mimic the function of the sensor network node/s, the central server to which the nodes transmit their sensor data, and the cloud simulation with SVM models which replicates the role of the external cloud-based machine learning service often utilised in similar scenarios. By the completion of the first development phase, I aim to have this representative network simulation up and running, marking the achievement of this objective.
3. Assemble the "SquirrelBox" agricultural sensor box and employ it for the collection of illuminance data	The aim is to gather labelled illuminance data, both valid and invalid, which is critical for training the SVMs, given their basis in supervised learning. I intend to assemble an agricultural sensor box, the "SquirrelBox," akin to those used in smart and precision agriculture, to collect acquire data. This objective will be deemed complete when the "SquirrelBox" is fully functional and have collected at least 1000 illuminance samples over several days.
4. Implement a simple SVM solution to detect anomalies in an agricultural sensor network	It is important to split the development part of the project into small manageable steps which can be easily tested since bugs might be more difficult to detect after implementing more complex features. Thus, first I will focus on non-privacy preserving SVMs. The completion of this objective will be marked by the successful creation of a basic SVM-based solution for detecting anomalies in light sensor recordings in the developed sensor network simulation. This should be implemented in Python using the Scikit-Learn library by the end of the project's first development phase.
5. Implement a privacy preserving SVM to detect malicious nodes in the agricultural sensor network simulation	The primary aim of my project is to develop a privacy preserving SVM solution that ensures ASN integrity while safeguarding the data used for SVM model training. To achieve this, I'll implement an SVM using the differentially private approach identified in the initial research phase and perform controlled experiments with adding noise to the training data. I'll consider this objective met if, by the end of the second development phase, I have an operational privacy preserving SVM solution that identifies malicious nodes. The success of this objective will be measured by the accuracy of the classifier and the detection of any data leakage determined using membership inference attacks.

Objective	Explanation
6. Implement the developed privacy preserving SVM to the real agricultural sensor network	This ia an optional ambitious objective which aims to further validate the efficacy of the developed privacy-preserving Support Vector Machine based solution. It will be achieved if, in the period between completing the prototype development and crafting the final summary and conclusion, I successfully integrate it into a real-world smart agricultural sensor network, composed of Arduino-based sensor devices. The success of this objective will be evaluated by comparing the performance of this real-world implementation with the computer-based prototype simulation.
7. Summarise and evaluate the performance of the created prototype and classifiers	Finally, to describe the overall project and evaluate the results, I will need to determine the performance of the created prototype and SVM classifiers. This objective will be achieved if I evaluate whether my designed solution is an effective method to detect malicious sensor nodes while keeping the data private. This evaluation will be done using the accuracy metric and effectiveness of membership attacks against the designed SVM models and therefore be the deciding factor of whether the overall project aim has been achieved.

1.2.3 Changes Since the Initial Project Proposal

In the initial project proposal, one objective was to analyse and summarise two published examples of potential sensor network attacks. However, later it was decided that it would be more productive to focus on data anomalies rather than specific attacks like denial of service, which many modern routers can handle.

Furthermore, the original proposal lacked an objective concerning data acquisition. However, after initial research, I found no suitable datasets featuring agricultural data from the UK with a detailed setting description. This necessitated the assembly of an agricultural sensor box and data acquisition (objective **3**).

Additionally, the initial idea in the project proposal was to explore homomorphic encryption-based approaches for preserving data privacy, however, most of the frequently used homomorphic encryption libraries like Blyss, Petlib, and Pythel do not support SVMs. These libraries only accommodate simpler algorithms and building a homomorphically encrypted SVM from the ground up would have exceeded the time constraints of this project. Moreover, while conducting background research, I learnt that homomorphic encryption, due to its computationally intensive nature, may not be viable for agricultural applications that often deal with large datasets. Due to that, the $\mathbf{5}^{\text{th}}$ objective was updated and the focus was shifted to differentially private SVMs instead.

Lastly, the fifth and sixth objectives were initially unified. For clarity, they have been separated into two distinct goals.

1.3 Dissertation Structure

In this chapter, the problem was presented, and project aims, and objectives were outlined. Moving forward, the paper unfolds as follows: chapter 2 lays the theoretical groundwork for Support Vector Machines (SVMs) and differential privacy, supplemented by a review of related literature. chapter 3 outlines the project specifications, detailing both functional and non-functional requirements, and the overall project methodological framework. Next, chapter 4 presents the steps taken during the development phase of the solution. Following this, in chapter 5, testing phase strategies are discussed. Next, chapter 6 describes project's feedback and release stages. Following this, in chapter 7 an overview of the strengths and weaknesses of the overall methodology of the project is presented. chapter 8 discusses results and evaluations. Finally, the paper concludes in chapter 9, where the final conclusions are drawn.

Chapter 2

Background

2.1 Introduction

This chapter provides a condensed overview of the background information and related literature. It initiates with the significance of data integrity and privacy in agricultural sensor networks, then transitions to Support Vector Machines (SVMs), often employed to tackle data integrity issues. Subsequently, the focus shifts to differential privacy and differentially private Support Vector Machines.

2.2 Background

2.2.1 Data Integrity and Privacy Concerns in ASNs

Agricultural Sensor Networks (ASNs) are inherently vulnerable to a range of environmental adversities. Factors such as humidity or temperature fluctuations, physical obstructions, and human presence can impact these networks, leading to communication disruptions and data loss [21, 62, 18]. Additionally, the open, distributed nature of ASNs, exposes them to individuals with malicious intent who can tamper with or compromise them, causing malfunctions and network data integrity compromise [15]. According to the IEEE International Conference on Collaboration and Internet Computing (CIC) 2020 conference paper "Cyber Attacks on Smart Farming Infrastructure", such data integrity compromise due to the automated nature of Smart Farming, can even result in agroterrorism attacks since it can create hazardous farming environments, damaging large areas of crops, flooding the farmlands, or even over-spraying pesticides thus causing unsafe consumption as well as economic deterioration [60]. And while hardware solutions such as robust sensor design [41] and resilient sensor deployment strategies [63, 40] have been proposed to tackle these issues, guaranteeing absolute data integrity in ASNs remains a significant challenge due to factors such as limited computation power and memory constraints on sensors [49].

Consequently, farmers often depend on outsourced machine learning to analyse sensor data. However, the significant data required for reliable operation raises privacy issues such as exposing farmers production details or unveiling specific historical farming operations, leading to farmers' reluctance in contributing their data for algorithm training [69]. Moreover, if the attackers discern the private individual data points used for training these models, for example via membership inference attacks [57], they could potentially exploit the system. This could be by using the information to evade the systems which ensure data integrity in order to damage the

crops, selling the sensitive historic data to competitors, or even orchestrating more targeted breaches, undermining the network's integrity. Therefore, both heightened privacy and security measures are essential to safeguard agricultural sensor network integrity.

2.2.2 Support Vector Machines

Support Vector Machines (SVMs) are among the most widely used supervised learning algorithms, applicable to both classification and regression tasks. They were pioneered by Vladimir Vapnik and his team at AT&T Bell Laboratories [26], are underpinned by the statistical learning (VC) theory [11], and have demonstrated effectiveness in a multitude of domains such as anomaly detection tasks within computer networks [34, 66, 73], medical systems [54], and smart grid networks [59, 16], underscoring their potential for this project to ensure the integrity of agricultural sensor networks.

Furthermore, numerous studies have highlighted the superior accuracy of SVMs in comparison to other models. For example, in the agricultural context, the study which focused on identifying soil texture classes in southwest China [71] highlighted that SVMs performed better than artificial neural network and classification tree algorithms. Additionally, other researchers, who conducted comparative analysis of Support Vector Machine (SVM), Genetic Algorithm (GA), Long Short Term Memory (LSTM) network, Decision Tree (DT), and K-Nearest Neighbour (KNN) Algorithms in real-time applications related to bank fraud, medicine, face detection, and student performance prediction, stated that LSTM network and SVM algorithm have projected a superior behavior over the rest in terms of accuracy [19]. Also, another comparative assessment study of Support Vector Machine (SVM), Genetic Algorithm (GA), Long Short Term Memory (LSTM) network, Decision Tree (DT), and K-Nearest Neighbour (KNN) Algorithms was done for water quality classification and demonstrated that the SVM algorithm presents the best performance with no errors in calibration and validation phases [46]. Given this existing literature and the successful applications of SVMs, their promising high accuracy was another key reason for choosing them for this study.

Additionally, Support Vector Machines (SVMs) are relatively efficient in terms of memory, since when they are trained, they only need to remember a subset of the training data (the support vectors) to make predictions, which is a desirable feature as the machine learning training datasets in agricultural settings contain a vast amount of data [27]. Also, they have proven their mettle with consistent, high-quality performance, even when confronted with noisy data - a common scenario many machine learning algorithms struggle to handle effectively [17, 14]. This capability gains paramount significance in the context of agriculture, where data quality can fluctuate due to a variety of environmental factors, making it inherently noisy.

Due to this, given their proven performance and robustness, Support Vector Machines (SVMs) hold great promise for preserving and augmenting the reliability of agricultural sensor networks. Consequently, these qualities were instrumental in the decision to select SVMs as the principal tool for this project.

2.2.3 Technical Background of Support Vector Machines

For a set of 'n' data points given as $(x_1, y_1), \ldots, (x_n, y_n)$, where every unique x_i corresponds to a feature vector and each y_i represents the corresponding class label, taking values of either 1 or -1, traditional linear SVMs aim to classify these points into two distinct groups. Their goal is to separate all the xi points with a label '1' into one group, and those with a label '-1' into another group [26]. This classification is accomplished by determining hyperplanes, which can be expressed as a set of points x x that satisfy the equation: $w^T x - b = 0$, where w is a normal vector to the hyperplane and and b is a bias term.

The goal of these hyperplanes is to maximise the margin, defined as the distance between the decision boundary and the nearest data points from each class. The parameter $\frac{b}{||w||}$ defines how much the hyperplane is offset from the origin along the normal vector w, and the data points that reside on or closest to the hyperplane are termed "support vectors" due to their role in defining the hyperplane's placement and orientation.

Finally, in SVMs, the decision function for a new instance x is computed as $w^t * x - b$. If the result is positive, then x is predicted to belong to the positive class, otherwise it is predicted to belong to the negative class. These concepts are graphically illustrated in Figure 2.1 bellow.

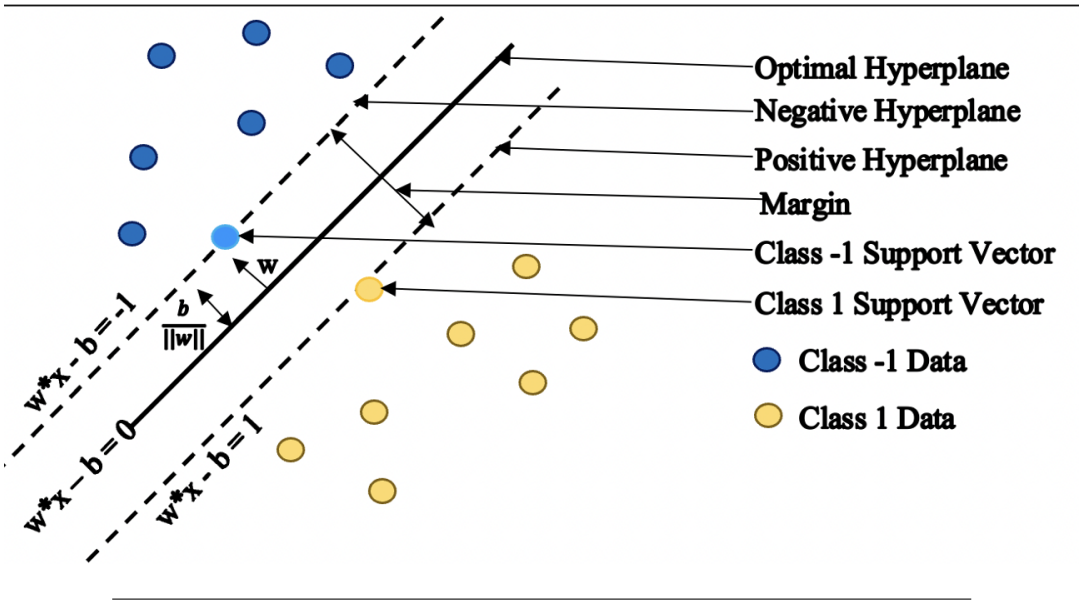

FIGURE 2.1: Graphical SVM illustration

There exist two common SVM types depending on the margin: soft and hard margin SVMs. Hard margin SVMs are used when the data is completely separable, soft margin SVMs, on the other hand, allow some misclassifications, aiming to find a good balance between perfect classification and model complexity. They are often employed when the data is noisy, contains outliers, or is not entirely separable, making them an ideal choice for this project, considering that agricultural data frequently

exhibits these traits due to factors like weather variability and other unforeseen conditions.

In soft margin SVMs, the objective is to not only maximise the margin, but also minimise the classification error. This trade-off is controlled by a parameter 'C', also known as the penalty term. A larger 'C' means less tolerance for misclassification, resulting in a smaller margin, and a smaller 'C' allows more misclassifications for a wider margin as illustrated in Figure 2.2.

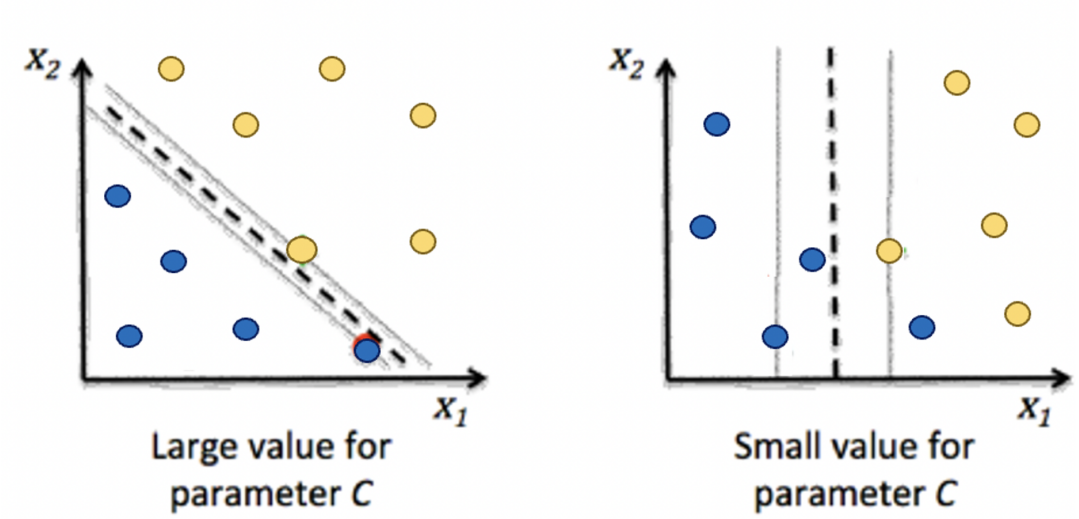

FIGURE 2.2: C Parameter in Soft Margin SVMs

The loss function (the measure of how well the model is performing) used in soft margin SVMs is called Hinge loss. Hinge loss is defined as $\max(0, 1 - y \cdot f(x))$, where y is the true label and $f(x)$ is the predicted output. If the predicted output has the same sign as the true label and the absolute difference between them is at least 1, the hinge loss is 0, indicating no penalty. If not, the Hinge loss is positive and contributes to the total error.

The concept of error minimization in soft margin SVMs corresponds to Empirical Risk Minimization (ERM), a principle in statistical learning theory. The idea is to find a function (in this case, the decision function of the SVM) that minimises the average loss (in this case, the hinge loss) on the training data. This corresponds to minimising the misclassifications while taking into account the soft margin.
Alongside these considerations, an important aspect of SVMs is regularisation, which controls the complexity of the model to prevent overfitting. The L2 regulariser, represented by $\|w\|^2$ in the objective function, penalises large weights, thereby preferring simpler models with smaller weight vectors. Taking all these elements into account, the objective function of the soft margin SVM can be represented as follows:

$$\min_{w} \underbrace{C \sum_{i=1}^{n} \max\left[1 - y_i \underbrace{(w^\top x_i + b)}_{h(x_i)}, 0\right]}_{\text{Hinge-Loss}} + \underbrace{\|w\|_{z^2}}_{\text{L2-regulariser}}$$

This function encapsulates the goal of the SVM: to find a balance between minimising the hinge loss (classification error) and the L2 regulariser (model complexity), under the chosen trade-off parameter 'C' [2].

Lastly, linear SVMs can be further extended and made more versatile through the use of kernel functions such as RBF or Polynomial which work with non-linearly separable data by implicitly mapping the data into a higher-dimensional space, where a linear hyperplane can be found to separate the classes as illustrated in Figure 2.3[72]. This flexibility adds another layer of potential for SVMs when applied to agricultural sensor networks, where the data may exhibit complex patterns and relationships.

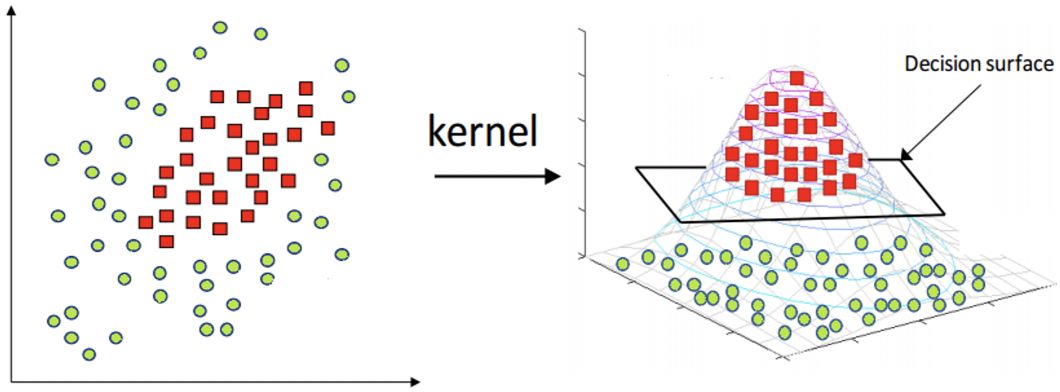

FIGURE 2.3: SVM Kernel Mapping Data into a Higher-Dimensional Space

2.2.4 Differential Privacy

Differential privacy is a mathematical framework for quantifying the privacy guarantees provided by an algorithm. Introduced by Dwork et al. [32], it has since become a gold standard for privacy-preserving data analysis.

The central concept of differential privacy is that the output of an algorithm should not change significantly when a single record is added to, or removed from the input dataset, providing a robust privacy guarantee, as it ensures that an adversary cannot determine whether a specific individual's data was included in the input dataset based on the algorithm's output.

More formally, a randomised mechanism (algorithm A(·)) is considered (ϵ, δ)-differentially private (where ϵ and δ are non-negative) if for all neighbouring databases D and D' (for databases differing in only one record), and for all events S, the following inequation holds: $\mathrm{P(A(D)} \in S) \leq e\varepsilon \cdot P(A(D') \in S) + \delta$

It is said that the mechanism satisfies pure differential privacy, if it satisfies (ϵ, 0)-differential privacy, and it satisfies approximate differential privacy (ADP), if it satisfies (ϵ, δ)-differential privacy for $\delta > 0$ [45, 31].

The parameter ϵ serves as a measure of the similarity between the distributions of $A(D)$ and $A(D)$, as illustrated in Figure 2.4 [47], and it also determines the robustness of the privacy assurance. Typically, ϵ is a minor constant ranging between 0 and 1 (for example, 0.01), and for such small values of ϵ, the exponential can be reasonably approximated as e≈1+ϵ. Hence, differential privacy implies that the probability of A(D) belonging to set S can be only marginally higher than the probability of A(D) being in set S (for instance, it could be 1.01 times the likelihood of A(D) falling within set S).

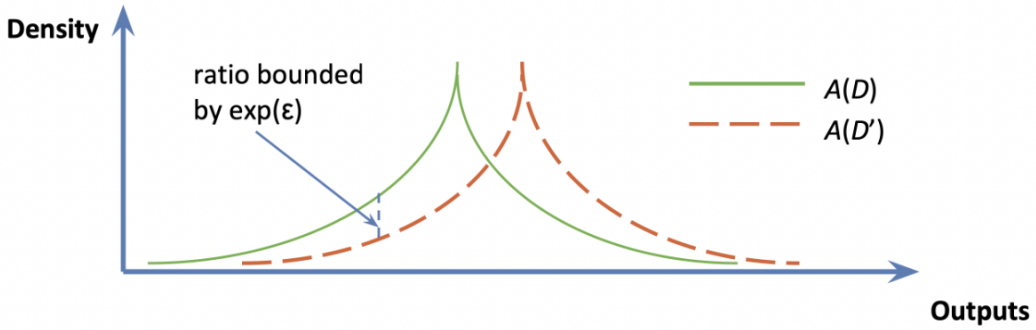

FIGURE 2.4: Differential Privacy Visualisation

2.2.5 Differential Privacy in Literature

In existing research, differential privacy is frequently suggested as an effective countermeasure against various attacks. For example, it has been identified as a potent defence against membership inference attacks [44, 51] and due to that has been practically applied in areas such as machine learning for genomic data [25] where such attacks can be detrimental. Additionally, it is used to help mitigate re-identification and linkage attacks, for example, U.S. Census Bureau started to use differential privacy with the 2020 Census data [22], Apple uses it in iOS and macOS devices for personal data such as search queries, emojis and health information [3], and Microsoft employs it when collecting telemetry data from Windows devices [28]. Additionally, recent research has introduced strategies for applying differential privacy to anomaly detection in Local Area Network (LAN) [52] activities and for enabling distributed anomaly detection architectures to maintain differential privacy [33]. However, despite these encouraging developments, it is important to note that these techniques have not yet been applied to agricultural sensor networks. Thus, there is a significant opportunity for exploration and innovation in this uncharted territory.

In the context of this project, the adoption of differential privacy would yield notable advantages. Incorporating differential privacy measures would increase the difficulty for attackers to infer the specific training data employed for the models

responsible for detecting anomalies in agricultural sensor networks. And in the absence of differential privacy, if attackers were able to easily deduce such information, it could result in a multitude of significant issues. Firstly, it could compromise the trust of the data providers. For instance, competitors could exploit the historical data of the farmers, gaining insights about their farms and crops at specific times, which could be sold or used to gain competitive advantages. Secondly, and potentially more concerning, is the possibility that this obtained knowledge could aid malicious actors in circumventing the anomaly detection systems. This could not only impair the ability to spot potential threats to crops, but it could also facilitate more targeted network breaches. Finally, the attackers could leverage inferred historic data used for training machine learning models to gather more specific information about the condition of a farm by correlating this data with other related data sets that may have been made publicly accessible by the farmers. To do so, potential attackers might exploit linkage attacks, providing them with a glimpse of the farm's state at a particular moment, compromising both its privacy and potentially its security.

2.2.6 Differentially Private SVMs

Differentially Private Support Vector Machines (DP-SVMs) combine the principles of SVMs and differential privacy to offer a supervised learning algorithm that protects the privacy of its training data. They were initially proposed by Benjamin I. P. Rubinstein, Peter L. Bartlett, Ling Huang, Nina Taft [53] who applied Laplace mechanism-based approach to add noise to the SVM's output function and demonstrated that their DP-SVM can provide robust privacy guarantees without significantly sacrificing accuracy. However, later, some researchers observed that the considerable amount of noise needed to maintain differential privacy in the algorithm often led to decreased performance. As a result, they introduced and developed a novel alternative method called objective perturbation [24], which has since emerged as the leading method for performing differentially private SVM classification and has been referenced by over 1,400 researchers in their respective studies. This particular method offers a unique approach to differentially private SVM solutions, as compared to others proposed in existing literature. Rather than applying noise to the output result, this technique mitigates concerns of accuracy loss by integrating noise directly into the objective function, the element optimised within SVMs. Consequently, this results in the necessity of a smaller amount of noise to maintain the ϵ-differential privacy of the algorithm, a clear advantage in preserving model performance while keeping data private.

In this method, instead of solving a conventional regularised objective function, denoted as:

$$\mathcal{L}(\theta, D) = \sum_{i=1}^{n} \ell(\theta, x_i, y_i) + \frac{\lambda}{2}||\theta||_2^2 \tag{2.1}$$

where (θ, x_i, y_i) is the loss function, λ is the regularisation parameter, and $\|\theta\|_2^2$ is the L2 norm of the parameter vector θ, the method focuses on a correlated problem: the components of the parameter vector θ are adjusted by a random amount, which can either penalise or reward them:

$$\mathcal{L}^{priv}(\theta, D) = \sum_{i=1}^{n} \ell(\theta, x_i, y_i) + \frac{\lambda}{2}||\theta||_2^2 + \langle b, \theta \rangle \tag{2.2}$$

where the parameter "b" follows the probability distribution, which can be either Gamma or Gaussian, depending on whether the objective is to achieve pure or approximate differential privacy.

2.2.7 Applications of Differentially Private SVMs

While the novel differentially private SVM objective function perturbation approach has been proven to outperform the output perturbation method [24], its application has been somewhat limited in scope. It has been used as a basis for federated learning based in-hospital mortality prediction system [39], for creating a scheme for privacy preserving SVM model training for distributed medical machine learning [30], and for developing a private empirical risk minimisation scheme for a healthcare system [29]. However, there remains a significant gap in the literature regarding its application within agricultural sensor networks as the distributed and federated privacy-preserving machine learning approaches mentioned before are not directly applicable for anomaly detection in agricultural sensor networks due to the demands of extensive local data processing and device communication, which most of these networks are not equipped to handle efficiently [42, 68].

However, this does not rule out the potential for privacy-preserving methods altogether. In fact, an alternative, simpler approach may be more feasible. Drawing inspiration from the success of differentially private SVMs in the realm of image classification [56], where the addition of Laplacian noise to the data enhanced privacy without highly impacting classification accuracy, a similar strategy could be adapted for agricultural sensor networks.

This would involve adding a small amount of random noise directly to the sensor data. It is important to note that this noise would not be carefully calibrated for differential privacy, but rather, it would be added to help mask specific data points. This could enhance the privacy of the data while possibly aiding in model generalisation, especially when dealing with smaller datasets [4], due to the inherent regularisation effect of noise addition and introduction of randomness that helps the model avoid overfitting to the training data.

Therefore, exploring the integration of both differentially private SVMs and minimal noise addition to the data appears to be a promising path. This combined approach would not only capitalise on the proven benefits of differential privacy in SVMs but also potentially gain from the added robustness and generalisation capabilities offered by noise injection.

2.3 Summary

This chapter presented a comprehensive review of relevant literature, beginning with an exploration of data integrity and privacy in agricultural sensor networks. The discussion then focused on the role of Support Vector Machines (SVMs) in managing data integrity issues. This was followed by an examination of differential

privacy and its application within SVMs. Lastly, the potential for integrating differentially private SVMs, and noise addition techniques to ensure agricultural sensor network integrity was suggested, signposting a promising approach for data privacy preservation and integrity in this context.

Chapter 3

Project Planning

3.1 Introduction

This chapter provides a high-level overview of the project's methodology plans. It starts by outlining the project plan and organisation, followed by a discussion on the selection of the software development model. The chapter then details both functional and non-functional project requirements, accompanied by justifications for the chosen tools, technologies, and methodologies that will be employed to fulfil these requirements.

3.2 Project Schedule

Before initiating the project and after establishing clear goals, a project schedule was created in order to ensure systematic progression of work, and enable efficient tracking of milestones. The project schedule, along with the correlation of all project objectives to respective task/project stages, is visually represented in Figure 3.1.

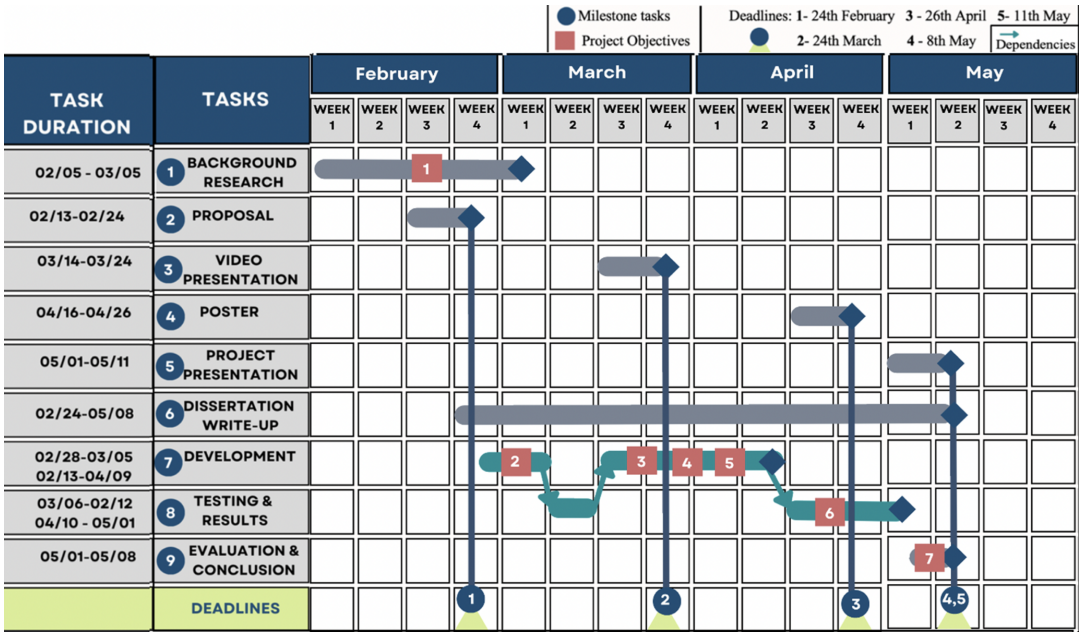

FIGURE 3.1: Project Plan

3.3 Project Plan Overview

As can be seen in Figure 3.1, before the software development process could begin, the first objective was to analyse and compare two instances of privacy preserving SVMs in the background research phase of the project to find the best approach to preserve the privacy of the data used to train the SVM models used in this project. Upon completion, the project was set to move into the software development and testing stages. The initial step involved creating an agricultural sensor network simulation (objective 2). Then, the plan was to assemble an agricultural sensor box, an integral component for collecting illuminance data (objective 3). With illuminance data in hand, the plan was to incorporate a simple Support Vector Machine (SVM) into the simulation network to identify light data anomalies (Objective 4). Then the subsequent phase intended to introduce a way to add noise to the training data and implement a privacy preserving SVM (objective 5) into the developed prototype solution. Following this, if time and circumstances allow, another goal was to implement this privacy preserving SVM in a real-world agricultural sensor network which uses LoRaWAN connectivity and sensor boxes (objective 6), thereby offering further validation and testing of the effectiveness of the proposed solution. Finally, during the last project phase, the plan was to summarise and evaluate the performance of the created prototype and SVM classifiers (objective 7).

3.4 Software Development Model

Upon establishing the project plan, the process to identify the most suitable software development methodology for the project was initiated. During this phase, both the Waterfall and Agile models, as depicted in Figure 3.2, were considered.

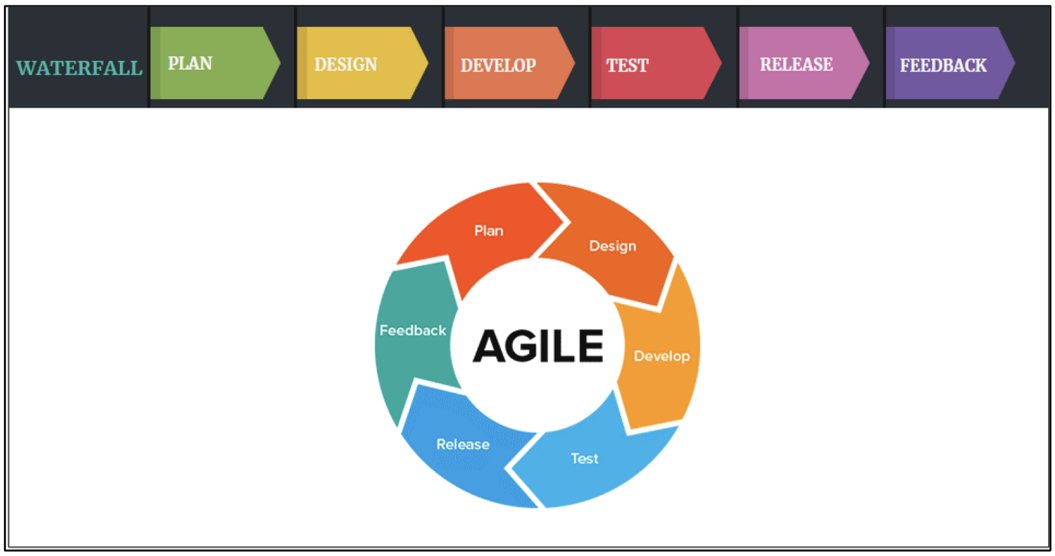

FIGURE 3.2: Comparison of Waterfall (Linear) and Agile (Iterative) Models

The Waterfall model was ultimately selected for this project since it emphasises a linear progression where each phase hinges on the successful completion of the previous one, perfectly aligning with the sequential nature of the defined project objectives. Additionally, compared to the iterative Agile model, which tends to be more suitable for projects with frequently changing requirements, the Waterfall model encourages thorough documentation, which is advantageous for maintaining a clear record of the project's progress.

3.5 Functional and Non-Functional Project Requirements

During the planning phase of the project, functional and non-functional requirements were defined to establish a clear vision and set the foundation for the project's success:

3.5.1 Functional requirements

1. SVMs should achieve a classification accuracy of at least 85%
2. A differentially private SVM should be employed to help preserve the privacy of the training data.
3. The system should possess a feature that sends automated alerts to system administrators when anomalies are detected.
4. An audit log of all detected anomalous nodes and data must be automatically updated whenever a new malicious node is detected to facilitate incident investigation.
5. There should be an option for users to adjust the level of privacy based on their privacy-accuracy trade-off preference.

3.5.2 Non-Functional requirements

1. The system's source code should contain comments and follow PEP8 [5] coding standards and guidelines.
2. The system's terminal interface must incorporate color-coding to enhance readability and user interaction. Different outputs, such as error messages and logs should be displayed in distinct colours to assist with immediate recognition and interpretation.

3.6 Chosen Tools, Technologies, and Techniques

Transitioning from the stage of defining functional and non-functional requirements, the design planning phase was initiated. A pivotal task in this stage was the selection of appropriate tools and technologies, which were crucial for the effective development and execution of the project. The specific set of core tools and technologies chosen, along with the justifications for choosing them, are detailed below:

3.6.1 Arduino "SquirrelBox" Sensor Box

The Arduino-based agricultural sensor box called "SquirrelBox" was selected for its ability to gather agricultural sensor data for SVM training. This feature was particularly critical due to the lack of publicly accessible data from agricultural sensor networks in the UK. Furthermore, the SquirrelBox's design closely emulates the Arduino microcontroller-based sensors commonly found in Smart Agriculture sensor networks. This similarity was another deciding factor in its selection since if time permits, it could be connected to the LoRaWAN router and provide an opportunity to test the developed solution in conditions that closely mirror real-world scenarios.

3.6.2 Git and Github

The Git version control system and the GitHub hosting service were adopted due to their comprehensive feature sets, allowing efficient sharing, code versioning and storage. Other similar systems were considered, such as SVN and Bitbucket, but Git and GitHub were chosen due to existing familiarity with these tools, which would contribute to a more efficient development process.

3.6.3 Python

Python was chosen over other languages such as Java and C for this project. The decision was primarily driven by Python's extensive libraries that facilitate machine learning, data manipulation, and numerical computations - all vital for the project's implementation.

3.6.4 Scikit-Learn Library

Scikit-Learn library was chosen due to its accessible and thorough documentation, ideal for those with limited machine learning knowledge, like me, its simplicity compared to lower-level libraries like TensorFlow and PyTorch, and support for SVMs, which were the chosen machine learning algorithm for this project.

3.6.5 Differentially Private Objective Function perturbation SVM

Currently, there are no publicly available libraries which support differentially private SVMs, as opposed to other simplier machine learning algorithms [12]. Due to that, the SVM objective function perturbation method proposed by K. Chaudhuri, C. Monteleoni, A.D Sarwate [24] was selected since this innovative technique demonstrates superior results, outperforming older output perturbation approaches, and promises robust performance while protecting individual data points, a balance that directly aligns with the project's objective of achieving a minimum of 85% predictive accuracy while ensuring differential privacy.

3.6.6 Adversarial Robustness Toolbox (ART) Library

Developed by renowned researchers at IBM, the ART Library [48] was selected for its capabilities in evaluating ML models' security and privacy, including via membership inference attacks. Its support for both Scikit-Learn and custom 'black-box' models made it ideal for testing the non-private and differentially private SVMs used in this project, facilitating the quantification of privacy loss. Additionally, given

my lack of machine learning knowledge prior this project, ART's detailed documentation further motivated its selection over other similar libraries like Privacy Meter and ML-Doctor.

3.6.7 ANSI escape codes

ANSI Escape Codes are a set of standard byte sequences that trigger specific actions when interpreted by character-based terminals [13]. Due to their ability to transform the terminal interface by altering text and background colours it was decided that they would be an ideal tool to incorporate color-coding to enhance readability and user interaction in order to help to fulfil the 2nd non-functional requirement.

3.7 Summary

In summary, this segment of the dissertation offered a comprehensive high-level overview of project plans and schedule, it emphasised the application of the Waterfall methodology in software development, and discussed the principal tools and technologies chosen for the project, alongside justifications for their selection in relation to completing the outlined objectives .

Chapter 4

Development Phase Methodology

4.1 Introduction

Following the Waterfall methodology, after completing the planning phase and selecting the appropriate tools, the next step involved defining the core components that would need to be developed. It was decided to split the development phase into steps, perfectly aligning with the defined 2,3,4,5,6 project objectives and sequentially complete them. Due to that, this chapter offers an overview of the development process. First, the process of developing an agricultural sensor network simulation and gathering illuminance data will be explained. After that, the implementation of both the simple Support Vector Machine (SVM) and the function to add noise to the machine learning model training datasets will be covered. Then implementation of the privacy-preserving SVM will be described, and the development of the method to assess the privacy-accuracy trade-off will be addressed. These sections will detail the software/system design, applied methodology, and the techniques that were employed during the process.

4.2 Agricultural Sensor Network Simulation

Initially, a decision was reached to create a simulation for an agricultural sensor network. The aim of this simulation is to mirror a real-world agricultural sensor network environment and provide a foundation for the prototype being developed. This prototype would demonstrate the project's capabilities to potential users and offer them a testing environment prior to deploying the solution to their actual sensor networks. Additionally, the "SquirrelBox" units use LoraWan connectivity, which posed an additional challenge creating and testing the prototype on the real network as only a single LoRaWAN router, specifically the WisGate Edge gateway, was available, and already engaged in other university research activities. Further complicating matters, a shortage in the UK market made it impossible to order additional units. Given these constraints, the decision was made to construct this simulation and implement the solution within it. This approach was deemed preferable to programming directly on the real router's MQTT file without the capacity for adequate testing.

4.2.1 System Design

The system architecture of the simulation agricultural sensor network was designed to reflect the real environment accurately. Central to this design are sensor nodes that transmit collected data, such as illuminance measurements, to a single, centralised server or router, which manages the data inflow from the nodes and forwards the

data to a simulated cloud system, which is a representation of either a local higher-capacity server or an external cloud server, capable of machine learning operations to determine any anomalies. To offer a visual understanding of this system's intricate workings, an illustrative representation of the architecture design is provided in Figure 4.1.

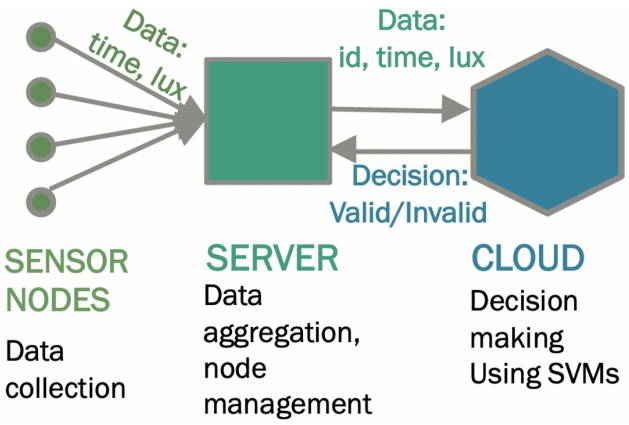

FIGURE 4.1: Simulation System Design

4.2.2 Sensor Node Implementation

The first part of the simulation development involved the creation of two categories of sensor nodes: those transmitting legitimate data (client.py) and those sending malicious data (malicious_client.py). These nodes were designed to transmit data to the server periodically, every 10 seconds. The data selection from specific CSV files hinges on the sensor type (reliable/malicious) and the system's current time (morning, afternoon, night). This time-dependent data selection was implemented to attain a more realistic simulation, capturing the daily variations often seen in agricultural settings, where for example, valid ranges of illumination levels can greatly differ between morning, afternoon, and night.

Originally, the data transmitted by the simulated nodes was artificial, serving as placeholder for testing purposes. However, it was subsequently replaced with authentic sensor readings, gathered with the assistance of the "SquirrelBox". The process of this data acquisition will be detailed in the next chapter.

The network connectivity amongst all components in the simulation was created using TCP socket programming. This choice was inspired by real-world agricultural scenarios, which often employ the MQTT messaging protocol - built on the foundation of TCP/IP [6].

For reference, the client.py script, which can be used to simulate a valid sensor, can be found in Appendix A.

4.2.3 Server Implementation

The simulated server (server.py) was developed to handle client communications, perform initial data filtering, transmit the data for further checks to the machine learning model hosted on the cloud, and then block the clients and inform administrators if the decision is made that the node/s are sending invalid data.

Similar to the sensor nodes, the networking was programmed using TCP sockets. Additionally, multi-threading was used to allow each sensor node to run/connect on a separate thread (parallel process), in order to send data to the server independently, like in a real-world scenario.

The server script contains two main functions :

- `handle_client()`
- `handle_cloud()`

For every successfully connected client, the `handle_client()` function is invoked. This function enables the server to perform initial checks on whether the node is in the blocklist (as recorded in the blocklist.csv file) and to continue communication only if it is not. Following the receipt of data (a message) from the sensor node, the server filters out invalid nodes by conducting a simple test to confirm that the data is within a valid range. For testing purposes, the valid range corresponds to that of the illuminance sensor used in the "SquirrelBox" (between 0 and 20,000 Lux). If the data falls within this range, the server then transmits all potentially valid data to the simulated cloud via a TCP connection, where further, more memory-intensive data checks are performed.

The server subsequently awaits the decision from the simulated cloud housing the machine learning model by calling the `handle_cloud()` function. If the returned decision suggests that the sensor node is transmitting potentially malicious or invalid data, the node is added to the blocklist file (blocklist.csv), which contains all nodes that have been sending potentially invalid sensor readings. Moreover, in the event of detected anomalies, system administrators are informed about these irregularities via an email. This email contains information about the detected invalid data, along with an attached file comprising an updated blocklist.

The functionality to send an email with the attached blocklist file was implemented using Gmail's free API with the Smtplib library, and the email form/body was designed using HTML. The administrator email addresses are read from the admin_email.csv file, thereby allowing users to easily update the administrator email addresses by simply modifying the contents of the file.

The complete simulation server script is available in Appendix B, Figure 4.2 bellow illustrates the designed email template, the HTML file of this template be found in Appendix C.

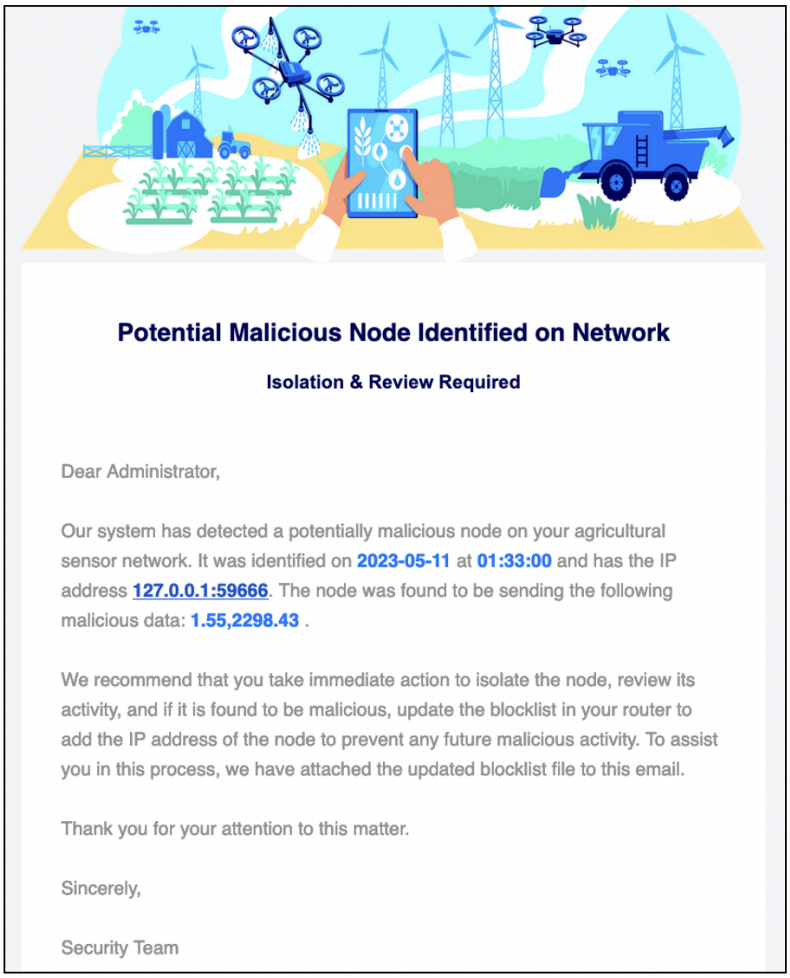

FIGURE 4.2: Designed Email Template

4.2.4 Cloud Implementation

The next crucial component in the system architecture was the cloud simulation (SVM.py), whose primary function was to allow the user to choose different SVM machine learning models for advanced data checks, host these models, and send the decision about the data validity back to the server.

First, the functionality to allow the user to choose the specific type of the machine learning (SVM) model they want to use was implemented. An illustration of the available options and the choices the user can make can be seen in Figure 4.3.

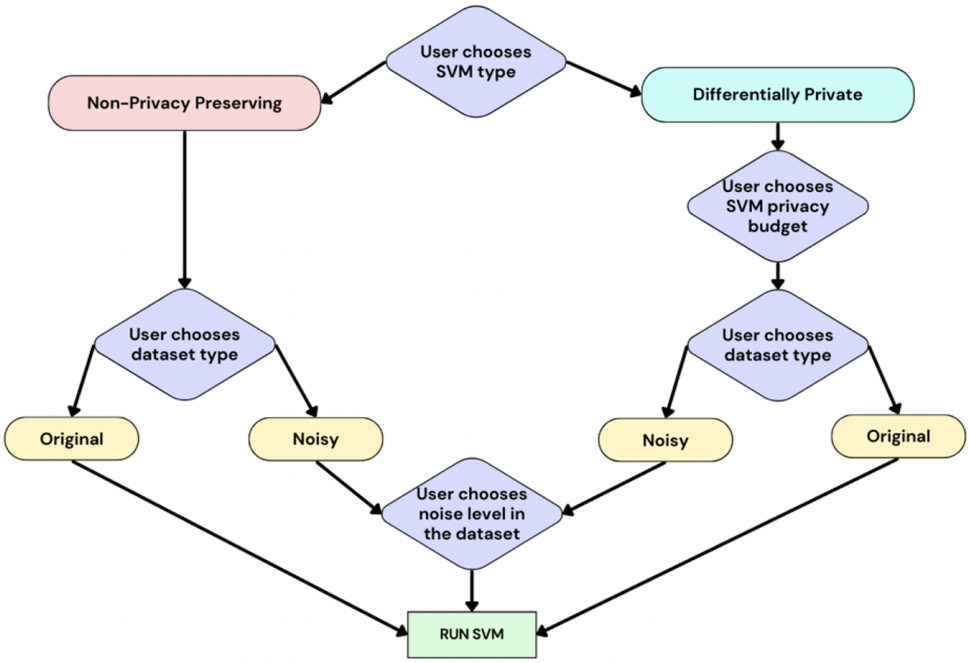

FIGURE 4.3: Available SVM Options

Initially, different placeholder functions were added to be called depending on these choices. For example, if the user chose to use a noisy training dataset, then the placeholder `privatised_training_dataset()` function would get called, or overwise the function `non_privatised_training_dataset()` would get invoked to prepare data for model training. Additionally, if the user chose to run a differentially private SVM, then placeholder custom functions for fitting (training the model) and predicting (determining whether or not the provided values are malicious) would be used instead of calling the equivalent non private SVM machine learning model functions from the Scikit-Learn library.

The process of developing the SVM models, which replaced these placeholder functions and implementing the noise addition functionality will be described in the "Development of the Simple SVM", "Development of the Privacy-Preserving SVM", and "Function for Adding Noise to Datasets" sections.

Additionally, since it was decided to split the training datasets into three different sections depending on the time of the day for a more accurate anomaly detection, the function `choose_training_dataset()` was added to check the current system time and read in the appropriate machine learning model training datasets depending whether it is night, morning, or afternoon.

An illustration showing the overall functionality of the developed cloud simulation together with the specific functions which have been created for the specific tasks can be seen in Figure 4.4, and the cloud simulation script is provided in Appendix D.

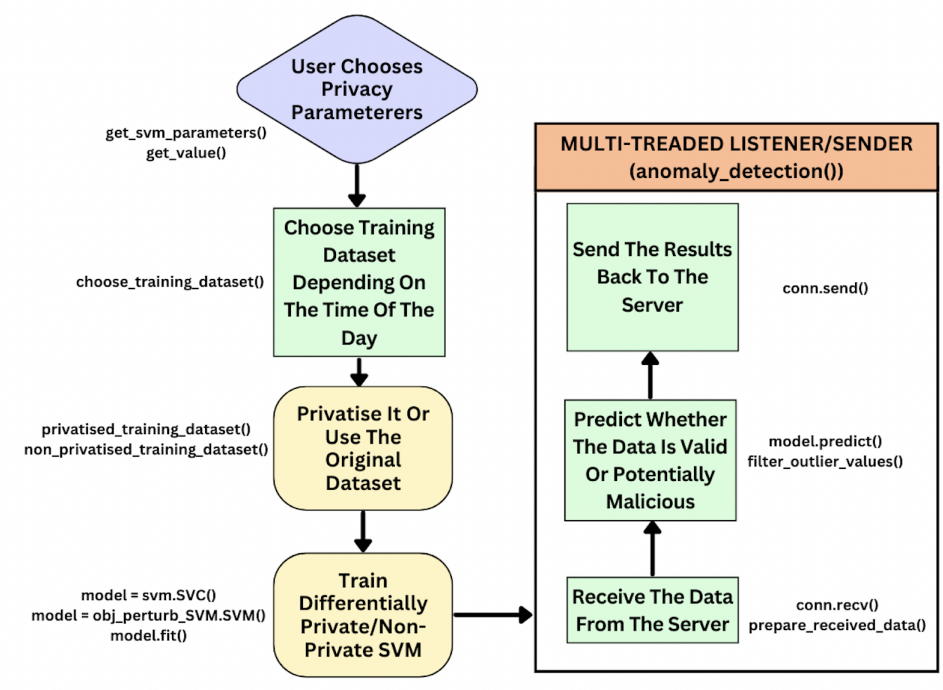

FIGURE 4.4: Overview of the Developed Cloud Simulation

Finally, the choices for the specific privacy budget (epsilon) values for differentially private SVMs, and the level of noise added to the training datasets also initially had temporary values for testing purposes, but at the end of the development phase, after the SVMs were created, they were replaced with more realistic options, allowing the farmers to choose different levels of privacy that they would like to achieve.

4.2.5 Implementation of System Terminal Interfaces

To meet the second non-functional requirement - enhancing readability and user interaction - ANSI escape codes were employed across all terminal interfaces. This included the simulated sensor nodes, server, and cloud. The use of these codes allowed for color-coding, effectively distinguishing between different types of outputs, such as informational messages and logs. The method used is demonstrated in Figure 4.5, while Figures 4.7,4.8,4.6 display examples of real terminal outputs from the working developed client, server, and cloud simulations.

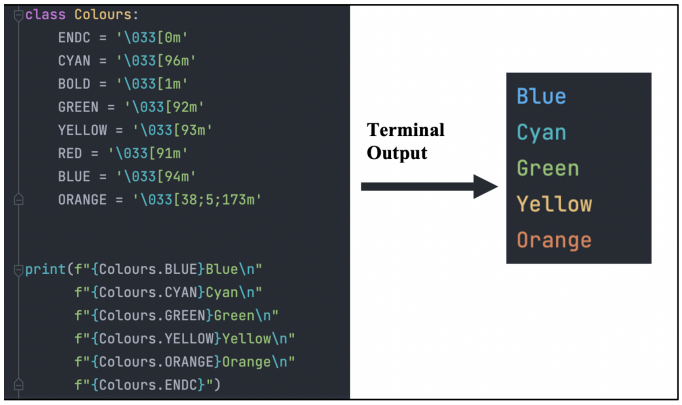

FIGURE 4.5: Using ANSI Escape Codes for Coloured Text

FIGURE 4.6: Terminal Output Display in the Client Simulation

FIGURE 4.7: Terminal Output Display in the Server Simulation

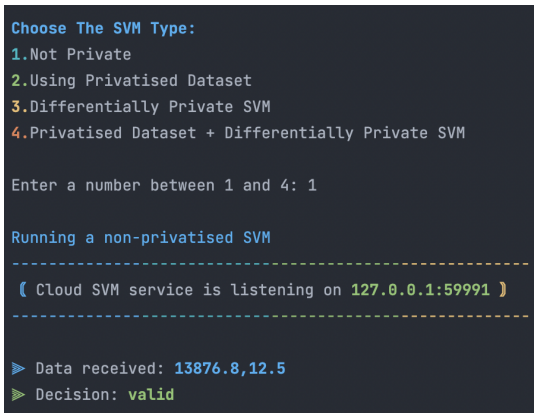

FIGURE 4.8: Terminal Output Display in the Cloud Simulation

4.3 Data Acquisition

Pivoting to the third project objective, the challenge was to gather labelled authentic illuminance data - both valid and invalid - for training SVMs. The decision to focus on gathering illuminance data was strategic and intended for broader applicability. Illuminance data, unlike other agricultural data such as soil pH and moisture levels, can potentially be reused by other farmers. This is because light exposure is generally uniform for farms in the same geographical area, especially for farms with outdoor sensors. Conversely, soil conditions can vary significantly even within a single farm due to factors such as crop type and soil composition. However, despite the broader usefulness of illuminance data, there was a surprising scarcity of this type of information available publicly, which meant I had to acquire this data myself. To do so, I assembled an Arduino-based sensor box called the "SquirrelBox".

4.3.1 Arduino "SquirrelBox" Sensor Box

To construct the “SquirrelBox”, the necessary components were first soldered to the motherboard. After the box was assembled, continuity tests were performed to ensure that the electrical connections within the sensor box were functioning properly. Following this, the sensor box was connected to a computer and different sensor Unit tests were ran to check for any issues or errors that could lead to inaccurate sensor readings. The schematic diagram of the “SquirrelBox”, including detailed information on the wiring and components, can be seen in Figure 4.9, Figure 4.10 shows the mother board before the components were soldered, and fully soldered SquirrelBox can be seen in Figure 4.11.

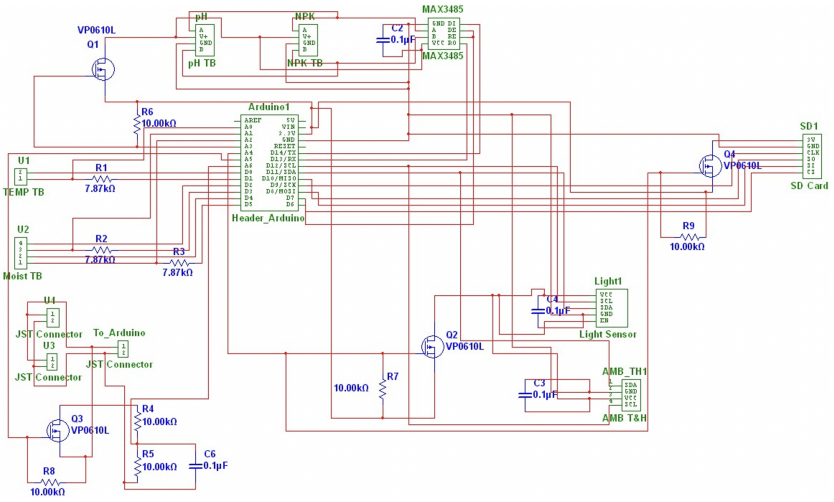

FIGURE 4.9: "SquirrelBox" schematic diagram

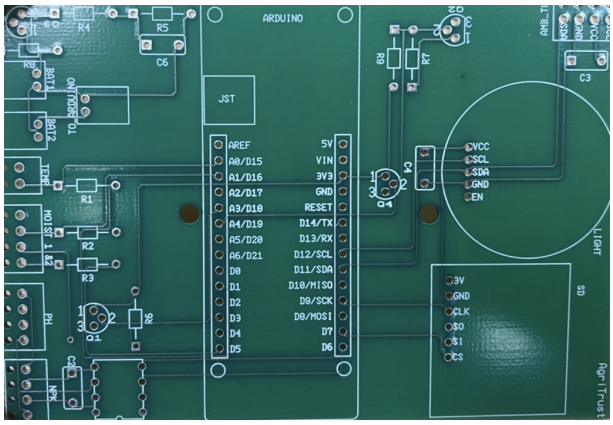

FIGURE 4.10: The motherboard of the "SquirrelBox"

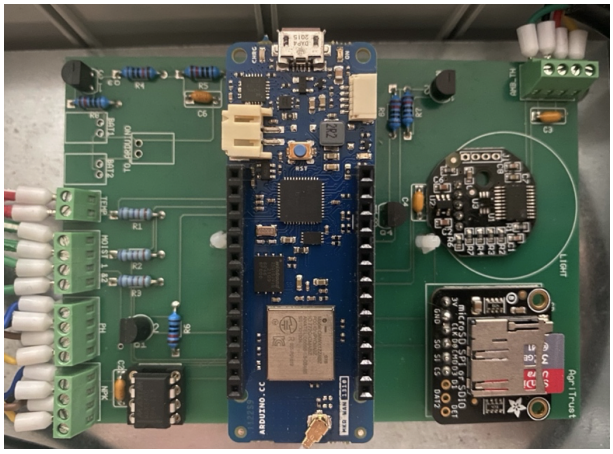

FIGURE 4.11: Fully soldered "SquirrelBox"

When it was confirmed that the sensor box could provide highly accurate readings, the next step involved using the SquirrelBox to collect illuminance data. To achieve this, the provided "SquirrelBox" C++ code was adapted to output the current hour, minute, and illuminance level recordings and to facilitate data processing and analysis, the ArduSpreadsheet Arduino IDE plugin [7], developed by Indrek Luuk, was utilised. The tool enabled the conversion of data from the serial Arduino monitor into CSV files, which offered a more accessible and practical format for subsequent machine learning model training.

4.3.2 Light Collection Setting and Rationale

The first steps in creating the illuminance datasets were defining the setting in which the data would be collected and choosing a suitable location for the sensor. After careful consideration, it was decided that the sensor will be placed indoors, in a room without any artificial lighting, on a south-facing windowsill, where it would receive consistent sunlight throughout the day. The reason for this was to protect the sensors and the computer the box was connected to since weather conditions could damage them if placed outside, and to allow for data collection that closely reflects the natural light conditions outdoors or in a small greenhouse without artificial lighting, similar to those used in small Smart agriculture farms, where light

anomalies can significantly affect plant growth due to the lack of artificial lighting, and where tasks such as watering or fertilisation are automated depending on illuminance readings together with other sensor data.

4.3.3 Temporal Characteristics of Light Data Collection

After determining the setting in which the data would be collected and after choosing the suitable location for the sensors, the next step was to establish the frequency and period for data collection. Because of the tight project schedule and the fact that the light sensor had to be connected to the computer, which was also used for work and dissertation write-up, it was determined that the best option was to record light data for 7 days, capturing the sensor readings every ten minutes. In total, 1008 light measurements were collected and stored in a CSV file format. The overall distribution of the captured data can be seen in Figure 4.12.

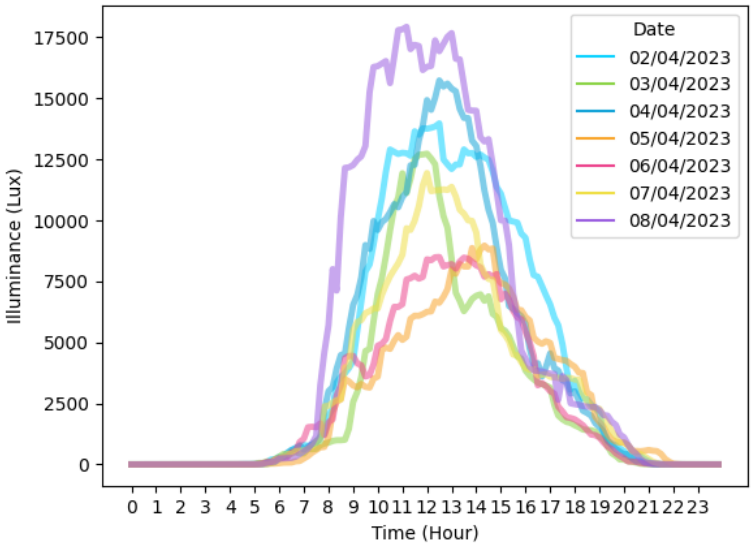

FIGURE 4.12: Collected Illuminace Data Distribution

4.3.4 Dataset Validation

Even though the soldered sensor box, which was used to collect the light data, was tested by performing continuity and Unity tests, it was still important to ensure that the datasets did not contain any missing, invalid, or incorrectly formatted values since they will be used for training and testing the created machine learning models. To achieve this, a small Python script, which checks for missing values, ensures that the 'Date' column contains correctly formatted information, verifies that only numerical data is present in the hour, minute, and lux value columns, and that these values fall within a reasonable range was written. The full data validation script can be found in Appendix E.

4.3.5 Anomalous Illuminance Recordings

Conducting experiments with SVMs necessitated the collection of both valid and invalid illuminance data. For invalid recordings, I manipulated conditions such as lighting and sensor placement and collected 503 sensor readings. However, the project's tight schedule posed a challenge in acquiring an equivalent number of thoroughly distributed invalid illuminance samples as the valid ones, and imbalanced datasets have been shown to skew the SVM models towards the minority class, affecting the model's performance in accurately predicting this class [64]. To mitigate this issue, common strategies such as under sampling and oversampling are often employed [20]. Consequently, I wrote a small script to oversample the collected anomalous data using the RandomOverSampler() function from the Scikit-Learn library. The full script can be seen in Appendix F and Figure 4.13 shows the overall data distribution of the resulting dataset.

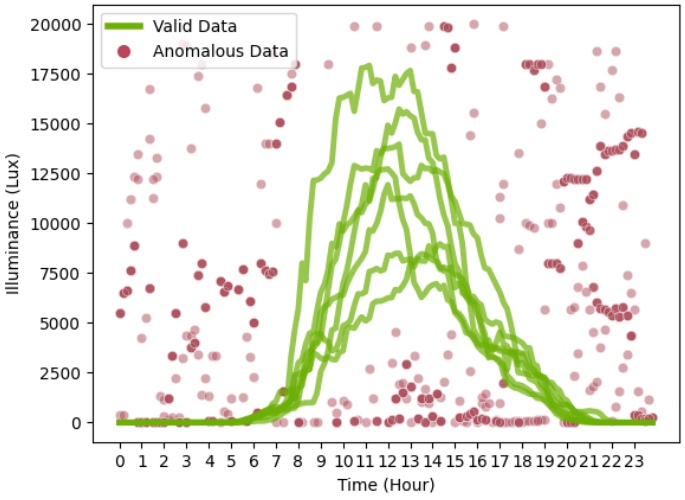

FIGURE 4.13: Final Illuminance Dataset Data Distribution

4.3.6 Dataset Splitting

During the background research, the differentially private SVM, which was essentially a linear SVM [24], was determined to be suitable for this project. Given that the overall outdoor illuminance data is not linearly separable, a strategic decision was made to partition the dataset into five smaller subsets for more effective application of this linear model. These subsets corresponded to distinct time frames of the day: night (9 PM - 6 AM), morning (6 AM - 12 PM), and afternoon (12 PM - 9 PM). To further refine the classification, the morning and afternoon data was subdivided based on the degree of anomaly, creating sets with lower (morning_0.csv, afternoon_0.csv) and higher (morning_1.csv, afternoon_1.csv) recordings compared to the valid data since SVMs are inherently a binary classier, but the data can be anomalous both because it is higher than the norm or lower than the norm. This approach, while more complex, was designed to better align with the linear SVM model's assumptions improving its applicability and performance for this specific study.

4.3.7 Data Pre-Processing

Following the strategic subdivision of the dataset, necessary pre-processing steps were applied to all the data used for model training in the relevant SVM training and testing scripts, which will be mentioned in the following chapters. The data was first divided into training and testing sets, reserving 30% for testing and using the remaining 70% for training. This split ratio was selected based on several empirical studies, suggesting it as an optimal choice [36, 65]. This was done to ensure that it would be possible to check if the created models can do accurate predictions on unseen data while being trained and tested on representative data. Subsequently, since SVMs consider distances between observations, scaling and normalisation tend to improve the performance of the models [61, 50] thus the data was normalized and scaled using Scikit-Learn's normaliser and standard scaler to ensure that all features have a similar scale and distribution.

4.4 Development of the Simple SVM

Once the distinct datasets for anomaly detection were successfully acquired, the next phase of the project involved the development of the initial machine learning model to complete the 4th project objective. Since I had no previous experience with machine learning, I utilised the Scikit-Learn library to design a linear SVM and find the most optimal C parameter. To do so, I used built-in grid search functionality and 10-fold cross validation. In more detail, how the testing for each training dataset was done is illustrated in Figure 4.14. As can be seen in the illustration, first the data was split into training and testing subsets, then pre-processed. Following this, the training data was used to test the performance of the algorithm using different pre-defined C parameters (0.001,0.01,0.1,1,10,100,1000) to find the most optimal one by doing grid search (testing each hyperparameter combination, in our case, just a single C value) and performing a 10 fold cross validation to ensure that the score of the model does not depend on the way the train and test sets were chosen.

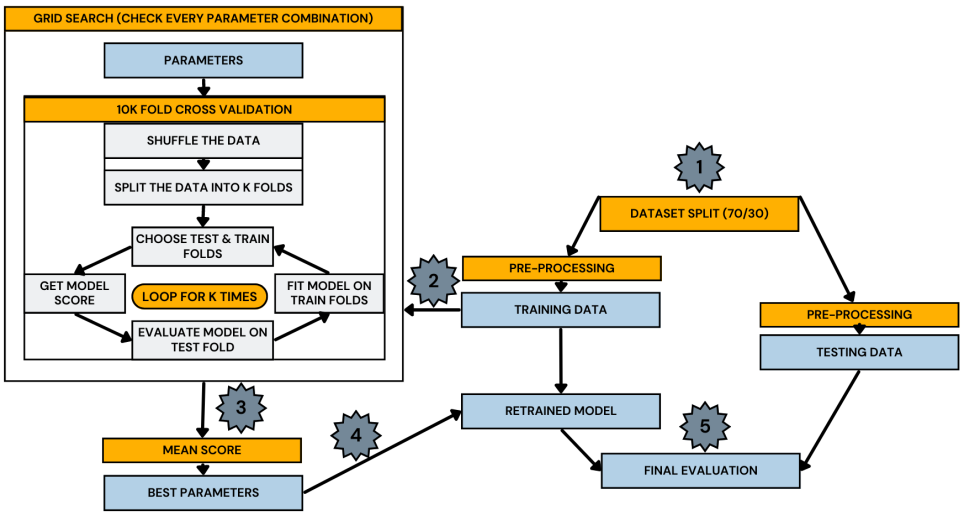

FIGURE 4.14: Grid Search with K-Fold Cross-Validation

After performing the 10K-Fold cross-validation, it was determined that 100 is an optimal C parameter for the non-privatised SVM model, the file with the full test results is included in the project files (simple_SVM_grid_search_results.txt), and the table summarising the tests can be found in the Appendix G.

Following this, after determining the most optimal hyperparameters, the final developed model was implemented into the created Cloud simulation as follows: after checking that the user chose to run the non-privacy preserving SVM, two models would get fitted with different data, one with higher than normal anomalous recordings and another one with lower than normal anomalous recordings depending on the time of the day. Then, the evaluation from both classifiers is checked, if the first classifier does not detect any anomalies (the data is not above the valid margin), then the second classifier is used to determine if the data is not lower than the valid margin to get the final decision.

4.5 Function for Adding Noise to Datasets

After the simple SVM model was developed, building on the insights from the background research, a strategic choice was made to experiment with simple noise addition to the data as this approach could possibly provide two potential advantages. First, to enhance the model's performance by discouraging overfitting. Second, to diminish the success rate of membership inference attacks, thereby enhancing the privacy of the data. Accordingly, scripts (abstract_data_privatiser.py and Laplace_dataset_privatiser.py - Appendix H) were written to achieve this and to replace the initially created dataset privatisation placeholder scripts in the Cloud simulation.

The abstract_data_privatiser.py script includes two key functions:
get_data_sensitivity_values() and privatise_data(). To enact data privacy, first the function get_data_sensitivity_values() should be called to determine the sensitivity of each column in the dataset. This is calculated by evaluating the range of each column, determined by the absolute difference between the maximum and minimum values. Following this, the second function should be invoked, supplying it with the original data and calculated sensitivity values. The second function calculates the amount of noise which should be added to each entry in the dataset, drawing it from the Laplacian distribution, modulated by a privacy budget parameter 'epsilon', and the calculated sensitivity level:

```
1 noise_value = np.random.laplace(self.mean_value,
  ↪  sensitivity_level/self.epsilon_value, 1)
2
3 return float(data_value+noise_value)
```

This process was designed to ensure that every data point was treated individually, respecting its specific sensitivity level, thus delivering a robust method for data privatisation.

4.6 Development of the Privacy-Preserving SVM

Following the incorporation of noise into the datasets, the next stage was to fulfil the fifth project objective: the implementation of the differentially private objective function perturbation-based SVM, which was outlined in the background research section [24]. This involved studying public lectures by Gautam Kamath [8] and examining the corresponding R implementation in the DPpack R package [9].

The adaptation process began with the implementation of the custom Huberised SVM loss function, as proposed by Oliver Chapelle in 2007 [23]. This function is a prerequisite for the differential privacy guarantees of the SVM algorithm to hold. It is defined as follows:

$$F_{\text{Huber}}(z) = \begin{cases} 0 & \text{if } z > 1+h, \\ \frac{1}{4h}(1+h-z)^2 & \text{if } |1-z| \leq z, \\ 1-z & \text{if } z < 1-h. \end{cases}$$

Here z is position of training data to predictor f, and h is a fixed constant [23], a hyperparameter which determines the point at which the loss function changes from quadratic to a linear function. According to the authors, it has to be equal to $\frac{1}{2c}$ (i.e c has to be equal to $\frac{1}{2h}$) to keep the loss function doubly-differentiable, which is needed to make the objective perturbation SVM function differentially private. The full implementation this loss function can be seen in Appendix I (function func()).

Next, the custom 'fit' function (Appendix I (model_fit())) was added to facilitate the training of the model. This custom function was developed in accordance with the proposed objective perturbation algorithm [24] :

Algorithm 2 ERM with objective perturbation

Inputs: Data $\mathcal{D} = \{z_i\}$, parameters $\varepsilon_p, \Lambda, c$.
Output: Approximate minimizer $\mathbf{f}_{\text{priv}}$.
Let $\varepsilon'_p = \varepsilon_p - \log(1 + \frac{2c}{n\Lambda} + \frac{c^2}{n^2\Lambda^2})$.
If $\varepsilon'_p > 0$, then $\Delta = 0$, else $\Delta = \frac{c}{n(e^{\varepsilon_p/4}-1)} - \Lambda$, and $\varepsilon'_p = \varepsilon_p/2$.
Draw a vector $\mathbf{b}$ according to (4) with $\beta = \varepsilon'_p/2$.
Compute $\mathbf{f}_{\text{priv}} = \text{argmin} J_{\text{priv}}(\mathbf{f}, \mathcal{D}) + \frac{1}{2}\Delta||\mathbf{f}||^2$.

FIGURE 4.15: Objective Perturbation Algorithm

Here, the authors of this algorithm first calculate the ε'_p and delta (Δ) values and then draw a noise vector $\mathbf{b}$ from a Laplace distribution with density $v(\mathbf{b}) = \frac{1}{\alpha}e^{-\beta||\mathbf{b}||}$, where $\beta = \varepsilon'_p$ and α is the normalizing value. Then, they add noise to the original objective function as follows: $J_{\text{priv}}(\mathbf{f}, D) = J(\mathbf{f}, D) + \frac{1}{n}\mathbf{b}^T\mathbf{f}$, where n is the number of samples in the dataset and T is used to denote the transpose of vector $\mathbf{b}$. Following this, the final private predictor $\mathbf{f}$ is calculated by solving the Empirical Minimisation

problem and adding a regularisation term.

After implementing the custom `model_fit()` function, in order to make predictions, the `predict()` function was written as well. It simply applies the trained SVM model to a given datapoint by calculating the raw score and turning it into a class prediction, achieved by returning a +1 or -1 sign based on whether the score is positive or negative, respectively, as can be seen in the code snippet below:

```
prediction = np.sign(np.dot(np.transpose(self.f), datapoint))
```

After implementing all the SVM functions, hyperparameter tuning was done using Grid Search and 10-Fold Cross-Validation similarly to the non-private SVM, except since the functions were custom and not from the Scikit-Learn library, it had to be implemented from scratch. The tests were done for the h values (equivalent to the c values used for the non-private svm tests, calculated as $h = \frac{1}{2c}$) and lambda values which were inverse of these h values. During the tests h=5 and lambda=0.002 were found as optimal values and the table summarising these tests can be found in Appendix G.

4.7 Development of the Membership Inference Attacks

After incorporating differentially private classifier, the last step in the development process was to create a way to quantify the privacy loss/gain to help empower farmers with insightful data for making informed decisions about system parameter selection and option tuning. To achieve this, a script for performing membership attacks was written (Appendix J). It works by training an additional classifier using the ART library [48]. This classifier conducts a black-box membership inference attack by querying the original models, obtaining the predictions, and then employing their probabilities or losses as learning inputs, as depicted in Figure 4.16.

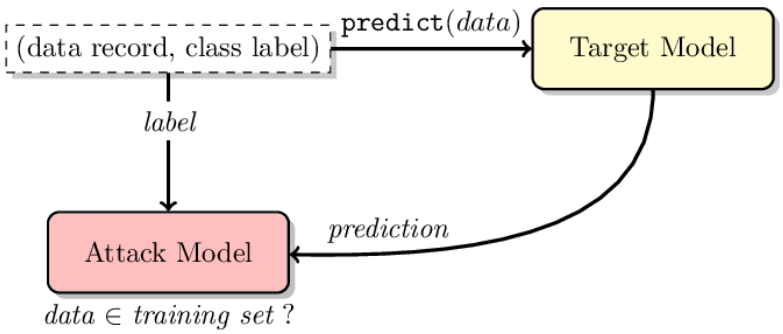

FIGURE 4.16: Illustration of a Membership Attack

4.8 Update to the Simulation Parameters

After the core components were developed, an initial accuracy evaluation was conducted to set parameter values. The parameters were adjusted to meet the functional requirement that the accuracy of SVMs should remain above 85%. Accordingly, differentially private epsilon values were updated to 0.9, 0.5, and 0.1. The

level of noise added to the dataset was also revised to 9, 5, 1. Then, privacy gain for each option was evaluated by conducting membership inference attacks. Initially, non-privatised datasets were used with a non-private Scikit-Learn classifier as a baseline. Subsequently, privatised datasets with different noise levels were employed, followed by differentially private SVMs with various privacy parameters. The privacy loss/gain for each option and setting will be extensively discussed in the "Results and Evaluation" section.

Chapter 5

Testing Phase Methodology

Throughout the development phase, a diligent approach to testing was applied to each separate component. For instance, Arduino boxes underwent continuity tests, while the dataset data was validated through a custom script. After the development phase concluded, additional manual tests were conducted to verify the system's functioning. Key functionalities such as the accurate calculations by the implemented differentially private SVM were confirmed. Print statements were employed to ensure the data sent between nodes, server, and SVM was as intended. Finally, the system's response to anomalous nodes was manually inspected to confirm that the server, upon detecting any anomalies, successfully sent an email. This rigorous testing methodology provided confidence in the overall performance and reliability of the system.

Chapter 6

Release and Feedback Phase Methodology

Upon completion of the testing phase, the Waterfall methodology dictates the next stages to be release and feedback collection. Hence, the code, together with thorough set-up instructions and documentation, has been made publicly available on the GitHub platform: https://github.com/Preffet/PPSVM

The public can access, review, and use the code, providing an open-source environment for continuous improvement. As for feedback, constructive criticisms, and suggestions are warmly welcomed from users and other developers who interact with the system via GitHub's built-in features: Pull requests and Discussions. Such feedback is invaluable, helping to identify potential issues, enhance the system's performance, and ensure its usability.

Chapter 7

Methodology Summary, Strengths and Limitations

In chapters 3-6, the project's development methodology was laid out in detail. Adhering to the Waterfall model, each phase cascaded into the next, covering requirements analysis, system design, implementation, testing, release, and feedback collection. The specific tools and methodologies chosen for each phase were also discussed.

The overall project's development process had its own strengths and limitations. The methodology of this project adhered to a waterfall model, which brought forth several strengths. It enhanced the reproducibility of each phase and the chronological structure of the model allowed for a clear progression through the project, making it easier to manage and maintain momentum.

However, the project faced a challenge when the router required for the agricultural sensor network was unavailable. The contingency plan, which was to develop a simulation of the agricultural sensor network, was successfully implemented in response. This experience demonstrated the project's resilience and adaptability, proving it could handle unforeseen obstacles effectively.

Reflecting on what could have been done better, a more robust contingency plan for hardware or equipment issues would have been beneficial, given the dependency of the project on specific tools.

Chapter 8

Results and Evaluation

8.1 Introduction

This chapter presents a thorough analysis of the project's outcomes, including an evaluation of the performance of the created SVM classifiers and the achievable privacy levels under different system parameters. Additionally, it provides an evaluation of the artefacts produced as well as a discussion on the limitations in evaluation process.

8.2 Evaluation of the Created Classifiers

In the initial evaluation, the nonprivatised SVMs displayed impressive accuracy, averaging 95.52% across all datasets and validating their effectiveness for anomaly detection in illuminance or similar linearly separable agricultural data. Results are depicted in Figure 8.1.

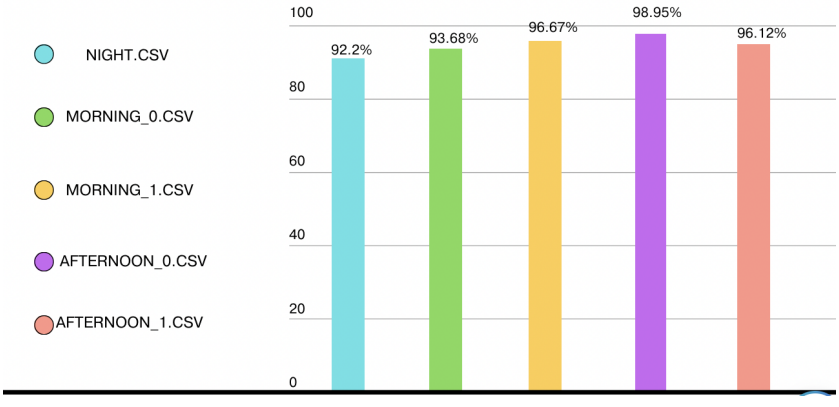

FIGURE 8.1: Designed SVM Model Performance Over all Datasets

Following the initial evaluation, the effect of adding Laplacian noise to the datasets was analysed, as prior research suggested potential performance and privacy improvement. The impact of this noise was assessed by conducting manual tests with different small values and observing their influence on SVMs' accuracy. Figure 8.2 illustrates the results: the addition of minimal noise occasionally led to an improvement in performance, but it was not a consistent effect. Furthermore, the overall

result was not beneficial, as there was a significant decrease in accuracy with the introduction of more noise.

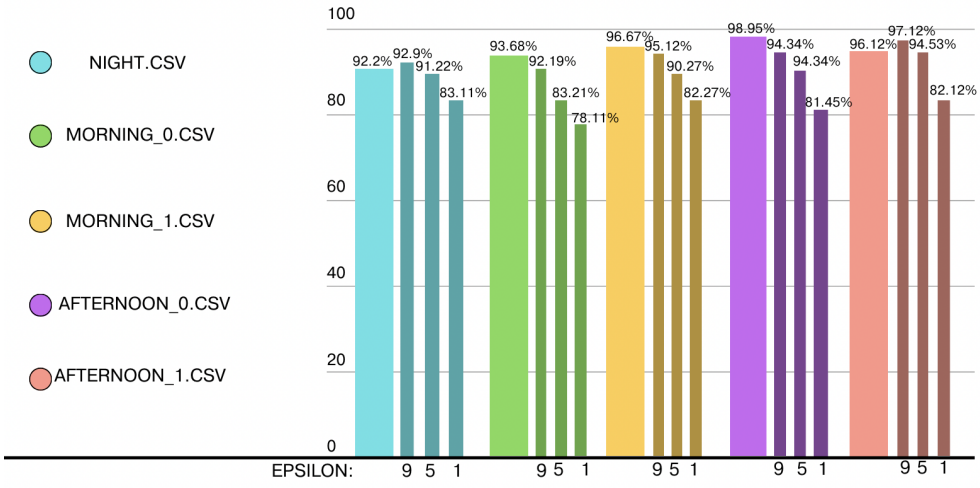

FIGURE 8.2: SVM Performance Using Noisy Datasets

Following this, the membership inference probabilities were computed for both the original datasets and those augmented with noise, specifically at a level equal to 1, a setting that had occasionally shown an improvement in performance. As illustrated in Figure 8.3, the impact of noise varied among datasets; in some cases, it led to an increase in privacy, while in others, it resulted in a decrease. Regardless of these fluctuations, the attacks were successful, with the highest success rate reaching 74.23%.

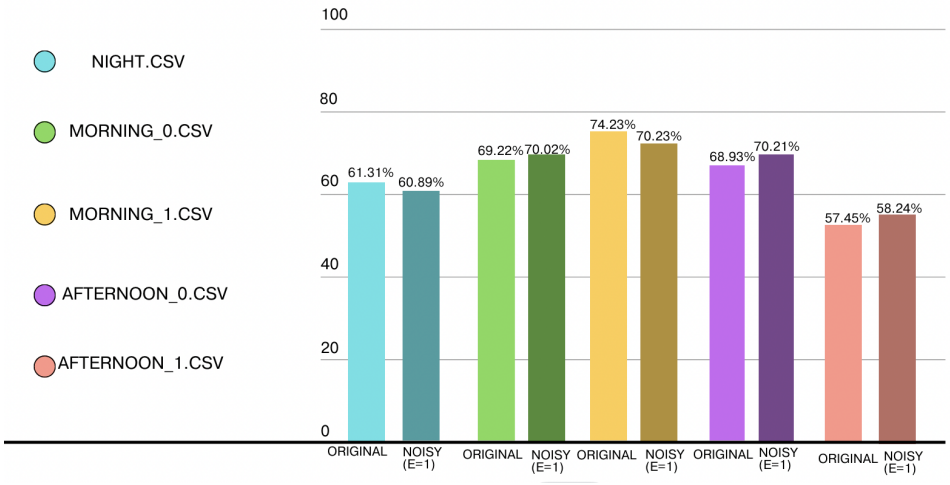

FIGURE 8.3: Membership Inference Attack Success Rates

Next, each dataset underwent manual evaluation for membership inference attacks using the differentially private SVM. The results can be seen in Figure 8.4 and were very promising; in some cases, the attacks were reduced by more than 20% with low epsilon values, as was the case for the dataset 'morning_0.csv'. Motivated by these results, the performance of the differentially private SVM was further evaluated on the 'morning_0.csv' dataset using epsilon of 0.1. Compared to the original performance (96.67%), the differentially private SVM achieved an accuracy of 91.3%. Although this resulted in a decrease in accuracy by 5.37%, it represented a significant gain in privacy. Therefore, the results suggest that the implemented differentially-private SVM is not only effective but also practically applicable in agricultural settings.

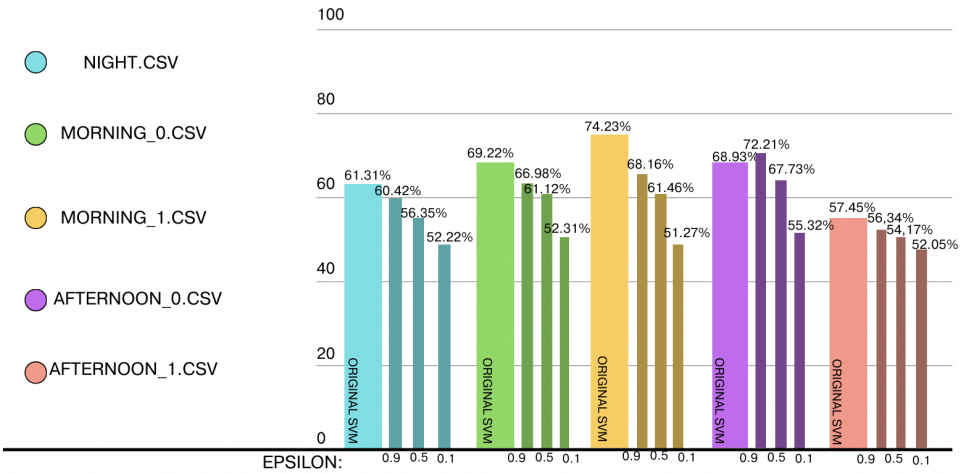

FIGURE 8.4: Membership Inference Attack Success Rates

8.3 Limitations In Evaluation

It is important to acknowledge the limitations of the evaluation conducted. Due to time constraints, evaluations were performed on only five different datasets, all containing illuminance data. For a more comprehensive assessment, it would be beneficial to include a wider range of datasets with additional features, representing different contexts such as soil moisture, pH level, and other relevant variables. This would provide a more diverse and realistic evaluation of the system's performance. Furthermore, it is worth noting that the literature on privacy attacks, specifically membership inference techniques, is constantly evolving. While the project incorporated an established technique, newer approaches, such as those utilising neural networks, have been proposed [58, 37]. To stay up to date with the latest attacks, it is recommended to explore and incorporate these emerging techniques in future iterations of the system since it would provide the farmers with a more accurate information about the system's resilience.

8.4 Artifacts Produced

The project has generated a variety of valuable artifacts that have practical implications, specifically for farmers interested in checking the validity of illuminance data within their agricultural sensor networks or those who wish to do privacy preserving machine learning with linear SVMs using their own datasets.

First, a substantial collection of over 1008 illuminance recordings was acquired. They closely reflect outdoor illuminance conditions in the UK around the April-May. This dataset not only served to validate the models, but also represents a valuable resource for further model testing, or direct application in farming operations.

Next, custom scripts were developed to balance and validate the datasets, which can be re-used in other machine learning projects.

Following this, a notable outcome of this project is the development of membership inference attack test script. This script is not limited to SVMs, because any machine learning model can be evaluated using them by simply exchanging the 'fit' and 'predict' functions of the models.

Next, agricultural sensor network simulation system was developed. It can serve as a testing bed where users can experiment and resolve any potential issues before implementing their machine learning algorithms into the real ASNs.

Additionally, at its current state, the developed simulation system effectively distinguishes between valid and invalid illuminance recordings outdoors in the UK between April-May. Yet, its potential goes beyond this immediate application since it also offers a solid starting point for farmers and other users to experiment with their own datasets using the developed models as the scripts for hyperparameter tuning and accessing both the model's accuracy and privacy loss with different parameters have been implemented.

8.5 Fullfillment of Project Requirements

The project successfully satisfied its functional requirements:

Requirement 1, which stated that SVMs should achieve a classification accuracy of at least 85%, was fulfilled. All SVMs achieved an accuracy around 95-96%. When different noise levels are introduced to the data based on user preference, the accuracy still remains at or above the required 85%.

Requirement 2 specified the use of a differentially private SVM to help preserve the privacy of the training data. This requirement was met as a differentially private SVM was implemented and offered as an option to the users.

Requirements 3 and 4, which required the system to send automated alerts upon detection of anomalies and update an audit log with information about detected

anomalous nodes, respectively, were also fulfilled. The information about any detected malicious nodes is added to a blocklist file, and users receive an email alert about the event, with the blocklist.csv file attached.

Requirement 5, which called for an option to adjust the level of privacy was accomplished. Users are provided with three different options for adding noise to the dataset and differentially private SVM.

Non-functional requirements were also fulfilled: code follows PEP8 standards and colour coding for terminal messages has been implemented as can be seen in Figures 4.7, 4.8, 4.6.

8.6 Summary

This section has conducted an in-depth examination of the results obtained from the project, incorporating a performance evaluation of the constructed SVM classifiers, as well as an analysis of the attainable privacy levels given different system settings. Additionally, it includes an assessment of the developed artefacts and provides a discussion on the limitations faced during the evaluation process.

Chapter 9

Conclusions and Future Directions

9.1 Introduction

This concluding section of this report will encompass a discussion on the extent to which the original project aim and objectives were met along with a summary of what has been learnt. Furthermore, potential areas for future work will be identified, highlighting future research opportunities.

9.2 Meeting Project Aims And Objectives

The extent to which the project objectives were met is as follows:

1. The first objective was achieved by identifying two differential privacy learning approaches: output and objective perturbation and analysing their applications.

2. The second objective was successfully fulfilled, with the proposed agricultural sensor network simulation being developed according to the specification.

3. The third objective was also completed successfully. The "SquirrelBox" agricultural sensor box was assembled and collected over 1008 illuminance readings.

4. The fourth objective was achieved by implementing and tuning a simple SVM classifier using the scikit-learn library, as discussed in the "Results and Evaluation" section, it showcased high accuracy.

5. The fifth objective was also fulfilled. A privacy-preserving differentially private SVM was developed and tuned.

6. The sixth objective, unfortunately, was not met. The intention was to implement the developed privacy-preserving SVM into a real agricultural sensor network. However, due to unforeseen circumstances including limited access to the necessary router, this implementation could not be carried out within the project timeframe.

7. The seventh objective was fully realised. Thorough summarisation and evaluation of the performance of the created prototype and classifiers was conducted, with the results indicating the effectiveness of the developed system.

Nevertheless, the overall objectives of the project were mostly achieved and lead to the fulfilment of the overall aim of the project, which was to design a prototype privacy-preserving SVM solution for ensuring agricultural sensor network data integrity.

9.3 Future Work

The developed system using linear SVMs successfully reduced membership attack chances while maintaining good performance. However, further improvements can be made. Future research could implement differentially private SVMs with non-linear Radial Basis Function (RBF) to enhance accuracy, and support differentially private multiclass classification for greater versatility.

Furthermore, testing the system within a real agricultural sensor network would provide robustness evaluation and identify potential areas for enhancement. Additionally, incorporating diverse datasets, such as pH levels and temperature readings, could improve predictive power.

Finally, other privacy-preserving methods like synthetic data generation could be explored to balance data utility and privacy preservation. These efforts would advance the system's capabilities, potentially providing a more effective solution for agricultural sensor networks.

9.4 Summary of What Has Been Learnt

All in all, embarking on this project, I had limited prior knowledge about machine learning, soldering, agricultural sensor networks, and differential privacy - the latter being only in the context of database queries rather than machine learning. This posed a significant challenge and established a steep learning curve.

Despite the difficulties, the process was deeply enlightening and enriching. I dove headfirst into unfamiliar territories, acquiring and honing skills in various domains. I navigated through the complexities of machine learning, grasped the practicality of soldering, explored the intricacies of agricultural sensor networks and privacy concerns in them, and delved deeper into the application of differential privacy in machine learning, which not only allowed me to improve my programming skills in Python, but explore different libraries such as Scikit-Learn and ART which I have not used before. However, the journey was not merely about acquiring knowledge, but also about developing a problem-solving mindset, resilience in the face of challenges, and the ability to synthesize cross-disciplinary information.

In essence, this project turned out to be a valuable learning experience, prompting both personal and academic growth. It not only equipped me with a new set of skills but also broadened my perspective, fostering a more profound understanding and appreciation of machine learning, agricultural sensor networks, and data privacy. The knowledge and experience gained throughout this project will undoubtedly serve as a solid foundation for my future endeavours.

Appendix A

Sensor Node Simulation Script

```
from datetime import datetime
import random
import socket
import time
import sys
import os

# Public variables
# IP address used for client-server connection
IP = socket.gethostbyname('localhost')
# port used for client-server connection
PORT = 1228
# full address
ADDR = (IP, PORT)
# message size
SIZE = 1024
# message format
FORMAT = "utf-8"


# ANSI escape codes to print coloured/bold text
class Colours:
    ENDC = '\033[0m'
    CYAN = '\033[96m'
    BOLD = '\033[1m'
    GREEN = '\033[92m'
    YELLOW = '\033[93m'
    RED = '\033[91m'
    BLUE = '\033[94m'
    ORANGE = '\033[38;5;173m'


# Program entry point
def main():
    # Try to connect to the server
    connected = False
    try:
        # setup sockets
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client.connect(ADDR)
```

```
41          connected = True
42          # Print the header - client IP address {IP}:{PORT}
43          print(f"\n       "
44                f" Client IP "
45                f"{Colours.BOLD}{Colours.BLUE}"
46                f"{client.getsockname()[0]}:"
47                f"{client.getsockname()[1]}"
48                f"{Colours.ENDC}")
49          # Print the header - connected to the server at {IP}:{PORT}
50          print(f"{Colours.BOLD}{Colours.BLUE}{Colours.ENDC}"
51                f" Connected to the server at {Colours.BOLD}{Colours.GREEN}"
52                f"{IP}:{PORT}{Colours.YELLOW} {Colours.ENDC}")
53          print(f"{Colours.BLUE}------------"
54                f"{Colours.CYAN}------------"
55                f"{Colours.GREEN}------------"
56                f"{Colours.YELLOW}----------"
57                f"{Colours.ENDC}\n")
58      # Catch the errors that occur when setting up the connection
59      except:
60          connected = False
61          print(f"\n{Colours.BOLD}{Colours.RED}"
62                f"Could not connect to the server. Quitting..."
63                f"{Colours.ENDC}")
64          sys.exit()
65
66      # when client successfully connects to the server,
67      # every 10 seconds send randomly chosen data from
68      # the valid.csv file to simulate an honest sensor node
69      while connected:
70          try:
71              # sleep for 10s
72              time.sleep(10)
73              # get the path to the current directory
74              dir_path = os.path.dirname(os.path.dirname(__file__))
75
76              # check time & pick a valid dataset
77              current_hour = datetime.now().hour
78
79              if (6 > current_hour >= 0) or (21 <= current_hour < 24):
80                  # get the dataset with valid night data
81                  path_to_data = dir_path + "/datasets/simulation_nodes/valid/night.csv"
82
83              elif 6 <= current_hour < 13:
84                  # get the dataset with valid data for the first half of the day
85                  path_to_data = dir_path + "/datasets/simulation_nodes/valid/day1.csv"
86
87              else:
88                  # get the dataset with valid data for the second half of the day
89                  path_to_data = dir_path + "/datasets/simulation_nodes/valid/day2.csv"
90
91              # Read a random line from the dataset containing valid sensor readings
```

```
92              with open(path_to_data) as f:
93                  lines = f.readlines()
94                  msg = random.choice(lines[1:])
95                  # print the data which will be sent
96                  print(f"{Colours.BOLD}{Colours.CYAN}"
97                        f"{Colours.ENDC} Sent: "
98                        f"{Colours.BOLD}{Colours.CYAN}"
99                        f"{msg}{Colours.ENDC}", end='')
100             # send the message to the server
101             client.send(msg.encode(FORMAT))
102             # receive the message from the server
103             msg = client.recv(SIZE).decode(FORMAT)
104
105             # check if the node got blocked, if yes,
106             # print the information
107             if msg == "blocked":
108                 # print the data the server received
109                 # and that the node got blocked
110                 print(f"\n{Colours.BOLD}{Colours.RED}{Colours.ENDC}"
111                       f" Blocked by the server for sending"
112                       f"{Colours.BOLD}{Colours.RED} "
113                       f"malicious data{Colours.ENDC}\n")
114                 # disconnect
115                 connected = False
116             # if node did not get blocked,
117             # print the data that the server received
118             else:
119                 # print the data the server received
120                 print(f"{Colours.BOLD}{Colours.YELLOW}\n{Colours.ENDC}"
121                       f" Decision received from the server:"
122                       f" {Colours.BOLD}{Colours.YELLOW}{msg}{Colours.ENDC}\n")
123
124         # Catch the errors that occur when sending messages
125         except:
126             print(f"\n{Colours.BOLD}{Colours.RED}"
127                   f"Could not send the message. Quitting..."
128                   f"{Colours.ENDC}")
129             sys.exit()
130
131
132 # Program entry point
133 if __name__ == "__main__":
134     main()
```

Appendix B

Server Simulation Script

```
import os
import socket
import smtplib
import threading
from datetime import datetime
from email.mime.application import MIMEApplication
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

# Public variables
# IP address used for client-server & server-SVM connection
IP = socket.gethostbyname('localhost')
# message size
SIZE = 1024
# port used for client-server connection
PORT = 1228
# full address for client-server connection
ADDR = (IP, PORT)
# port used for server-cloud(SVM) connection
SVM_PORT = 59991
# full address for connection to cloud(SVM)
SVM_ADDR = (IP, SVM_PORT)
# message format
FORMAT = "utf-8"


# ANSI escape codes to print coloured/bold text
class Colours:
    ENDC = '\033[0m'
    CYAN = '\033[96m'
    BOLD = '\033[1m'
    GREEN = '\033[92m'
    YELLOW = '\033[93m'
    RED = '\033[91m'
    BLUE = '\033[94m'
    ORANGE = '\033[38;5;173m'
```

```
41 # ------------------------------------------------------------------------------ #
42 # ------------------------------------------------------------------------------ #
43
44 # function to send an email to the administrators informing
45 # them about the detected malicious node
46 def send_email(date, time, ip, data):
47
48     # get the path to the admin email addresses
49     path_to_emails = os.path.dirname(os.path.dirname(__file__))
50     path_to_emails = path_to_emails + "/Email/admin_email.csv"
51     file = open(path_to_emails, "r")
52     admin_emails_string = file.read()
53     admin_emails_list = admin_emails_string.split(",")
54     file.close()
55
56     port = 465  # For SSL
57     # not an actual gmail account password,
58     # it is an app password specifically created
59     # to authenticate this application
60     password = 'gcjdhpydrjstnibz'
61     sender_email = 'a15764291@gmail.com'
62
63     # get the path to the email html document
64     email_html = os.path.dirname(os.path.dirname(__file__))
65     email_html = email_html + "/Email/simulation_email.html"
66     with open(email_html, 'r', encoding='utf-8') as f:
67         html_string = f.read()
68
69     # replace placeholder text with the actual data
70     html_string = html_string\
71         .replace("{date}", date)\
72         .replace("{time}", time)\
73         .replace("{ip}", ip)\
74         .replace("{data}", data)
75
76     # format the email
77     message = MIMEMultipart()
78     message['Subject'] = "Potential Malicious Node Identified on Network"
79     message['From'] = sender_email
80     message['To'] = admin_emails_string
81
82     # attach the defined html document
83     message.attach(MIMEText(html_string, "html"))
84
85     # get the path to the blocklist.csv file
86     path_to_blocklist_csv = os.path.dirname(__file__)
87     path_to_blocklist_csv = path_to_blocklist_csv
88     + "/detection_system_files/blocklist.csv"
89
90     # attach the blocklist file
91     attach_file_to_email(message, path_to_blocklist_csv)
```

```
92      # convert email to text
93      email_string = message.as_string()
94
95
96      # send the email
97      server = smtplib.SMTP_SSL("smtp.gmail.com", port)
98      server.login(sender_email, password)
99      server.sendmail(sender_email, admin_emails_list, email_string)
100     server.quit()
101
102 # ------------------------------------------------------------------------------ #
103 # ------------------------------------------------------------------------------ #
104
105 # function to attach the blocklist csv
106 # file to the email
107 def attach_file_to_email(email_message, filename):
108     # Open the attachment file for reading in binary mode,
109     # and make it into MIMEApplication type
110     with open(filename, "rb") as f:
111         file_attachment = MIMEApplication(f.read())
112     # Add header/name to the attachments
113     file_attachment.add_header(
114         "Content-Disposition",
115         f"attachment; filename= blocklist.csv",
116     )
117     # Attach the file to the message
118     email_message.attach(file_attachment)
119
120 # ------------------------------------------------------------------------------ #
121 # ------------------------------------------------------------------------------ #
122
123 # function to handle the communication
124 # between the server and the client
125 def handle_client(conn, addr):
126
127     # print information when a new connection is received
128     # new connection from {IP}:{PORT}
129     print(f"\n{Colours.BOLD}{Colours.CYAN}"
130           f"{Colours.ENDC}"
131           f" New connection from "
132           f"{Colours.BOLD}{Colours.CYAN}"
133           f"{addr[0]}:{addr[1]}\n", end='')
134
135     # get the path to the clients.csv file
136     path_to_clients_csv = os.path.dirname(__file__)
137     path_to_clients_csv = path_to_clients_csv + "/detection_system_files/clients.csv"
138
139     # add the client to the clients.csv
140     # and assign id if the client does not already exist
141     with open(path_to_clients_csv, "r+") as file:
142         id = len(file.readlines())+1
```

```
143         if not(f"{addr[0]}:{addr[1]}" in file.read()):
144             with open(path_to_clients_csv, "a") as f:
145                 f.write(f"{id},{addr[0]}:{addr[1]}\n")
146     connected = True
147 
148 
149     # main function loop,
150     # runs while the client is connected
151     while connected:
152         malicious = False
153         # get the path to the blocklist.csv file
154         path_to_blocklist_csv = os.path.dirname(__file__)
155         path_to_blocklist_csv = path_to_blocklist_csv
156         + "/detection_system_files/blocklist.csv"
157         # check if the connected client is not in the blocklist,
158         # if not, receive the message, else close the connection
159         with open(path_to_blocklist_csv, "a+") as blocklist:
160             if not (f"{addr[0]}:{addr[1]}" in blocklist.read()):
161                 msg = conn.recv(SIZE)
162 
163             # if the message is empty, close the connection
164             # but do not add the client to the blocklist
165             if not(len(msg) <= 0):
166 
167                 # Print the message: Message from {IP}:{PORT} : {MESSAGE}
168                 print(f"\n{Colours.BOLD}{Colours.YELLOW}{Colours.ENDC}"
169                       f"{Colours.ENDC} Message from {addr[0]}:{addr[1]}:"
170                       f"{Colours.BOLD}{Colours.YELLOW}"
171                       f" {msg.decode(FORMAT)}{Colours.ENDC}", end='')
172 
173                 # send received data to the cloud to be checked
174                 # and get the decision whether the data is anomalous
175                 # if it is, inform the administrators and add the ip
176                 # to the blocklist
177                 decision = handle_cloud(msg)
178                 if not decision == "valid":
179                     malicious = True
180                     data = [float(x) for x in msg.decode().split(",")]
181 
182                     # print that malicious data was received: {data}
183                     print(f"\n{Colours.BOLD}{Colours.RED}{Colours.ENDC}"
184                           f" Malicious data received: "
185                           f"{Colours.BOLD}{Colours.RED}"
186                           f"{float(data[0])},{float(data[1])}"
187                           f"{Colours.ENDC}", end='')
188 
189                     # add ip to the blocklist
190                     current_date_and_time = datetime.now()
191                     blocklist.write(f"{addr[0]}:{addr[1]},{current_date_and_time}\n")
192 
193                     # print: IP added to the blocklist {IP}
```

```
194                     print(f"\n{Colours.BOLD}{Colours.RED}{Colours.ENDC}"
195                           f" IP added to the blocklist: {Colours.BOLD}"
196                           f"{Colours.RED}{addr[0]}:{addr[1]}"
197                           f"{Colours.ENDC}", end='\n')
198
199
200
201                     # inform the node that it got blocked
202                     message = "blocked"
203                     conn.send(message.encode(FORMAT))
204                     # email the administrators the
205                     information about the malicious node
206                     # and updated blocklist
207                     send_email(str(current_date_and_time)[0:10],
208                                str(current_date_and_time)[11:19],
209                                f"{addr[0]}:{addr[1]}", msg.decode())
210                     # close the connection
211                     connected = False
212                     conn.close()
213
214                 # if the data is valid, inform the client
215                 else:
216                     message = "valid"
217                     conn.send(message.encode(FORMAT))
218
219             # close connection if received message is empty
220             else:
221                 connected = False
222                 conn.close()
223     # close connection if the node was malicious
224     conn.close()
225
226     # once the connection is closed, print information:
227     # connection closed {IP}:{PORT},
228     # reason: {reason for closing the connection}
229     # also, update the number of currently active connections
230     reason = "closed by the client" if not malicious else "malicious node"
231     print(f"{Colours.BOLD}{Colours.RED}{Colours.ENDC}"
232           f" Connection closed {Colours.BOLD}{Colours.RED}"
233           f"{addr[0]}:{addr[1]}{Colours.ENDC}")
234     print(f"   Reason: "
235           f"{Colours.BOLD}{Colours.RED}"
236           f"{reason}{Colours.ENDC} \n", end='')
237     # Print the number of currently active connections
238     print(f"{Colours.BOLD}{Colours.GREEN}{Colours.ENDC}"
239           f" Active connections: {Colours.BOLD}"
240           f"{Colours.GREEN}{threading.activeCount() - 2}\n{Colours.ENDC}", end='')
241
242 # ------------------------------------------------------------------------ #
243 # ------------------------------------------------------------------------ #
244
```

```
245 # function to handle the communication
246 # between the server and the cloud (SVM)
247 def handle_cloud(msg):
248     decision = "valid"
249     # set up the server-cloud connection
250     cloud_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
251     cloud_server.connect(SVM_ADDR)
252 
253     data_to_send = [float(x) for x in msg.decode(FORMAT).split(",")]
254 
255 
256     # do an initial check if the values are
257     # within a reasonable range
258     time = data_to_send[1]
259     lux = data_to_send[0]
260 
261     if time < 0 or time > 24 or lux < 0 or lux > 20000:
262         decision = "anomalous"
263 
264     # if initial checks are passed, send the data to the cloud
265     if decision != "anomalous":
266         # send received data to the cloud
267         cloud_server.send(msg)
268 
269         # print information: sent data to SVM: {data}
270         print(f"\n{Colours.BOLD}{Colours.YELLOW}{Colours.ENDC}"
271               f"{Colours.ENDC} Sent data to SVM:"
272               f"{Colours.BOLD}{Colours.YELLOW}"
273               f" {(data_to_send[0])},{(data_to_send[1])}{Colours.ENDC}")
274 
275         # receive the decision if the data is malicious
276         decision = cloud_server.recv(SIZE).decode(FORMAT)
277         # print information about the decision,
278         # decision: {decision}
279         if decision == "valid":
280             print(f"{Colours.BOLD}{Colours.GREEN}{Colours.ENDC}"
281                   f"{Colours.ENDC} Returned decision:"
282                   f"{Colours.BOLD}{Colours.GREEN} {decision}{Colours.ENDC}\n ")
283         else:
284             print(f"{Colours.BOLD}{Colours.RED}{Colours.ENDC}"
285                   f"{Colours.ENDC} Returned decision:"
286                   f"{Colours.BOLD}{Colours.RED} {decision}{Colours.ENDC}\n ")
287     return decision
288 
289 # ------------------------------------------------------------------------ #
290 # ------------------------------------------------------------------------ #
291 
292 # program entry point
293 def main():
294     try:
295         server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
296         server.bind(ADDR)
297         server.listen()
298         # Print information: Listening on {IP}:{PORT}
299         print(f"\n{Colours.BOLD}{Colours.BLUE}"
300               f"{Colours.ENDC}"
301               f" The server is listening on "
302               f"{Colours.BOLD}{Colours.GREEN}"
303               f"{IP}:{PORT}"
304               f"{Colours.YELLOW} "
305               f"{Colours.ENDC}")
306 
307         print(f"{Colours.BLUE}------------"
308               f"{Colours.CYAN}------------"
309               f"{Colours.GREEN}----------"
310               f"{Colours.YELLOW}------------"
311               f"{Colours.ENDC}")
312 
313 
314         # Main program loop
315         while True:
316 
317             # Wait for the client to connect and accept the connection
318             conn, addr = server.accept()
319 
320             # Create a separate thread to handle the client
321             thread = threading.Thread(target=handle_client, args=(conn, addr))
322             thread.start()
323 
324             # Print information about active connections.
325             # Active connections: {number of active connections}
326              print(f"{Colours.BOLD}{Colours.GREEN}"
327                    f"{Colours.ENDC}"
328                    f" Active connections: "
329                    f"{Colours.BOLD}{Colours.GREEN}"
330                    f"{threading.activeCount() - 1}"
331                    f"\n")
332 
333 
334     # Quit if errors occur
335     except:
336         os._exit(1)
337 
338 
339 # Program entry point
340 if __name__ == "__main__":
341     main()
342 
343 
```

Appendix C

Email HTML Template

```
<html lang="en">
  <body style="text-align: center; margin: 0; padding-top: 10px;
  ↪  padding-bottom: 10px; padding-left: 0; padding-right: 0;
  ↪  -webkit-text-size-adjust: 100%;background-color: #f2f4f6; color:
  ↪  #000000" align="center">

  <!-- Fallback force center content -->
  <div style="text-align: center;">

    <!-- image -->
    <img style="width: 600px; max-width: 600px; height: 300px;
    ↪  max-height: 350px; text-align: center;" alt=" "
    ↪  src="https://cdni.iconscout.com
    /illustration/premium/thumb/digital
    -agriculture-3859571-3315214.png" align="center" width="50"
↪   height="450">
    <!-- image -->

    <!-- single column section -->
    <table align="center" style="text-align: center; vertical-align:
    ↪  top; width: 600px; max-width: 600px; background-color:
    ↪  #ffffff;" width="600">
        <tbody>
          <tr>
            <td style="width: 596px; vertical-align: top; padding-left:
            ↪  30px; padding-right: 30px; padding-top: 30px;
            ↪  padding-bottom: 40px;" width="596">

              <h1 style="font-size: 20px; line-height: 24px;
              ↪  font-family: 'Helvetica', Arial, sans-serif;
              ↪  font-weight: 600; text-decoration: none; color:
              ↪  #010359;">Potential Malicious Node Identified on
              ↪  Network</h1>
              <h3 style="color:#010359;">Isolation & Review
              ↪  Required</h3><br>

              <p style="font-size: 15px; line-height: 24px;
              ↪  font-family: 'Helvetica', Arial, sans-serif;
              ↪  font-weight: 400; text-decoration: none; color:
              ↪  #919293;text-align:left">Dear Administrator,<br><br>
```

```
                Our system has detected a potentially malicious node on
                 ↪  your agricultural sensor network.
                It was identified on <b
                 ↪  style="color:#0277FF;">{date}</b> at <b
                 ↪  style="color:#0277FF;">{time}</b> and has the IP
                 ↪  address <b style="color:#0277FF;">{ip}</b>.
                The node was found to be sending the following
                 ↪  malicious data:
                <b style="color:#0277FF;">{data}</b>.<br><br>
                We recommend that you take immediate action to isolate
                 ↪  the node, review its activity,
                and if it is found to be malicious, update the
                 ↪  blocklist in your router to
                add the IP address of the node to prevent any future
                 ↪  malicious activity.
                To assist you in this process, we have attached the
                 ↪  updated blocklist file to this email.<br><br>
                Thank you for your attention to this matter.<br><br>
                Sincerely,<br><br>
                Security Team</p>
                <br>
              <p style="font-size: 15px; line-height: 24px;
               ↪  font-family: 'Helvetica', Arial, sans-serif;
               ↪  font-weight: 400; text-decoration: none; color:
               ↪  #919293;">If you have any further questions feel free
               ↪  to</p>

              <!-- Start button -->
              <a href="mailto:a15764291@gmail.com" target="_blank"
               ↪  style="background-color:#3ED1A7; font-size: 15px;
               ↪  line-height: 22px; font-family: 'Helvetica', Arial,
               ↪  sans-serif; font-weight: normal; text-decoration:
               ↪  none; padding: 12px 15px; color: #ffffff;
               ↪  border-radius: 5px; display: inline-block;
               ↪  mso-padding-alt: 0;">
                  <!--[if mso]>
                  <i style="letter-spacing: 25px; mso-font-width:
↪  -100%; mso-text-raise: 30pt;"> </i>
                <![endif]-->

                  <span style="mso-text-raise: 15pt; color:
                   ↪  #ffffff;">Contact us</span>
              </a>
              <!-- End button -->
            </td>
          </tr>
        </tbody>
      </table>
      <!-- End single column section -->

  </div>
```

```
  </body>
</html>
```

Appendix D

Cloud Simulation Script

```
# Data processing
import pickle as pkl
import numpy as np
from numpy import where
import pandas as pd
# Model and performance
from sklearn import svm
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# networking
import socket
# multithreading
import threading
# dp data privatisation
from privacy_preserving_svms import objective_function_perturbation_SVM
 ↪  as obj_perturb_SVM
from privacy_preserving_svms.Laplace_dataset_privatiser import
 ↪  DataConverter, LaplacePrivacyPreserver


# ANSI escape codes to print coloured/bold text
class colours:
    ENDC = '\033[0m'
    CYAN = '\033[96m'
    BOLD = '\033[1m'
    GREEN = '\033[92m'
    YELLOW = '\033[93m'
    RED = '\033[91m'
    BLUE = '\033[94m'
    ORANGE = '\033[38;5;173m'


# public variables
# message size
SIZE = 1024
# port of the server
SERVER_PORT = 59991
# message format
FORMAT = "utf-8"
# available choices for epsilon values when
# running SVMs in privacy-preserving way
```

```
39 EPSILON_VALUES = [0.9, 0.5, 0.1]
40 # privacy preserving SVM lambda and h values
41 SVM_H = 5
42 SVM_LAMBDA = 0.2
43
44
45 # function to get the chosen SVM type as
46 # well as the privacy epsilon parameters from the user
47 def get_svm_parameters():
48     # small function to print all available epsilon options
49     def print_options(epsilon_values):
50         print(f"{colours.BOLD}{colours.BLUE}1.
          ↪  {colours.ENDC}{epsilon_values[0]}\n"
51               f"{colours.BOLD}{colours.CYAN}2.
                 ↪  {colours.ENDC}{epsilon_values[1]}\n"
52               f"{colours.BOLD}{colours.GREEN}3.
                 ↪  {colours.ENDC}{epsilon_values[2]}\n")
53
54     # small function to check if the entered value is correct
55     # and if it is, return it
56     def get_value(number_up_to):
57         while True:
58             try:
59                 number = input(f"Enter a number between 1 and
                   ↪  {number_up_to}: ")
60                 if number.isdigit():
61                     number = int(number)
62                 else:
63                     raise ValueError()
64                 if 1 <= number <= number_up_to:
65                     break
66                 raise ValueError()
67             except ValueError:
68                 print(f"Input must be an integer between 1 and
                   ↪  {number_up_to}.")
69         return number
70
71     # get the parameters from the user
72     # 1.Ask the user to choose the SVM type:
73     # - Not privacy preserving
74     # - Preserving dataset privacy
75     # - Differentially private SVM
76     # - Privatised dataset + differentially private svm
77     print(f"\n{colours.BOLD}{colours.BLUE}Choose The SVM
       ↪  Type:{colours.ENDC}\n"
78           f"{colours.BOLD}{colours.CYAN}1.{colours.ENDC}Not Private\n"
79           f"{colours.BOLD}{colours.GREEN}2.{colours.ENDC}Using
             ↪  Privatised Dataset\n"
80
             ↪  f"{colours.BOLD}{colours.YELLOW}3.{colours.ENDC}Differentially
             ↪  Private SVM\n"
```

```
81          f"{colours.BOLD}{colours.ORANGE}4.{colours.ENDC}Privatised
             ↪ Dataset + Differentially Private SVM\n")
82      svm_type = get_value(4)
83
84      # 2.Ask the user to choose the privacy parameter
85      # if they choose option 2, 3, or 4
86      # (privatised dataset,
87      # differentially private SVM
88      # privatised dataset + differentially private SVM)
89
90      # if it is option 2 or 3 ask for one
91      # epsilon value
92      epsilon1 = 0
93      if svm_type in (2, 3):
94          print(f"\n{colours.BOLD}{colours.BLUE}Choose The Epsilon Value
             ↪ ", end='')
95          if svm_type == 2:
96              print(f"(default is 5):{colours.ENDC}")
97              print_options(epsilon_values=[i * 10 for i in
                 ↪ EPSILON_VALUES])
98              epsilon1 = EPSILON_VALUES[get_value(3) - 1] * 10
99          if svm_type == 3:
100             print(f"(default is 0.5):{colours.ENDC}")
101             print_options(epsilon_values=EPSILON_VALUES)
102             epsilon1 = EPSILON_VALUES[get_value(3) - 1]
103
104     # if it is option 4 ask for two
105     # epsilon values
106     # one for dataset privatisation and
107     # another for SVM
108     epsilon2 = 0
109     if svm_type == 4:
110         print(f"\n{colours.BOLD}{colours.BLUE}Choose the Epsilon Value
             ↪ for Dataset Privatisation ")
111         print(f"(default is 5.0):{colours.ENDC}")
112         print_options(epsilon_values=[i * 10 for i in EPSILON_VALUES])
113         epsilon1 = EPSILON_VALUES[get_value(3) - 1] * 10
114         print(f"\n{colours.BOLD}{colours.BLUE}Choose the Epsilon Value
             ↪ for Privacy-Preserving SVM ")
115         print(f"(default is 0.5):{colours.ENDC}")
116         print_options(epsilon_values=EPSILON_VALUES)
117         epsilon2 = EPSILON_VALUES[get_value(3) - 1]
118
119     # Print information about the chosen parameters
120     # non privatised svm
121     if svm_type == 1:
122         print(f"{colours.BLUE}"
123               f"\nRunning a non-privatised SVM"
124               f"{colours.ENDC}")
125     # svm using privatised dataset
126     if svm_type == 2:
```

```
127         print(f"{colours.BLUE}"
128               f"\nRunning a SVM with a privatised dataset with epsilon"
129               f" {epsilon1}{colours.ENDC}")
130     # differentially private svm
131     if svm_type == 3:
132         print(f"{colours.BLUE}"
133               f"\nRunning a Differentially-Private SVM with epsilon"
134               f" {epsilon1}{colours.ENDC}")
135     # differentially private svm with a privatised dataset
136     if svm_type == 4:
137         print(f"{colours.BLUE}"
138               f"\nRunning a Differentially-Private SVM (epsilon
                  ↪  {epsilon2})\n"
139               f"with a Privatised Dataset (epsilon {epsilon1})"
140               f"{colours.ENDC}")
141     # separator line (-----)
142     print(f"{colours.BLUE}--------------{colours.CYAN}"
143           f"--------------{colours.GREEN}--------------"
144           f"{colours.YELLOW}--------------{colours.ENDC}")
145
146     # return the SVM choice and epsilon values
147     return svm_type, epsilon1, epsilon2
148
149
150 # ------------------------------------------------------- #
151 # ------------------------------------------------------- #
152
153 # Function to choose & load the correct training
154 # datasets depending on the system time
155 def choose_training_dataset():
156     # Get the current hour, at the moment, it is set to 23 for testing
        ↪  purposes, but
157     # for real time testing, change current_hour variable value to
        ↪  datetime.now()
158     # and then go to client.py file and change the current_hour
        ↪  variable to
159     # datetime.now() too. These both values must match.
160     current_hour = 10  # datetime.now().hour
161     df2 = pd.DataFrame()
162     # if it is night
163     if (6 > current_hour >= 0) or (21 <= current_hour < 24):
164         df1 = pd.read_csv('../datasets/training/balanced/night.csv',
165                           header=0, sep=',')
166
167     # if it is first half of the day
168     elif 6 <= current_hour < 13:
169         df1 =
            ↪  pd.read_csv('../datasets/training/balanced/morning_0.csv',
170                           header=0, sep=',')
171         df2 =
            ↪  pd.read_csv('../datasets/training/balanced/morning_1.csv',
```

```
172                             header=0, sep=',')
173     # if it is second half of the day
174     else:
175         df1 =
            ↪ pd.read_csv('../datasets/training/balanced/evening_0.csv',
176                             header=0, sep=',')
177         df2 =
            ↪ pd.read_csv('../datasets/training/balanced/evening_1.csv',
178                             header=0, sep=',')
179     # return the required dataframes
180     return df1, df2
181
182
183 # function to privatise the dataset
184 # by adding laplace noise
185 def privatised_training_dataset(eps):
186     # import the dataset (with the headers)
187     df =
        ↪ pd.read_csv('../datasets/all_data/all_data_including_anomalous.csv',
188                       header=0, sep=',')
189
190     # define the scaler
191     scaler = StandardScaler()
192
193     # get X and y values from the dataset
194     X = df.loc[:, ['Lux', 'Float time value']]
195     y = df['Label']
196
197     # convert to numpy array
198     X = X.values
199     y = y.values
200
201     # define data converter
202     ad = DataConverter()
203     data_input = ad.convert_from_original(X)
204     target_values = ad.convert_from_original(y)
205     # define the data privatiser
206     data_privatiser = LaplacePrivacyPreserver(eps * 10)
207     # get data sensitivity value
208     input_sensitivity =
        ↪ data_privatiser.get_data_sensitivity_values(data_input)
209     privatised_data = data_privatiser.privatise_data(
210         data_input,
211         sensitivity_values_list=input_sensitivity)
212     # normalise the data
213     privatised_data =
        ↪ scaler.fit_transform(pd.DataFrame(privatised_data,
        ↪ columns=['Lux', 'Float time value']))
214     privatised_data = pd.DataFrame(privatised_data, columns=['Lux',
        ↪ 'Float time value'])
215     # could test noisy labels too, but that is future work
```

```
216     privatised_y_vals = pd.DataFrame(target_values, columns=['Label'])
217     # return the privatised dataset and scaler
218     return privatised_data, privatised_y_vals, scaler
219
220
221 # ------------------------------------------------------- #
222 # ------------------------------------------------------- #
223
224
225 # function get the training data from the not privatised dataset
226 def non_privatised_training_dataset(df, name):
227     # define the scaler
228     scaler = StandardScaler()
229     # get X and y values from the dataset
230     X = df.loc[:, ['Lux', 'Float time value']]
231     y = df['Label']
232     # normalise the training data and convert it back to a pd dataframe
233     X_train = scaler.fit_transform(X)
234     X_train = pd.DataFrame(X_train, columns=['Lux', 'Float time
        ↪ value'])
235     # save the scaler
236     import pickle as pkl
237     with open(f"detection_system_files/scalers/{name}scaler.pkl", "wb")
        ↪ as outfile:
238         pkl.dump(scaler, outfile)
239
240     # return training X,y values
241     return X_train, y
242
243
244 # ------------------------------------------------------- #
245 # ------------------------------------------------------- #
246
247
248 # function to prepare the received data for predictions
249 def prepare_received_data(msg, name):
250     with open(f"detection_system_files/scalers/{name}scaler.pkl", "rb")
        ↪ as infile:
251         scaler = pkl.load(infile)
252     decoded_received_data = [float(x) for x in msg.split(",")]
253
254     # Print the received message
255     print(f"\n{colours.BOLD}{colours.BLUE}{colours.ENDC} Data received:
        ↪ {colours.BOLD}{colours.BLUE}{decoded_received_data[0]},{decoded_received_data[1]}{c
        ↪ end = '')
256
257     # convert the data to a dataframe
258     decoded_received_data = pd.DataFrame([decoded_received_data],
        ↪ columns=['Lux', 'Float time value'])
259
260     # scale it
```

```
261     decoded_received_data = scaler.transform(decoded_received_data)
262     # convert to df again
263     data_to_be_predicted = pd.DataFrame(decoded_received_data,
        ↪  columns=['Lux', 'Float time value'])
264     # return pd dataframe containing the data to be predicted
265     return data_to_be_predicted
266 
267 
268 # ------------------------------------------------------- #
269 # ------------------------------------------------------- #
270 
271 # helper function to filter out
272 # outlier values after making a prediction
273 # and return the decision/s
274 def filter_outlier_values(prediction, data_to_be_predicted):
275     # filter outlier indexes
276     outlier_index = where(prediction == -1)
277     # filter outlier values
278     outlier_values = data_to_be_predicted.iloc[outlier_index]
279     # determine if the received data is anomalous
280 
281     if not outlier_values.size == 0:
282         decision = "anomalous"
283         # print that the received data is anomalous
284         # decision: anomalous
285         print(f"\n{colours.BOLD}{colours.RED}{colours.ENDC}"
286               f" Decision: {colours.BOLD}{colours.RED}"
287               f"anomalous{colours.ENDC}\n")
288     else:
289         decision = "valid"
290         # print that the received data is valid
291         # decision: valid
292         print(f"\n{colours.BOLD}{colours.GREEN}{colours.ENDC}"
293               f" Decision: {colours.BOLD}{colours.GREEN}"
294               f"valid{colours.ENDC}\n")
295     return decision
296 
297 
298 # ------------------------------------------------------- #
299 # ------------------------------------------------------- #
300 
301 
302 # multithreaded function where the anomaly detection is
303 # done using the chosen SVM type and epsilon values.
304 # The dataset is chosen depending on a static value, but to simulate
305 # a real world scenario, it can be chosen based on real system time
306 # (see choose_training_dataset() function for instructions)
307 def anomaly_detection(conn, addr, parameters):
308     # get model training datasets
309     df1, df2 = choose_training_dataset()[0],
        ↪  choose_training_dataset()[1]
```

```
310
311     # 1. dataset specification
312     # if the user choose to use a not privatised dataset
313     if parameters[0] == 1 or parameters[0] == 3:
314         # retrieve x,y train values and scaler for the first classifier
315         returned_data_info = non_privatised_training_dataset(df1,
            ↪ "df1")
316         X_train_1 = returned_data_info[0]
317         y_1 = returned_data_info[1]
318
319         import pickle as pkl
320         with open(f"detection_system_files/scalers/df1scaler.pkl",
            ↪ "rb") as infile:
321             scaler_1 = pkl.load(infile)
322
323         # retrieve x,y train values and scaler for the second
            ↪ classifier
324         if not df2.empty:
325             returned_data_info = non_privatised_training_dataset(df2,
                ↪ "df2")
326             X_train_2 = returned_data_info[0]
327             y_2 = returned_data_info[1]
328             with open(f"detection_system_files/scalers/df2scaler.pkl",
                ↪ "rb") as infile:
329                 scaler_2 = pkl.load(infile)
330
331     # if the user chose to use privatised dataset
332     if parameters[0] == 2 or parameters[0] == 4:
333         # get the privatised X_train and y
334         # values and scaler for the 1st classifier
335         dataset_info = privatised_training_dataset(parameters[1])
336         X_train_1 = dataset_info[0]
337         y_1 = dataset_info[1]
338         scaler_1 = dataset_info[2]
339         if not df2.empty:
340             # get the privatised X_train and y values and scaler
341             # or the 2nd classifier
342             dataset_info = privatised_training_dataset(parameters[1])
343             X_train_2 = dataset_info[0]
344             y_2 = dataset_info[1]
345             scaler_2 = dataset_info[2]
346
347     # 2. model specification
348
349     # if the user chose to use
350     # non-privacy preserving SVM
351     if parameters[0] == 1 or parameters[0] == 2:
352         # linear svm
353         model_1 = svm.SVC(kernel='linear', C=100.0)
354         # fit the training data to the 1st classifier
355         model_1.fit(X_train_1, np.ravel(y_1))
```

```
356             # fit the training data to the 2nd classifier
357             if not df2.empty:
358                 # linear svm
359                 model_2 = svm.SVC(kernel='linear', C=100.0)
360                 # fit the training data for the 2nd classifier
361                 model_2.fit(X_train_2, np.ravel(y_2))
362
363         # if the user chose to use the privacy-preserving SVM
364         if parameters[0] == 3 or parameters[0] == 4:
365             model_1 = obj_perturb_SVM.SVM(privatised=True,
                ↪  lambda_value=SVM_LAMBDA, h_val=SVM_H)
366             # just add the y values to get all the data
367             X_train_1["Label"] = y_1
368             model_1.model_fit(epsilon_p=parameters[1], data=X_train_1)
369             if not df2.empty:
370                 model_2 = obj_perturb_SVM.SVM(privatised=True,
                    ↪  lambda_value=SVM_LAMBDA, h_val=SVM_H)
371                 # just add the y values to get all the data
372                 X_train_2["Label"] = y_2
373                 model_2.model_fit(epsilon_p=parameters[1], data=X_train_2)
374
375         # 3. receive a message, make a prediction
376         # send back results to the server
377         # 3.1. receive the message
378         msg = conn.recv(SIZE).decode(FORMAT)
379         if not (len(msg) <= 0):
380             # 3.2. make predictions
381             # prepare data for the 1st prediction
382             data_to_be_predicted_1 = prepare_received_data(msg, "df1")
383             # 3.2 make the prediction
384             prediction_1 = model_1.make_prediction(data_to_be_predicted_1)
385             # get the decision
386             decision = filter_outlier_values(prediction_1,
                ↪  data_to_be_predicted_1)
387             # if not anomalous, make the second prediction
388             if decision != "anomalous" and not df2.empty:
389                 # prepare data for the 2nd prediction
390                 data_to_be_predicted_2 = prepare_received_data(msg, "df2")
391                 # make the prediction
392                 prediction_2 =
                    ↪  model_2.make_prediction(data_to_be_predicted_2)
393                 # get the decision
394                 decision = filter_outlier_values(prediction_2,
                    ↪  data_to_be_predicted_2)
395
396             # 3.3 send back the prediction to the router (server)
397             conn.send(decision.encode(FORMAT))
398         else:
399             return
400
401
```

```
402 # ------------------------------------------------------ #
403 # ------------------------------------------------------ #
404
405 # Program Entry Point
406 def main():
407     # make the user choose the type of SVM
408     # they want to use as well as the epsilon privacy value
409     parameters = get_svm_parameters()
410
411     # set up the connection between the server and the cloud SVM
412     # using sockets
413     ip_addr = socket.gethostbyname('localhost')
414     address = (ip_addr, SERVER_PORT)
415     svm_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
416     svm_server.bind(address)
417     svm_server.listen()
418
419     # Print information: Listening on {IP}:{PORT}
420     # ----------------------------------------
421     print(f"{colours.BOLD}{colours.BLUE}{colours.ENDC}"
422           f" Cloud SVM service is listening on {colours.BOLD}"
423           f"{colours.GREEN}{ip_addr}:{SERVER_PORT}"
424           f"{colours.YELLOW} {colours.ENDC}")
425     print(f"{colours.BLUE}--------------{colours.CYAN}"
426           f"--------------{colours.GREEN}--------------"
427           f"{colours.YELLOW}--------------{colours.ENDC}")
428
429     # Main program loop
430     while True:
431         # Wait for the server to connect and accept the connection
432         conn, addr = svm_server.accept()
433
434         # Do anomaly detection
435         anomaly_detection(conn, addr, parameters)
436
437         # Create a separate thread to handle separate data
438         # when doing anomaly detection. This is done so that the
439         # SVM could support multiple nodes and predict their data at
440         # the same time
441         thread = threading.Thread(target=anomaly_detection, args=(conn,
            ↪  addr, parameters))
442         thread.start()
443
444
445 if __name__ == "__main__":
446     main()
```

Appendix E

Dataset Validation Script

```
import pandas as pd
from datetime import datetime

# ANSI escape codes to print coloured text
class Colours:
    ENDC = '\033[0m'
    BLUE = '\033[94m'


# function to locate empty and NaN columns in the dataset
def check_for_empty_cells(data_frame):
    missing_cols, missing_rows = (
        (data_frame.isnull().sum(x) | data_frame.eq('').sum(x))
                .loc[lambda x: x.gt(0)].index
            for x in (0, 1))

    if not data_frame.loc[missing_rows, missing_cols].empty:
        # print the columns with the located empty cells
        print("Columns with empty or NaN values found:")
        print(data_frame.loc[missing_rows, missing_cols])
        return 1
    else:
        return 0


# function to validate lux values
def validate_lux(val):
    try:
        # check if it is a numerical value
        float(val['Lux'])
        # check if it is >=0 and <=200 000
        if not 0 <= float(val['Lux']) <= 200000:
                invalid = True
                print("\nInvalid Lux value found:")
                print(val)
    except:
        print("\nNon-numeric Lux value found:")
        print(val)


```

```
41  # function to check if the
42  # date values are in a correct format
43  def validate_date(val):
44      # check the data type
45      try:
46          datetime.strptime(val['Date'], '%m/%d/%Y')
47      except:
48          print("Invalid date found:")
49          print(val)
50
51
52  """
53  Program entry point. First read the data,
54  then check for empty cells, if not found,
55  check if numeric fields are actually numeric,
56  after that, check their range. Finally,
57  validate the date fields.
58  """
59  def main():
60      print(f"{Colours.BLUE}Running this script will provide information
        ↪  about any invalid data present in the dataset.")
61      print(f"If this is the only message displayed, then all the data in
        ↪  the dataset is valid.{Colours.ENDC}")
62      # open the file which contains the dataset for reading
63      df = pd.read_csv(
64          "./datasets/all_data/all_collected_valid_data.csv",
65              sep=",",
66              header=0)
67      # convert values to strings
68      df['Hour'] = df['Hour'].astype(str)
69      df['Minute'] = df['Minute'].astype(str)
70      df['Lux'] = df['Lux'].astype(str)
71
72      # if no empty/NaN cells exist,
73      # check data format and values
74      if not (check_for_empty_cells(df)):
75
76          # check if all hour fields are numeric (int)
77          if not (df[~df['Hour'].map(lambda x: x.isnumeric())]).empty:
78
79              print("Non-numeric hour values found:")
80              print(df[~df['Hour'].map(lambda x: x.isnumeric())])
81
82          # in they are, check if the data is between 0 and 23
83          else:
84              # convert hours to integers
85              df['Hour'] = df['Hour'].astype(int)
86              filtered_invalid_hour = df[~df['Hour'].between(0, 23)]
87              if not filtered_invalid_hour.empty:
88
89                  print("Invalid hour values found:")
```

```
                print(filtered_invalid_hour)

        # check if all minute fields are numeric (int)
        if not (df[~df['Minute'].map(lambda x: x.isnumeric())]).empty:

            print("Non-numeric minute values found:")
            print(df[~df['Hour'].map(lambda x: x.isnumeric())])

        # if they are, check if the data is between 0 and 59 minutes
        else:
            df['Minute'] = df['Minute'].astype(int)
            filtered_invalid_mins = df[~df['Minute'].between(0, 59)]
            if not filtered_invalid_mins.empty:
                print("\nInvalid minute values found:")
                print(filtered_invalid_mins)

        # Check if the date field is valid
        df.apply(lambda x: validate_date(x), axis=1)

        # check if all lux fields are numeric (int/float)
        # and between 0 and 200 000 lux
        df.apply(lambda x: validate_lux(x), axis=1)

"""
Program entry point
"""
if __name__ == "__main__":
    main()
```

Appendix F

Dataset Balancing Script

```python
import pandas as pd
from imblearn.over_sampling import RandomOverSampler

# load the chosen dataset
dataset_name = "all_data_including_anomalous"
df = pd.read_csv(f'../datasets/all_data/{dataset_name}.csv', header=0, sep=',')
# get x values and labels
X = df.drop(['Label'], axis=1)
y = df['Label']

# balance out the classes
# change to RandomOversampler to do oversampling instead
ros = RandomOverSampler(sampling_strategy="auto")
X_res, y_res = ros.fit_resample(X, y)

# save the dataset to a file
X_res.columns = [
    'Date',
    'Hour',
    'Minute',
    'Lux',
    'Float time value']
dataset = X_res.assign(Label=y_res.reset_index(drop=True))
dataset.to_csv(f"balanced-{dataset_name}.csv", index=False)
```

Appendix G

SVM Parameter Tuning Results

Dataset /C value	0.001	0.01	0.1	1.0	10.0	100.0	1000.0
night.csv	51.2%	70.61%	71.51%	85.13%	88.43%	92.2%	89.99%
morning_0.csv	56.55%	86.89%	87.8%	92.12%	93.87%	93.68%	92.12%
morning_1.csv	51.96%	85.19%	90.15%	94.04%	96.54%	96.67%	96.07%
afternoon_0.csv	51.42%	74.36%	91.91%	96.58%	98.72%	98.94%	98.22%
afternoon_1.csv	76.51%	83.87%	96.81%	97.65%	96.5%	96.12%	96.54%

TABLE G.1: Simple SVM C Parameter Tuning

λ (h) /λ (v)	500	50	5	0.5	0.05	0.005	0.0005
0.002	80.02%	86.65%	83.65%	87.98%	83.73%	76.7%	85.9%
0.02	90.65%	93.43%	92.65%	90.96%	79.98%	76.8%	81.6%
0.2	89.94%	94.11%	94.54%	93.54%	80.76%	79.9%	88.87%
2	82.65%	93.76%	93.78%	93.55%	92.76%	89.67%	92.9%
20	81.54%	90.65%	93.01%	92.67%	90.0%	89.3%	80.0%
200	89.93%	93.09%	89.67%	90.09%	88.67%	88.8%	88.5%
2000	84.76%	89.95%	91.87%	89.98%	80.45%	87.5%	80.6%

TABLE G.2: Differentially Private SVM Parameter Tuning

Appendix H

abstract_dataset_privatiser.py and Laplace_dataset_privatiser.py

abstract_dataset_privatiser Script

```
class ABCPrivacyPreserver():
    # privatise the provided raw data using different
    # calculated sensitivity values for each column (if there are
    ↪ several)
    def privatise_data(self, raw_values, sensitivity_value=0.01,
    ↪ sensitivity_values_list=None):
        if type(raw_values) == list:
            # return an empty list if provided list is empty
            if len(raw_values) == 0:
                return []
            privatised_data = []
            # if there is no sensitivity values list provided,
            # get one for the given raw data
            if type(raw_values[0]) == list and sensitivity_values_list
            ↪ is None:
                sensitivity_values_list =
                ↪ self.getSensitivityList(raw_values)
            # initialise the counter
            # used to index into the sensitivity_value list when
            # the sensitivity value for a single value is not provided
            ↪ as a float
            counter = 0
            # If the current value entry in the list is not a list,
            # set the sensitivity value to either the sensitivity
            # _value if it is a float, or the value in the
            ↪ sensitivity_value
            # list at the current counter index.
            for value_entry in raw_values:
                if type(value_entry) is list:
                    sensitivity_val = sensitivity_values_list
                else:
                    if type(sensitivity_value) is float:
                        sensitivity_val = sensitivity_value
                    else:
                        sensitivity_val = sensitivity_value[counter]
                # append the privatised data to the list
```

```
                privatised_data.append(self.privatise_data(value_entry,
                ↪  sensitivity_val))
                counter += 1
            return privatised_data
        # add noise to a single value if no list is provided
        else:
            return self.privatise_single_value(raw_values,
            ↪  sensitivity_value)

    # get the list of sensitivity values for the provided data
    def get_data_sensitivity_values(self, raw_data_list):
        data_sensitivity_vals_list = []
        for i in range(len(raw_data_list[0])):
            # get all data in the i-th column
            column_data = [row[i] for row in raw_data_list]
            # get the sensitivity value for that column
            single_sensitivity_val =
            ↪  abs(max(column_data)-min(column_data))
            # append to all the sensitivity values
            data_sensitivity_vals_list.append(single_sensitivity_val)
        return data_sensitivity_vals_list
```

Laplace_dataset_privatiser.py script

```
from abc import ABC

import numpy as np
from privacy_preserving_svms.abstract_data_privatiser import
↪  ABCPrivacyPreserver


# class which is responsible for preparing
# dataset for privatisation and adding
# laplace noise to increase the privacy of
# individual entries
class LaplacePrivacyPreserver(ABCPrivacyPreserver, ABC):
    # default values
    mean_value = 0.0
    epsilon_value = 1.0

    def __init__(self, epsilon_val=1.0):
        # check if provided epsilon value is valid
        if type(epsilon_val) != float:
          raise ValueError('Epsilon value has to be a float')
        if epsilon_val <= 0.0:
          raise ValueError('Epsilon value has to be >0.0')
        self.epsilon_value = epsilon_val

    def privatise_single_value(self, data, sensitivity_level=1.0):
```

```
25          # convert it to a float
26          try:
27            float_value = float(data)
28          except:
29            raise ValueError('The value to be sanitised has to be float')
30          # define the sensitivity value:
31          # how much of an impact can an individual value
32          # have on the outcome of the queries?
33          sensitivity_level = max(0.001, sensitivity_level)
34          # add noise to the value:
35          # epsilon attribute represents the privacy budget,
36          # which is a measure of how much privacy is being
37          # provided to the data. Together with the sensitivity
38          # it determines the scale of the noise added to the data.
39          # the noise is drawn from the Laplace distribution
40          noise_value = np.random.laplace(self.mean_value,
            ↪  sensitivity_level / self.epsilon_value, 1)[0]
41          return float(float_value + noise_value)
42
43
44  class DataConverter:
45      # convert np array into a list
46      def convert_from_original(self, original_data):
47          if type(original_data) == np.ndarray:
48              converted_data = []
49              # loop through all the data entries
50              for value in original_data:
51
                    ↪  converted_data.append(self.convert_from_original(value))
52              return converted_data
53          else:
54              return self.covert_to_float(original_data)
55
56      # convert the data to a float
57      def covert_to_float(self, data):
58          try:
59              return float(data)
60          except:
61              # inform the user if errors occur
62              raise ValueError('Data could not be converted to float')
```

Appendix I

Objective Perturbation SVM

Implementation of differentially private SVM with objective function perturbation. Based on the paper: "Differentially Private Empirical Risk Minimization" [24].

Code Implementation is based on explanation written in R: [9], and lectures by Gautam Kamath (CS 860 - Algorithms for Private Data Analysis, Differentially Private ERM (Part 1, Part 2, Part 3))[8], and the talk "Differentially Private Empirical Risk Minimization" by Prateek Jain [10].

```
1
2  import numpy as np
3  from scipy import optimize
4
5  """
6  Implementation of differentially private SVM.
7  This class which allows to train and evaluate
8  both: differentially private sum
9  (using objective perturbation) and plain
10 sum with modified loss function (Huber)
11
12 Credits:
13 Chaudhuri, K., Monteleoni, C. and Sarwate, A. (2011)
14 'Differentially Private Empirical Risk Minimization',
15 Journal of Machine Learning Research, 12, pp. 1069-1109.
16 Available at:
   ↪ https://jmlr.org/papers/volume12/chaudhuri11a/chaudhuri11a.pdf.
17
18 Algorithm 2 ERM with objective perturbation,
19
20 Implementation in R:
21 (Explanations)
22 https://search.r-project.org/CRAN/refmans/DPpack/html/sumDP.html
23 """
24
25
26 class SVM:
27     def __init__(self, privatised, lambda_value, h_val):
28         self.private = privatised
29         self.h = h_val
30         self.lambda_value = lambda_value
31         self.c = 1/(self.h*2)
```

```
32
33      """
34      fit the SVM model (train)
35      """
36      def model_fit(self, epsilon_p, data):
37          # dataset labels
38          train_y = data["Label"]
39          # dataset features
40          train_x = data.drop(columns=["Label"])
41
42          train_x = train_x.values
43          train_y = train_y.values
44          # number of features
45          self.n = train_x.shape[0]
46          self.num_of_features = train_x.shape[1]
47          # at first guess the optimisation value
48          initial_f = np.ones(self.num_of_features)
49
50          if self.private:
51              # calculate epsilon p prime
52              c_squared = self.c ** 2
53              n_squared = self.n ** 2
54              lambda_squared = self.lambda_value ** 2
55
56              first_term = 2 * self.c / (self.n * self.lambda_value)
57              second_term = c_squared / (n_squared * lambda_squared)
58
59              epsilon_p_prime = epsilon_p - np.log(1 + first_term +
                ↪  second_term)
60
61              # determine the delta value
62              if epsilon_p_prime > 0:
63                  delta_val = 0
64
65              else:
66                  exp_term = np.exp(epsilon_p / 4) - 1
67                  numerator = self.c / (self.n * exp_term)
68                  delta_val = numerator - self.lambda_value
69                  epsilon_p_prime = epsilon_p / 2
70
71              # create noise according to epsilon_p_prime
72              scale_factor = 2 / epsilon_p_prime
73              size_parameter = self.num_of_features
74              noise = np.random.exponential(scale_factor, size_parameter)
75
76              # use BFGS method to optimise the function
77              initial_guess = initial_f
78              additional_args = (train_x, train_y, noise)
79              display = False
80              solution = optimize.fmin_bfgs(self.func, initial_guess,
                ↪  args=additional_args, disp=display)
```

```
81              f = solution
82
83              norm_f_squared = np.linalg.norm(f) ** 2
84              self.f = f + (delta_val * norm_f_squared) / 2
85
86          else:
87              # if the training is not privatised
88              # we should pass a vector with all 0s
89              noise = np.zeros(self.num_of_features)
90              # use the BFGS method to optimise the function
91              solution = optimize.fmin_bfgs(self.func, initial_f,
                ↪  args=(train_x, train_y, noise), disp=False)
92              self.f = solution
93
94      """
95          function to find
96          the empirical loss
97      """
98          def func(self, f, train_x, train_y, noise):
99              # find the location of the training data to prediction function
100             z = np.dot(f.T, train_x.T) * train_y
101
102             # Specify the conditions
103             conditions_list = [
104                 z > 1 + self.h,
105                 np.abs(1 - z) <= self.h,
106                 z < 1 - self.h
107             ]
108
109             # Specify the corresponding choices
110             choices_list = [
111                 0,
112                 (1 / (4 * self.h)) * (1 + self.h - z),
113                 1 - z
114             ]
115
116             # Select the loss based on the conditions and choices
117             loss_value = np.select(conditions_list, choices_list)
118
119             # return regularized emperical loss, noise is added in this
                ↪  step
120             return np.mean(loss_value + (self.lambda_value / 2 *
                ↪  (np.linalg.norm(f) ** 2)) + ((1 / self.n) *
                ↪  np.dot(np.transpose(noise), f)))
121
122         #  make a prediction
123         def make_prediction(self, data_sample):
124             # flatten the data if it is a numpy array
125             flattened_data = data_sample.values.flatten()\
126                 if not isinstance(data_sample, np.ndarray) else
                    ↪  data_sample.flatten()
```

```
127         # make the prediction
128         prediction = np.sign(np.dot(self.f.T, flattened_data))
129         return prediction
130 
131 
132     """
133     Prediction function.
134     It calculates the location of the testing data relative to the
    ↪  predictor vector f
135     and return the error rate
136     """
137     def evaluate(self, test_data):
138         # test data labels
139         df_test_y = test_data["Label"]
140         # test data features
141         df_test_x = test_data.drop(columns="Label")
142 
143         test_x = df_test_x.values
144         test_y = df_test_y.values
145 
146         error_number = 0
147         for idx, datapoint in enumerate(test_x):
148             prediction = np.sign(np.dot(np.transpose(self.f),
                ↪  datapoint))
149             if prediction != test_y[idx]:
150                 error_number += 1
151 
152         return error_number/test_x.shape[0]
153 
```

Appendix J

Membership Inference Test Script

```
from __future__ import absolute_import, division, print_function,
    unicode_literals
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from privacy_preserving_svms import objective_function_perturbation_SVM
    as obj_perturb_svm
from art.estimators.classification import BlackBoxClassifier
import numpy as np
from art.attacks.inference.membership_inference import
    MembershipInferenceBlackBoxRuleBased

# import the dataset (with the headers)
df = pd.read_csv('./datasets/training/balanced/morning_0.csv',
    header=0, sep=',')
#df =
    pd.read_csv('./datasets/training/balanced/membership_inference_original.csv',
    header=0, sep=',')

# define the scaler
scaler = StandardScaler()

# get X and y values from the dataset
X = df.loc[:, ['Lux','Float time value']]
y = df['Label']
# convert to numpy array
X = X.values
y = y.values
y = np.where(y == 0, -1, y)
# split the data into train and test datasets
# 70% for training, 30% for predictions
x_train, x_test, y_train, y_test = train_test_split(X, y,
    test_size=0.50,random_state=1)

# define non private model
model2 = SVC(C=100)
x_train = scaler.fit_transform(x_train)
```

```
model2.fit(x_train, y_train)

x_test = scaler.transform(x_test)
results = model2.predict(x_test)
# define private model
model_1 = obj_perturb_svm.SVM(privatised=True, lambda_value=0.2,
 ↪  h_val=1)

# fit private model
model_1.fit_membership_inference(x_train, y_train, epsilon_p=0.3)

# encoding
def manual_to_categorical(labels, nb_classes):
    result = np.zeros((len(labels), nb_classes))
    for i, label in enumerate(labels):
        if label == -1:
            result[i, 0] = 1
        else:  # assuming label is 1
            result[i, 1] = 1
    return result

# private model prediction helper function
def predict(x):
    x = np.array(x)
    predictions = model_1.membership_inference_predict(x)
    predictions = list(map(int, predictions))
    return manual_to_categorical(predictions, nb_classes=2)

# fit the custom (differentially private)
# classifier into the ART library black box model
classifier = BlackBoxClassifier(predict, x_test.shape,2)
# define the attack
attack = MembershipInferenceBlackBoxRuleBased(classifier)

# infer attacked feature
inferred_train = attack.infer(x_train, y_train)
inferred_test = attack.infer(x_test, y_test)

# check accuracy
train_acc = np.sum(inferred_train) / len(inferred_train)
test_acc = 1 - (np.sum(inferred_test) / len(inferred_test))
acc = (train_acc * len(inferred_train) + test_acc * len(inferred_test))
 ↪  / (len(inferred_train) + len(inferred_test))
print(f"Members Accuracy: {train_acc:.4f}")
print(f"Non Members Accuracy {test_acc:.4f}")
print(f"Attack Accuracy {acc:.4f}")


test_data = pd.DataFrame({
    'Lux': x_test[:, 0],
    'Time float value': x_test[:, 1],
```

```
    'Label': y_test
})

```

Bibliography

[1] URL: https://www.precedenceresearch.com/precision-farming-market (visited on 05/17/2023).

[2] URL: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote10.html (visited on 05/27/2023).

[3] en-US. URL: https://machinelearning.apple.com/research/learning-with-privacy-at-scale.

[4] en. URL: https://www.linkedin.com/pulse/injecting-noise-input-mahitha-meka (visited on 05/11/2023).

[5] URL: https://peps.python.org/pep-0008/.

[6] URL: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

[7] URL: https://circuitjournal.com/arduino-serial-to-spreadsheet.

[8] URL: http://www.gautamkamath.com/CS860-fa2020.html (visited on 05/17/2023).

[9] URL: https://search.r-project.org/CRAN/refmans/DPpack/html/svmDP.html (visited on 05/17/2023).

[10] 2013. URL: https://www.youtube.com/watch?v=PgNpNcR6afY (visited on 05/17/2023).

[11] en. Page Version ID: 1156332283. 2023. URL: https://en.wikipedia.org/w/index.php?title=Support_vector_machine&oldid=1156332283.

[12] Python. 2023. URL: https://github.com/IBM/differential-privacy-library (visited on 05/17/2023).

[13] en. Page Version ID: 1154861468. 2023. URL: https://en.wikipedia.org/w/index.php?title=ANSI_escape_code&oldid=1154861468.

[14] Khalid M. Al-Gethami, Mousa T. Al-Akhras, and Mohammed Alawairdhi. "Empirical Evaluation of Noise Influence on Supervised Machine Learning Algorithms Using Intrusion Detection Datasets". en. In: *Security and Communication Networks* 2021 (2021), e8836057. ISSN: 1939-0114. DOI: 10.1155/2021/8836057. URL: https://www.hindawi.com/journals/scn/2021/8836057/.

[15] Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri. *Foundations of Security Analysis and Design V: FOSAD 2008/2009 Tutorial Lectures*. en. Google-Books-ID: MU1tCQAAQBAJ. Springer, 2009. ISBN: 9783642038297.

[16] Abdelaziz Amara korba and Nour Karabadji. "Smart Grid Energy Fraud Detection Using SVM". In: June 2019, pp. 1–6. DOI: 10.1109/ICNAS.2019.8807832.

[17] Abhinav Atla et al. "Sensitivity of Different Machine Learning Algorithms to Noise". In: *J. Comput. Sci. Coll.* 26.5 (2011), 96–103. ISSN: 1937-4771.

[18] Kenneth Bannister, Gianni Giorgetti, and Kornepati Sandeep. "Wireless sensor networking for hot applications: Effects of temperature on signal strength, data collection and localization". In: (Jan. 2008).

[19] Malti Bansal, Apoorva Goyal, and Apoorva Choudhary. "A comparative analysis of K-Nearest Neighbor, Genetic, Support Vector Machine, Decision Tree, and Long Short Term Memory algorithms in machine learning". en. In: *Decision Analytics Journal* 3 (2022), p. 100071. ISSN: 2772-6622. DOI: 10.1016/j.dajour.2022.100071. URL: https://www.sciencedirect.com/science/article/pii/S2772662222000261.

[20] Rukshan Batuwita and Vasile Palade. "CHAPTER 6 CLASS IMBALANCE LEARNING METHODS FOR SUPPORT VECTOR MACHINES". In: 2012.

[21] Carlo Alberto Boano et al. "The Impact of Temperature on Outdoor Industrial Sensornet Applications". In: *IEEE Transactions on Industrial Informatics* 6.3 (2010), pp. 451–459. DOI: 10.1109/TII.2009.2035111.

[22] US Census Bureau. *2020 Decennial Census: Processing the Count: Disclosure Avoidance Modernization*. URL: https://www.census.gov/programs-surveys/decennial-census/decade/2020/planning-management/process/disclosure-avoidance.html.

[23] Olivier Chapelle. "Training a Support Vector Machine in the Primal". In: *Neural Computation* 19.5 (2007), pp. 1155–1178. DOI: 10.1162/neco.2007.19.5.1155.

[24] Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarwate. "Differentially Private Empirical Risk Minimization". In: (2011). arXiv:0912.0071 [cs]. DOI: 10.48550/arXiv.0912.0071. URL: http://arxiv.org/abs/0912.0071.

[25] Junjie Chen, Wendy Hui Wang, and Xinghua Shi. "Differential Privacy Protection Against Membership Inference Attack on Machine Learning for Genomic Data". eng. In: *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing* 26 (2021), 26–37. ISSN: 2335-6936.

[26] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". en. In: *Machine Learning* 20.3 (1995), 273–297. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/BF00994018. URL: http://link.springer.com/10.1007/BF00994018.

[27] Ania Cravero et al. "Data Type and Data Sources for Agricultural Big Data and Machine Learning". en. In: *Sustainability* 14.23 (2022), p. 16131. ISSN: 2071-1050. DOI: 10.3390/su142316131. URL: https://www.mdpi.com/2071-1050/14/23/16131.

[28] Bolin Ding, Janardhan (Jana) Kulkarni, and Sergey Yekhanin. "Collecting Telemetry Data Privately". en-US. In: 2017. URL: https://www.microsoft.com/en-us/research/publication/collecting-telemetry-data-privately/.

[29] J. Ding et al. "Private Empirical Risk Minimization With Analytic Gaussian Mechanism for Healthcare System". In: *IEEE Transactions on Big Data* 8.04 (2022), pp. 1107–1117. ISSN: 2332-7790. DOI: 10.1109/TBDATA.2020.2997732.

[30] Jiahao Ding et al. "Differentially Private ADMM for Distributed Medical Machine Learning". In: (2020). arXiv:1901.02094 [cs, stat]. URL: http://arxiv.org/abs/1901.02094.

[31] Cynthia Dwork. "Differential Privacy". In: *Automata, Languages and Programming*. Springer, 2006, pp. 1–12. URL: http://audentia-gestion.fr/MICROSOFT/dwork.pdf.

[32] Cynthia Dwork and Aaron Roth. *The Algorithmic Foundations of Differential Privacy*. Now Publishers Inc., 2014. URL: https://www.cis.upenn.edu/~aaroth/Papers/privacybook.pdf.

[33] Riccardo M. G. Ferrari, Kwassi H. Degue, and Jerome Le Ny. "Differentially Private Anomaly Detection for Interconnected Systems". en. In: *Safety, Security and Privacy for Cyber-Physical Systems*. Ed. by Riccardo M.G. Ferrari and André M. H. Teixeira. Lecture Notes in Control and Information Sciences. Cham: Springer International Publishing, 2021, 203–230. ISBN: 9783030650483. DOI: 10.1007/978-3-030-65048-3_10. URL: https://doi.org/10.1007/978-3-030-65048-3_10.

[34] Carolina Fortuna and Blaž Fortuna. "ANOMALY DETECTION IN COMPUTER NETWORKS USING LINEAR SVMs". In: (Jan. 2007).

[35] Zeinab Ghodsi, Mir Masoud Kheirkhah Zarkesh, and Bagher Ghermezcheshmeh. "Comparison of Accuracy Between Support Vector Machine and Random Forest Classifiers for Land Use and Crop Mapping Using Multi-Temporal Sentinel-2 Images". In: *Iranian Journal of Remote Sensing & GIS* 12.4 (2021), pp. 73–92. ISSN: 2008-5966. DOI: 10.52547/gisj.12.4.73. eprint: https://gisj.sbu.ac.ir/article_100688_d29775c3f7a4b801054d094e4bac62c8.pdf. URL: https://gisj.sbu.ac.ir/article_100688.html.

[36] Afshin Gholamy, Vladik Kreinovich, and Olga Kosheleva. "Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation". In: *Departmental Technical Reports (CS)* (2018). URL: https://scholarworks.utep.edu/cs_techrep/1209.

[37] Seira Hidano, Takao Murakami, and Yusuke Kawamoto. "TransMIA: Membership Inference Attacks Using Transfer Shadow Training". In: *2021 International Joint Conference on Neural Networks (IJCNN)*. arXiv:2011.14661 [cs]. 2021, 1–10. DOI: 10.1109/IJCNN52387.2021.9534207. URL: http://arxiv.org/abs/2011.14661.

[38] Shotaro Ishii and David Ljunggren. "A Comparative Analysis of Robustness to Noise in Machine Learning Classifiers". PhD thesis. 2021. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-302532.

[39] Raouf Kerkouche et al. "Privacy-Preserving and Bandwidth-Efficient Federated Learning: An Application to In-Hospital Mortality Prediction". In: *CHIL 2021 - ACM Conference on Health, Inference, and Learning*. virtual event, France: ACM, Apr. 2021, pp. 1–11. DOI: 10.1145/3450439.3451859. URL: https://inria.hal.science/hal-03160473.

[40] Randhir Kumar et al. "SP2F: A secured privacy-preserving framework for smart agricultural Unmanned Aerial Vehicles". en. In: *Computer Networks* 187 (2021), p. 107819. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2021.107819. URL: https://www.sciencedirect.com/science/article/pii/S1389128621000086.

[41] Franz Kuntke et al. "LoRaWAN security issues and mitigation options by the example of agricultural IoT scenarios". en. In: *Transactions on Emerging Telecommunications Technologies* 33.5 (2022). ISSN: 2161-3915, 2161-3915. DOI: 10.1002/ett.4452. URL: https://onlinelibrary.wiley.com/doi/10.1002/ett.4452.

[42] Tian Li et al. "Federated Learning: Challenges, Methods, and Future Directions". In: *IEEE Signal Processing Magazine* 37.3 (2020), 50–60. ISSN: 1558-0792. DOI: 10.1109/MSP.2020.2975749.

[43] R. Bhavani M. Reddaiah. "Comparison of Decision Tree Algorithm and Support Vector Machine for Efficient Soil Data Classification in Terms of Accuracy". en. In: *Journal of Survey in Fisheries Sciences* 10.1S (2023), 1964–1972. ISSN: 2368-7487. DOI: 10.17762/sfs.v10i1S.430. URL: https://sifisheriessciences.com/journal/index.php/journal/article/view/430.

[44] H. Brendan McMahan et al. "Learning Differentially Private Language Models Without Losing Accuracy". In: *CoRR* abs/1710.06963 (2017). arXiv: 1710.06963. URL: http://arxiv.org/abs/1710.06963.

[45] Sebastian Meiser. "Approximate and Probabilistic Differential Privacy Definitions". In: *IACR Cryptol. ePrint Arch.* 2018 (2018), p. 277.

[46] Fereshteh Modaresi and Shahab Araghinejad. "A Comparative Assessment of Support Vector Machines, Probabilistic Neural Networks, and K-Nearest Neighbor Algorithms for Water Quality Classification". In: *Water Resources Management* 28 (Sept. 2014), pp. 4095–4111. DOI: 10.1007/s11269-014-0730-z.

[47] Hardt Moritz. *DS 102 Data, Inference, and Decisions*. 2019. URL: https://data102.org/fa19/assets/notes/notes24.pdf.

[48] Maria-Irina Nicolae et al. "Adversarial Robustness Toolbox v0.2.2". In: *CoRR* abs/1807.01069 (2018). arXiv: 1807.01069. URL: http://arxiv.org/abs/1807.01069.

[49] Tamoghna Ojha, Sudip Misra, and Narendra Singh Raghuwanshi. "Wireless sensor networks for agriculture: The state-of-the-art in practice and future challenges". In: *Computers and Electronics in Agriculture* 118 (2015), pp. 66–84. ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2015.08.011. URL: https://www.sciencedirect.com/science/article/pii/S0168169915002379.

[50] Dilber Uzun Ozsahin et al. "Impact of feature scaling on machine learning models for the diagnosis of diabetes". In: *2022 International Conference on Artificial Intelligence in Everything (AIE)*. 2022, pp. 87–94. DOI: 10.1109/AIE57029.2022.00024.

[51] Md.Atiqur Rahman et al. "Membership Inference Attack against Differentially Private Deep Learning Model". In: *Trans. Data Priv.* 11 (2018), pp. 61–79.

[52] Norrathep Rattanavipanon et al. "Detecting Anomalous LAN Activities under Differential Privacy". In: *Security and Communication Networks* 2022 (2022). Ed. by George Drosatos, pp. 1–15. DOI: 10.1155/2022/1403200. URL: https://doi.org/10.1155%2F2022%2F1403200.

[53] Benjamin I. P. Rubinstein et al. "Learning in a Large Function Space: Privacy-Preserving Mechanisms for SVM Learning". In: (2009). arXiv:0911.5708 [cs]. URL: http://arxiv.org/abs/0911.5708.

[54] Osman Salem et al. "Anomaly Detection in Medical Wireless Sensor Networks using SVM and Linear Regression Models:" ng. In: *International Journal of E-Health and Medical Communications* 5.1 (2014), 20–45. ISSN: 1947-315X, 1947-3168. DOI: 10.4018/ijehmc.2014010102. URL: https://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/ijehmc.2014010102.

[55] Prof. Rahul Satpute et al. "REVIEW PAPER ON WIRELESS SENSOR NETWORKS FOR AGRICULTURE". In: *International Journal Of Creative Research Thoughts (IJCRT)* (). ISSN: 2320-2882. URL: https://ijcrt.org/papers/IJCRTJ020007.pdf.

[56] Makhamisa Senekane. "Differentially Private Image Classification Using Support Vector Machine and Differential Privacy". In: *Machine Learning and Knowledge Extraction* 1 (Feb. 2019), pp. 483–491. DOI: 10.3390/make1010029.

[57] Reza Shokri et al. "Membership Inference Attacks against Machine Learning Models". In: (2017). arXiv:1610.05820 [cs, stat]. URL: http://arxiv.org/abs/1610.05820.

[58] Reza Shokri et al. "Membership Inference Attacks against Machine Learning Models". In: (2017). arXiv:1610.05820 [cs, stat]. URL: http://arxiv.org/abs/1610.05820.

[59] Saurabh Shukla, Subhasis Thakur, and John G. Breslin. "Anomaly Detection in Smart Grid Network Using FC-Based Blockchain Model and Linear SVM". In: *Machine Learning, Optimization, and Data Science: 7th International Conference, LOD 2021, Grasmere, UK, October 4–8, 2021, Revised Selected Papers, Part I*. Berlin, Heidelberg: Springer-Verlag, 2021, 157–171. ISBN: 9783030954666. DOI: 10.1007/978-3-030-95467-3_13. URL: https://doi.org/10.1007/978-3-030-95467-3_13.

[60] Sina Sontowski et al. "Cyber Attacks on Smart Farming Infrastructure". In: Dec. 2020. DOI: 10.1109/CIC50333.2020.00025.

[61] Dave Sotelo. *Effect of Feature Standardization on Linear Support Vector Machines*. en. 2017. URL: https://towardsdatascience.com/effect-of-feature-standardization-on-linear-support-vector-machines-13213765b812.

[62] J. Thelen, D. Goense, and Koen Langendoen. "Radio wave propagation in potato fields". In: (Jan. 2005).

[63] Li Ting et al. "A secure framework for IoT-based smart climate agriculture system: Toward blockchain and edge computing". en. In: *Journal of Intelligent Systems* 31.1 (2022), 221–236. ISSN: 2191-026X. DOI: 10.1515/jisys-2022-0012. URL: https://www.degruyter.com/document/doi/10.1515/jisys-2022-0012/html.

[64] K Veropoulos, ICG Campbell, and N Cristianini. "Controlling the Sensitivity of Support Vector Machines". In: *Proceedings of the International Joint Conference on Artificial Intelligence, Stockholm, Sweden (IJCAI99)* (1999), 55–60.

[65] Borislava Vrigazova. "The Proportion for Splitting Data into Training and Test Set for the Bootstrap in Classification Problems". en. In: *Business Systems Research Journal* 12.1 (2021), 228–242. DOI: 10.2478/bsrj-2021-0015. URL: https://sciendo.com/article/10.2478/bsrj-2021-0015.

[66] GuiPing Wang, Shu Chen, and Jun Liu. "Anomaly-based Intrusion Detection using Multiclass-SVM with Parameters Optimized by PSO". In: *International Journal of Security and its Applications* 9 (June 2015), pp. 227–242. DOI: 10.14257/ijsia.2015.9.6.22.

[67] Ning Wang, Naiqian Zhang, and Maohua Wang. "Wireless sensors in agriculture and food industry—Recent development and future perspective". en. In: *Computers and Electronics in Agriculture* 50.1 (2006), 1–14. ISSN: 0168-1699. DOI: 10.1016/j.compag.2005.09.003. URL: https://www.sciencedirect.com/science/article/pii/S0168169905001572.

[68] Jie Wen et al. "A survey on federated learning: challenges and applications". en. In: *International Journal of Machine Learning and Cybernetics* 14.2 (2023), 513–535. ISSN: 1868-808X. DOI: 10.1007/s13042-022-01647-y. URL: https://doi.org/10.1007/s13042-022-01647-y.

[69] Leanne Wiseman et al. "Farmers and their data: An examination of farmers' reluctance to share their data through the lens of the laws impacting smart farming". In: *NJAS - Wageningen Journal of Life Sciences* 90-91 (2019), p. 100301. ISSN: 1573-5214. DOI: https://doi.org/10.1016/j.njas.2019.04.007. URL: https://www.sciencedirect.com/science/article/pii/S1573521418302616.

[70] Sjaak Wolfert et al. "Big Data in Smart Farming – A review". en. In: *Agricultural Systems* 153 (2017), 69–80. ISSN: 0308-521X. DOI: 10.1016/j.agsy.2017.01.023. URL: https://www.sciencedirect.com/science/article/pii/S0308521X16303754.

[71] Wei Wu et al. "A comparison of support vector machines, artificial neural network and classification tree for identifying soil texture classes in southwest China". In: *Computers and Electronics in Agriculture* 144 (2018), 86–93. ISSN: 0168-1699. DOI: 10.1016/j.compag.2017.11.037. URL: http://www.scopus.com/inward/record.url?scp=85036471221&partnerID=8YFLogxK.

[72] Grace Zhang. *What is the kernel trick? Why is it important?* en. 2018. URL: https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d.

[73] Ming Zhang, Boyi Xu, and Jie Gong. "An Anomaly Detection Model Based on One-Class SVM to Detect Network Intrusions". In: *2015 11th International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*. 2015, 102–107. DOI: 10.1109/MSN.2015.40.