Bachelor thesis
Intelligenta System, 15 hp

Classification of physical exercises using Machine Learning

Halmstad, June 20, 2023
Rasmus Nordin & Isak Axelsson
Classification of physical exercises is an important task in many applications, particularly within health services. Innowearable AB has developed a device called Inno-X that collects data using an accelerometer and sEMG sensors. To optimize Inno-X, a Machine Learning AI must be implemented for real-time exercise classification, balancing simplicity and flexibility for maximum market impact. This enhances efficiency and accuracy in analysis. This thesis investigates how raw data from Inno-X can be used to implement a pipeline and a machine-learning AI with the purpose of classifying physical exercises in real-time. Starting from implementing a protocol for collecting data to a finished end-to-end pipeline and AI that can perform the classification, this thesis includes all the steps in between. Comparison of different machine learning algorithms and the execution of transitioning from a training environment to a real-time environment has led to the obtained result. The highest accuracy achieved in the training and real-time environment was 96.98% and 90.00%, respectively. This thesis concludes that the more complex machine-learning algorithms perform better in the training environment and the less complex algorithms perform better in the real-time environment.
2 Abstrakt

Klassificering av fysiska övningar är en viktig uppgift inom många applikationer, särskilt inom hälsovården. Innowerable AB har utvecklat en enhet som heter Inno-X som samlar in data med hjälp av en accelerometer och sEMG-sensor. För att optimala Inno-X är det nödvändigt att en maskinlärnings AI implementeras för klassificering av fysiska övningar i realtid, som balanserar enkelhet och flexibilitet för maximal marknadspåverkan. Detta förbättrar effektiviteten och noggrannheten i analysen. Denna avhandling undersöker hur rådata från Inno-X kan användas för att implementera en pipeline och en maskinlärnings AI med syfte att klassificera fysiska övningar i realtid. Från att implementera ett protokoll för insamling av data till en färdig end-to-end pipeline och AI som kan utföra klassificeringen, inkluderar denna avhandling alla stegen däremellan. Jämförelse av olika maskinlärningsalgoritmer och utförandet av övergången från träningsmiljö till realtidsmiljö har lett till det erhållna resultatet. Den högsta noggrannheten som uppnåddes i tränings- och realtidsmiljö var 96,98% respektive 90,00%. Denna avhandling drar slutsatsen att de mer komplexa maskinlärningsalgoritmerna presterar bättre i träningsmiljön och de mindre komplexa algoritmerna presterar bättre i realtidsmiljön.
## Contents

1 Abstract .................................................. 3  

2 Abstrakt .................................................. 4  

List of Figures .............................................. 7  

3 Introduction .............................................. 9  
   3.1 Goals .................................................. 9  
   3.2 Requirements ........................................ 9  

4 Background .............................................. 11  
   4.1 Inno-X .................................................. 11  
       4.1.1 Accelerometer ................................... 11  
       4.1.2 Surface Electromyography (sEMG) ............... 11  
   4.2 Machine Learning (ML) ............................... 12  
       4.2.1 Supervised Machine Learning (SL) .............. 12  
       4.2.2 Machine Learning Pipeline ...................... 13  
   4.3 Algorithms ........................................... 14  
       4.3.1 Decision Tree .................................... 15  
       4.3.2 Multilayer Perceptrons ......................... 16  
       4.3.3 Support Vector Machine ......................... 18  
       4.3.4 Ensemble .......................................... 20  

5 Method .................................................... 21  
   5.1 Collection of data .................................... 21  
   5.2 Setup and materials .................................. 22  
   5.3 Training Environment ................................. 22  
   5.4 Creation of dataset ................................... 24  
   5.5 Creation of pipeline .................................. 24  
   5.6 From training environment to real-time classification 24  
   5.7 Trial and error ....................................... 25  
   5.8 How to analyze the result ............................ 25  
   5.9 Flowchart for real-time classification .............. 25  

6 Result ..................................................... 27  
   6.1 Training Environment ................................. 27  
   6.2 Real-time environment ............................... 30  
   6.3 Pipeline ............................................... 34  

7 Discussion ............................................... 35  
   7.1 Performance ........................................... 35  
   7.2 Shortcomings and potential for improvement ........... 37  
   7.3 Comparison with existing works ...................... 37  
   7.4 Social Requirements ................................... 38  

8 Conclusion .............................................. 39
References
**List of Figures**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Inno-X</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>Pipeline</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>Decision Tree</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>Operations performed by each neuron.</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>Support Vector Machine</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>Leg extension procedure [17]</td>
<td>21</td>
</tr>
<tr>
<td>7</td>
<td>Wall Sit procedure [17]</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>Squat Jump procedure [17]</td>
<td>22</td>
</tr>
<tr>
<td>9</td>
<td>Acceleration measured by Inno-X during LE and WS exercises</td>
<td>23</td>
</tr>
<tr>
<td>10</td>
<td>Flowchart for the training environment</td>
<td>23</td>
</tr>
<tr>
<td>11</td>
<td>Flowchart of the real-time classification</td>
<td>26</td>
</tr>
<tr>
<td>12</td>
<td>Confusion matrix for SVM in training environment</td>
<td>28</td>
</tr>
<tr>
<td>13</td>
<td>Confusion matrix for DT in training environment</td>
<td>28</td>
</tr>
<tr>
<td>14</td>
<td>Confusion matrix for MLP in training environment</td>
<td>29</td>
</tr>
<tr>
<td>15</td>
<td>Confusion matrix for RF in training environment</td>
<td>29</td>
</tr>
<tr>
<td>16</td>
<td>Confusion matrix for Ensemble in training environment</td>
<td>30</td>
</tr>
<tr>
<td>17</td>
<td>Confusion matrix for SVM in real-time environment</td>
<td>31</td>
</tr>
<tr>
<td>18</td>
<td>Confusion matrix for DT in real-time environment</td>
<td>31</td>
</tr>
<tr>
<td>19</td>
<td>Confusion matrix for MLP in real-time environment</td>
<td>32</td>
</tr>
<tr>
<td>20</td>
<td>Confusion matrix for RF in real-time environment</td>
<td>32</td>
</tr>
<tr>
<td>21</td>
<td>Confusion matrix for Ensemble in real-time environment</td>
<td>33</td>
</tr>
<tr>
<td>22</td>
<td>Final version of pipeline</td>
<td>34</td>
</tr>
</tbody>
</table>
Acronyms

AI - Artificial Intelligence
DT - Decision Tree
EMG - Electromyography
GD - Gradient descent
LE - Leg Extension
MLP - Multilayer Perceptron
ML - Machine Learning
RF - Random Forest
Relu - Rectified Linear Unit
SL - Supervised Machine Learning
SJ - Squat Jump
SVM - Support Vector Machine
sEMG - Surface Electromyography
WS - Wall Squat

Keywords

Machine Learning / Maskininlärning
Supervised learning / Övervakad inlärning
Classification / Klassifikation
Artificial Intelligence / Artificiell intelligens
Classification of physical exercises / Klassifikation av fysiska övningar
Electromyography / Elektromyografi
Accelerometer / Accelerometer
3 Introduction

Dedicated professional athletes consistently aim to optimize their physical performance by pushing their bodies to their limits. However, a subtle balance exists between achieving peak physical shape and over-exerting oneself, which can cause physical harm and inhibit the body’s ability to perform. Traditionally, athletes have had to rely on subjective assessments of their physical fatigue to determine their training regimes. Nevertheless, technological advances have led to the development of new and more objective methods for evaluating physical performance.

Innowearable AB has developed a new product called Inno-X [1] for this purpose. Equipped with an accelerometer and sEMG sensor, it collects data from the user and can help map the athlete’s daily shape and determine the body’s fatigue, readiness, and recovery. For full effectiveness in the usage of Inno-X, it is necessary to implement a Machine Learning AI tuned for classifying the physical exercises1. In order to have the greatest impact on the market, the Inno-X should be both simple and flexible. Implementing a Machine Learning AI that can classify the exercises in real-time will make the product more time-efficient and accurate in its analysis.

This project aims to develop and implement a robust and efficient machine learning pipeline that accurately classifies the considered exercises based on the data collected by Inno-X. The project aims to explore how different classification algorithms perform this task with the intention of optimizing the result.

Overall, the main objective of this thesis is to enhance our comprehension of the factors that lead to a robust and efficient machine learning pipeline designed to classify physical exercises based on information from an accelerometer and sEMG sensor.

3.1 Goals

The main goal of this thesis is to investigate how the data from an accelerometer and sEMG sensor can be used to classify three different physical exercises and implement such a machine learning AI. The second objective is to investigate the factors contributing to an efficient and effective pipeline in a manner of high accuracy within a reasonable time cost. The study aims to identify the key characteristics and components that lead to a successful pipeline and to understand the potential areas for improvement in this specific project. A robust pipeline is assessed by its accuracy, while its efficiency can be measured by its speed in making predictions.

3.2 Requirements

The project connected to this thesis is focused on simplifying the utilization of Inno-X by enhancing its user-friendliness and time efficiency. According to the guidelines from Innowarable AB, the simplification of the utilization requires real-time classification of physical exercises using data collected from sensors. The sensors include an accelerometer and surface electromyography whose measurements constitute the

1Physical exercises refers to: Leg Extension (LE), Wall Squat (WS), Squat Jump (SJ)
raw data input to the pipeline. The pipeline output consists of four classes corresponding to the three exercises: LE, WS, SJ, and no exercise (NE). The classification of the exercises must stay within a reasonable time frame that does not negatively impact the user experience. Hence, a limit of 15 seconds per classification is set.

This is a 2+1 project performed in close collaboration with Joel Ekstrand. Joel’s main role is to take care of the feature extraction process in the pipeline. Hence, we refer to his thesis for more information regarding feature extraction.
4 Background

4.1 Inno-X

The device Inno-X (Fig. 1) developed by Innowearable AB is equipped with an accelerometer and sEMG sensor. It has a sampling frequency of 1000 HZ. The product aims to give insights into the user’s neuromuscular fatigue, readiness, and recovery. This is done by performing three different exercises while monitored by Inno-X [1] and then analyzing the electromyography. The three exercises include Leg Extension (LE), Wall Squat (WS), and Squat Jump (SJ). For each person using the Inno-X device, the first thing the user does is to create a baseline of when the body is a good daily shape. This baseline EMG can then be compared with the daily EMG in order to draw conclusions about the daily shape.

![Inno-X](image)

Figure 1: Inno-X

4.1.1 Accelerometer

An accelerometer measures the acceleration in relation to free fall. The sensor used in this project is a 3-axis accelerometer [2]. It can give information about the acceleration in a 3-dimensional space\(^2\). The raw data from the accelerometer is defined as an acceleration amplitude at a given time point for each axis.

4.1.2 Surface Electromyography (sEMG)

Electromyography (EMG) is a technique used to measure the neuromuscular response in the form of waves. EMG detects the electric potential produced when a nerve is stimulated by the muscle. One way to record this muscle activity is by applying electrodes on the surface above the muscle, which is called surface electromyography (sEMG) [3]. sEMG measures the potential voltage in relation to

\(^2\)x-dimension, y-dimension, z-dimension
4.2 Machine Learning (ML)

Machine learning (ML) is a subfield of artificial intelligence (AI) that aims to enable machines to learn and make decisions from data without being explicitly programmed. ML algorithms use statistical techniques to learn patterns and relationships in data and use these patterns to make predictions or classifications on new data. There are three main types of machine learning: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning involves training an algorithm using labeled examples where the desired output is known. Unsupervised learning involves finding patterns and relationships in unlabeled data with the unknown desired output. Reinforcement learning involves training an algorithm to make decisions based on feedback from its environment. A fundamental basis of machine learning is access to a large amount of data [4]. A larger training data set exposes the ML to more different scenarios, which are beneficial for high precision of the ML [5]. The more data available for training, the more accurately the ML algorithm can learn to generalize from the training data to new, unseen data. However, it’s important that the data used for training is diverse, reliable, and representative of the problem the ML intends to solve. A biased or incomplete training set can lead to poor performance and subjective decision-making [6]. The process of training the model can be compared to the process at a university course. First, during the course, you are trained on training data and learning to solve a problem. Then a test is used for validating the model, which is done by letting the model classify the validation data. If the model then passes the test, it can be deployed in the real world.

Overfitting is a common occurrence in ML. There is a risk that the model used in the ML can get overfitted, meaning that the generalization gets too tailored to the training data. The problem with this is that when testing the model in the training environment, it shows higher accuracy than it would in the deployment. This is due to its lacking ability to classify new unseen data. Therefore, it is important to have insight into the model’s accuracy on the training data and validation data [7].

ML has been widely used in the last couple of years in the field of physical exercise classification. Accelerometers and similar sensors are common in smartphones and collect data almost continuously. [8] used such data for activity recognition and achieved a recognition rate of 96.11%. In [9] a similar accuracy was achieved for activity classification using built-in sensors of smartphones. This demonstrates that one can accurately classify physical exercises using the data from these sensors and machine learning.

4.2.1 Supervised Machine Learning (SL)

Supervised machine learning (SL) is a subfield of ML and can be separated into two types of problems classification and regression. This thesis focuses only on classifi-
cation since the output will be discrete\(^3\). The main characteristic defining SL is that the training data has predefined labels. This means that all feature vectors correspond to a labeled designated data point. By adjusting the weights of parameters, the model tunes the algorithm to the correct output.

An essential aspect of SL is the choice of the appropriate algorithm. Some common algorithms used in SL classification are Decision Tree (DT), Naïve Bayes (NB), Support Vector Machine (SVM), and Random Forest (RF). In [10], the authors compared different SL algorithms for disease prediction and concluded that there was a significant variation in the results depending on the algorithm employed. The authors concluded that (RF) outperformed Support Vector Machine (SVM) and Naïve Bayes algorithm. Of the 17 studies included in their study, RF had the highest accuracy at 53%, followed by SVM at 41%. The quality of the features used to train the model is also crucial. Selecting the most relevant and informative features for the task explicitly affects the accuracy of the ML [4].

The inferred function produced from the training set should be able to map new inputs. The challenge is then to classify new unseen scenarios properly which requires the algorithm to generalize the inputs. When the model has been fitted, the next step in the process is validation. The model is validated on a test set. Depending on the requirements of the ML, the validation will give an insight into how well the model coped with the new data and whether it is good enough for further usage.

### 4.2.2 Machine Learning Pipeline

An ML pipeline is shortly described as the end-to-end workflow. It is a systematic process that involves several interconnected stages designed to turn raw data into a trained and tested ML model that can make predictions or classifications on new unseen data. In [11], a standard pipeline model is presented. Depending on the specifics of the objective, the pipeline may look different. The pipeline shown in (Fig. 2) is a generalized model.

ML pipelines start with data ingestion, where raw data from various sources is processed into a format that can be used by the following components. It is crucial to ensure that the data is representative and checked for anomalies, as this can significantly impact the accuracy of the ML model. Preprocessing includes cleaning, handling missing values, feature scaling, and transforming the data for ML algorithms, acting as a filter for unwanted attributes. In [12], it was concluded that preprocessing significantly affected classification performance. Feature engineering produces relevant features to improve model accuracy, but more features do not always mean better accuracy. It is important to choose features with a clear positive impact and avoid redundancy caused by highly correlated features. The next stage is model selection, where an appropriate algorithm is chosen based on the problem’s nature and the data type. After the model has been selected, the next stage is training, where the model is trained on the data using various techniques, such as cross-validation, to prevent overfitting. Once the model is trained, it is evaluated.

\(^3\)The outputs are divided into four classes
on a separate test dataset to estimate its performance on new, unseen data. The final stage is deployment, where the model is deployed in a production environment and used to make predictions or classifications on new data [4], [11].

4.3 Algorithms

Although various supervised machine learning algorithms exist for addressing specific problems, this thesis focuses on DT, Multilayer Perceptron (MLP), and SVM algorithms. In [13], MLP and SVM were successfully utilized to categorize physical exercises based on sEMG data. The primary focus of this work was to distinguish between different activities, such as sitting, lying, walking, walking upstairs, and walking downstairs. It used both time and frequency domains in its feature extraction. [13] showed that overall, the frequency domain features achieved higher accuracy than time domain features. [13] used a publicly available dataset and did not collect any data of its own compared to the current work where new data was collected. [13] obtained a accuracy of 94.35% with DT, 86.3% with SVM, and 96.48% with MLP.

In [14], MLP and DT were successfully utilized to categorize exercises using an sEMG sensor, with MLP yielding 100% accuracy and DT achieving 92.87% accuracy, respectively [14]. A notable distinction between this thesis and [14] is the difference in the data collection frequency. The authors of [14] collected their data at 50 and 200 Hz, whereas the data collection frequency for this thesis was set at 1000 Hz. Another difference is the set of considered exercises. [14] used his sensor on the wrist and also classified other exercises than the current work.

Since both [13] and [14] showed high accuracy with MLP, DT, and SVM, these three algorithms were chosen to be used in this thesis.
4.3.1 Decision Tree

A Decision Tree is an algorithm used in supervised ML for both classification and regression tasks. A DT consists of a hierarchical tree structure where the hierarchy has three different types of nodes: root, internal, and leaf [4]. The root node is at the top of the hierarchy, internal nodes are better described as decision nodes, and leaf nodes represent the different outcomes from the dataset. DT is built on the principle of "divide and conquer" and recursive partitioning (RP). [4]. In RP, data is divided into a smaller subset of classes. The algorithm starts at the root and selects the feature that most effectively predicts the target classes. The algorithm continues with this process until a stopping criterion is reached. An example of a DT can be seen in Fig. 3. A stopping criterion could be a limit on the maximum depth or number of leaf nodes in the tree. These stopping criteria are set as input arguments for the model. The DT algorithm performs best in less complex environments where the amount of features is more limited. It performs best when it can draw orthogonal decision boundaries making it less flexible for high variation in the training data. Orthogonal boundaries refer to splitting points created by the decision tree. These boundaries are orthogonal to the specific feature axis. In more complex environments, Random Forest can limit this instability by averaging predictions over many trees [7].

There are different ways to find the best decision for each feature, but the most commonly used are Entropy, Information gain, and Gini impurity. Entropy measures the impurity of sample values. The formula for entropy is given as [15]:

$$ Entropy(S) = - \sum_{c \in C} p(c) \log_2 p(c) $$

where $S$ represents the data set, $c$ is the classes in $S$, and $p(c)$ indicates the fraction of data points in class $c$ out of the total data points present in the dataset.

Figure 3: Decision Tree
Entropy takes values between 1 and 0. To choose the best attribute and make the optimal DT, the attribute with the smallest amount of entropy should be the first decision in a DT [4]. Entropy is favorable in environments where information theory is applicable.

Information gain tells the difference before and after a split on an attribute has been made. Information gain is calculated by using the formula [15]:

$$\text{Information gain}(S, a) = \text{Entropy}(S) - \sum_{v \in v(a)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$  \hspace{1cm} (2)

where $v$ stands for a specific attribute or a class label, $\frac{|S_v|}{|S|}$ represents the proportion of the values in $S_v$ to the number of values in the dataset $S$, and "a" represents a candidate attribute to split the dataset "S" on. Information gain tends to work well in many scenarios and usually favors attributes with a large number of distinct values.

Gini impurity is a metric that is calculated as the probability of misclassifying random data points in a data set. When using Gini impurity, the aim is to measure the attribute that gives the lowest Gini impurity. It is calculated using the following formula [15]:

$$Gini(D) = 1 - \sum_{i} (P_i)^2$$  \hspace{1cm} (3)

where $D$ represents the dataset, $i$ represents classes in the dataset, and $p_i$ is the probability of the sample belonging to class $i$ [4]. Gini can be a good choice when the class distribution is imbalanced.

Generally, Gini is to some extent computationally faster than the other criterions. However, Gini tends to isolate the most frequent class in its own branch whereas entropy tends to produce slightly more balanced trees. Gini is the default criterion for DT, however, it can be a good idea to compare different criterions in order to find the best one.

### 4.3.2 Multilayer Perceptrons

Multilayer Perceptron is a feedforward network model composed of an input layer, hidden layers, and an output layer [4]. It can approximate non-linear functions and implement non-linear discrimination. The input layer has the number of neurons equal to the number of inputs, and the output layer has the number of neurons equal to the number of output classes. The hidden layer contains all neurons between the input- and output layers. The name hidden originates from that they are not directly exposed to the input or output. When the data is passed forward through the network, the output to the next neuron is calculated by equation (4):

$$y_i = \sum_{j=1}^{d} \omega_{ij}x_j + b$$  \hspace{1cm} (4)
where $\omega_{ij}$ represent the weight of the connection from input $x_j$ to output $y_i$; $x_j$, $j = 0, ..., d$ are inputs; $y_i$, $i = 1, ..., k$ are outputs; $k$ and $d$ represent how many neurons are in each layer, and $b$ is the bias (see Fig. 4).

Fig. 3 shows the basic design of how each neuron in an MLP operates in a network. The MLP network contains many neurons connected to each other belonging to the different layers. It learns by adjusting and adapting the weights so that the network tries to minimize the difference between its output and the desired output. For a given node, the inputs are multiplied with the weights and summed together, and then passed through an activation function. The most common activation functions are Sigmoid and ReLU (Rectified Linear Unit). The activation functions decide whether a neuron should be activated or not [7].

MLP uses gradient descent (GD) presented by equation (5) where $\theta$ represents the weights in the MLP, and it is the variable that is optimized during the training process. $\eta$ is the learning rate. $\nabla f(\theta^k)$ represents the gradient of the cost function with respect to the weights at the current position $\theta^k$. The gradient points in the direction of the steepest increase in the cost function, and its negative points in the direction of the steepest decrease. It determines the magnitude and direction of the weight update at each iteration. Backpropagation is used to calculate the gradient $\nabla f(\theta^k)$. It involves propagating the error back through the network, using the chain rule. This allows computing the gradients of the cost function with respect to the weights in each layer. $\omega$ in equation (4) and $\theta$ in equation (5) refers to the same concept of weights associated with the features in a model.

$$\theta^{k+1} = \theta^k - \eta \nabla f(\theta^k)$$

A cost function is the difference between the predictions of the MLP algorithm and the true classes. The GD equation is used when fitting the model to the training data. GD is an iterative optimization algorithm capable of finding optimal parameters to minimize a cost function. The model will iterate until it converges to a minimum of the cost function, typically achieved when the cost function is close to zero.
or equal to zero, resulting in the smallest possible error [7].

To tune the MLP, some different parameters can be set for finding the optimal solution. Some of these parameters include activation function, learning rate, and optimization algorithm.

The Sigmoid activation function seen in equation (6) takes $x$ as an input which is the summation of the synaptic weights and calculates the neuron’s activation state. The function is limited in networks with many layers due to the vanishing gradient problem. The vanishing gradient problem leads to the lower layers of the network staying unchanged, meaning that the training never converges to a good result. The opposite can happen as well, and it is called exploding gradient descent problem. In such a situation, the connection weights grow bigger and bigger, which results in the algorithm diverging from the desired minimum. It most commonly occurs in regression ML tasks [7].

$$f(x) = \frac{1}{1 + e^{-x}}$$  \hspace{1cm} (6)

The Relu activation function seen in (equation (7)) takes $x$ as an input and calculates the neuron’s activation state. If $x$ is less than zero it returns zero, else it returns $x$. The network learns from its mistakes and improves its predictions over time while simultaneously making it computationally effective since it does not activate all neurons at the same time. The neurons will only be deactivated from back-propagation. The advantages of using Relu are the efficiency regarding computation, and that it converges faster due to its linear, non-saturating property. ReLU has a limitation since negative inputs can cause a neuron to become inactive, i.e., ”die”, reducing the model’s capacity to fit the data [7].

$$R(x) = \max(0, x)$$ \hspace{1cm} (7)

The learning rate ($\eta$ in equation (2)) is a hyperparameter that controls how big a learning step the model will take when the weights are updated during fitting and optimizing the model. A value that is too small can lead to a long learning process in which it may not converge, while a value that is too large can result in missing an optimal set of weights [7].

4.3.3 Support Vector Machine

The support vector machine algorithm is widely used as a machine learning classifier. The algorithm thrives in high dimensions and has flexibility in that it can use different kernels. A trained model has fast predictions, although the training requires a high cost in computation. The more advanced setting for an SVM has a big impact on the accuracy, making it important to get the right parameters. Incorrect parameters may lead to computation problems. For small data sets, SVM can be a powerful algorithm, but for larger data sets it has a risk of over-fitting [7].
SVM seeks a hyperplane in N-dimensional space (N — number of features) that differentiates data points. A variety of hyperplanes could be employed to separate the two or more data point classes [16]. The objective is to identify the hyperplane with the largest margin, or the largest distance between data points of both classes. Depending on which side of a hyperplane a data point resides on, a hyperplane establishes a boundary between decisions. The dimension is determined by the number of features retrieved from the raw data. The support vectors are utilized to alter the hyperplane, hence maximizing the distance between data points [16].

In SVM training, the margin can be optimized with either a hard- or soft margin. These two algorithms differ in how they handle data items that are difficult to classify or that break the margin limitation. Finding a hyperplane that separates the two classes without permitting misclassifications or margin constraint violations is the purpose of hard margin optimization. Hence, all data points must lie outside of the margin, and classes cannot overlap. Soft margin optimization loosens the margin constraint, permitting certain data points to fall within the margin or even on the incorrect side of the hyperplane. This allows for some misclassification, which aids the model in fitting the data points and generalizing them to new cases. The soft margin can be controlled by setting a parameter C. A higher value of C increases the tolerance for misclassification. Depending on the data, various optimization strategies can be employed. If the data is noisy or the classes are not clearly distinguished, a soft margin is employed. A hard margin is used if the data is devoid of these characteristics. The equation $f(x) = (w \cdot x) + b = 0$ is the mathematical interpretation of a hyperplane, where $x$ is a feature vector in a space $\mathbb{R}^D$ with $D$ dimensions, $w$ is a weight vector of the same dimensionality as $x$ and $b$ is the bias term[16].

During the training of SVM, the input data is represented as pairs of feature
4.3 Algorithms

vectors and corresponding labels: \( x_i \) and \( y_i \), where \( i \in 1, 2, ..., L \) and \( L \) are the number of data points. The feature vectors are \( D \)-dimensional, denoted as \( x_i \in \mathbb{R}^D \). The objective of the hyperplane is to orient itself as far away from the support vectors as possible, given that the support vectors are closest to the data points. Hence, the support vectors can be described as [16]:

\[
(w \cdot x_i) + b \geq 1 \text{ for } y_i = 1 
\]

\[
(w \cdot x_i) + b \leq -1 \text{ for } y_i = -1 
\]

where \( w \) is an orthogonal vector to any vector on the hyperplane. Combining the two equations (8) and (9), we have \( y_i((w \cdot x_i) + b) \geq 0 \). This allows the support vectors to be written as hyperplanes \( h_1 \) and \( h_2 \). To get the greatest distance between these support vectors and the origin hyperplane, it finds the greatest distance from the origin. Since \( h_1 = (-1 - b)/|w| \) and \( h_2 = (1 - b)/|w| \), the margin can be described as \( M = (1 - b)/|w| - (-1 - b)/|w| \) resulting in \( M = 2/|w| \). SVM seeks to reduce the term \(|w|\) to maximize the margin between the data points [16].

4.3.4 Ensemble

The ensemble is an ML algorithm that combines 2 or more ML algorithms to create a single algorithm. There are different methods on how to combine the different ML algorithms. The more popular ones are voting, bagging, and RF.

A voting ensemble algorithm works by letting the different ML algorithms vote. The prediction that got the most votes is the prediction of the ensemble algorithm. This makes lower-performing ML algorithms perform better.

Bagging is an ML technique that aims to reduce the variance and overfitting of a model by training several instances of the same algorithm on different subsets. Each instance generates a prediction, and the final output is the average of all predictions. Bagging works by taking multiple random samples with replacements from the training data, and training a separate model on each sample. By using multiple models, the variance of the predictions is reduced, and the bias is kept low. This approach allows for each model to be slightly biased, but the aggregation of predictions helps to reduce the overall bias.

RF uses bagging but it only uses DT as its ML algorithm. This allows for both to get a better algorithm for complex data and to control how each tree grows. RF gets extra randomness due to it not looking for the best feature or attribute to split on. Instead, RF searches for the best point among a random subset of features. This results in higher bias but, as a trade-off, in lower variance. Generally, this leads to better results.
5  Method

5.1 Collection of data

Data is an indispensable component of every ML model. The quality of data gathered directly influences the outcome of the model. An important aspect of the data collection is ensuring the data has no bias. In [6], the authors discuss the importance of diversity for objective decision-making. Due to this fact, a diverse group of individuals varying in gender, height, age, and athleticism was used to collect data in the current project. The group included 5 males in age 20-30 years and one female in age 25. All individuals varied from average to good athletic shape. This was done to obtain more comprehensive data. The variance can facilitate the development of a more profound ML model and enable a broader range of potential users.

To facilitate the labeling of data from Inno-X, it was chosen to collect the data with each exercise separately instead of doing everything in the same sequence. A protocol was established to ensure acceptable quality of the data. Each collection began from a stationary stand-up position while being monitored by the Inno-X, which was strapped around the leg. One of the three exercises was executed, then the collection was finished in a stationary stand-up position. The process is illustrated in Fig. 6, 7, 8.

This order of sequences was preferred since it allowed for a pause between each exercise to identify the beginning and end. Data with subpar quality was also collected to assess the ML model’s performance, such as improper use of Inno-X or other factors that may impact the data quality. This was done to observe the model’s predictive capabilities under non-perfect conditions. The ultimate goal was to obtain 500 data files for each exercise to develop a robust ML model.

Figure 6: Leg extension procedure [17]
5.2 Setup and materials

Already established and well-used programs and frameworks have been used in the project. The data collected by Inno-X was sampled at 1000 Hz and directly written to a CSV file. To ensure that the collected data maintained the desired quality, all data was plotted in Matlab, as seen in Fig. 10.

Python is a high-level programming language with many libraries, making it one of the most popular languages for data science. Python was chosen as the primary programming language because of its ease of use, flexibility, and ability to handle data efficiently. A well-established framework for ML in Python is scikit-learn [18] which provides tools to execute tasks including classification, regression, clustering, and dimensionality reduction. With Jupyter Notebook [19], it is possible to create and share documents containing live code, equations, visual diagrams, and computations, allowing one to manage enormous datasets collaboratively and efficiently.

5.3 Training Environment

The initial version of the training environment was implemented in its most basic form. The constructed training set only included one feature extraction for each feature. This was intended to yield a result and confirmation that the exercises could be separated and classified. After laying the groundwork for the training environment, the following step was to expand and enhance it. The objective was to
5 METHOD

5.3 Training Environment

Figure 9: Acceleration measured by Inno-X during LE and WS exercises

gradually make it more representative of the actual setting where the product will be used.

Figure 10: Flowchart for the training environment

The data files were divided into smaller subsets of data and labeled thereafter.
This was necessary because it is hard to know when a user starts and stops an exercise [14]. This was accomplished by slicing the files into smaller data chunks. In this manner, the training environment simulates real-time data by sending data in chunks rather than as complete exercises. Jupyter was used in the training environment since it allows for blocks of code that can be run instead of running the entire program. This allowed a more efficient way to evaluate the different algorithms individually.

By following the protocol for collecting data, a program was made to automate the labeling of the data windows. Since the first part of an exercise started with standing up and preparing for the exercises, the first window was thrown away to make sure that only clean data was included. The middle part was labeled as the particular exercise, and the last part was also thrown away.

5.4 Creation of dataset

The format of the dataset needs to satisfy scikit-learn’s requirements. The format consists of vectors that contain different features corresponding to a specific class. All feature vectors, X were extracted from the raw data as NumPy-arrays, corresponding to a class Y. Y consists of the three different exercises: LE, WS, SJ, and no exercise. Each vector X was then inserted into a matrix that constitutes the data set.

The data set was then divided randomly into a training set consisting of 80%, $x_{train}, y_{train}$, and a validation set consisting of 20% of the data, $x_{test}, y_{test}$. The model was then fitted with the training set $x_{train}$ and $y_{train}$, and afterwards validated with $x_{test}, y_{test}$. In order to determine which algorithm was the most suitable for the classifications, the models were reviewed and ranked based on two distinct characteristics, computational expense in a time-based space and classification precision in accuracy. Each algorithm was performed under identical conditions and with identical data to provide a valid outcome. To detect overfitting, the result was analyzed with cross-validation.

5.5 Creation of pipeline

The first tries of the created machine-learning models were tested on raw data without any preprocessing. The accuracy reached at that point was around 70%. By normalizing the data using the scikit-learns normalize function, the accuracy went up by a significance of 5-7%. The effect of the normalization of raw data made the feature extraction more efficient due to the normalized data. The next step in the pipeline after the feature extraction was to scale the data.

5.6 From training environment to real-time classification

The training data had preset inputs and outputs, whereas such information is unavailable during real-time classification. Assuming that exercises will be executed randomly, we have devised a solution by training our model with smaller subsets of each exercise. This approach enables the model to adjust more effectively when an individual changes exercise. In addition, ”no exercise” has been added as an output.
to the model to assist in classifying when no exercise is performed. For real-time classification, we decided to utilize two concurrently operating threads, with one thread sending input for feature extraction and the other – performing feature extraction and sending it to the model for prediction. When Inno-X has sent sufficient data to build a subsection, the data is delivered to a queue. The other thread can then grab it and begin extracting features, which are then being transmitted to the model for prediction.

5.7 Trial and error

The background of this project has helped in the progression of the project. In some respects, the previous knowledge and ideas are insufficient for the desired result. The project has progressed by testing and evaluating different methods and finding solutions. This has led to a trial-and-error method and has been a successful approach.

5.8 How to analyze the result

In this project, the aim was to achieve a confidence level of 95%. A confusion matrix presents the predictions made by the model in a graphical way. The accuracy is also presented as a percentage of the correct predictions versus total predictions. The user-friendliness was measured as computational cost in the time-space. An effective and efficient pipeline is the result of the combination between time and accuracy.

5.9 Flowchart for real-time classification

Fig. 11 shows the whole process of real-time classification. The process begins when Inno-x starts collecting data from a user. A separate thread from the main thread is then started, which gets raw data from Inno-x as a matrix that has the size of $4 \times \text{window size}$ where window size is the number of samples included in the window. window size will hereafter be referred to as a window. The window was set to 2500 samples which equals 2.5 seconds. Due to that, ML algorithms tend to converge faster and perform better with values between 0 and 1. The preprocessing of the raw data $x$ includes normalization of the data as follows: $X_{\text{norm}} = \frac{x}{||x||_2}$ where $||x||_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$. When the data is preprocessed, the window is ready for feature extraction. For more depth regarding the feature extraction, we refer to Joel’s thesis [17]. The next step in the process is to scale the feature extraction by calculating $X_{\text{scaled}} = \frac{x-\mu}{\sigma}$ where $\mu$ is the mean and $\sigma$ is the standard deviation of $X$. This step improves the model’s optimization and helps to minimize the cost function faster which measures the error or discrepancy between the predicted output and the actual output. The final step before predicting the window is to do a Principle Component Analysis (PCA) of the extracted features in order to reduce dimensionality.

The next step is the model prediction. The output will be one of the four classes NE, LE, WS, and SJ. The trained models are now deployed for prediction in real-time.

EMG and acceleration in x-, y-, and z-direction
Figure 11: Flowchart of the real-time classification
6 Result

The results achieved in this thesis are presented using confusion matrices for the training environment, and for the real-time classification, another dimension is added for computational cost in the form of time. The confusion matrices should be interpreted as the model’s prediction looking from top to bottom and the actual exercise from left to right.

6.1 Training Environment

The training environment consists of a data set, including 1849 windows of data corresponding to the different exercises.

<table>
<thead>
<tr>
<th>No Exercise</th>
<th>Squat Jump</th>
<th>Wall Sit</th>
<th>Leg Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>145</td>
<td>188</td>
<td>711</td>
<td>805</td>
</tr>
</tbody>
</table>

Table 1: Structure of data

Each window consists of 2500 data points for each feature collected by the sensors from Inno-X, making it a $4 \times 2500$ matrix. The data set was then divided into a training set consisting of 80% of the windows and a validation set consisting of the remaining 20%. The result presented in the confusion matrices is the model’s prediction on the validation set.

In table 2, the performance of each model is shown. Accuracy Train refers to the model’s ability to predict the training data. Accuracy Val refers to the model’s ability to predict the validation data. It can be fast determined that MLP and RF are the best-performing algorithms with an accuracy of 95.88% and 96.43%, respectively on the validation data. An ensemble stacking these algorithms achieved the highest accuracy of 96.98%. The corresponding confusion matrix for each model is shown in the figure mentioned in table 2.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy Train</th>
<th>Accuracy Val</th>
<th>Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>94.6%</td>
<td>86.26%</td>
<td>12</td>
</tr>
<tr>
<td>DT</td>
<td>100%</td>
<td>90.66%</td>
<td>13</td>
</tr>
<tr>
<td>MLP</td>
<td>95%</td>
<td>95.88%</td>
<td>14</td>
</tr>
<tr>
<td>RF</td>
<td>90%</td>
<td>96.43%</td>
<td>15</td>
</tr>
<tr>
<td>Ensemble</td>
<td>99%</td>
<td>96.98%</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 2: Accuracy of models in training environment
6.1 Training Environment

Figure 12: Confusion matrix for SVM in training environment

Figure 13: Confusion matrix for DT in training environment
6 RESULT

6.1 Training Environment

Figure 14: Confusion matrix for MLP in training environment

Figure 15: Confusion matrix for RF in training environment
6.2 Real-time environment

In the real-time environment, the data used for prediction is gathered directly from the Inno-X device and predicted every 2.5 seconds. The windows that are predicted vary in size from $\text{window size} = 2500 \pm 100$ due to delays or interference with the program. The time column included in table 3 is a mean value calculated over 50 predictions.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Time [s]</th>
<th>Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>90%</td>
<td>0.030</td>
<td>17</td>
</tr>
<tr>
<td>DT</td>
<td>78%</td>
<td>0.028</td>
<td>18</td>
</tr>
<tr>
<td>MLP</td>
<td>76%</td>
<td>0.031</td>
<td>19</td>
</tr>
<tr>
<td>RF</td>
<td>74%</td>
<td>0.046</td>
<td>20</td>
</tr>
<tr>
<td>Ensemble</td>
<td>74%</td>
<td>0.060</td>
<td>21</td>
</tr>
</tbody>
</table>

Table 3: Accuracy of models in real-time environment
6.2 Real-time environment

Figure 17: Confusion matrix for SVM in real-time environment

Figure 18: Confusion matrix for DT in real-time environment
6.2 Real-time environment

Figure 19: Confusion matrix for MLP in real-time environment

Figure 20: Confusion matrix for RF in real-time environment
Figure 21: Confusion matrix for Ensemble in real-time environment
6.3 Pipeline

The final version of the pipeline is shown in Fig. 22. The end-to-end workflow presented is the groundwork of all models. The result of this pipeline directly corresponds to the achieved accuracy and computational time.

Figure 22: Final version of pipeline
7 Discussion

7.1 Performance

The performance of the different models in the training environment varied over a span of 12 percentage points. Three of the five tested models achieved a higher accuracy than the 95 percent that was set as a goal. The simplest models, SVM and DT, were the ones that had the lowest accuracy. An interesting aspect is that at the beginning of the project, these algorithms had the highest accuracy of the five. Aligned with the background work, this was not so strange since those algorithms usually perform better in simpler environments. But as the room of feature vectors got more complex, their performance was outperformed by the more complex models. This intersection point became very clear when at some point we added 5 times as many feature vectors.

All the models had problems with separating Leg Extension and Wall Sit. This was an expected problem, and when plotting the data points for these exercises, it is easy to understand why. The reason for this is the similarity of the exercises. If performed correctly, there should be minimal difference in acceleration since it is constant and at the same angles, leaving a lot of weight on the EMG features for separating these two exercises. When the data points overlap like this, the algorithms have difficulties with isolating the classes. Additional feature extraction is required for a more significant separation. More feature vectors make the computation heavier and more complex. When at the same time, the data points are overlapping, the need for a more complex algorithm that can generalize from such an environment is beneficial. Such algorithms, in this case, are the RF and MLP.

All models are high-performance in the time domain. The requirement for this project was to make predictions for at longest 15 seconds per prediction. Each of the model’s predictions is made in under a tenth of a second (See table 3). A reason for this is that the pipeline is implemented with a separate thread, allowing the Inno-X device to collect and stream data while our model can predict exercises at the same time. This contributes to the user-friendliness of the usability of Inno-X, which was one of the goals of this project. Informing the athlete that the device and program are aware of the type of exercise being performed can enhance the likelihood of collecting clean data with minimal noise, which enables a more accurate analysis of the EMG.

The real-time environment versus the training environment shows a deviant result from what we expected. The algorithms which was the worst performer in the training environment had the highest accuracy in the real-time environment. The SVM classifier had an increase of 4 percentage points. The reasons for this are unknown. As previously mentioned, there is a gray zone in the real-time environment where there is a sequence between the different exercises that are expected to be mixed up since it was not included in the training environment. Therefore, the lower accuracy for all models except SVM can be explained and seems to be reasonable. As for SVM, it obviously thrives well in the new real-time environment and has an easier time separating the classes than the other models. One explana-
tion could be that the more complex algorithms such as MLP, RF, and Ensemble require more training data for better generalization, whereas the SVM can handle a complex feature environment with less training data. To test this hypothesis, one could have trained the models on a dataset that is ten times larger to observe how the performance varies.

The algorithms used in this thesis show some interesting outcomes. In our case, it seems like the more complex algorithms such as RF and MLP easier got overfitted which affected the performance in the real-time environment. Reasons for this may be due to the need for these algorithms to generalize from more features. Usually, these algorithms also require a bigger dataset to get fitted from. These two aspects have most likely been the biggest reasons for the difference in the environments. Regarding the SVM and DT, their performance is more aligned when it comes to overfitting, although they are likely overfitted as well. DT has a tendency to get overfitted in noisy environments which we expect our environment to be. SVM is generally more robust which could be an explanation to why the DT performed better on the validation data but not in the real-time environment. See Table. 2 and 3. These algorithms usually perform better in more simple feature spaces which we believe our environment is. This may be the reason why they performed better than MLP and RF in a real-time environment.

In table 2 the difference in the accuracy of the model’s predictions on training data versus validation data can be seen. DT reaching 100 percent is an indication that it likely is overfitted. The same thing can be said about the ensemble. Comparing this to their performance in the real-time environment, it is clear that the translation produces a gap in accuracy which also indicates that these are overfitted. For MLP and RF it can also be noted that their performance on the validation data is higher than on the training data. Usually, it is expected that the performance on the training data is higher than on the validation data. In this case, it can be seen that the generalization ability to a real-time environment is poor. Both MLP and RF perform significantly worse in real-time. SVM is the only model that shows great generalization ability. As expected, it has a lower accuracy on the validation data. A small gap between these and still lower on the validation data is usually a good indication of good generalization ability. The translation from training- to a real-time environment also shows that it handles new unseen data well.

A self-critical examination of the project as a whole gives some insight into what could have been improved. The fact that the training environment provided such a high accuracy gave a false sense that the same applies to the real-time environment. For improving this transition between environments, more time is needed to adjust the pipeline. Looking at the final version of the pipeline seen in Fig. 22, the step from deployment to feedback is very important. Being able to know in advance how all modules should be constructed is difficult. Hence, more loops in the pipeline workflow are required for improving the models in the real-time environment. Therefore, this project can be improved further by keeping on working in that last step of the pipeline.
7.2 Shortcomings and potential for improvement

The placement of Inno-X on the leg might have affected the data more than we realized in the beginning. Our first thought was that if we changed the leg that we strapped on the device, the more diversity we would get in the data. This is true, but what we did not think of is that the acceleration in the z- or y-axis would change direction. Furthermore, we did not have a protocol for how to place Inno-X on the leg, leading to it sitting upside down in some cases. Hence, the acceleration in the x-direction was not consistent.

One potential factor that could have influenced the model’s performance is the overfitting of the data. Mainly, all data collection has been done by the authors of this thesis. We have been aware of this problem since the start, but due to the lack of time and contacts, we had to prioritize. For an even better generalization, a higher diversity in the data would create a more solid training environment. A reason for the high accuracy in the training environment on the more complex algorithms may be due to overfitting. A good approach for further investigation would have been to carefully analyze the training data predictions and the validation data predictions.

A deliberate shortcoming of this project is the time between the exercises. We made a choice at the beginning of the project to collect data and divide it into windows. The advantage of that strategy was the easiness of labeling the data. By doing this, each window contains only clean data of the specific exercise even though ”no exercise” is a class. The data used for training the model is clean data from standing still. Therefore, it exits a grey zone when switching between exercises where the person might walk or lose balance when preparing for the next exercise. This causes the accuracy in the real-time environment to be lower since this gray zone data is thrown away in the training environment.

The final version of the concluded pipeline for this project shown in Fig. 22 is an ongoing workflow that can be improved further. All included modules have been well-crafted and fine-tuned to reach the goals of this project. Despite so, there is still room for improvement to reach higher accuracy. The last percent towards achieving 100 percent accuracy lies in confusion between the two exercises, leg extension, and wall sit. To separate these classes further, the feature extraction part most likely has the most significant impact. Since the acceleration in all directions is similar while executing these exercises, the separation relies mainly on the EMG features. By adding more features from the EMG both in the time and frequency domain, the distance between the data points in the feature vector room will most likely be increased.

7.3 Comparison with existing works

In [8], a similar project for activity recognition was done. The authors made classifications of 6 different classes by using raw data from only a three-axial accelerometer. The study made a comparison between time- and frequency domains as well as original features and features after PCA. The classifiers used are DT, SVM, and MLP, which makes the similarities to our project high. A significant difference is that the
authors in [8] only performed their classification in a training environment. Another difference is that the activities included standing, sitting, lying, stairs up, stairs down, and walking. Looking more closely at Fig. 4 in [8], one can see that the overlapping between exercises exits but not to the same extent as between LE and WS.

With the similarities and differences in mind, the average accuracy achieved in [8] was 94.36% for the original features, 98.57% with features after PCA, and 83.48% respectively 92.38% in time and frequency domains. The accuracy is very similar to what was achieved in the training environment in this thesis. Looking at the feature extraction and what models were used, the similarities align as they should. An interesting difference is that the dimensionality reduction had a negative impact on our accuracy, whereas in [8], it was positive. A reason for this might be that we were careful of which features were used by adding them one by one in order to see how they affected the accuracy. By doing that, we ensured that all features had a significant variance. The usage of accelerometer data in both time- and frequency domains was something that we did not do in our project. The achieved accuracy in the time contra frequency domain was almost 10 percentage points, which is a noteworthy higher. Definitely, something that should be looked into working further on this project.

In [9], the same data was used for activity recognition using SVM and k-Nearest Neighbors (KNN). The authors also divided the feature extraction into two different sets, with one being the time domain and the other – the frequency domain. Similar to [8], the frequency domain features achieved a higher accuracy but this time with around 5 percentage points. The combination of both domains gave the best result.

### 7.4 Social Requirements

The product Inno-X serves a social health purpose where the main goal is to help athletes to not overexert the body in training. By using the Inno-X before exercising it can be seen as a preventive action for knowing the body’s readiness and fatigue. If reaching a wider audience Inno-X can help the society to prevent injuries connected to training which can relieve the health care. It can be economic advantages for using Inno-X since preventing injuries is beneficial for society but also sport clubs with elite athletes that train every day. It is therefore important that the analysis of the fatigue and readiness are of high accuracy and gives the needed insight for adjusting exercise patterns after the body’s boundaries.


8 Conclusion

This thesis presents an approach to how to classify physical exercises using raw data from a 3-axial accelerometer and sEMG sensor. The developed pipeline lays the ground for the workflow of this project and the classification. The data required for classification were collected by the authors of this thesis using the Inno-X device developed by Innowearable AB. The goal was to develop a pipeline that can make predictions within the guidelines that were given by Innowearable AB. The guidelines included a time limit for each prediction that was set to a maximum of 15 seconds per prediction. The accuracy goal was set to 95% by the authors of this thesis, in both training- and real-time environments. The pipeline's robustness and efficiency are defined by its accuracy and speed.

The main conclusions of this thesis are given below. Strict protocols of data collection are of high importance for ensuring the quality of training data. Both factors, how the data is collected and how and where the Inno-X device is placed on the leg, affect the quality of the collected data. Diversity in the data is important to avoid biases and to get as good generalization as possible. Data preprocessing, normalization, and scaling of features have a significant positive effect on the ML accuracy. By testing different ML algorithms, we were able to analyze and find which algorithm performs best for different feature spaces. This also contributed to a good understanding of how different models behave in different environments. By tuning these models, testing different hyperparameters, and performing a grid search, we were able to increase the accuracy by a few percentage points. Plotting the prediction in confusion matrices gave an informative insight into how the models confused the different classes. In the training environment, the pipeline was well worked out, which resulted in high accuracy for most of the models. MLP, RF, and Ensemble all reached a higher accuracy than 95%. The real-time environment was not as elaborated, which on average resulted in a lower accuracy. One factor that contributed to this is the grey zone between the classes. This occurs when the user switched between exercises, for example, from LE to NE. Why this is a grey zone is due to that this exact sequence was thrown away in the training environment for the purpose of only having clean data. In the real-time environment, SVM was the best-performing model. A possible explanation is SVM's ability to handle complex feature spaces having less training data. By adding 10 times more training data, we expect the real-time environment to better correlate with the training environment in terms of which models perform best.

Although the ML algorithms achieved sufficient accuracy in the training environment, the accuracy could not be translated effectively to the real-time environment. The accuracy results in the training environment are similar to the result found in both [13] and [14]. Compared to the mentioned works, this thesis has two original contributions: the classification of specific physical exercises and the utilization of raw accelerometer data in combination with sEMG data.

This thesis has laid a solid foundation for the development of a robust AI solution that can be utilized as a final product for Innowearable AB's Inno-X device. By refining the pipeline and enhancing the modules for the real-time environment, the
accuracy can be improved beyond the 95% limit. Collecting additional training data is a critical factor in creating a more robust real-time model with improved generalization ability. Expanding the feature set in the frequency domain, as was done in [8] and [9], would also be a valuable addition for future work.
bibliography