UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR MEDIZINISCHE INFORMATIK

From the Institute for Medical Informatics at the University of Lübeck
Head of Institute: Prof. Dr. rer. nat. habil. Heinz Handels

Privacy Concerns in Models about Physical Activity

Bedenken hinsichtlich der Privatsphäre in Modellen der körperlichen Aktivität

Masterthesis
in the course of studies in Medical Informatics
at the University of Lübeck

submitted by
Niclas Kath

issued by
Prof. Dr.-Ing. habil. Marcin Grzegorzek

and supervised by
M. Sc. Muhammad Adeel Nisar

Prof. Dr. ret. nat. Esfandiar Mohammadi

Lübeck, March 9, 2021

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Lübeck, the March 9, 2021

Abstract

Machine learning is achieving better and better results in activity recognition. However, these models implicitly store information about their training data, which can lead to data leakage. In this work, a versatile approach for recognizing activities of daily living based on rank pooling, which is highly vulnerable to membership inference attacks, was made robust against these attacks. In doing so, the support vector regression used was designed to be differentially private. Although the dataset used is difficult to classify, this work achieves very good results. With the methods used, it is possible to reduce the success rate of the attack by 71.44% while the accuracy is reduced by only 12.07%. In the experiments, Objective Perturbation was shown to perform better. Furthermore, support vector machines with differential privacy were investigated, here, it was shown that the previous approaches do not achieve satisfactory results on the data set used.

Kurzfassung

Machine Learning erzielt in der Aktivitätserkennung immer bessere Ergebnisse. Jedoch speichern diese Modelle implizit Informationen über ihre Trainingsdaten, was zu einem Datenleck führen kann. Im Rahmen dieser Arbeit wurde ein vielsprechender Ansatz zur Erkennung von Aktivitäten des täglichen Lebens basierend auf Rankpooling, welcher stark anfällig für Membership Inference Attacks ist, robust gegen diese Angriffe gemacht. Dabei wurde die eingesetzte Support Vector Regression Differentially Private gestaltet. Obwohl der genutzte Datensatz nur schwer klassifizierbar ist, erzielt diese Arbeit sehr gute Ergebnisse. Mit den eingesetzten Methoden ist es möglich die Erfolgsrate des Angriffes um 71.44% zu reduzieren während die die Genauigkeit nur um 12.07% gesenkt wird. In den Experimenten hat sich gezeigt, dass die Objective Perturbation etwas besser funktioniert. Im Weiteren wurden Support Vector Machines mit Differential Privacy untersucht, hier zeigte sich, dass die bisherigen Ansätze keine zufrieden stellenden Ergebnisse auf dem genutzten Datensatz erzielen.

Contents

List of Figures

List of Tables

List of abbreviations

Abbreviation	
ADL	Activity of Daily Living
BFGS	Broyden–Fletcher–Goldfarb–Shanno algorithm
CCCV	CalibratedClassifierCV
CNN	Convolutional Neural network
CogAge	Cognitive Village
DNN	Deep Neural Network
IQR	Interquartile Range
LSTM	Long Short-Term Memory
MBSGD	Minibatch Stochastic Gradient Descent
MIA	Membership Inference Attack
ML	Machine Learning
OvO	One-vs-One
OvR	One-vs-Rest
PPML	Privacy-Preserving Machine Learning
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
SVR	Support Vector Regression

Chapter 1

Introduction

1.1 Problem Statement and Motivation

The age distribution of the world population is increasingly shifting towards elderly people [1]. An increased age leads to new challenges in life. Firstly, elderly people want to stay as independent as possible, which leads to the conflict that a balance between independence and provided aid has to be found. To achieve this, it is necessary to measure the independence of an elderly person. Germany has such a system which expresses the required help of a person to cope the day. It is called level of care. However, the assessment in this system is a time-consuming task. An automatic system which can undertake some parts of this assignment would be a beneficial assistance, not only for elderly people but also for the healthcare people who make this assignment.
Another challenge, which is caused by the aging of the population and the wish of elderly people, is the risk of injury. One of the most crucial problems for elderly people is the risk of falling. Falling in your daily life is not really dangerous for younger people as they still have the reflexes to cushion the fall. However, with increased age, these reflexes become slower and the risk for injury increases. If a fall has happened, it is important that medical assistance comes in a short amount of time. Elderly people who have fallen and injured themselves often can not call for this medical assistance as their injury prohibits them from doing so. A system which can automatically identify a fall and call for assistance enhances the life quality for elderly people.

There are promising research results in activity recognition [2] and the detection of falling [3]. Most of the promising results in those areas are received through machine learning (ML). ML can achieve sufficient results if the used training data has a good quality and quantity. The results of ML heavily depend on the used training data. In general, the more data is available the higher is the chance that the model generalizes well. However, ML methods learn implicitly information from the training data, which they can leak if no countermeasures are implemented. This is especially an issue if the training data contains sensible data. An attacker can extract this learned information out of the ML model. In the scope of medical data this is an serious issue as mostly all data used for training is highly sensible. This leads to several problems. Publishing novel approaches in the area of ML is problematic as these models can leak information. Furthermore, it prohibits the usage of promising ML models in a real world scenario. It does not only prohibit the usage from an ethical standpoint but also from an legal standpoint [4]. Security of data is nowadays required by law which products have to satisfy. Therefore, these models have to be modified in a way that they cannot leak information. Privacy-preserving machine learning (PPML) tries to close this gap [5]. PPML has to protect the training data which not only enables the publishing of novel techniques but also prepares the way for deployment which ultimately can help people in their life [6].

One of the most used ML algorithms is the Support Vector Machine (SVM). It is a classifier which can work really well in many scenarios and often does not require a great amount of training samples. Many frameworks are relying on the libsvm framework [7] to provide an SVM. This implementation is fast and accurate. However, it is vulnerable to malicious attacks. For example, the membership inference attack (MIA) by Salem et al. [8] which has an accuracy up to 84% on the Cognitive Village (CogAge) Dataset. With this accuracy, the libsvm implementation is not suitable for sensitive data as it leaks a great amount of its information. In general, SVMs are very vulnerable to the extraction of information from them [9].

1.2 Contribution of this work

This work extends the work by Nisar et al. [10] which showed promising results in the recognition of activities of daily living (ADL) by taking the presented approach and extending it by differential privacy. The approach is highly vulnerable to a MIA which achieves an accuracy of 84%. It means nearly all training samples can be identified for an attacker.
Providing a differential privacy approach one step closer to being used in daily life as preserving the privacy of users is required by federal law. This work evaluates different options to provide differential privacy. The main focus of the evaluation is the trade-off between utility and protection. Therefore, it is desired to achieve an accuracy which is as close as possible to the presented work by Nisar et al. [10] while still protecting the data.
The data set is difficult to classify as the reference approach by Nisar et al. [10] achieves only an accuracy of 63.64%. Therefore, it is difficult to provide differential privacy without dropping the accuracy that much that it is not usable anymore. This work achieves this trade-off between accuracy and protection as it drops the MIA accuracy by 31% to 57.95% accuracy which is nearly nonfunctional. In contrast, the accuracy only drops by 13% to 55.31%. Improvement in both areas is achieved by implementing a differentially private Support Vector Regression (SVR). Differential privacy for SVM has also been evaluated but has not shown similar promising results.

1.3 Overview

This work explains in Chapter 2 the theoretical basics of the used ML techniques and a brief introduction to differential privacy.
Chapter 3 sets its focus on the security evaluation of the proposed approach by Nisar et al. [10] and the implementation of differentially private algorithms which are also introduced in that chapter. There are three algorithms:

1. Output perturbation. An approach which perturbs the output of the classifier with noise.
2. Objective perturbation. This approach perturbs the optimization function itself and therefore adds noise not only after training but also during the training phase.
3. Minibatch Stochastic Gradient Descent. A gradient descent based SVM which is trained with differential privacy gradient descent. This approach also explains how the SVM is transformed into a gradient descent problem.

The following Chapters 4 and 5 evaluate these algorithms and assess them.

1.4 Related Work

Preserving privacy is crucial in many cases. Even if it does not look like at first glance, there is information that can be gained from data that can leak sensitive information about the subject. For example, eye tracking is an interesting area for virtual reality. It can reduce the amount needed computation power as the scene has only been rendered at the area the user is looking at. On the other side eye-tracking can be used to improve user interfaces as the developer can get information through eye tracking, like, for example, which are the areas where a user is mostly looking at. On the first glance, having information about the eye movement of a single subject seems harmless. However, eye movement is linked to mental disorders like Alzheimer's or Parkinson's. Extracting this information out of the data is not something anyone who took part in data acquisition would like to share. Steil et al. [11] presented in their work a privacy preserving method, which tracks the eye movement and achieves sufficient accuracy while preserving privacy.
Zhang et al. [12] presented a differential privacy support vector machine (SVM) which uses the sequential optimization algorithm to optimize itself. The advantage of this approach it provides a differential privacy SVM while using its fastest optimization algorithm.
One of the first differential privacy SVM was presented by Rubinstein et al. [13]. It showed that an SVM can provide differential privacy without any restrictions which kernel functions can be used. This is an important finding, as the usage of different kernel functions to classify nonlinear data is one of the key advantages of an SVM.
Differential privacy for convolutional deep belief networks for human behavior modeling

was presented by Phan et al. [14]. It showed that choosing an $\epsilon < 0.8$ carries a large accuracy loss. However, the differentially private approach still achieves an accuracy of 75% for $\epsilon = 0.5$ but the used data set is not very difficult to classify.

Chapter 2

Fundamentals

2.1 Machine Learning

ML is the training of a system to deal with unseen data. Therefore, it is a part of artificial intelligence. It can be used for numerous application scenarios, such as classification, pattern recognition, or translation [15].
ML can be divided into two classes, supervised techniques and unsupervised techniques. Supervised techniques train a system on labeled training data which then can deal with unknown data. The accuracy of these systems is heavily depending on the quality of the training data. Due to the high increase in computation power in recent years, ML has gained interest in many different use cases. For example, the majority of the proposed methods for sensor-based activity recognition are based on ML [2].

2.1.1 Support Vector Machine

An SVM is a classifier based on supervised ML. Firstly published in 1995 [16, 17], SVMs have since gained high popularity in several fields of areas like pattern recognition [18], image classification [19] or computational biology [20].
To classify data in a binary classification task, an SVM is searching for a hyperplane which separates the data points into two parts. This hyperplane can be represented as follows:

$$h(x) = w^T x + b \tag{2.1}$$

In Equation 2.1 the hyperplane is represented by the SVM's weights w and SVM's bias b. Multiplying the weights with a point x and adding the bias b results in a floating number which reflects the position of the point x on the hyperplane. If $h(x) = 0$ then x lies on the hyperplane otherwise x falls on either side of the hyperplane depending on the sign. As this output value reflects the distance of the point to the hyperplane, the value can also be interpreted as a confidence value. A big distance from the hyperplane can be interpreted as a high confidence that this point belongs to the corresponding class. However, if a good degree of certainty is wanted, the SVM has to be calibrated via Platt Scaling [21] (see Section 2.1.3) to do so.

Since there is a theoretically indefinite number of possible hyperplanes which fulfill the requirement of separating the two classes, an SVM is searching for the optimal hyperplane. To do so, the SVM is trying to maximize the distance between the hyperplane and the outliers of each class. These outliers are called support vectors. The distance between the hyperplane and the support vectors is named as the margin. A standard SVM has a margin of 1, therefore the following equation counts for data points:

$$\begin{aligned} w^T x_i - b = 1 \quad \text{if } y_i = 1 \\ w^T x_i - b = -1 \quad \text{if } y_i = -1 \end{aligned} \tag{2.2}$$

In Equation 2.2 w and b represent the SVM's weights and bias, whereas x_i and y_i represent the i-th data point and its corresponding class label. Equation 2.2 can be transformed into one inequality:

$$y_i(w^T x_i + b) \geq 1 \tag{2.3}$$

Since the goal of an SVM is to find the best hyperplane, it maximizes the margin. Using Equation 2.3 this can be expressed as:

$$\begin{gathered} \min \frac{||w^2||}{2} \\ \text{s.t Linear Constraint: } y_i(w^T x_i + b) \geq 1 \end{gathered} \tag{2.4}$$

This form of an SVM is also called the hardmargin SVM as it purely tries to maximize the margin between the hyperplane and the points. Furthermore, this is a convex optimization problem, which is a necessity for the privacy guarantees in Section 2.3.

However, a hardmargin SVM can only deal with linear separable data. Since most real world data is not linearly separable, some adjustments have to be made to work with that nonlinearity. Cortes and Vapnik [17] proposed several techniques to implement a nonlinear SVM. The first proposition is the introduction of a slack variable ξ to create a so-called softmargin SVM for Equation 2.3:

$$y_i(w^T x_i + b) \geq 1 - \xi_i \quad \xi_i \geq 0 \tag{2.5}$$

The slack variable ξ_i allows a data point x_i to violate the margin. This introduction makes it possible to change the hardmargin optimization problem in Equation 2.4 to the softmargin optimization problem:

$$\begin{gathered} \min \frac{||w^2||}{2} + C \sum_{i}^{N} \xi_i \\ \text{s.t Linear Constraint: } y_i(w^T x_i + b) \geq 1 - \xi_i \end{gathered} \tag{2.6}$$

In Equation 2.6 the parameter C defines the influence of the slack variables and must satisfy the condition $C > 0$. If $C \to 0$, the softmargin SVM becomes a hardmargin SVM like in Equation 2.4. If $C \to \infty$, the margin has no effect and the constraint dominates the equation. This can be interpreted as a loss function, as it maps the miss classification to a floating number. Therefore, C is a hyperparameter of the softmargin SVM and has to be set according to the data set. However, in most cases $C = 1.0$ performs sufficient.

By looking at Equation 2.5, it can be seen that only data points, where $y_i(w^T x_i + b) < 1$, carry a loss as they require a $\xi_i > 0$ to satisfy the linear constraint in Equation 2.6. Therefore, Equation 2.5 can be rewritten as the so-called hinge loss [22]:

$$\mathcal{L}(x) = \max(0, 1 - y_i(w^T x_i + b)) \tag{2.7}$$

To calculate the loss, the data point x_i is multiplied with the weight w, some bias b is added and the result is afterwards multiplied with the class label, which in a binary SVM is either -1 or 1. If the data point is on the correct side of the hyperplane, only a small loss is added for points with a distance smaller than 1 to the hyperplane. Data points with a distance greater than 1 on the correct side of the hyperplane do not carry a loss. Data points on the wrong side of the hyperplane have a linear increasing loss regarding their distance to the hyperplane.

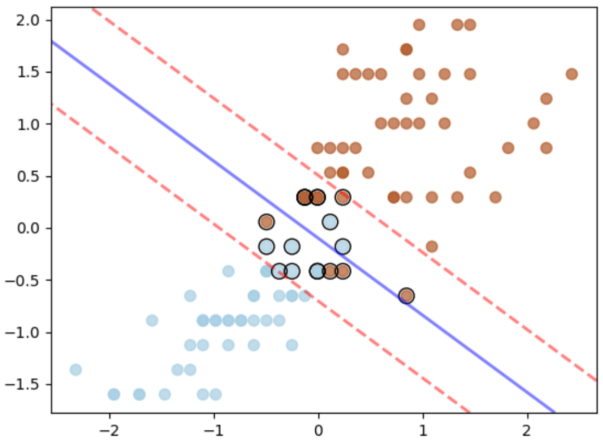

Figure 2.1: A graphical example of a softmargin SVM with $C = 1.0$. The hyperplane is colored in blue, the margins are dotted. There are some missclassifications which are allowed in a softmargin SVM, since this data is not linearly separable. Points with a black edging are the support vectors.

Another approach to deal with nonlinear data is its transformation into a higher dimensional space where it can be separated. There are several different kernels to achieve this objective, with the most popular one being the Radial Basis Function. This kernel has a very good projection into higher dimensional space. However, it comes with a cost of an increase not only in the form of computation but also in the number of hyperparameters.

2.1.2 Multiclass Support Vector Machine

Originally, Support Vector Machines were designed only for binary classification. However, many real world use cases are multiclass classification problems. There are some approaches for using SVM in multiclass classification: The two most commonly used approaches are the One-vs-Rest (OvR) classification [23] and the One-vs-One (OvO) classification [24]. For example, LibSVM, which is the reference SVM implementation and is internally used by many SVM frameworks, provides for multiclass classification OvR and OvO [7]. Even though both approaches were originally introduced for different classifiers, it was shown in [25, 26], they can also be used for SVMs .
In OvR classification for each class a binary SVM is trained where the corresponding class is labeled as 1, and all the samples, which do not belong in this class, are labeled as -1. The prediction is based on a winner-takes-it-all approach. This means the highest positive output is selected as the winner. This is caused by the fact that the distance can be interpreted as how well a data point fits in this class. The greater the distance is, the better the data point fit in this class. In a rare case, when all SVMs predict negative distances and the sample falls in the rest category, the class with the smallest negative output is selected. The advantage of this approach is that it trains as many SVMs as class labels and is therefore, especially for scenarios with many classes, efficient in its computational time.
In contrast, OvO classification trains for each class combination a binary SVM. This means each class is trained against each other class. Since the training data has to be split for each of the class combinations, it is necessary to have a certain number of samples for each class. The prediction is done according to a voting scheme, where each binary SVM predicts a class which then is awarded with a vote. The class with the most votes is considered as the final prediction. The main advantage of this approach is the more accurate classification, since there are many more SVMs than in the OvR approach. The disadvantage is that the number of binary SVMs scales with the number of classes according to the formula $n(n-1)/2$, where n is the number of classes. In scenarios where the number of classes is high, the training time can become an issue.

In addition to these two approaches based on binary classification, there is another approach to implement a multiclass classification by transforming the hinge loss to a multiclass hinge loss [27]:

$$\mathcal{L}(y) = \max(0, 1 + \max_{y \neq t} w_y x - w_t x) \tag{2.8}$$

In Equation 2.8 t is the target label, w_y and w_t are the weights of the SVM. This approach does not perform better than OvR and OvO [26]. Moreover, it is only with some adjustments compatible with the Sequential Minimization Algorithm (SMO) [28], which is the reference algorithm for an SVM. Therefore, multiclass SVM still relies on either OvR or OvO.

2.1.3 Platt Scaling

An SVM predicts only class labels. However, in some cases probabilistic predictions are needed. An SVM can be calibrated to provide probabilistic predictions via Platt Scaling [21]. Platt Scaling is fitting a Sigmoid function for this.

$$P(y = 1|f) = \frac{1}{1 + exp(Af + B)} \tag{2.9}$$

Platt Scaling calibrates a binary SVM with Equation 2.9 for the probabilistic output. To calibrate a multiclass SVM, Platt Scaling has to be adjusted. In OvO case, this can be done by Pairwise Coupling [29] and in the OvR case by extending the coupling [30]. Since an OvO SVM can be transformed into an OvR SVM, the extension of the coupling is often used for multiclass calibration as that algorithm is more efficient and easier to implement.

2.1.4 Support Vector Regression

Support Vector Regression (SVR) is an extension of the well-known SVM for regression analysis [31]. The optimization function of a hardmargin SVR is as follows:

$$\begin{aligned} &\min \frac{||w^2||}{2} \\ \text{Constraint: } &|y_i - (w^T x_i + b)| \leq \epsilon \end{aligned} \tag{2.10}$$

In Equation 2.10 y_i denotes the target value of the predictor x_i, w denotes the weights and b denotes the bias. The parameter ϵ defines the distance from the regression function in which all predictors should lie in. Figure 2.2 shows the graphical representation of ϵ-tube which bounds all predictors. Unlike other regression algorithms, an SVR allows this ϵ deviation from the outcome value.
Since there might be some cases where not all predictors can lie in this ϵ-tube, the softmargin approach of an SVM can also be used for regression analysis. Unlike the softmargin SVM, two slack variables ξ_i and ξ_i^* are introduced [32]. This allows to individually tune the allowed outliers above and below the ϵ-tube. However, in most implementations $\xi_i = \xi_i^*$ is used, because having different ξ_i and ξ_i^* increases the number of hyperparameters and complicates the training process.
The introduction of these slack variables ξ_i and ξ_i^* changes Equation 2.4 to:

$$\begin{aligned} &\min \frac{||w^2||}{2} + C \sum_i^N (\xi_i + \xi_i^*) \\ \text{Constraint: } &y_i - w^T x_i - b \leq \epsilon + \xi_i \\ &w^T x_i + b - y_i \leq \epsilon + \xi_i^* \\ &\xi_i, \xi_i^* \geq 0 \end{aligned} \tag{2.11}$$

The parameter C with $C > 0$ defines the tolerance for points outside of ϵ. If $C \to 0$ Equation 2.11 approaches Equation 2.10. If $C \to \infty$, the tolerance for the data points

outside of ϵ increases and they dominate the equation. Usually, $C = 1.0$ is considered a suitable.
The loss of an SVR is known as the ϵ-insensitive loss function and can be expressed as [32]:

$$\mathcal{L}(X, \epsilon) = \begin{cases} 0 & \text{if} |y_i - (w^T x_i + b)| \leq \epsilon \\ |y_i - (w^T x_i + b)| - \epsilon & \text{otherwise} \end{cases} \tag{2.12}$$

One of the advantages of an SVM is the usage of kernels for classifying nonlinear data. Since an SVR is in its basic a modified SVM, as they both rely on the Support Vector algorithm, an SVR can also use the kernel trick to deal with nonlinear regression analysis. All SVM kernels can also be used for an SVR [7].

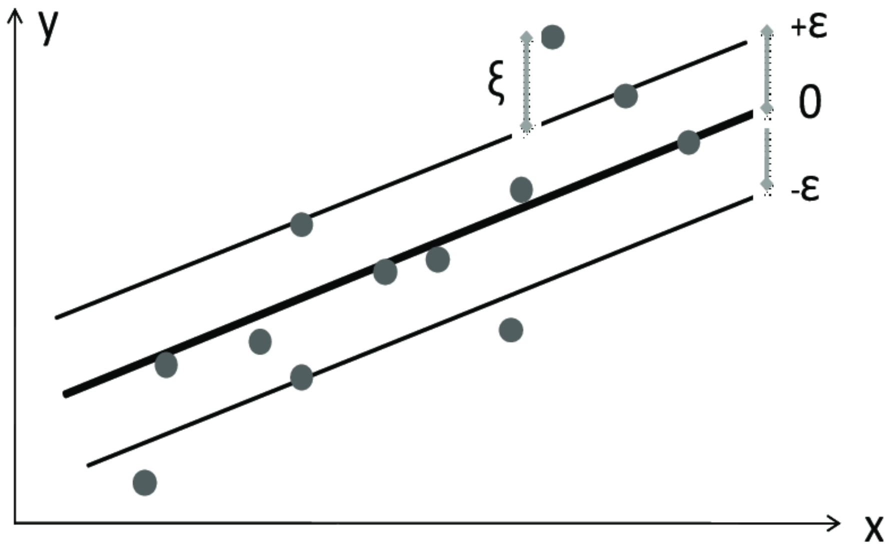

Figure 2.2: Graphical representation of an SVR [33]. The SVR optimizes its regression line in the way that all points should lie in the tube which is stretched by the parameter ϵ. The parameter ξ is the distance to the ϵ-boundary.

2.2 Activity Recognition

Activity Recognition can be understood as the classification of activities performed by humans. Due to the increase in accessibility of cheap sensors in the form of smartphones or smartwatches, activity recognition has gained much traction in research groups recently. Most of the smartphones have built-in motion sensors which make it possible to record data of daily life without needing to setup complex sensor environments [34]. It has led to a high increase in collected data which is benefiting the ML algorithms. Since the collected data consists mostly out of raw sensor data, activity recognition is very familiar to pattern recognition. The focus in activity recognition has recently

shifted to the usage of deep neural networks (DNN). As the amount of data has increased and especially the computational power of modern hardware, deep learning became sufficient for recognizing activities [2].
Activity recognition based on raw sensor data is a challenging task for classifiers. Different activities can be performed simultaneously and activities can overlap. This means the activities share common parts among each other. For example, drinking water and washing teeth share both the movement of a glass to the mouth.

Due to these challenges, new approaches in activity recognition distinguish between atomic activity and composite activities [10, 35]. In this distinction, a compositive activity consists of several atomic activities. In the scope of activity recognition, these approaches first classify the atomic activities and afterwards the composite activities. The advantage of this approach is that it can deal with overlapping activities well. For example, the composite activity "preparing food" consists of several atomic activities like standing, walking, cutting, opening a door, taking, and closing a door.

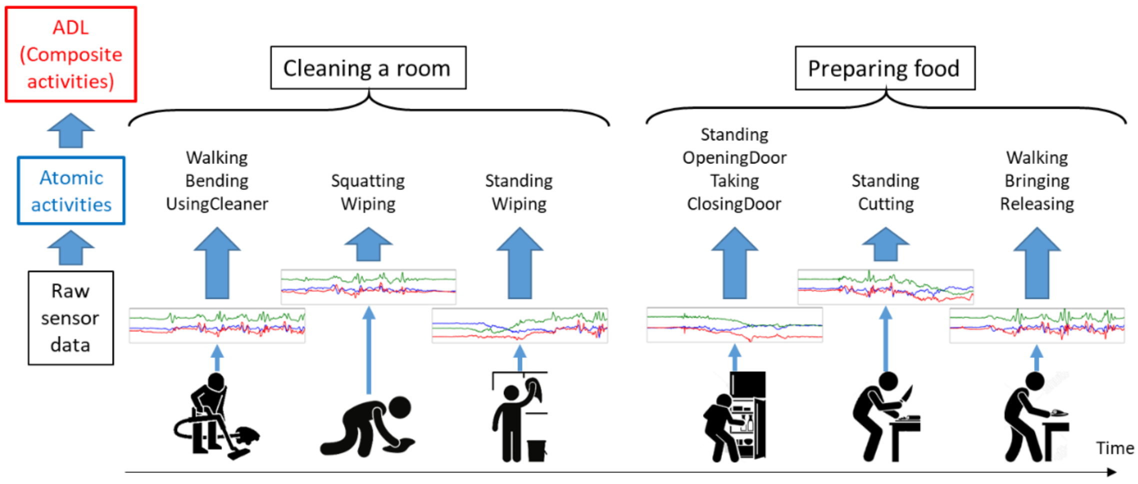
ADL (Composite activities)
Atomic activities
Raw sensor data
Cleaning a room
Preparing food
Walking Bending UsingCleaner
Squatting Wiping
Standing Wiping
Standing OpeningDoor Taking ClosingDoor
Standing Cutting
Walking Bringing Releasing
Time

Figure 2.3: Schematic model of the Activities of Daily Living (ADL) recognition [10]. The raw sensor data is collected with a smartphone. Out of this data the atomic activities are learned via Rank Pooling. In the last step the composite activities are learned with an SVM.

Using this distinction between atomic and composite activities, it is possible to use the rank pooling approach to maximize the precision of the activity recognition.

2.2.1 Rank Pooling

Rank Pooling is a temporal pooling method for activity recognition [36]. It uses the temporal dynamics of the input data to recognize activities or actions. Even though it was originally proposed for video data, it is also suitable for sensor-based timeseries [10]. Figure 2.4 shows a graphical scheme of rank pooling with video data as input.
Rank pooling takes the sensor data of the atomic activities as input. It searches for

the temporal evolution of this input which ultimately can be related to the composite activities. There are several options to find that evolution, however, the time-varying mean vectors approach suits in most cases well. Time varying mean vectors have one advantage over other approaches like moving average: They can not only find dependencies in past to future frames, they can also find dependencies in future to past frames [36].
After these features are generated, the actual rank pooling can be done. The rank pooling is done with an SVR, which learns the evolution of the frames regarding their appearance in the video. This is stored in the parameters of the SVR. After training the SVR, these parameters are used as the input for the classifier, which in this case is an SVM. This SVM learns the parameters of the SVR with the labeled composite activities to perform the actual activity recognition.

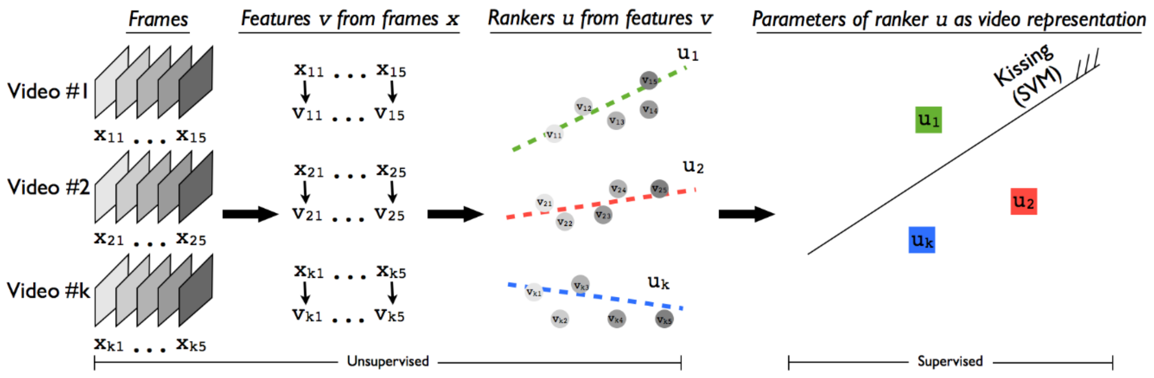

Figure 2.4: The steps of rank pooling [36]. Each video can be interpreted as a dataset. Each frame is extracted from the video and for each frame one feature is generated. These features are then used as the input for the ranking machines, which is an SVR in the case of ADL recognition. Finally the weights of the SVRs are used to train an SVM which learns the representation of the composite activities.

2.3 Privacy in Machine Learning

Privacy is important in many areas. ML is no exception. Due to the increase of usage of ML in several areas, privacy has become an issue. In many areas, ML relies on sensitive data which shall not leak out to the public. Therefore, privacy preserving methods for ML are an important research area and have gained much attention recently.
Data leakage is a dangerous problem. The leakage can be exploited in attacks against the ML algorithm which have to goal to extract that data. In contrast to attacks which try to extract information about the training data out of a ML algorithm, there are also attacks which try to harm the classifier itself. For example, adversarial attacks [37] try to inflict a false classification by designing malicious inputs. Depending on the attack scenario, different defense mechanisms and privacy parameters are needed. However,

there are also attacks which do not directly attack the ML model itself. They attack the system at different points during the usage of the model. This is especially a problem for online models. This work focuses on attacks which try to extract information out of ML model and do not take vulnerabilities at different points in the usage of the model into consideration.
One of the main reasons ML models may leak information is their overfitting on the training. To prevent this overfitting, a class of regularized empirical risk minimization classifiers is introduced. These classifiers can be defined as follows [38]:

$$J(f, D) = \frac{1}{n} \sum_{i=1}^{N} \mathcal{L}(f(x_i, y_i) + \Delta N(f) \tag{2.13}$$

They add to the loss function $\mathcal{L}$ a regularization term to penalize overfitting. This is a small privacy preserving method.

To express privacy, the concept of $\epsilon-$differential privacy was introduced by Dwork et al. [39]: p

Definition 2.3.1 (ϵ-Differential Privacy) *Let D be the set of all databases, Obs be the set of observations, and R(Obs) be the set of random variables over Obs. A randomized algorithm $\mathcal{M} : D \to R(Obs)$ for any pair of databases D and D' which only differ at most at one row is $\epsilon-$differentially private if for and for all tests $S \subseteq Obs$:*

$$Pr[\mathcal{M}(D) \in S] \leq exp(\epsilon) Pr[\mathcal{M}(D') \in S]$$

Definition 2.3.1 expresses that the probabilities Pr of all potential outcomes S of an algorithm $\mathcal{M}$ performed on D and D' only differ at most exp(ϵ).
Algorithms which satisfy the definition of $\epsilon-$differential privacy guarantee a strong robustness against known attacks. The parameter ϵ defines the privacy. A small ϵ results in high privacy but also results in low accuracy. If ϵ approaches 0, the privacy is highly protected, but the classifier does not produce accurate results. Therefore, ϵ must be chosen in such a way in consideration of privacy and accuracy. Privacy is achieved by adding noise to the algorithm. This randomness guarantees that a single data point can only have so much influence that changing it does not leak any information for an attacker who only has access to the output of the model.
The concept of $\epsilon-$differential privacy can be extended to the concept of (ϵ,δ)-differential privacy [40].

Definition 2.3.2 ((ϵ,δ)-Differential Privacy) *Let D be the set of all databases, Obs be the set of observations, and R(Obs) be the set of random variables over Obs. A randomized algorithm $\mathcal{M} : D \to R(Obs)$ for any pair of databases D and D' which only differ at most at one row is $(\epsilon, \delta)-$differentially private if for and for all tests $S \subseteq Obs$:*

$$Pr[\mathcal{M}(D) \in S] \leq exp(\epsilon) Pr[\mathcal{M}(D') \in S] + \delta$$

Definition 2.3.2 expresses that the probabilities Pr of all potential outcomes S of an algorithm $\mathcal{M}$ performed on D and D' only differ at most $\exp(\epsilon)$ with an probability δ that information is leaked accidentally. It is not as strict as plain $\epsilon-$differential privacy and can achieve better accuracy for the classifier while still providing a strong privacy.

Theorem 2.3.1 (Post-Processing) *Let $\mathcal{M} : \mathbb{N}^{|\mathcal{X}|} \rightarrow R$ be (ϵ, δ)-differentially private. Let $\boldsymbol{f}$ be a function with $f : \mathcal{R} \rightarrow \mathcal{R}'$. Then $\boldsymbol{f} \circ \mathcal{M} : \mathbb{N}^{|\mathcal{X}|} \rightarrow R'$ is (ϵ, δ)-differentially private.*

Theorem 2.3.1 expresses that an algorithm which is performed on the output of an (ϵ_i, δ_i)-differentially private algorithm cannot add leakage [41].

Theorem 2.3.2 applies for the composition of algorithms with (ϵ, δ)-differential privacy [41].

Theorem 2.3.2 (Composition) *The composition of n (ϵ_i, δ_i)-differentially private algorithms is $(\sum_i^n \epsilon_i, \sum_i^n \delta_i)$-differentially private.*

This theorem is especially important for multiclass classification which is based on OvO-classification or OvR-classification. In that case, ϵ and δ have to scaled down for each binary classifier to achieve (ϵ,δ)-differential privacy in the multiclass classification algorithm.

2.3.1 Privacy-Preserving Mechanism

In recent times, increasingly privacy-preserving algorithms have been published. In the scope of this work, the output perturbation, which is introduced as algorithm 1 and the objective perturbation as algorithm 2 are presented [38]. Furthermore, for the SVM experiments a (ϵ,δ)-differentially private SGD is evaluated, which is introduced in algorithm 3 [42].

Output perturbation and objective perturbation choose a predictor $\mathbf{f}$ which minimizes the regularized empirical loss, defined in Equation 2.13.

Both algorithms have the following restriction for their input X:

$$\forall x \in X : ||x||_2 \leq 1 \tag{2.14}$$

To preserve privacy, both algorithms rely on the addition of noise at different points during the training phase. This noise is represented by the noise vector $\mathbf{b}$, drawn from the Laplace distribution and is a random noise with density [38]:

$$v(\mathrm{b}) = \frac{1}{\alpha}\mathsf{e}^{-\beta||\mathrm{b}||_2} \tag{2.15}$$

In Equation 2.15, α is a normalizing constant and β is a parameter defined by the relevant algorithm.

The first algorithm 1 is the output perturbation that adds noise to the weights of the classifier.

Algorithm 1 ERM with output perturbation

```
    Input: Data X, parameters $\epsilon_p, \lambda$
    Output: minimizer $\mathbf{f}_{\mathbf{priv}}$
1: Draw vector $\mathbf{b}$ according to 2.15 with $\beta = \frac{n\lambda\epsilon_p}{2}$
2: Compute $\mathbf{f}_{\text{priv}}$ = argmin $J(\mathbf{f},\text{X})+\mathbf{b}$
```

Algorithm 2 is the objective perturbation that adds noise to the objective function itself and minimizes the perturbed objective function. This perturbation is as expressed as [38]:

$$J_{\text{priv}}(\mathbf{f}, X) = J(\mathbf{f}, X) + \frac{1}{n}\mathbf{b}^T\mathbf{f} \tag{2.16}$$

In Equation 2.16, $\mathbf{b}$ has a density according to Equation 2.15 with $\beta = \epsilon_p$. With this definition of the perturbed objective function, the objective perturbation can be defined as shown in algorithm 2.

Algorithm 2 ERM with objective perturbation

```
    Input: Data X, parameters $\epsilon_p, \lambda, c$
    Output: minimizer $\mathbf{f}_{\mathbf{priv}}$
1: $\epsilon'_p \leftarrow \epsilon_p - \log(1 + \frac{2c}{n\lambda} + \frac{c^2}{n^2\lambda^2})$
2: if $\epsilon'_p > 0$ then
3:     $\Delta \leftarrow 0$
4: else
5:     $\Delta \leftarrow c/(n(\mathrm{e}^{\epsilon_p/4} - 1)) - \lambda$
6:     $\epsilon'_p \leftarrow \epsilon_p/2$
7: end if
8: Draw vector $\mathbf{b}$ according to 2.15 with ß=$\epsilon'_p/2$
9: Compute $\mathbf{f}_{\text{priv}}$ = argmin $J_{\text{priv}}(\mathbf{f},\text{X})+\frac{1}{2}\Delta||\mathbf{f}||^2$
```

The case distinction in algorithm 2 at lines 2 and 4, is needed for the proof that this algorithm provides ϵ-differential privacy [38].

In the scope of this work, the SVM is also implemented with an SGD approach. An (ϵ,δ)-differentially private SGD is presented by Abadi et al. [42], and implemented in algorithm 3. Compared to a non-differentially private SGD, this algorithm clips the gradients. This means each gradient greater than the clipping bound C is clipped in its length to C. Clipping is required to fulfill the differential privacy as it bounds the influence of each sample on the gradient descent [42]. The clipped gradient is perturbed with noise drawn from a multivariate Gaussian distribution.

Algorithm 3 differentially private SGD

```
    Input: Data X, parameters: privacy $\epsilon_p$, clipping bound $C$, learning rate $lr$
    Output: $weights$
1: Initialize $weights$ randomly
2: Compute Gradient
3: if ||Gradient|| > 1.0 then
4:     Clip Gradient
5: end if
6: Add noise
7: Descent
```

2.4 Membership Inference Attack

A Membership Inference Attack (MIA) is an attack which tries to determine whether a sample was used for training the model or not [43]. Although this does not sound that much inflicted in the first moment, in scenarios where the training data consists out of sensitive data, this can become an issue. An attacker can gain a type of information out of this knowledge. For example, if this training set was used to train a model which is learning if a certain treatment method has success for a certain disease, you can assume that each sample in that training data is from persons who suffer from that disease, hence an attacker now knows that this person, whose sample he got his hands on, suffers from that disease.
A MIA exploit the fact that models in general have a higher confidence on their training samples than on samples which were not used for training. Therefore, a MIA performs best if the victim model, which is the model the MIA attacks, produces likelihood vectors as output and not just the labels. These likelihood vectors should show a much higher likelihood for a single class in case the sample was used to train the model. If the sample was not in the training set, the likelihoods are more equally distributed. The following two steps are required to perform MIA.
Firstly, a shadow model is trained on similar data. The higher the similarity of the data to the training data of the victim model, the more accurate is the attack. The MIA performs best in a white-box scenario, where an attacker has knowledge of the victim's layout. It can also work in the blackbox scenario, where an attacker does not have any knowledge. In the later case, an attacker might use several shadow models to limit this restriction [43]. It is to use similar data to train this shadow model as it was used in the training of the victim model.
Once the shadow models are trained, the actual attack model can also be trained. It acts as a binary classifier by taking the outputs of the shadow model and predicting whether these outputs were part of the training samples or not. Since this is a binary

classification task, the attack model itself can be a small fully connected neural network with 3 layers. Figure 2.5 shows a MIA schematic.

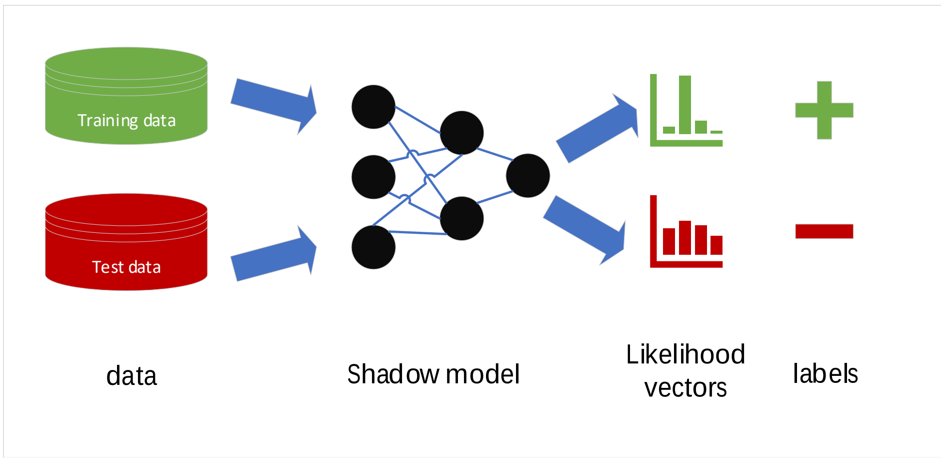

Figure 2.5: Scheme of a Membership Inference Attack. A similar data set is split into training and testing set. A shadow model is trained on this training set. After training is done, the shadow model predicts the whole data set. The predicted likelihood vectors show a pattern: Likelihood vectors for training samples have a high confidence for a single class. In contrast likelihood vectors for testing samples does not have this high confidence. Training samples are labeled for the attack model which is learning based on the likelihood vectors and the corresponding labels.

Chapter 3

Experimental Setup

This chapter describes the implementation of privacy preserving algorithms. Firstly, the implementation and results of the MIA are shown. Implementation of the MIA, SVM, and SVR can be found at `https://github.com/nkath/differentiallyprivatesv`.

3.1 Setup and Implementation

3.1.1 Membership Inference Attack

In the first step, the current approach presented by Nisar et al. [10] has to be investigated if it is vulnerable to membership inference attacks.
The recognition of ADLs in that approach is achieved by implementing an SVR and an SVM in the sklearn framework [44] which provides an interface to the C++ implementation of those two ML algorithms in the libsvm framework [7]. Since most membership inference attacks exploit overfitting on the training data, the activity recognition has to produce not only labels but also probabilities. The sklearn SVM implementation provides two options to calibrate itself to produce probabilistic outputs:

- Self-calibrating the SVM
- Usage of the CalibratedClassifierCV(CCCV) class

Self-calibrating the SVM only requires the parameter **probability** to be set to true, which internally uses a k-fold cross-validation to train and calibrate the classifier.
This calibration method has the consequence that the probabilities heavily overfit for training samples. This can be seen in Table 3.2 which shows the calibrated probabilities 3.2 for the first training sample in the training-testing split of the CogAge data set. The corresponding output of the SVM's decision function is shown in Table 3.1. There are two labels which have a great distance from the hyperplane, label 0 and label 3. Naturally, it is expected that label 0 has the highest probability and label 3 has a small probability. However, the probabilities in the SVM self-calibration heavily overfit for label 0 as it has a probability of 94.7% while label 3 has a probability of only 1.1%. Interestingly, this is the same probability which is predicted for label 4 and 5, which have a much smaller distance from the hyperplane.
The usage of the CCCV has two options to perform the calibration. Either a pretrained classifier is calibrated or the CCCV trains and calibrate a classifier. The later option

is used in this work. Hereby, 5-fold cross-validation is used by default. This method has a better mapping of the distances from Table 3.1 to probabilities in Table 3.2. It predicts a probability of 62.3% for label 0 and 31.0% for label 3. In contrast to the SVM self-calibration, this calibration method does not overfit heavily towards label 0.

In contrast, the CCCV takes this into consideration as label 0 still has the highest probability with 62.3% but also has some probability for label 3 with 24.1%.

Table 3.1: The 0 vs Rest binary SVM has the greatest distance from the hyperplane, therefore label 0 is the output label. 5 vs Rest has also a great distance from the hyperplane.

0 vs Rest	1 vs Rest	2 vs Rest	3 vs Rest	4 vs Rest	5 vs Rest	6 vs Rest
6.29	1.78	-0.25	5.26	3.95	0.75	2.83

Table 3.2: The self-calibrated SVM in table 3.2 heavily favors Label 0 with its probability prediction, while the CCCV does not as it also has some probability for label 3. In case the CCCV is trained on the same data set as the classifier, which is the approximation of self-calibrated SVM, the CCCV shows a similar overfitting on label 0.

Calibration	Label 0	Label 1	Label 2	Label 3	Label 4	Label 5	Label 6
SVM	94.7%	1%	0.5%	1.1%	1.1%	1.1%	0.5%
CCCV	62.3%	0.2%	0.3%	31.0%	0%	1.9%	4.3%

The most well-known Membership Inference Attack is the attack suggested by Shokri et al. [43]. This attack creates for each class a shadow model and an attack model. In the scope of this work, the focus lies on white-box attacks, therefore the attacker knows the architecture of the victim model. This means an attacker can use the same architecture of the victim model as the shadow model. The shadow models have to be trained on a different set of samples than the victim model.
Since the CogAge data set is rather small with a total of 890 samples, training the shadow models can become an issue as there are not enough samples for each shadow model. There is a simplification of this attack suggested by Salem et al. [8]. Instead of creating a shadow model for each class, this attack only uses a single shadow model which is called S. This shadow model S is trained on the data set X_1 which is one half of the data set X. The other half of X is used for the testing set X_2. Shadow model S predicts the probabilities for the data set X after it was trained. The probabilities for each sample are represented in descending order, and the highest 3 possibilities are considered, the rest is cut off. This probability array is the input for the attack model A. The label input for A is a column vector Y with the amount of samples in X which has a value of 1 at the positions of the samples which were used for training and 0 for those which were not used for training. The attack model A can consist of 3 fully connected layers with a decreasing number of neurons. After training for around 100

epochs, the attack model A can be used to attack the victim model. Therefore, the output of the victim model is given to A which then predicts the membership of the samples belonging to this output.

Algorithm 4 Salem MIA Attack

```
    Input: data X, label y
 1: $X_1, X_2 \leftarrow$split($X$)
 2: Train Shadow model $S$ on $X_1$
 3: $prediction \leftarrow$S.predictproba($X$)
 4: $attackdata \leftarrow$ topThree($prediction$)
 5: $truetrain \leftarrow$ones(len($X_1$)
 6: $falsetrain \leftarrow$zeros(len($X_2$)
 7: $attacklabel \leftarrow$stack($truetrain, falsetrain$)
 8: Train attack model $A$ on $attackdata$ and $attacklabel$
 9:
```

For the demonstration purpose of the MIA, the shadow model S and attack model A are trained on the testing data set of the victim model. Therefore, this work focuses on the hold-out setting, where half of the data is used for training and the other half is used for testing. This setup is used as the MIA relies on similar data to train on and there are no other public available data sets which are similar to the CogAge data set. The training data set is used to train the victim model, the testing data set is used to train the shadow model S.
The victim model then predicts the probabilities for each sample of the CogAge data set and the attack model A predicts out of these probabilities of the samples those were used for training.
The victim model is the approach by Nisar et al. [10], which combines a LinearSVR and SVM in the sklearn-framework [44]. To be more precise, the model which is being attacked, is the SVM implementation. The shadow model is the same SVM implementation which the victim model is using. The attack model is implemented in the TensorFlow framework [45]. This framework provides an easy to use interface to build and use neural networks. The attack model is a 3-layer fully connected neural network with a decreasing number of neurons in each layer. It starts with 128 neurons in the first layer, 64 in the second and has one output neuron in the last layer. Since it is a binary classifier, the attack model is trained with binary cross-entropy as its loss function. Therefore, the output layer is activated with the sigmoid function, so the output of the attack model is interpreted as a training sample if the output is ≥ 0.5 and as a non-training sample if the output is < 0.5.

Table 3.3 shows the results of the MIA for different probability calibration methods for the activity recognition approach by Nisar et al. [10] on the CogAge data set. The attack performs best if the SVM is self-calibrated. In that case, the attack achieves an accuracy of 83%. This is due to the strong overfitting on the training samples of this calibration method, as shown in Table 3.2. This benefits the attack, since it relies

on overfitting on the training samples. This finding strengthens the cause to make the SVR private in rank pooling
Theorem 2.3.2 allows it to make the SVR differentially private and achieve differential privacy even after the SVM is applied. In the scope of this work, it was also evaluated what happens if the SVM is made differentially private. However, in this case the CCCV implementation is used to generate the probabilistic output to have a fair comparison. If the training of the classifier and calibration is done with the 5-fold cross-validation approach of the CCCV, the attack has only an accuracy of 59%.

Table 3.3: This table shows the results of the MIA attack. When the SVM is self-calibrated, the attack can distinguish with an accuracy of 84% if the sample was used for training or not. If the CCCV is used for calibration the attack has an accuracy of 59%.

Calibration	accuracy
SVM	84%
CCCV	59 %

Even though 59% might seem low, the attack works. Nevertheless, there are attacks which are not as simplified as the MIA by Salem et al. [8]. For example, a label-only MIA does not rely on the probabilistic output [46]. These attacks might perform better on the ADL recognition and therefore it is plausible to make this activity recognition have ϵ-differential privacy.

3.1.2 Implementation

The implementations of the support vector algorithms for output and objective perturbation rely on the so-called bias trick. The bias trick allows to transform a classifier in the form of

$$f(x_i, W, b) = Wx_i + b \tag{3.1}$$

to a classifier in the form of

$$f(x_i, W) = Wx_i \tag{3.2}$$

The bias trick combines the two parameters W and b into W by adding an extra dimension to W. This additional dimension stores the previous parameter b. To perform this adaption, x_i is extended with a additional dimension which holds the constant 1. This conversion is crucial for the ongoing sections, as the proposed implementations in algorithms 7 and 8 are using the bias trick and therefore refer to the layout presented in Equation 3.2.

As pointed out in Section 2.1.2 there are different options to implement the multiclass SVM. This work uses the OvO approach for training. Prediction is done via transforming the OvO output to a OvR output which is needed to use the CCCV class.

To implement the ϵ-differentially private SVM and SVR, their loss functions have to approximated. As pointed out in Section 2.3, the privacy preserving algorithms by Chaudhuri et al. require a differentiable loss function [38]. Since the derivative of the SVR's loss function (see 2.12) is discontinuous at $|y_i(w^T x_i + b)| = \epsilon$, the loss function has to be approximated with a differentiable loss function. This can be achieved by the approximation as described in Equation 3.3, which is called the smooth ϵ-insensitive loss [47]:

$$\mathcal{L}(X, \epsilon) = \log(1 + \mathrm{e}^{X-\epsilon}) + \log(1 + \mathrm{e}^{-X-\epsilon}) - 2\log(1 + \mathrm{e}^{-\epsilon}) \tag{3.3}$$

This approximation is smooth and differentiable at all points, and therefore, satisfies the privacy guarantees of algorithms 1 and 2. Figure 3.1 plots this smooth ϵ-insensitive loss against the ϵ-insensitive loss to show the difference of both loss functions.

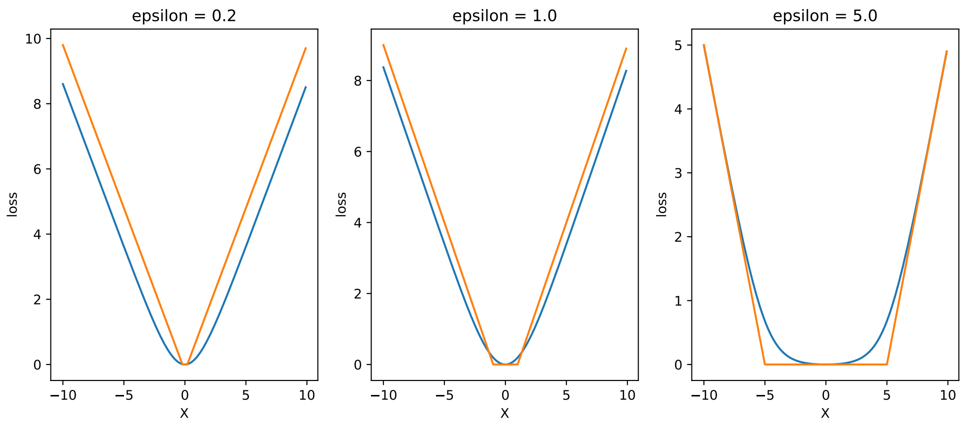

Figure 3.1: Comparison of the smooth ϵ-insensitive loss from Equation 3.3 in blue against the ϵ-insensitive loss from Equation 2.12 in orange. For $\epsilon = 0.2$ the smooth ϵ-insensitive loss is slightly below the ϵ-insensitive loss with a constant offset of 1.15. This offset is decreased down to 0.4 for $\epsilon = 1$ and for $\epsilon = 5.0$ it is not possible to distinguish both lines for $|X| > 6.5$, but there is some difference where $X = \epsilon$.

The plotting shows that Equation 3.3 is a good approximation of Equation 2.12. The smooth ϵ-insensitive loss approximate the ϵ-insensitive loss well for different ϵ values. In the case $\epsilon = 1.0$ the smooth approximation is nearly not distinguishable from the nonsmooth function. If $\epsilon = 0.2$, which is often used, the smooth approximation has a slight offset compared to the nonsmooth function. The plotting of $\epsilon = 5.0$ shows that the smooth ϵ-insensitive loss approximates the ϵ-insensitive loss even for a high ϵ.

The hinge loss is not differentiable at $y_i(w^T x_i + b) = 1$. There are some proposed approximations of the hinge loss which are differentiable. In this work, the approximation by Chapelle [48] is used with $z = y_i(w^T x_i + b)$, which is called Huber loss in the further:

$$\mathcal{L}(z,h) = \begin{cases} 0 & \text{if } z > 1+h \\ \frac{1}{4h}(1+h-z)^2 & \text{if } |1-z| \leq h \\ 1-z & \text{if } z < 1-h \end{cases} \tag{3.4}$$

Figure 3.2 plots this approximation against the hinge loss. The plotting shows that the Huber loss is a good approximation as the Huber loss approximates the non-differentiable spot $z = 0$ well for small h. For $h \to 0$ Huber loss approaches the hinge loss. A typical value for h is 0.5 [48].

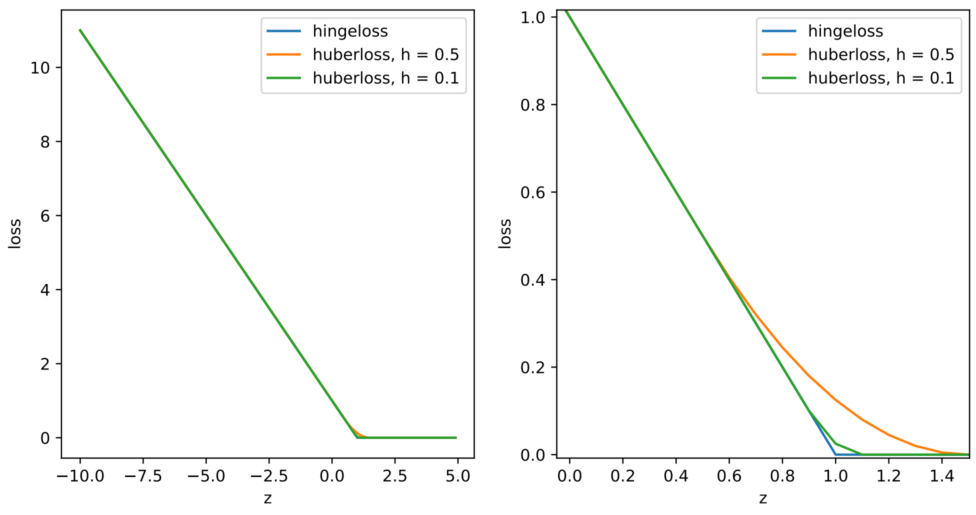

Figure 3.2: Comparison of the smooth Huber loss approximation against the hinge loss. The left image shows that the Huberloss has the same course of the function. The right image is zoomed on $z = 1.0$. If h is chosen small, the course is really similar to the hinge loss.

As pointed out in Section 2.1.4, SVR and SVM are similar to each other. Their biggest difference is the usage of different loss functions. This similarity is used in the implementation of algorithms 1 and 2, as they share most of their code, regardless if an SVM or SVR is implemented. Their main difference is the calling of different loss functions in the respective implementation. Algorithm 5 shows this backbone. The implementation has three input parameters, the data X, the corresponding label vector y and a lossparameter z, which is either the h in the Huber loss or the ϵ in the smooth ϵ-insensitive loss. The difference between an SVM and an SVR implementation is the loss function, which is called and calculated. In case algorithm 5 is used for an SVM, the loss is calculated accordingly to the Huber loss in Equation 3.4. If algorithm 5 is used for an SVR, the ϵ-insensitive loss from Equation 3.3 is used.

Algorithm 5 Support Vector algorithm

```
    Input: data X, label y, lossparameter $z$
    Output: Support Vector Algorithm f
1: $weights \leftarrow$ zeros(len($X$[0])
2: $n \leftarrow$ len($X$)
3: function OBJECTIVE FUNCTION($X$)
4:     $loss \leftarrow$calculateLoss($y(weights^T X), z$)
5:     $f \leftarrow loss$
6:     return f
7: end function
8: $f \leftarrow$ minimize(objective Function($X$))
9: return f
```

As this support vector algorithm is a minimization problem, the minimization can be solved with a quasi-Newton algorithm. There are several algorithms of the quasi-Newton class which can solve this optimization. In this work, that optimization is solved with the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm [49], which is one of the most advanced algorithms in the class of quasi-Newton algorithms. The BFGS algorithm is provided by the scipy framework in Python [50].
The algorithm runs until the maximum number of iterations is achieved or the minimization has converged. As the support vector algorithms are a convex optimization problem, the termination is mostly caused by the converging, not the maximum number of iterations.

The differentially private algorithms by Chaudhuri et al. [38] use Laplace-like noise to achieve protection. Algorithm 6 shows the implementation of the noise generation.

Algorithm 6 Noise Generation

```
    Input: dim d, scale $\lambda$
    Output: noise vector n
1: $normn \leftarrow$gammanoise($dim, \frac{1}{\lambda}, 1$)
2: $r \leftarrow$normalDistribution($0, 1, d$)
3: $n \leftarrow \frac{r}{||r||} * normn$
4: return $n$
```

This algorithm 6 takes as input the dimension d, which is the dimension of the output noise vector, and the scale parameter λ. It draws the noise vector from the Gaussian distribution with λ as the scale parameter and d as the size.

Algorithm 1 can be expressed by using Python and scikit-learn as shown in algorithm 7. Besides the inputs which were used in algorithm 5, two privacy parameters ϵ and λ are added. These are to scale the noise in the output perturbation approach, as explained in algorithm 1.
In the first step, the weight matrix *weights* is initialized. It has the size of a data

point and is initialized with zeros. If Algorithm 7 is used for a OvO SVM, the noise can be drawn from the Gaussian distribution[39], otherwise it is drawn accordingly to Algorithm 6. The scale parameter σ for Gaussian noise is calculated accordingly to:

$$\sigma = \frac{2C \cdot c\sqrt{2\log(\frac{1.25}{\delta})} \cdot \sqrt{|classes|(|classes-1)/2}}{\epsilon} \tag{3.5}$$

In Equation 3.5 C, denotes the softmargin parameter and c is the clipping bound. The term $\sqrt{|classes|(|classes-1)/2}$ is needed to satisfy Theorem 2.3.2. It has to be noted that the calculation of σ is really sensitive to the chosen C. Therefore, C is set to 0.03 which is a good trade-off between accuracy between accuracy and having enough noise. After the optimization is finished, the optimized weights array f is perturbed with the noise vector to achieve (ϵ,δ)-differential privacy.

Algorithm 7 Support Vector algorithm with output perturbation

```
    Input: data X, label y, lossparameter $z$, privacy parameter $\epsilon$, $\lambda$
    Output: Support Vector Algorithm f
 1: $weights \leftarrow$ zeros(len($X[0]$))
 2: if SVM then
 3:     $\sigma \leftarrow (2C \cdot c\sqrt{2\log(1.25/\delta)} \cdot \sqrt{|classes|(|classes|-1)/2})/\epsilon$
 4:     $noise \leftarrow$ GaussianNoise($\sigma$,dim($X$))
 5: else
 6:     $noise \leftarrow$ noise($\epsilon$,dim($X$)) (see Alg. 6)
 7: end if
 8: function OBJECTIVE FUNCTION($X$)
 9:     $loss \leftarrow$calculateLoss($y(weights^T X), z$)
10:     $f \leftarrow loss + 0.5\lambda(||f||_2)^2$
11:     return f
12: end function
13: $f \leftarrow$minimize(objective Function($X$))
14: return f + noise
```

The implementation of the objective perturbation algorithm in Python is shown in Algorithm 8. If Algorithm 8 is used for a multiclass SVM, ϵ has to be scaled down to satisfy Theorem 2.3.2. This is done with the scale factor 1/10 which was calculated using the implementation of Probability Buckets[1][51].
The main difference to the implementation of algorithm 7 is the perturbing of the optimization function accordingly to Equation 2.16. In addition, the regularization is added. Besides this difference, the calculation of the scale parameter for the noise generation is implemented.

[1]https://github.com/sommerda/privacybuckets

Algorithm 8 Support Vector algorithm with objective perturbation

```
    Input: data X, label y, lossparameter $z$, privacy parameters $\epsilon, \lambda$
    Output: Support Vector Algorithm f
 1: $weights \leftarrow$ zeros(len($X$)
 2: $n \leftarrow$ len($X$)
 3: $\epsilon_p' \leftarrow \epsilon_p - \log(1 + \frac{2c}{n\lambda} + \frac{c^2}{n^2\lambda^2})$
 4: if $\epsilon_p' > 0$ then
 5:     $\Delta \leftarrow 0$
 6: else
 7:     $\Delta \leftarrow c/(n(\mathrm{e}^{\epsilon_p/4} - 1))$
 8:     $\epsilon_p' \leftarrow \epsilon_p/2$
 9: end if
10: $noise \leftarrow$ noiseGeneration(dim($X$),$\epsilon_p/2$) (see alg. 6)
11: function OBJECTIVE FUNCTION($X$)
12:     $loss \leftarrow$calculateLoss($y(weights^T X), z$)
13:     $f \leftarrow \frac{1}{n} loss + 0.5\lambda(||f||_2)^2 + \frac{1}{n} noise^T f + \frac{1}{2}\Delta(||f||_2)^2$
14:     return f
15: end function
16: $f \leftarrow$minimize(objective Function($X$))
17: return f
```

Besides these two implementations, the SVM is also implemented with a gradient descent approach which provides (ϵ,δ)-differential privacy [42]. The gradient descent is implemented within the PyTorch framework [52]. PyTorch is, like TensorFlow as well, a ML framework. PyTorch is used in this implementation to calculate and update the gradients and to perform the gradient descent.
Algorithm 9 shows this implementation which is called Mini-Batch Stochastic Gradient Descent (MBSGD). In each iteration, a random batch is drawn. The random batch has the size of 32 in this implementation. From this batch the loss is calculated and based on that the gradient is calculated. The gradient is multiplied with the learnrate lr which in this case is really high with a value of 1. This is required as otherwise the added noise is too dominant. This is caused by the fact that the σ, which is $\sigma = ((|batchsize|/n) * \sqrt{epochs * (|classes| * (|classes| - 1)/2) * \log(1/\delta) * C)/\epsilon}$ [41], scales with the ratio of $|batchsize|/n$ where n is the number of samples in the data set. Since the CogAge data set is rather small, that ratio is quite close to 1 which leads to a great σ. To compensate this, the learning rate is increased and therefore the length of the gradients. This results in most gradients having the length of 1.0, when they are clipped or being close to 1.0. If the length of the gradients is 1.0 or close to 1.0 the noise is no longer dominant and the algorithm is more stable. With the calculated σ the noise is drawn from a multivariate normal distribution. Finally, that noise is combined with the gradient which then is used to update the weights. After this perturbation, the next epoch can start.

Algorithm 9 Mini-Batch SGD Support Vector Machine

```
    Input: data X, label y, clip parameter $C$, privacy parameters $\epsilon, \delta$
    Output: Support Vector ,Algorithm f
 1: $n \leftarrow$ len($X$)
 2: $weights \leftarrow$random($n$)
 3: $batchsize \leftarrow 32$
 4: $lr \leftarrow 1$
 5: for epoch in epochs do
 6:     pick random batch of size $batchsize$
 7:     calculateLoss
 8:     $gradient \leftarrow$calculate gradient
 9:     $gradient \leftarrow lr * gradient$
10:     if $||gradient|| > 1.0$ then
11:         $gradient/ = ||gradient||$
12:     end if
13:     $\sigma \leftarrow (batchsize/n)\sqrt{epochs * (|classes| * (|classes| - 1)/2) * log(1/\delta} * (C/\epsilon)$
14:     $noise \leftarrow$ normalDistribution(mean($gradient$),cov=$\sigma$)
15:     $weights \leftarrow gradient + noise$
16: end for
```

3.1.3 data set

All experiments are carried out on the Cognitive Village (CogAge) data set. In this data set, atomic and composite activities were collected using a smart phone, smart watch, and smart glasses [10].

- LG G5 smartphone to capture the body movement. It was placed in a subject's front left pocket of the jeans and provided the following sensory modalities: a gyroscope, gravity sensor, magnetometer, linear accelerometer, and 3-axis accelerometer.
- Huawei watch to capture the movement of the hand. It was placed on a subject's left arm and provided a magnetometer and 3-axis accelerometer.
- JINS MEME glasses to capture the head movement with 3-axis accelerometer.

The data set consists of 7 composite activities and 61 atomic activities. A total of 9700 instances of atomic activity were recorded on 8 different subjects. Over 1000 instances of composite activities were collected with 6 different subjects, however, some instances had missing sensory data and were therefore removed from the data set. Therefore, in total 890 instances of composite activities were used for the data set. [10]. The composite activities contain:

- Brushing Teeth
- Cleaning Room

- Handling Medication
- Preparing Food
- Styling Hair
- Using Phone
- Washing Hands

3.2 Evaluation Methodology

The evaluation is carried out from two points of view, one is the accuracy and the other is the defense against attacks. All experiments are carried out on the CogAge data set which is described in Section 3.1.3. Different values for ϵ are evaluated, these are: $\epsilon = [0.5; 1.0; 2.0; 4.0; 8.0]$. The experiments are divided into three parts:

1. SVR with output perturbation
2. SVR with objective perturbation
3. SVM with differential privacy

The SVR is first evaluated with two named algorithms. Afterwards, SVM results are represented, even though they are a bit out of the scope of this work. Table 3.4 shows the version of each library used for the experiments.

Table 3.4: Overview of the frameworks used and their version number.

Tool	Version
numpy	1.18.1
Python	3.6.8
scikit-learn	0.23.0
scipy	1.4.1
TensorFlow	2.1.0 GPU
torch	1.7.1+cu101
torchvision	1.7.1+cu101

The accuracy results of this implementation are compared with the results of the rank pooling approach by Nisar et al. [10]. In that work, several pooling techniques were looked at and evaluated by the average F1 score, accuracy, and the weighted F1 score. This work uses the same measures. The work by Nisar et al. [10] was done in the sklearn framework, which provides an interface to the SVR and SVM implementation from the libsvm framework [44, 7]. That work presented results for two scenarios, a hold-out setting and a leave-one-subject-out cross-validation. This work uses the hold-out setting for evaluation as this setting suits best for setting up the attack, as described in Section 3.1. Nisar et al. [10] suggested different pooling techniques. This work specifies the evaluation of the the fused rank pooling with max and average pooling and uses the pooling technique $RP_{Hellinger_{power_F}WD_REV}$. Besides the named measures, this work also evaluates the statistical distribution of the (ϵ,δ)-differentially private SVM for each chosen ϵ by plotting the distribution in boxplots. It has to be noted that this work uses the definition that the whiskers in a boxplot are defined as interquartile range (IQR) $\cdot$ 1.5. In this work, δ is fixed to 0.0001.

The second evaluation is the defense against a MIA. In that scenario, the accuracy of the MIA is the evaluation parameter. The benchmark in this evaluation is 50%, as this is the accuracy achieved by an attacker who purely guesses, if a sample was used for training or not. Therefore, the closer the accuracy is to 50% the more private is the implementation. To measure the improvement in the defense, the loss of the MIA accuracy is calculated. Therefore, the 50% mark is used as the baseline to measure this improvement. For example, as shown, against the nonprivate approach the MIA has a accuracy of 84%. An implementation which pushes this rate down to 50% would have a MIA loss of 100% as now the MIA does not function anymore. The accuracy of the MIA is plotted against the accuracy of the (ϵ,δ)-differentially private implementation to present the relation between utility of the classifier and privacy.

Chapter 4

Results

This chapter presents the results of the carried out work. This chapter is divided into three parts, the first part evaluates the SVR with output perturbation. Afterwards, the SVR with objective perturbation is evaluated. Finally, Section 4.3 presents some results of a differentially private SVM which was investigated in the scope of this work.

Table 4.1 shows the accuracy results for the work presented by Nisar et al. 4.1. As pointed out in chapter 3.2, these results are used to compare the accuracy results of the differential privacy algorithms.

Table 4.1: Results of the work by Nisar et al. [10], which are used to compare the result of this.

Average F1	Accuracy	Weighted F1
63.65	63.64	63.68

4.1 SVR with Output Perturbation

The experimental results of the SVR with output perturbation can be seen in Table 4.2. Accuracy change and MIA change are regarding the reference approach.

Table 4.2: Experimental results for SVR with output perturbation. With $\epsilon = 2.0$ the attack nearly completely fails while the accuracy loss with -27.77% is moderate. $\epsilon = 4.0$ results in acceptable accuracy loss with -12.49% while still having good protection. $\epsilon = 1.0$ has not acceptable accuracy loss and $\epsilon = 8.0$ offers to less protection.

ϵ	Accuracy	Average F1	Weighted F1	MIA	Accuracy change	MIA change
0.5	26.14	23.48	23.63	54.24	-58.93%	-88.53%
1.0	35.74	32.27	32.14	53.29	-43.88%	-90.32%
2.0	45.97	42.41	42.12	53.29	-27.77%	-90.32%
4.0	55.78	52.55	51.77	59.91	-12.35%	-70.85%
8.0	56.87	54.25	53.7	78.46	-10.64%	-16.29%

Table 4.4 shows the standard error for accuracy and MIA accuracy from Table 4.2.

Table 4.3: Standard Errors for accuracy and MIA accuracy from Table 4.2.

ϵ	Accuracy Standard Error	MIA Standard Error
0.5	0.44	1.09
1.0	0.39	0.61
2.0	0.35	0.85
4.0	0.38	1.30
8.0	0.2	0.85

For $\epsilon = 0.5$ the accuracy is low with 26.14%. Compared to the reference approach, the accuracy is more than halved with losing -58.93% of its accuracy. Even though $\epsilon = 0.5$ shows less protection against MIA than $\epsilon = 1.0$ and $\epsilon = 2.0$, it is in the range of the standard error as shown in Table 4.3. Interestingly, $\epsilon = 1.0$ and $\epsilon = 2.0$ provide on average the same protection against the tested MIA. However, they have different standard errors. In both cases, the attack is close to having an accuracy of 50% which means it is not successful at all. However, if accuracy of the ADL recognition is compared, there is a large difference between $\epsilon = 1.0$ and $\epsilon = 2.0$. In case $\epsilon = 1.0$ the classifier has only an accuracy of 35.74% which an loss of -43.83% compared to the nonprivate SVR. In this case the approach loses nearly half of its accuracy. In contrast, $\epsilon = 2.0$ provides much better utility as the accuracy loss is only -27.77% to an accuracy of 45.97%. However, this translates to classifying more than half of the samples falsely.
To achieve good accuracy results, ϵ has to be set to 4.0. In this case, the accuracy drops only by -12.49% to a total accuracy of 55.78%. This is an acceptable loss of accuracy. On the other hand, $\epsilon = 4.0$ still offers a good protection against the MIA. With an accuracy of 59.91% most of the training samples are protected against this attack. If ϵ is increased to 8.0 the accuracy drop-off is not much reduced in comparison to $\epsilon = 4.0$, as the accuracy loss is then -10.64% or a total accuracy of 56.87%. However, this small increase in accuracy comes with a large penalty regarding the protection against the MIA. For $\epsilon = 8.0$ MIA nearly achieves the same accuracy as in the nonprivate approach. With a MIA accuracy of 78.46% it is merely below the MIA accuracy of the nonprivate approach, which sits at 84%. It can be said that for $\epsilon = 8.0$ the SVR with output perturbation offers nearly no protection against a MIA. Good protection against MIA is given for $\epsilon \leq 4.0$.

Comparing the F1 scores of the SVR with output perturbation with the F1 scores of the nonprivate approach shows that they are not as close to their corresponding accuracy. While the average F1 score and weighted F1 score are only diverging in the second decimal place for the nonprivate approach, in this approach with output perturbation they diverge quite much. Compared to the accuracy in Table 4.2 average F1 score is on average 3.41 lower and weighted F1 score 3.78. In contrast to the nonprivate approach, in this experiment, the average F1 score is slightly higher than the nonprivate approach.

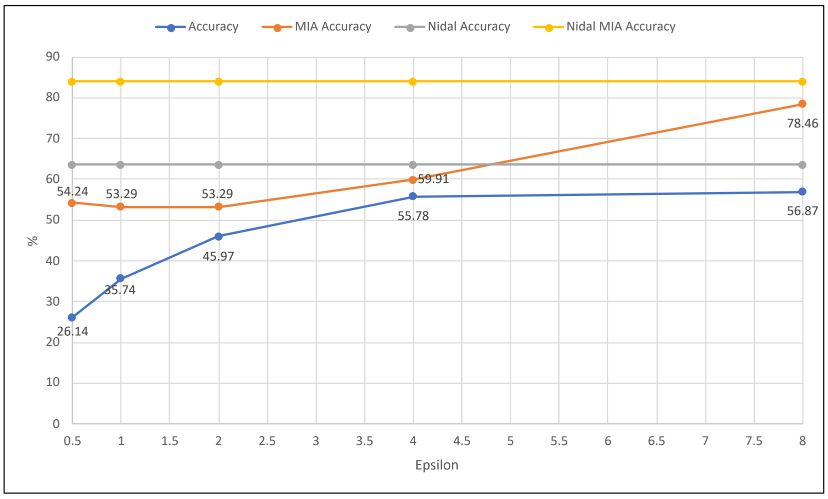

Figure 4.1: Self-calibrated SVM accuracy (blue) and the MIA accuracy (orange) against it. For $\epsilon \leq 4.0$ the SVR with output perturbation provides reasonable protection against the MIA, if $\epsilon = 8.0$ the MIA performs nearly as good as against the reference implementation. The accuracy increases with increasing ϵ. For $\epsilon = 4.0$ the SVR with output perturbation provides reasonable protection against the MIA and accuracy of the SVM.

Figure 4.1 plots the accuracy of the SVM against the MIA accuracy for the evaluated ϵ. The plotting shows that an increased ϵ results in better accuracy but also in an increased accuracy of the MIA. However, for $\epsilon \leq 4.0$ the MIA has a accuracy $< 60\%$ which is a good defense against the investigated attack. Furthermore, the plotting shows really well that with an increased accuracy and MIA accuracy are approaching the nonprivate reference values.

In Figure 4.2 the accuracy and MIA accuracy for the SVR with output perturbation are plotted regarding their drop-off to the reference approach. The plotting shows really well that for increasing ϵ the accuracy increases and once increasing ϵ does not increase the accuracy anymore, the MIA accuracy increases heavily.

Figure 4.3 plots the boxplots of each evaluated 4.2. It shows a steady increase of accuracy for increasing ϵ. Furthermore, the distribution of the values is close to each average value as each boxplot has a small IQR and the whiskers are not far off from the average value. With increasing ϵ the IQR slightly decreases which is expected as an increased ϵ means less noise and therefore the range of possible results should decrease. Interestingly, $\epsilon = 4.0$ has an outlier with an accuracy of 60.05% which has an accuracy better than any accuracy $\epsilon = 8.0$ achieves. In addition, $\epsilon = 4.0$ has also an outlier on the negative side with an accuracy of 51.67%. Furthermore, the upper whisker for $\epsilon = 4.0$ is higher than the upper whisker for $\epsilon = 8.0$. All other boxplots do not show show a fluctuation which is interesting to note. However, the IQR for $\epsilon = 4.0$ is small, which hints that the calculated noise is on average in the same range. Furthermore, the upper quartile is below the average of $\epsilon = 8.0$ which decreases the impact of these

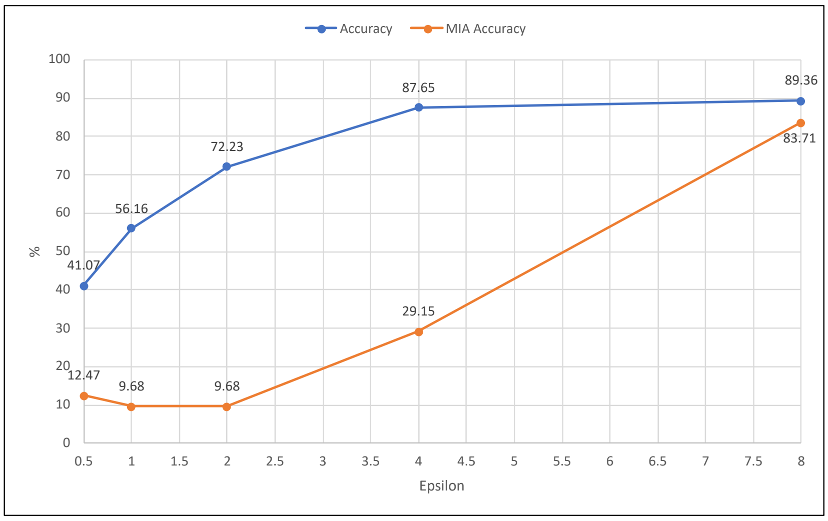

Figure 4.2: Accuracy (blue) and MIA accuracy (orange) plotted in regard to the reference implementation. For $\epsilon = 4.0$ this implementation achieves 87.65% of the reference accuracy while the MIA accuracy is only 29.15% of the reference.

outliers on the evaluation.

Interestingly, for $\epsilon = 1.0$ the lower whisker is much closer to the average value than the upper whisker. This is a good sign, as $\epsilon = 1.0$ does not provide any far-off outlier to the negative. For $\epsilon = 2.0$ the boxplot looks like expected with an similar IQR and distances to the outlier as $\epsilon = 1.0$ shows. Clearly, the accuracy is much higher than for $\epsilon = 1.0$.

Figure 4.4 shows the corresponding average F1 score (blue) and weighted F1 score (orange) for the SVR with output perturbation. The F1 scores for the nonprivate implementation are shown in grey and yellow. The grey line is hardly visible as it is covered by the yellow line since the average F1 score and weighted F1 score in the reference approach only diverge in the second decimal place.
F1 scores for the SVR implementation are steadily increasing with an increasing ϵ. Similar to the accuracy in Figure 4.1 the slope decreases with increasing ϵ but it is more steep in the increase from $\epsilon = 4.0$ to $\epsilon = 8.0$. In contrast to the accuracy plotting in Figure 4.1 the distance between reference plotting and plotting of this implementation is larger. As already mentioned, the F1 scores for this implementation are not as close to the accuracy as in the reference implementation. Therefore, the distance to the reference F1 scores is larger even if the accuracy plot in Figure 4.1 approaches its nonprivate counterpart.

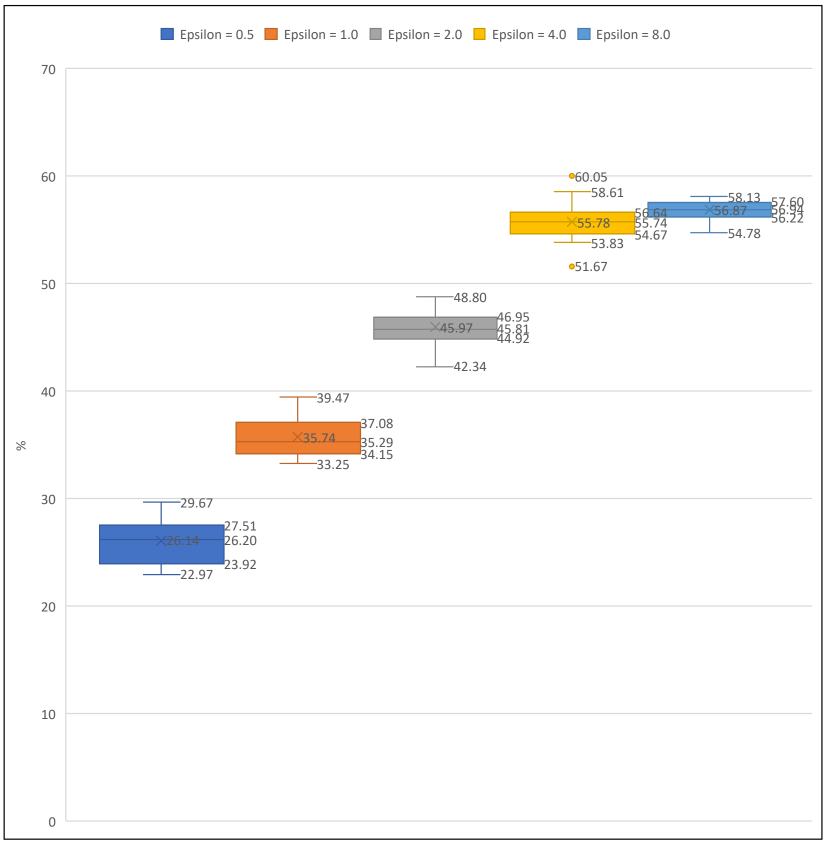

Figure 4.3: Boxplots for the SVR with output perturbation. With increasing ϵ the whisker are getting closer to the average value. IQR is decreasing slightly with increasing ϵ.

4.2 SVR with Objective Perturbation

Conclusively, the SVR with objective perturbation is evaluated. Table 4.4 shows the results. The corresponding standard errors are shown in Table 4.5.

The presented results are similar to the results of the output perturbation. With increasing ϵ the accuracy starts to increase with the downside that the accuracy of the MIA increases. Interestingly, the increase in the accuracy for $\epsilon = 0.5$ to $\epsilon = 2.0$ is nearly linear. However, for $\epsilon = 0.5$ the accuracy is more than halved with a loss of -57.73% compared to the reference approach for not much more protection than higher a ϵ. There is one interesting anomaly which is $\epsilon = 2.0$. If the results are compared to the results of $\epsilon = 1.0$ the accuracy and F1 scores show the expected behavior as they increase. However, the accuracy of the MIA slightly decreases on average. While the MIA has a accuracy of 55.75% for $\epsilon = 1.0$, it is only 54.5% for $\epsilon = 2.0$. This behavior is unexpected as normally higher ϵ means less privacy and therefore the accuracy should increase as well. However, the difference is relatively small and in the range of the standard error, which is a plausible explanation for this. If ϵ is set to 4.0, SVR with objective perturbation performs best in the trade-off between accuracy and protection. The accuracy is good enough with 55.96% which is only an loss of -12.07%. This loss

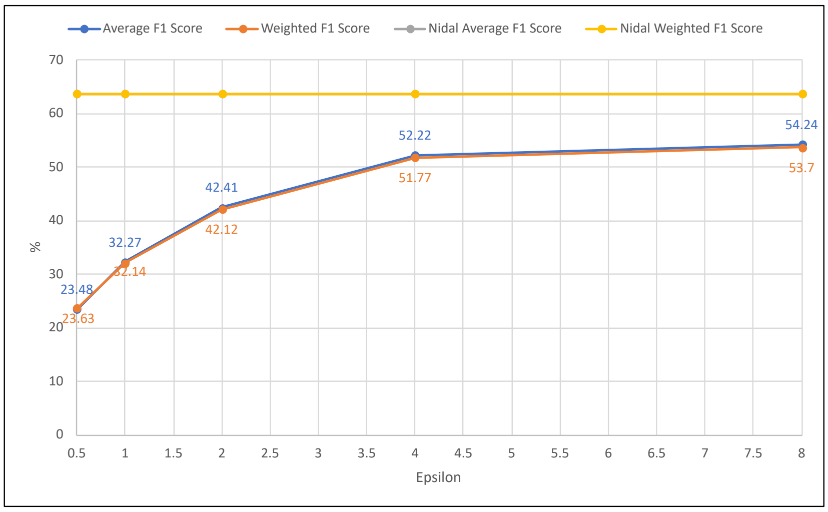

Figure 4.4: Average F1 score (blue) and weighted F1 score (orange). Both are close to each other. The F1 scores (grey and yellow) of the implementation from Nisar et al. [10] are plotted as an reference. Average F1 score from Nisar is not visible as it is covered by the weighted F1 score.

Table 4.4: Experimental results for SVR with objective perturbation. With increasing ϵ the accuracy increases while MIA accuracy increases as well.

ϵ	Accuracy	Average F1	Weighted F1	MIA	Accuracy change	MIA change
0.5	26.9	23.99	24.01	54	-57.73%	-88.24%
1.0	33.55	32.55	32.41	55.75	-47.28%	-83.09%
2.0	46.32	42.87	42.59	54.5	-27.22%	-86.76%
4.0	55.96	52.81	52.34	59.71	-12.07%	-71.44%
8.0	56.93	54.12	53.59	79.92	-10.54%	-12%

is totally acceptable. On the other hand, the accuracy is decreased by -71.44% which results in a accuracy of 59.71%. Choosing $\epsilon = 4.0$ provides a sufficient defense as an attacker cannot distinguish between training samples and testing samples. In contrast, $\epsilon = 8.0$ provides only 12% more protection compared to the nonprivate implementation while carrying an accuracy loss of -10.54%. This is similar to what was shown for the SVR with output perturbation. Increasing ϵ from 4.0 to 8.0 does not provide much more accuracy as it only increases from 55.96% to 56.93% but reduces the protection of the implementation drastically.
F1 scores show similar behavior as in the SVR with output perturbation. The average F1 score is slightly higher than the weighted F1 score. The difference between the average F1 score and weighted F1 score is slightly increasing as ϵ increases. Differently to the nonprivate reference implementation, the F1 scores for the SVR with objective perturbation are much more different to the accuracy. When the nonprivate F1 scores only diverge in the second decimal place, the F1 scores for the SVR with objective perturbation can be as much as 3.62 percentage points lower.

Table 4.5: Standard Errors for accuracy and MIA accuracy from Table 4.4.

ϵ	Accuracy Standard Error	MIA Standard Error
0.5	0.46	1.13
1.0	0.31	0.9
2.0	0.49	1.06
4.0	0.31	1.1
8.0	0.25	0.93

Figure 4.5 plots the accuracy and MIA accuracy from Table 4.4 against the reference nonprivate implementation. Accuracy is plotted in blue and the reference accuracy in grey. MIA accuracy for this implementation is plotted in orange and the corresponding accuracy for the nonprivate implementation in grey. For increasing ϵ the accuracy closes the gap to the nonprivate implementation. If ϵ is set to 4.0 or 8.0 the plot is really close to the nonprivate implementation.
Looking at the accuracy in orange the dip for $\epsilon = 2.0$ can be seen. However, the plotting shows that this dip is really small. If ϵ is increased beyond 4.0, the accuracy has a steep ascent towards the nonprivate reference. For $\epsilon = 8.0$ the gap is nearly closed.

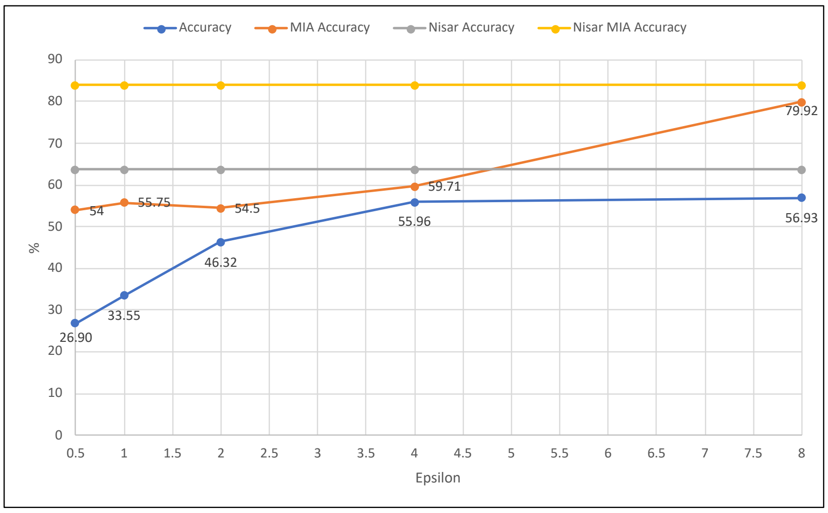

Figure 4.5: Self-calibrated SVM accuracy (blue) and the MIA accuracy (orange) against it. For $\epsilon \leq 4.0$ the SVR with objective perturbation provides reasonable protection against the MIA, if $\epsilon = 8.0$ the MIA performs nearly as good as against the reference implementation. The accuracy increases with increasing ϵ. For $\epsilon = 4.0$ the SVR with objective perturbation provides reasonable protection against the MIA and accuracy of the SVM.

Figure 4.6 plots the accuracy (blue) and the MIA accuracy (orange) regarding the reference approach, which is shown in Table 4.4 in the last two columns. The plotting shows very well that $\epsilon = 4.0$ is an excellent trade-off between accuracy and protection

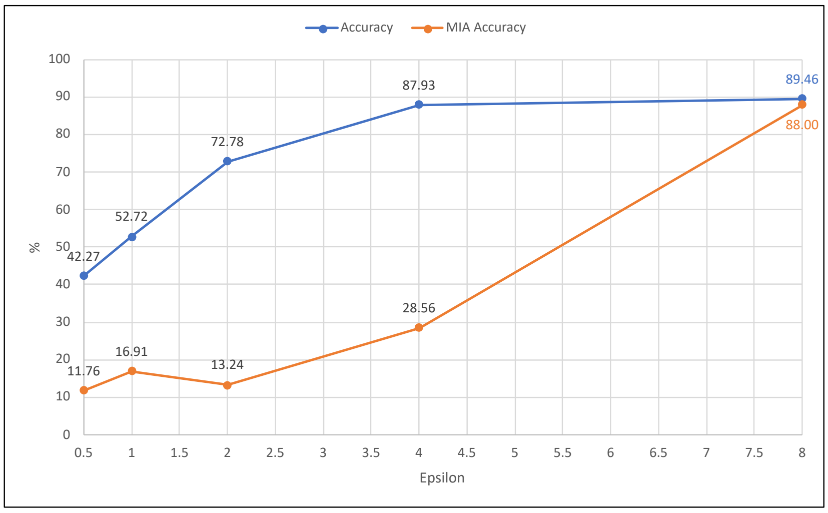

Figure 4.6: Accuracy (blue) and MIA accuracy (orange) plotted in regard to the reference implementation. For $\epsilon = 4.0$ this implementation achieves 87.93% of the reference accuracy while the MIA accuracy is only 28.56% of the reference.

against the MIA. Furthermore, the plots show well that increasing ϵ to 8.0 carries a large loss in protection while providing nearly no better accuracy.

In Figure 4.7 the boxplots for each evaluated ϵ in the SVR with objective perturbation are displayed. In contrast to the boxplots in Figure 4.3 the distance between the lower and upper whisker is slightly less. Especially $\epsilon = 1.0$ shows very small fluctuation. The distance from the average to the upper and lower whisker and as well to the first and third quartile is nearly the same. Therefore, the boxplot is nearly mirrored at the median line. For $\epsilon = 2.0$ it is slightly different. While the distances from the average to the first and third quartile are nearly identically, the lower whisker at 42.82% has a greater distance to the average at 46.32% than the upper whisker at 49.28%. Furthermore, there is one negative outlier at 40.43%. The boxplot $\epsilon = 0.5$ shows that choosing ϵ that small for the SVR in rank pooling is not a viable option for the accuracy. It is low and the upper whisker is far off the median and there is one negative outlier with 21.53$ which is not much better than just randomly selecting an output. At least the lower whisker is much more close to the average. The boxplots for $\epsilon = 4.0$ and $\epsilon = 8.0$ are quite similar to each other. Most parts of the quartiles overlap each other. This demonstrates really well that increasing ϵ to 8.0 does not increase the accuracy much.
Similar to the SVR with output perturbation, the IQR decreases with increasing ϵ. While the IQR is 4.31 for $\epsilon = 1.0$ it decreases to 1.79 for $\epsilon = 8.0$.

Next, the F1 scores for the SVR with objective perturbation are plotted in Figure 4.8 against the reference F1 scores. Average F1 score is plotted in blue, its counterpart in grey, which is covered by the reference weighted F1 score in yellow. Weighted F1 score for this implementation is plotted in orange. The plotting shows that the F1 scores increase for increasing ϵ. However, even for $\epsilon = 8.0$ the distance to the reference

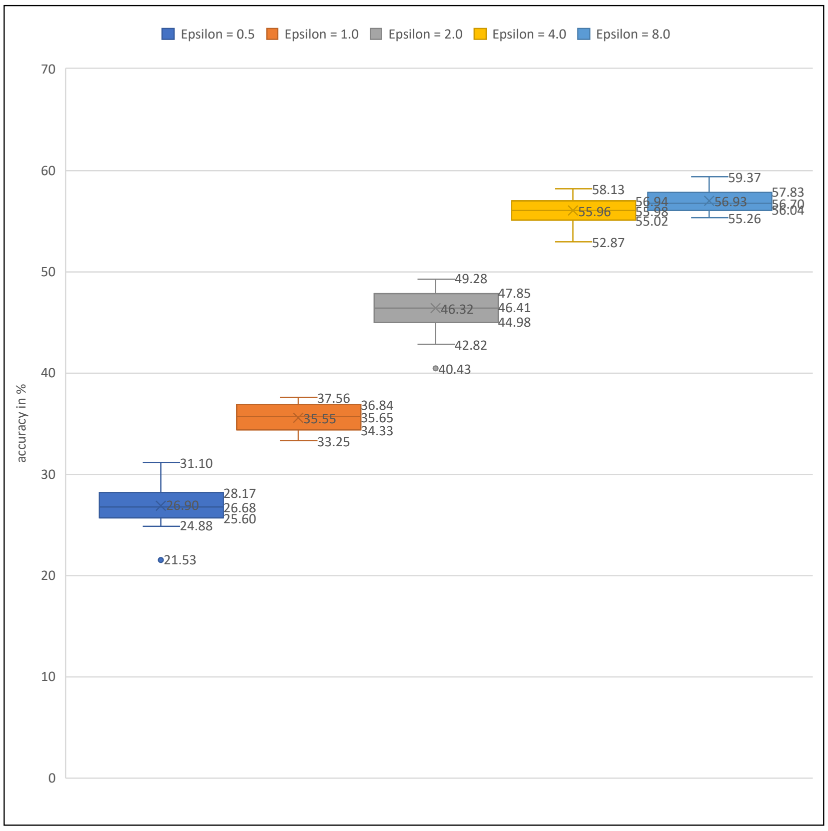

Figure 4.7: Boxplots for the SVR with objective perturbation. With increasing ϵ the IQR and the distance from lower to upper whisker is decreasing. The plotting shows really well that there is not much difference between $\epsilon = 4.0$ and $\epsilon = 8.0$ from an accuracy's perspective.

F1 scores is still quite much.
The difference between the average F1 score and weighted F1 score for this implementation is more than in the reference implementation but they are still close to each other. The plotting in Figure 4.8 shows this as the blue and orange plots overlap each other.

4.3 Differentially Private SVM

In addition to the implementation of a differentially private SVR, this work has also explored the implementation of an differentially private SVM. During the research, it became obvious that making the SVR differentially private is not enough to protect the training data in this ADL recognition. The results shown in this section are independent from the previously shown results. They are much more technical results to show what a differentially private SVM can produce. However, the results of these expirements might be helpful for different scenarios. SVMs are still commonly used in different areas and medical data is no exception. In the medical field, there is of-

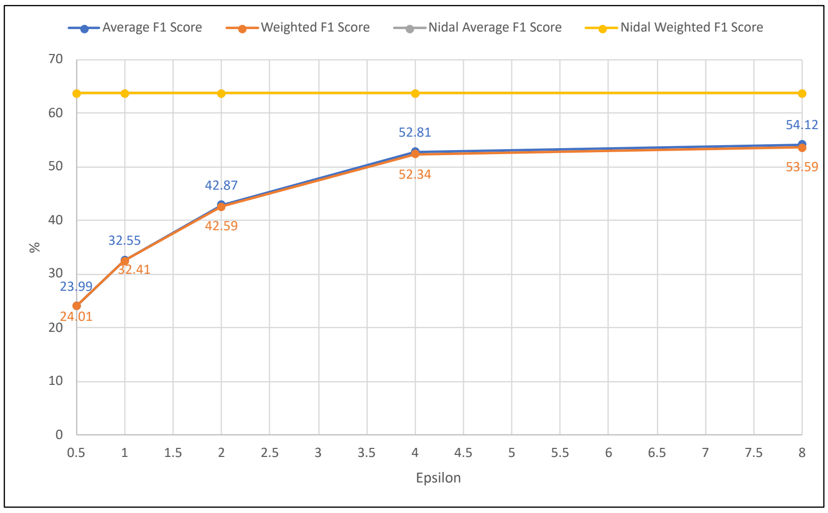

Figure 4.8: F1 scores for the SVR with objective perturbation. Both scores are close to each other. F1 scores for objective perturbation are not as close as The F1 scores (grey and yellow) of the implementation from Nisar et al. [10] are plotted as an reference. Average F1 score from Nisar is not visible as it is covered by the weighted F1 score.

ten not much data which makes the usage of deep learning based algorithms difficult. However, SVM can often produce good results even if there is not much training data. This makes SVMs still interesting for new research in the medical field.

The first algorithm, which is evaluated, is the output perturbation. Table 4.2 shows the results. For $\epsilon = 0.5$ the SVM with output perturbation the accuracy is not much better than just randomly guessing an outcome.

Table 4.6: Results for SVM with output perturbation. ϵ has to ≥ 4.0 to achieve good accuracy. In that case MIA accuracy is not decreased by much.

ϵ	Accuracy	Average F1	Weighted F1	MIA Accuracy	Accuracy change
0.5	20.69	18.85	18.8	49.97	-67.49%
1.0	27.27	25.73	25.63	47	-57.15%
2.0	41.39	38.95	38.71	49.12	-34.96%
4.0	52.96	50.56	50.25	54.67	-16.78%
8.0	56.84	54.1	53.69	58.1	-10.69%

The accuracy of this (ϵ,δ)-differential privacy SVM is plotted against the accuracy of the MIA. The MIA by Salem et al. [8] does not work anymore for $\epsilon \leq 2.0$. If $\epsilon = 1.0$ the SVM has only an accuracy of 27.27%, which is only twice as good as the randomly guessing of a label. Therefore, if $\epsilon = 1.0$ the classifier has lost most of its utility and is not suitable for classification. However, against the MIA by Salem et al. [8], it is reasonable to choose the greatest ϵ in which the attack does not work. Out of the evaluated ϵ this is the case for $\epsilon = 2.0$. In that case, the accuracy is 41.39% which is way better than the accuracy for $\epsilon = 2.0$. If ϵ is chosen even higher, the accuracy of the SVM is slowly approaching the accuracy of the nonprivate SVM. The accuracy is 52.96% for $\epsilon = 4.0$ and 56.84% for $\epsilon = 8.0$. However, the increase of the accuracy is slowly decreasing, the higher ϵ is chosen. This is expected, as the output perturbation should approach the nonprivate version for high ϵ which is happening as the difference for $\epsilon = 8.0$ is with 56.84% to the 63.64% of the nonprivate SVM reasonable close. Compared to the SVM results from Chaudhuri et al. [38] this experiments perform worse in the accuracy, however it has to be not in her work only a binary SVM was evaluated and the used dataset was easier to classify. It has to be noted that in this work the presented (ϵ,δ)-differential privacy SVM implementations will not exactly achieve the same optimum as the nonprivate SVM implementation in the libsvm. This is due to the use of the Huber loss which is an approximation of the hinge loss.

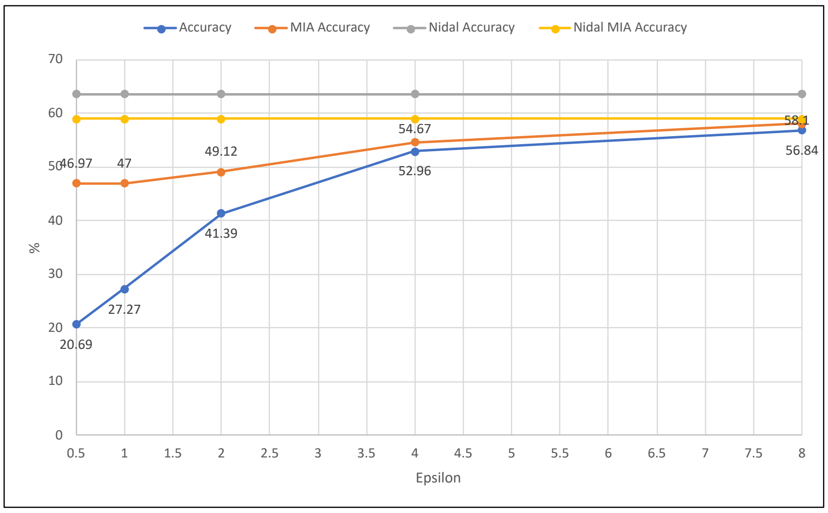

Figure 4.9: MIA attack does not work for $\epsilon \leq 2.0$. In $\epsilon \geq 4.0$ the accuracy approaches the nonprivate accuracy, however the MIA works again. In that case, SVM accuracy and MIA accuracy increase with same slope.

The price of an accuracy close to the nonprivate SVM is the loss in protection. The MIA attack starts to work again for $\epsilon = 4.0$ and $\epsilon = 8.0$. Interestingly, the increase of the classifier accuracy and MIA accuracy from $\epsilon = 4.0$ to $\epsilon = 8.0$ is a nearly parallel line. If $\epsilon = 8.0$ the MIA accuracy is with 58.1% nearly identical to the 59% of the nonprivate SVM.

Figure 4.10 plots the boxplot for the SVM with output perturbation for each evaluated ϵ. It can be seen that the range of the achieved accuracy decreases if ϵ increases as the distance between lower and upper whisker decreases with an increasing ϵ. This is the behavior which is expected as an increased ϵ decreases the amount of noise. Less noise should result in less fluctuations, which is the case for the SVM with output perturbation. In the case $\epsilon = 1.0$ the lowest accuracy is 17.46% and the highest accuracy is 35.41%. This means that the highest accuracy is twice the minimum. This

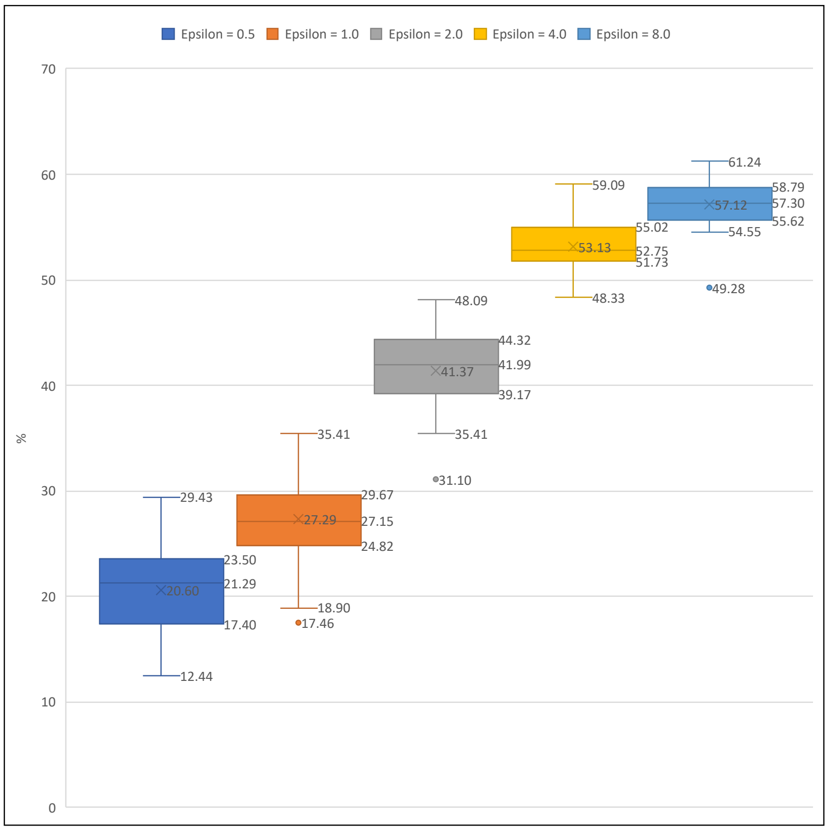

Figure 4.10: Boxplots for SVM with output perturbation. As ϵ increases the distance between the whiskers and the IQR decreases. If $\epsilon = 8.0$ the upper whisker is close to the nonprivate SVM.

negative outlier is close to the first quartile of $\epsilon = 0.5$, which hints a high fluctuations. Even if the outliers are not taken into consideration, the upper whisker is nearly twice the lower whisker. Moreover, $\epsilon = 1.0$ has its whiskers furthest away from the median compared to the other ϵ. Interestingly, if a boxplot has outliers, they lie below the lower whisker. No boxplot has an outlier above the upper whisker. One anomaly is the outlier in accuracy with 49.28% in case $\epsilon = 8.0$. Normally, you would not expect such far-off outliers for a high $\epsilon = 8.0$. This outlier has nearly the same accuracy as the lower whisker for $\epsilon = 4.0$.

Figure 4.11 shows the F1 scores of the SVM with output perturbation based on the nonprivate SVR. The graphs of F1 scores look similar to the graph of accuracy in Figure 4.9. Compared to the F1 scores in the reference approach by Nisar et al. [10], it can be seen that the F1 scores for the output perturbation are slightly below the accuracy of the SVM. Here, on average, the F1 scores are 1 percentage point below the accuracy. In contrast, the reference approach only has a deviation in the second decimal place. However, the F1 scores of the SVM with output perturbation are not so different as to be of concern. The reference and this implementation share that the weighted F1 score is slightly below the average F1 score, which is due to the not equal distribution of the classes in the training set. However, in this implementation this

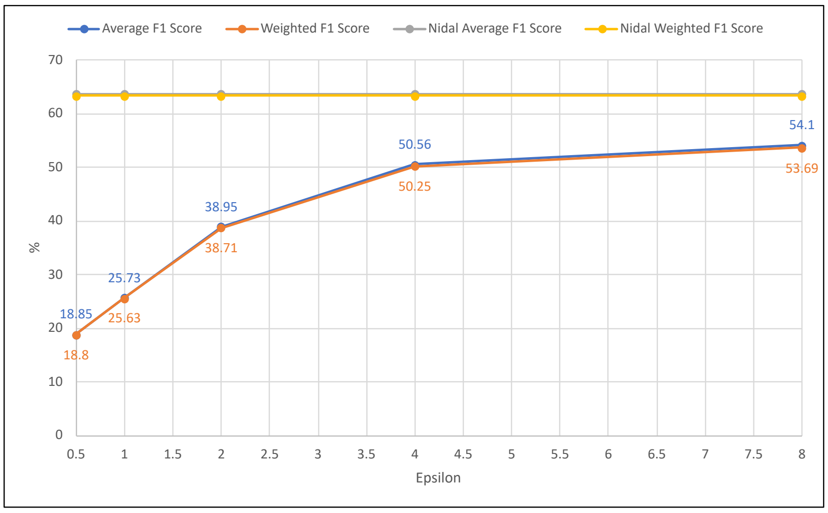

Figure 4.11: Weighted 1 score for the SVM with output perturbation (orange) and average F1 score (blue). They show similar scores which at most differ 0.41 percentage points. Both plots have a similar shape as the accuracy plot in Figure 4.9.

deviation is slightly higher than in the reference approach.

Next, the MBSGD is evaluated. Table 4.7 shows the results of the experiments.

Table 4.7: Results for MBSGD SVM. It performs for small ϵ well, but does not approach nonprivate reference accuracy.

ϵ	Accuracy	Average F1	Weighted F1	MIA Accuracy	Accuracy change
0.5	32.69	31.39	31.29	48.13	-48.63%
1.0	36.68	35.09	43.99	49.59	-42.36%
2.0	41.94	40.51	40.37	54.57	-34.1%
4.0	44.26	42.71	42.55	56.61	-30.45%
8.0	46.32	44.42	44.25	61.58	-27.22%

Figure 4.12 shows the accuracy of this approach and the corresponding MIA accuracy. Compared to the previous evaluation of the output perturbation, this algorithm performs better accuracy-wise for $\epsilon \leq 2.0$ but worse for $\epsilon \geq 2.0$. The MBSGD has an accuracy of 36.68% for $\epsilon = 1.0$ and an accuracy of 41.94%. Especially, the accuracy for $\epsilon = 1.0$ is way better than the accuracy of the SVM with output perturbation. However, the MBSGD accuracy does not really increase if ϵ is increased beyond 2.0. The accuracy only increases from 41.94% if $\epsilon = 2.0$ to 46.32& if $\epsilon = 8.0$. Graphically spoken, this increase is very flat in Figure 4.12. Compared to the graph of SVM with output perturbation in Figure 4.9, the increase there also flattens with an increasing ϵ but not as much as in Figure 4.12. In contrast to the output perturbation, the MIA starts to work again if $\epsilon = 2.0$ in the MBSGD. Therefore, if a full protection against this MIA is wanted, ϵ has to be chosen < 2.0. Similar to the output perturbation,

the MIA accuracy increases with the same slope as the SVM accuracy. However, the slope from $\epsilon = 4.0$ to $\epsilon = 8.0$ is a little more steep. Interestingly, if $\epsilon = 8.0$ MIA slightly performs better against the MBSGD than it does against the nonprivate SVM implementation.

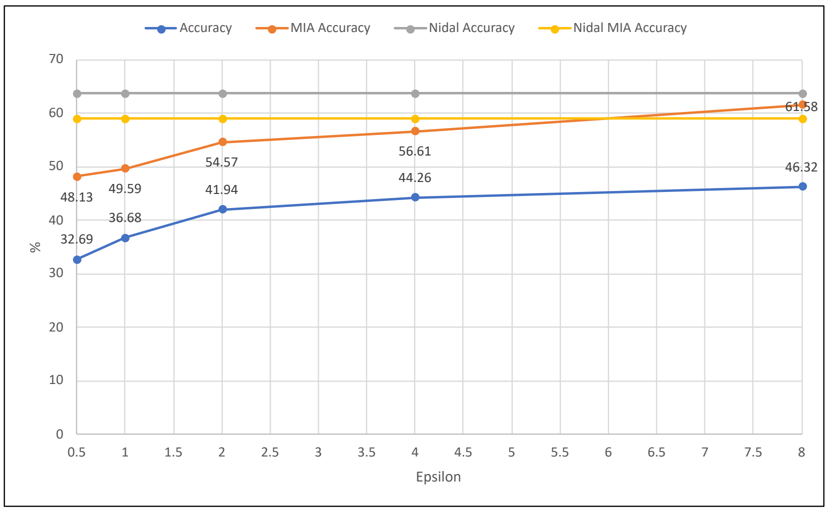

Figure 4.12: Accuracy of MBSGD (blue) and the MIA accuracy (orange). The MIA attack works for $\epsilon \geq 2.0$ while the accuracy of the MBSGD does not really increase. MBSGD with $\epsilon = 8.0$ performs worse against the MIA than the nonprivate SVM.

Figure 4.13 shows that the accuracy results show a large range of the upper and lower whisker. This range, in contrast to the output perturbation, does not shrink significantly if ϵ increases. If only the IQR is looked at, this shrinks by a small amount if ϵ increases. However, if the IQR of each ϵ is compared to its equivalent in the output perturbation, it is significantly greater. In addition, the IQR shrinks only for a small amount if ϵ increases. Furthermore, interesting is the fact that the upper quartiles for $\epsilon = 2.0$ and $\epsilon = 4.0$ are close to each other at 46.53% and 47.13% respectively. Both boxplots share this also for their upper whisker is close to each other at 52.15. What really stands out is the fact that the lower whisker for $\epsilon = 2.0$ is below the whisker of $\epsilon = 1.0$. In contrast, $\epsilon = 4.0$ and $\epsilon = 8.0$ do not have this far off lower whisker.

Figure 4.14 shows the weighted F1 score and the average F1 score for the MBSGD. The difference between the weighted F1 score and average f1 score is really small. The difference is at most 0.14%. Since the classes in the testing set are mostly equally split with a distribution from 13.9% to 15.3% this difference between the weighted F1 score and the average F1 score is expected.

Experiments with SVM with objective perturbation have shown that scaling ϵ with $\frac{1}{10}$ to satisfy Theorem 2.3.2 in the OvO classification is not tight enough as a boundary. The accuracy of this SVM implementation breaks down completely and shows no utility at all.

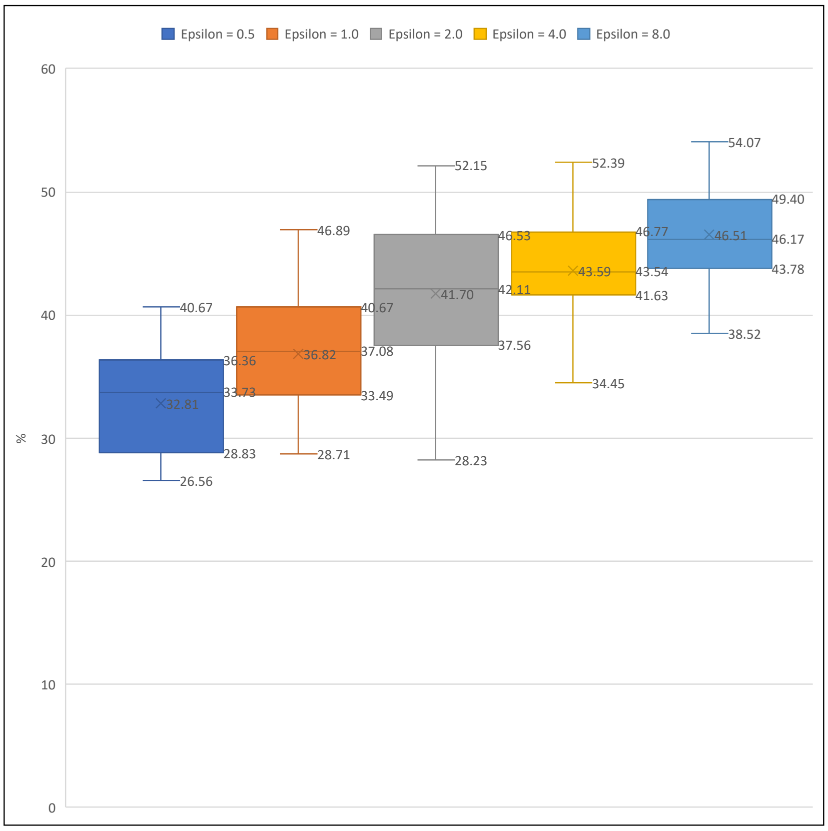

Figure 4.13: Boxplots for the MBSGD. Each boxplot has a great distance between its whiskers. IQR does not decrease much if ϵ increases.

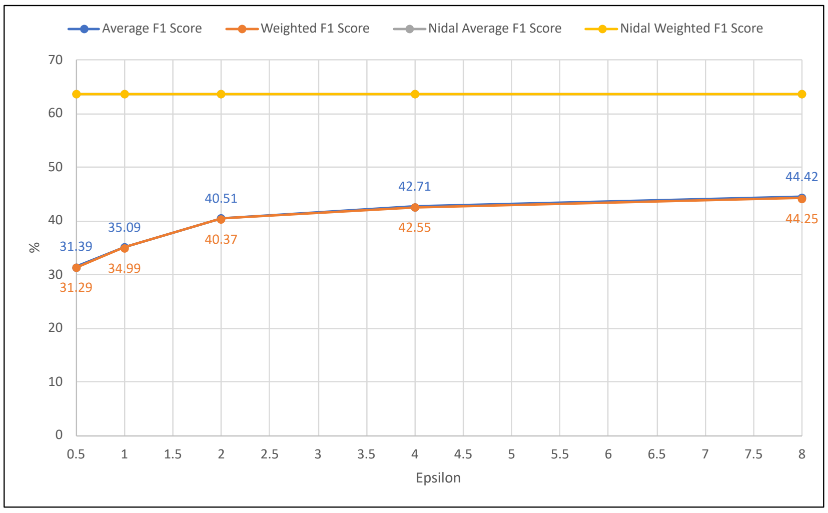

Figure 4.14: Average F1 score (blue) and Weighted F1 score (orange)for the MBSGD. They have nearly identical values, the most difference is 0.17 percentage points if $\epsilon = 8.0$. Both scores have a similar graph to the accuracy graph in Figure 4.12.

Chapter 5

Discussion

This work has shown that rank pooling is vulnerable to MIA. To use rank pooling in open scenarios, it has to be protected against such attacks. This work has evaluated two algorithms which provide privacy. However, using these algorithms requires to make a trade-off between accuracy and protection. Therefore, one goal of this work was to evaluate if it is possible to find a trade-off between accuracy and protection. Hence, the generated noise should be as less as possible while still being enough to mask the used data.

Firstly, both evaluated setups, SVR with the output perturbation and SVR with the objective perturbation, have shown that they can provide so much protection against the MIA that it nearly does not succeed at all. However, this protection comes with the cost that the accuracy breaks down to its half. Interestingly, the output perturbation performs slightly better on average, but the objective perturbation is more stable in its results. As the boxplot in Figure 4.7 shows, the majority of the results for the objective perturbation are close to the average. SVR with output perturbation should perform slightly perform worse than the objective perturbation as the objective perturbation is the more advanced algorithm. However, the output perturbation performs slightly better for on average $\epsilon = 1.0$. Even though the difference in the median is nearly negligible with 35.74% to 35.55%, output perturbation has the single better results as its upper whisker is at 39.47% compared to 37.56%. Objective perturbation is a bit more stable as the IQR and the distance between the whiskers is smaller. However, since these algorithms rely on randomness, choosing the algorithm which is more stable in its results is preferable. Even though the output perturbation has shown a higher upper whisker, this does not automatically guarantee that it cannot happen for the lower whisker.
Both algorithms have shown interesting results for $\epsilon = 2.0$. While the output perturbation shows the same protection as for $\epsilon = 1.0$, the objective perturbation performs even slightly better.
The results have shown that both algorithms can provide a good trade-off between accuracy and protection. For $\epsilon = 4.0$ the objective perturbation has an accuracy loss of -12.07% which fully acceptable. On the other side, the MIA accuracy decreases by -71.44% which makes the attack close to being nonefficient. More protection can be achieved by decreasing ϵ to 2.0. In that case, the MIA accuracy is decreased by -86.76% for the objective perturbation and -90.32%. However, this comes with accuracy drop of -27.22% and -27.77% respectively. This loss is too much in respect

to the protection gain. If more protection is wanted than what $\epsilon = 4.0$ provides it is recommended to set $\epsilon = 2.0 < \epsilon \leq 4.0$.
The objective perturbation has performed on average slightly better than the output perturbation in the performed experiments. This proves the theoretical approach that the objective perturbation is the more advanced algorithm and should therefore result in better results.

Finally, this work also evaluated implementing the output perturbation and objective perturbation for an SVM. Even though these results are not directly in the scope of this work, they show some interesting findings. Firstly, in contrast to the SVR, the objective perturbation does not produce any acceptable results. This is probably caused by choosing the scaling of $\epsilon = \epsilon/10$ to satisfy Theorem 2.3.2. It is not a tight boundary and therefore finding a tighter boundary should increase the results heavily. For both working algorithms, the output perturbation and the MBSGD, the results show a large fluctuation in the results. The boxplots have shown large distances between the whiskers. One possible cause for this is the usage of OvO. Since the noise has to be applied to each binary classifier, the chance for choosing unlucky randomness increases. In the SVR experiments, this problem is tackled by averaging over 20 runs and there is clearly only once randomness is applied. It would be interesting to see if increasing the number of iterations decreases this fluctuation. Another possible solution to fix this problem is to change to OvR training which was briefly experimented with. It was discarded since the reference implementation of an SVM by sklearn also relies on OvO training and the accuracy of the classifier was decreased by a considerable amount.
For the output perturbation using Gaussian noise heavily increased the accuracy of the SVM compared to using Laplacian Noise. If SVM with output perturbation is compared against the MBSGD, the MBSGD performs much better for small ϵ. While the SVM with output perturbation has in that case a very low accuracy, MBSGD has never an accuracy below 50% compared to the reference approach. However, this comes at the price of a large distance between the whisker and a high IQR. This leads to the conclusion that the MBSGD is not that stable for an SVM.
Finally, one note, which is out of the context of this work, but might be interesting, is that the training time for these algorithms is a problem. Neither of the evaluated algorithms takes usage of the SMO-algorithm which is heavily recommended for an efficient implementation.

Chapter 6

Conclusion and Future Work

All in All the experiments have shown that is it possible to achieve differential privacy for the recognition of ADL and the rank pooling approach. However, it comes with a cost in accuracy. This trade-off is the most difficult for someone who wants to implement differential privacy. This work has shown that you can achieve 83.73% accuracy of the nonprivate implementation while achieving a 71.44% better protection. However, this protection is only for the evaluated MIA. Differential privacy and possible attacks are still an ongoing research area and in the foreseeable future there will be probably attacks which work even better against an SVM or an SVR. On the other hand, research on differential privacy is catching up fast and future work can and should increase not only the protection but also the accuracy.

In the future, the accuracy of the classifier has to be improved. The work by Nisar et al. [10] showed that rank pooling can be used for the recognition of ADL. However, having an accuracy of at best 63.64% in a hold-out setting and 68.54% in a leave-one-subject-out cross-validation is not enough for real world usage. The approach has to generalize more to classify data from unseen subjects better. Furthermore, increasing the accuracy in the nonprivate setting will also increase the accuracy of the algorithms with differential privacy since their accuracy is depending on the nonprivate optimum. There are some prototypical approaches to increase the accuracy which look promising. For example, instead of using a holdout-setting, where half of the subjects are used for training and the other half is used for testing, the split is applied differently. In this case, each subject performs the same tasks at different times. Now, instead of splitting based on the subjects, splitting occurs based on the recording time. That means, all data from the first recording are used for training and all data from the second recording are used for testing. This increases the number of different subjects the model is trained and should benefit the generalization of the model. However, the splitting between subjects instead of recordings is a bit more close to a possible real world usage.

Another approach to increase is to change the classifier from an SVM to a classifier based on a DNN. As briefly mentioned in Section 2.2, deep learning has shown good results in the field of activity recognition. In the scope of ADL, two models are interesting: the Long short-term memory (LSTM) architecture and a convolution neural network (CNN). It is of course possible to build a CNN LSTM but in the scope of this future work, a CNN is not built in the LSTM architecture.
LSTM networks have the ability to identify temporal correlations. This makes them

intriguing for activity recognition as the sensor data for activities has temporal correlations since activities take some time. Rank pooling takes also advantage of this temporal correlation. First, experiments with LSTM are showing promising results.

Using neural networks has one disadvantage. Neural networks have a nonconvex loss function. In contrast to convex loss functions, they can have several local optimums. On the other hand, convex loss functions only have a single local optimum, which is also the global optimum. Finding the global minimum of convex loss functions is computationally appealing. Finding the global minimum of nonconvex loss functions is not. Therefore, mainly gradient descent is used to find a local minimum in nonconvex loss functions. On the first hand, this does not sound bothersome since in most cases finding a local minimum instead of the global minimum is good enough to achieve satisfying results.
However, from a security point of view using nonconvex loss functions is a problem. Guaranteeing privacy in nonconvex loss functions is much more difficult than in convex loss function. Since gradient descent can be stuck in a local minimum which it cannot escape since its separated from other optima by a mountain or a ridge, using gradient descent in nonconvex loss functions results in a large amount of leakage. Therefore, nonconvex loss functions require different algorithms which guarantee differential privacy. For example, in this work the presented ERM with output perturbation and ERM with objective perturbation require a convex loss function.

Further work should also investigate finding a better scaling for the objective perturbation. The work of Kifer et al. [53] showed promising improvements for the objective perturbation but requires difficult calculations to use the correct parameters.
Furthermore, objective perturbation requires the usage of the naive implementation of an SVM. This leads to long training times for even a small amount of data. The calculation of an SVM can be improved with the SMO-algorithm, which is currently the fastest algorithm for an SVM. Currently, there is no research on implementing the objective perturbation into the SMO-algorithm. Finding such an implementation can improve the results of the objective perturbation drastically.

Bibliography

[1] Hannah Ritchie. “Age Structure”. In: *Our World in Data* (2019). https://ourworldindata.org/age-structure.

[2] Jindong Wang et al. “Deep learning for sensor-based activity recognition: A survey”. In: *Pattern Recognition Letters* 119 (2019), pp. 3–11.

[3] Yuxi Wang, Kaishun Wu, and Lionel M Ni. “Wifall: Device-free fall detection by wireless networks”. In: *IEEE Transactions on Mobile Computing* 16.2 (2016), pp. 581–594.

[4] W Nicholson Price and I Glenn Cohen. “Privacy in the age of medical big data”. In: *Nature medicine* 25.1 (2019), pp. 37–43.

[5] Mohammad Al-Rubaie and J Morris Chang. “Privacy-preserving machine learning: Threats and solutions”. In: *IEEE Security & Privacy* 17.2 (2019), pp. 49–58.

[6] Georgios A Kaissis et al. “Secure, privacy-preserving and federated machine learning in medical imaging”. In: *Nature Machine Intelligence* 2.6 (2020), pp. 305–311.

[7] Chih-Chung Chang and Chih-Jen Lin. “LIBSVM: A library for support vector machines”. In: *ACM transactions on intelligent systems and technology (TIST)* 2.3 (2011), pp. 1–27.

[8] Ahmed Salem et al. “ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models”. In: *CoRR* abs/1806.01246 (2018). arXiv: `1806.01246`.

[9] Giuseppe Ateniese et al. *Hacking Smart Machines with Smarter Ones: How to Extract Meaningful Data from Machine Learning Classifiers.* 2013. arXiv: `1306.4447 [cs.CR]`.

[10] Muhammad Adeel Nisar et al. “Rank Pooling Approach for Wearable Sensor-Based ADLs Recognition”. In: *Sensors* 20.12 (2020), p. 3463.

[11] Julian Steil et al. “Privacy-aware eye tracking using differential privacy”. In: *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications.* 2019, pp. 1–9.

[12] Yaling Zhang, Zhifeng Hao, and Shangping Wang. “A differential privacy support vector machine classifier based on dual variable perturbation”. In: *IEEE Access* 7 (2019), pp. 98238–98251.

[13] Benjamin IP Rubinstein et al. “Learning in a large function space: Privacy-preserving mechanisms for SVM learning”. In: *arXiv preprint arXiv:0911.5708* (2009).

[14] NhatHai Phan, Xintao Wu, and Dejing Dou. “Preserving differential privacy in convolutional deep belief networks”. In: *Machine learning* 106.9 (2017), pp. 1681–1704.

[15] Christopher M Bishop. *Pattern recognition and machine learning.* springer, 2006.

[16] Vladimir N Vapnik. *The nature of statistical learning theory.* 1995.

[17] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), pp. 273–297.

[18] Christopher JC Burges. “A tutorial on support vector machines for pattern recognition”. In: *Data mining and knowledge discovery* 2.2 (1998), pp. 121–167.

[19] Olivier Chapelle, Patrick Haffner, and Vladimir N Vapnik. “Support vector machines for histogram-based image classification”. In: *IEEE transactions on Neural Networks* 10.5 (1999), pp. 1055–1064.

[20] William Stafford Noble et al. “Support vector machine applications in computational biology”. In: *Kernel methods in computational biology* 71 (2004), p. 92.

[21] John Platt et al. “Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods”. In: *Advances in large margin classifiers* 10.3 (1999), pp. 61–74.

[22] Lorenzo Rosasco et al. “Are loss functions all the same?” In: *Neural computation* 16.5 (2004), pp. 1063–1076.

[23] Léon Bottou et al. “Comparison of classifier methods: a case study in handwritten digit recognition”. In: *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3-Conference C: Signal Processing (Cat. No. 94CH3440-5)*. Vol. 2. IEEE. 1994, pp. 77–82.

[24] S. Knerr, L. Personnaz, and G. Dreyfus. “Single-layer learning revisited: a stepwise procedure for building and training a neural network”. In: *Neurocomputing*. Ed. by Françoise Fogelman Soulié and Jeanny Hérault. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 41–50. ISBN: 978-3-642-76153-9.

[25] Ulrich Kressel. “Pairwise classification and support vector machines”. In: *Advances in kernel methods: support vector learning* (1998), pp. 255–268.

[26] Chih-Wei Hsu and Chih-Jen Lin. “A comparison of methods for multiclass support vector machines”. In: *IEEE transactions on Neural Networks* 13.2 (2002), pp. 415–425.

[27] Koby Crammer and Yoram Singer. “On the algorithmic implementation of multiclass kernel-based vector machines”. In: *Journal of machine learning research* 2.Dec (2001), pp. 265–292.

[28] John Platt. “Sequential minimal optimization: A fast algorithm for training support vector machines”. In: (1998).

[29] Ting-Fan Wu, Chih-Jen Lin, and Ruby C Weng. “Probability estimates for multi-class classification by pairwise coupling”. In: *Journal of Machine Learning Research* 5.Aug (2004), pp. 975–1005.

[30] Bianca Zadrozny. “Reducing multiclass to binary by coupling probability estimates”. In: *Advances in neural information processing systems*. 2002, pp. 1041–1048.

[31] Alex J Smola and Bernhard Schölkopf. *Learning with kernels*. Vol. 4. Citeseer, 1998.

[32] Alex J Smola and Bernhard Schölkopf. “A tutorial on support vector regression”. In: *Statistics and computing* 14.3 (2004), pp. 199–222.

[33] Patrick Koch et al. “Optimization of Support Vector Regression Models for Stormwater Prediction”. In: Jan. 2010, pp. 146–160.

[34] Muhammad Shoaib et al. “Complex human activity recognition using smartphone and wrist-worn motion sensors”. In: *Sensors* 16.4 (2016), p. 426.

[35] Oscar D Lara and Miguel A Labrador. “A survey on human activity recognition using wearable sensors”. In: *IEEE communications surveys & tutorials* 15.3 (2012), pp. 1192–1209.

[36] Basura Fernando et al. “Rank Pooling for Action Recognition”. In: *CoRR* abs/1512.01848 (2015). arXiv: `1512.01848`.

[37] Christian Szegedy et al. “Intriguing properties of neural networks”. In: (2014). arXiv: `1312.6199 [cs.CV]`.

[38] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. “Differentially private empirical risk minimization.” In: *Journal of Machine Learning Research* 12.3 (2011).

[39] Cynthia Dwork et al. “Calibrating noise to sensitivity in private data analysis”. In: *Theory of cryptography conference*. Springer. 2006, pp. 265–284.

[40] Cynthia Dwork et al. “Our Data, Ourselves: Privacy Via Distributed Noise Generation”. In: *Advances in Cryptology - EUROCRYPT 2006*. Ed. by Serge Vaudenay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 486–503. ISBN: 978-3-540-34547-3.

[41] Cynthia Dwork, Aaron Roth, et al. "The algorithmic foundations of differential privacy." In: *Foundations and Trends in Theoretical Computer Science* 9.3-4 (2014), pp. 211–407.

[42] Martin Abadi et al. "Deep learning with differential privacy". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 308–318.

[43] Reza Shokri et al. "Membership inference attacks against machine learning models". In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, pp. 3–18.

[44] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[45] Martin Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[46] Christopher A Choquette Choo et al. "Label-only membership inference attacks". In: *arXiv preprint arXiv:2007.14321* (2020).

[47] Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. "Smooth ε-insensitive regression by loss symmetrization". In: *Journal of Machine Learning Research* 6.May (2005), pp. 711–741.

[48] Olivier Chapelle. "Training a support vector machine in the primal". In: *Neural computation* 19.5 (2007), pp. 1155–1178.

[49] D Byatt, ID Coope, and CJ Price. "Effect of limited precision on the BFGS quasi-Newton algorithm". In: *ANZIAM Journal* 45 (2003), pp. 283–295.

[50] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[51] Sebastian Meiser and Esfandiar Mohammadi. "Tight on budget? tight bounds for r-fold approximate differential privacy". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 247–264.

[52] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035.

[53] Daniel Kifer, Adam Smith, and Abhradeep Thakurta. "Private convex empirical risk minimization and high-dimensional regression". In: *Conference on Learning Theory*. JMLR Workshop and Conference Proceedings. 2012, pp. 25–1.