Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Hesham Hussen

Prediction of depressive episodes using biomedical sensors with machine learning techniques

Fakultaet Technik und Informatik
Department Informations und
Elektrotechnik

Faculty of Engineering and Computer Science
Department of information and
Electrical Engineering

Hesham Hussen

Prediction of depressive episodes using biomedical sensors with machine learning techniques

Bachelorthesis based on the examination and study regulations for the Bachelor of Engineering degree programme Information Engineering

at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Science Hamburg

Supervising examiner: Prof. Dr.-Ing. Joerg Dahlkemper
Second examiner: Dr. rer. nat. Andreas Martens

Date of delivery: 9. April 2020

Hesham Hussen

Title of the paper

Prediction of depressive episodes using biomedical sensors with machine learning techniques

Keywords

Machine learning, Depression Detection, Data preprocessing, Health Data, CNN, RNN, GRU, Random Forest, Ensemble Methods, Linear SVR, Python, Tensorflow

Abstract

The behaviours of patients with depression are usually difficult to predict because the patients demonstrate the symptoms of a depressive episode without warning at unexpected times. The goal of this thesis is to examine different machine learning approaches to detect such times and predict possible depressive episodes. The work also comprises a preprocessing pipeline for transforming and organizing the input data. The final assessment indicates current and future application possibilities of machine learning in the depression detection context.

Hesham Hussen

Thema der Arbeit

Prediction of depressive episodes using biomedical sensors with machine learning techniques

Stichworte

Maschinelles Lernen, Depression Vorhersagen, Data Vorverarbeitung, Health Data, CNN, RNN, GRU, Random Forest, Ensemblemethoden, Linear SVR, Python, Tensorflow

Kurzzusammenfassung

Das Verhalten von Patienten mit einer Depression ist in der Regel schwer vorherzusagen, da die Patienten die Symptome einer depressiven Episode ohne Vorwarnung zu unerwarteten Zeiten zeigen. Das Ziel dieser Arbeit ist es, verschiedene Ansaetze des maschinellen Lernens zu untersuchen, um solche Zeiten zu erkennen und moegliche depressive Episoden vorherzusagen. Die Arbeit umfasst auch eine Vorverarbeitungs-Pipeline zur Transformation und Organisation der Eingabedaten. Die abschliessende Bewertung zeigt aktuelle und zukuenftige Anwendungsmoeglichkeiten des maschinellen Lernens im Kontext der Depressionserkennung auf.

Contents

List of Tables

List of Figures

Listings

1. Introduction

1.1. Motivation

Depression is playing an increasingly important role in today's society and is nowadays frequently diagnosed as a disease. Around 5.3 million people in Germany suffer from depression each year, and around 17 percent of German adults will experience a persistent depressive disorder in their lifetime [1]. Increasingly, people complain about feelings of loneliness, indecisiveness, and powerlessness and run with the risk of depression. Moreover, a recent study shows that employees in Germany are taking more and more sick leave due to depression, causing experts to call for more investment in early recognition and prevention [2].

Adesso AG is tackling this problem with the so-called "STEADY Project" to connect the digital world with the healthcare area. Mainly, the project is a digital health portal aimed at people suffering from depression. The portal distinguishes between three user groups "patient", "doctor" and "scientists". Patients are thus provided with a platform that contains both data collection and data evaluation as essential components. A suitable form for the collection of vital data is a fitness bracelet. Additionally, digital questionnaires are used to obtain information about a patient's condition concerning their mood and well-being. The recording is usually carried out over a long period to enable as precise analysis as possible.

1.2. Problem statement

A report from 2017 estimated that the average person in Germany must wait three months for an initial appointment [3] with a registered psychotherapist, then three more months to get a regular appointment, which in some cases is a fatally long time. Moreover, the only way now to diagnose a patient with depression is through meticulously conducted sessions with psychiatrists and experts, wherein many questionnaires and tests are evaluated and assessed, a process that is both time and resource consuming.

Since modern problems require modern solutions, a desirable approach would reduce the need

for direct contact between patients and psychiatrists and at the same time, reaches a credible prognosis of the depression status.

1.3. Thesis Goal and Structure

The aim of this Bachelor thesis is the analysis of personal health data from potential depressed patients and predict their depression levels. For this purpose, a number of machine learning models shall be examined to see which one fits the data better and compare their respective behaviours. Based on the collected data, these models shall be able to classify patients - either with classification or regression - on a depression scale. This would be helpful as it will solve waiting lists bottlenecks by prioritising the process; patients with high depression rates would have a high priority, low depression rate mean they can wait some time. Moreover, it will alleviate the workload of the doctors, which will make them able to take in more patients and, at the same time, provide more thoroughly attention and support.

This thesis is structured as follows: after analysing the state-of-the-art of depression prediction methodologies based on biomedical sensor systems in this chapter, chapter 3 presents an introduction to depression categorisation and clinical diagnoses instruments, thereupon, giving an overview of the project STEADY as the environment wherein the following work shall be realised. A Thorough illustration of the datasets and the preprocessing pipelines follows in chapter 4. Chapter ?? goes through the implementation of the chosen machine learning models and present their results and performances. Findings and evaluation of the results are presented in Chapter 6. Finally, Chapter 7 concludes with a summary of the entire work and overview of perspectives for future work.

2. State of the Art

Depression is one of the most prevalent mental health problems among adults; however, its diagnosis and tracking methods still rely mainly on assessing self-reported depressive symptoms, methods that originated more than fifty years ago. These methods, which usually involve filling out surveys or engaging in face-to-face interviews, provide limited accuracy and reliability and are costly to track and scale [23].

2.1. Related work

For the past decades, significant efforts have been invested to find algorithms that detect depression. For instance, there had been several studies on human actions to identify depression symptoms. A research on facial actions and vocal prosody has achieved accuracy of 88% in detecting the symptoms [24]. Clinical depression is associated with dull, monotonous and lifeless speech with a lack of expression [31]. Reflected changes in speech quality can indicate affective disorders including depression [32, 33]. Depressed subjects experience physiological fluctuations that alter vocal fold and vocal tract airflow modifying speech properties [34]. Depressed speakers exhibit quantifiable changes in spectral, prosodic, articulatory and phonetic properties [35, 36]. Studies have subjectively and objectively evaluated speech parameters as indicators of depression, severity and treatment efficacy [37]. Jarrold et.al. conduct a research on brain health by using data mining tools to identify clinical depression and have also achieved a high range of accuracy from 73% to 97% [25]. However, the outcomes of these studies are not able to detect the symptoms without interactions between the doctor and patient [26].

A study by Burns MN et. al. found that phone sensor data could detect social patterns among depressed patients, but this was a small study with only 8 participants [27]. Other studies have found that phone sensors were effective at detecting social and sleep behaviours among patients with depression [28, 29], and such features correlated significantly with severity of depressive symptoms [30].

A linguistic analysis for detecting depression was performed by Ang Li et al. [38]. A content analysis of depression-related Tweets was performed by Patricia A. et al [39]. A nationally representative study among U.S. young adults was done by Brian et al using multiple social media platforms [40]. Affective content analysis of Online Depression Communities was done by Thin Nguyen et al. [41]. The main aim of these systems is to efficiently design the algorithm for detection of the depression stigma. The data set is collected from websites such as Weibo and Twitter. First the collected dataset is analysed with the help of syntax and semantics analysis which gives the sense of depression stigma among posts posted by different age groups. In this process the syntax is analysed for finding certain key words and relevance of those key words is made with the help of semantic analysis which finds the general emotion of the paragraph via understanding the emotion of the text also known as Emotion Detection Systems or sentiment analysis. Then the posts are classified according to the depression symptoms [21].

2.2. Machine learning methods

Machine Learning is one of the tools of Artificial Intelligence [42, 43]. Artificial Intelligence is a simulation done with the help of machines, mainly computers of human intelligence. This simulation includes constant learning, reasoning, and adapting by self-correction. Machine Learning gives the machines the ability to teach themselves without anyone instructing it what to do [45]. Its algorithms learn themselves from the given data and apply the gained knowledge to make predictions, it also has algorithms that are experts in finding the pattern from the data. Moreover, they are very good at discovering the best combination of features from the data [46]. In depression prediction studies, it is better to focus on the overall pattern in the data rather than looking at individual attributes [44].

The Machine Learning models based on the way they handle the data are classified into three major categories: Supervised Learning where the data is labeled, and the output is known already, Unsupervised Learning where the data is unlabelled, and the output is decided later from the inferences, and Reinforcement Learning which is based on the feedback mechanism, the algorithm is in such a way that it interacts with the environment, finds the rewards or errors [47]. The process of prediction or classification involves four stages: collecting data, pre-processing it to handle the missing values and reduce the noise, select the essential features, then implement a model suitable for the data in hand. In the following sections, a number of prominent algorithms are discussed to give more insights about their internal functionalities and why they are of relevance for this work.

Random Forest

Random forest is the most widely used machine learning model that can be used for both regression and classification. Random forest is an ensemble of several decision trees (see figure 2.1), and the prediction is made as a result of taking an average of all the predictions from the decision trees [48].

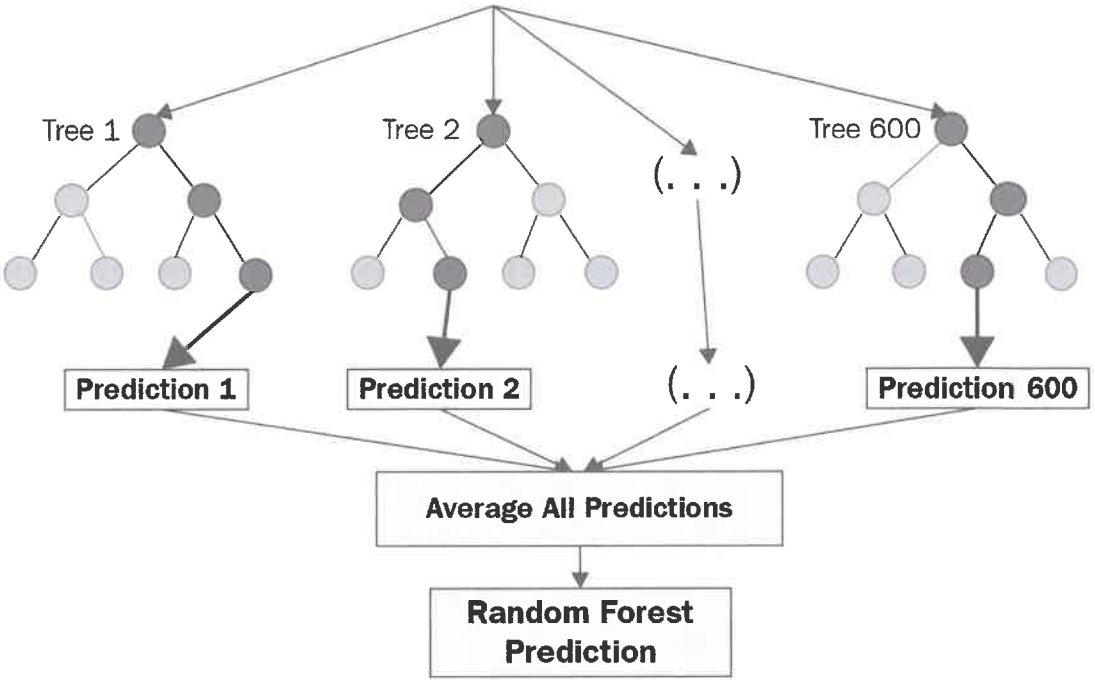


Figure 2.1.: Random Forest Structure [50].

The Random Forest algorithm introduces more randomness when growing trees; instead of searching for the very best feature when splitting a node, it searches for the best feature among a random subset of features. This behaviour results in a greater tree diversity, which trades higher bias for a lower variance, generally yielding an overall better model. Another great quality of Random Forest is that they can measure the importance of each feature. Scikit-Learn[1], for example, measure a feature's importance by looking at how much the tree nodes that use that that feature reduce impurity (or MAE) on average (across all trees in the forest). More precisely, it is a weighted average, where each node's weight is equal to the number of training samples that are associated with it.

[1]Scikit-learn is a free software machine learning library for the Python programming language.

Support Vector Machines

Support Vector Machines (SVM) is a versatile machine learning model that is capable of performing linear or nonlinear classification, regression, and even outlier detection [12, p. 153]. There are two main categories for support vector machines: support vector classification (SVC) and support vector regression (SVR). SVM is a learning system using a high dimensional feature space. It yields prediction functions that are expanded on a subset of support vectors. SVM can generalize complicated gray level structures with only a very few support vectors and thus provides a new mechanism for image compression. A version of a SVM for regression has been proposed in 1997 by Vapnik, Steven Golowich, and Alex Smola [13]. This method is called support vector regression (SVR). The model produced by support vector classification only depends on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by SVR only depends on a subset of the training data, because the cost function for building the model ignores any training data that is close (within a threshold ε) to the model prediction [14], such cost function is called ε-insensitive cost function and the model called ε-SV regression model. For a multidimensional data with the shape $(x_1, y_1), ...(x_l, y_l)$ where X denotes the

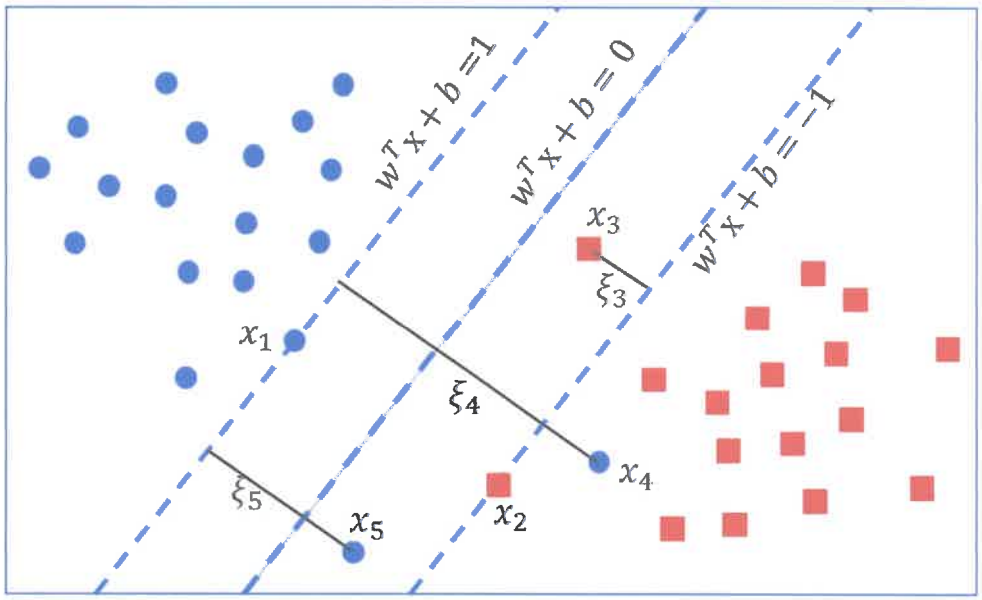


Figure 2.2.: Soft margin loss setting for linear SVR with $\varepsilon = 1$ [22]

space of input features and y the targets for such inputs, the SVR model goal is to find a function $f(x)$ that has at most ε deviation from the actually obtained targets y_i and at the same time as flat as possible. This model does not regard any errors as long as they are less than ε but it does not accept any deviation larger than this margin (see figure 2.2) [15].

$$f(x) = w \cdot x + b \text{ with } w \in X, b \in R \tag{2.1}$$

Flatness in the case of equation 2.1 means that it is required to minimize the norm $||w||^2$. This can be written as a convex optimization problem:

$$\text{minimize } ||w||^2$$

$$\text{subject to } \begin{cases} y_i - (w \cdot x_i) - b \leq \varepsilon \\ (w \cdot x_i) + b - y_i \leq \varepsilon \end{cases} \tag{2.2}$$

These constraints requires prefect separation between classified classes and does not allow any errors, and in some cases, make the problem unsolvable. A solution to this problem was provided by Corets and Vapnik [16] in the form of slack variables ξ_i, ξ_i^* to cope with otherwise infeasible constraints of the optimisation problem (equation 2.2) and allow some errors. Hence, equation 2.3 was stated by Vapnik [15].

$$\text{minimize } ||w||^2 + C \sum_{i=1}^{l} (\xi_i + \xi_i^*)$$

$$\text{subject to } \begin{cases} y_i - (w \cdot x_i) - b \leq \varepsilon + \xi_i \\ (w \cdot x_i) + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases} \tag{2.3}$$

The constant C determines the trade-off between the flatness of f and the amount up to which deviations larger than ε are tolerated [15].

Convolutional Neural networks

Convolutional Neural Network (CNN) is a neural network architecture that is especially suited for 2-dimensional data structures, e.g. images. The core concept behind CNNs is to model the invariance of visual features to translation, rotation or even illumination. This allows to recognise certain patterns in the data, even when images are shifted, rotated or flipped upside down. The main building block of a CNN is the *convolutional layer*: neurones in the first convolutional layer are not densely connected to each neurone in the input neurones (Like they are for a classic Neural Network), rather, they are only connected to pixels that represent their receptive field (see Figure 2.3). In turn, each neurone in the second convolutional layer is connected only to neurones located within a small rectangle in the first layer. This architecture allows the network to concentrate on low-level features in the first hidden layer, then assemble them into higher-level features in the next hidden layer, and so on. This hierarchical structure is common in real-world images, which is one of the reasons why CNNs work so well for image recognition [12, p. 63].

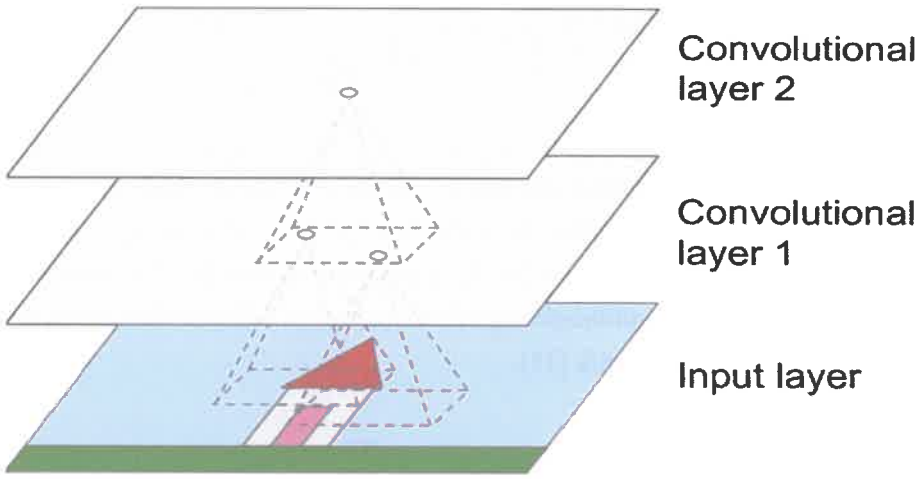


Figure 2.3.: CNN layers with rectangular local receptive fields.

Unlike classic Neural Network (NN) , CNN operate over volumes, that means these networks prefers the input to be in 3D shape. This quality allowed native interaction with visual data, as most images data consists of three channeled coloured images.

1D Convolutional Neural networks

The main point to be taken from CNN is that they discover patterns in visual data through the process of simultaneously applying different filters (or convolutional kernels) on the data, making them able to detect those patterns and features anywhere in its input. In the same analogy, 1D convnets can be used in time sequence forecasting where 1D patches (subsequences) are extracted form complete sequences (see figure 2.4). The same properties that make convnets excel at computer vision also make them highly relevant for sequence processing, as time can be treated as spatial dimension like the height or width of 2D image.

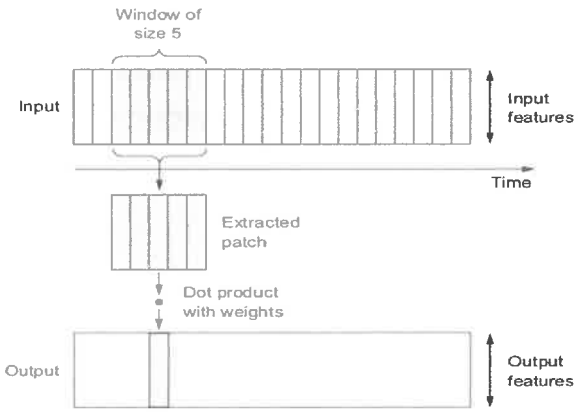


Figure 2.4.: How 1D convolution works: each output tilmestep is obtained from temporal patch in the input sequence [51, p. 225]

Such 1D convolution layers can recognise local patterns in a time sequence. Because the same transformation is performed on every patch. a pattern learned at a certain window can be later recognised at a preceding position. [51, p. 225]

Recurrent Neural Networks

A Recurrent Neural Network (RNN) looks very much as a classic NN, except it has connections pointing backwards. A simple RNN would be composed of one neurone receiving inputs, producing an output and sending that output back to itself. RNNs are a special architecture of NNs, that can effectively incorporate temporal dependencies within the input data. This can be achieved by unrolling an NN on the temporal axis, where the network at each time step is provided with feedback connections from previous time steps.

Training a recurrent NN with gradient descent requires back-propagating gradients through the entire architecture in order to calculate the partial derivatives of the loss function with respect to each parameter of the model. As the chain rule is applied many times in back-propagation, the gradients flowing through the network can either become very large or very small. Due to the complex structure of RNNs, the architecture suffers from vanishing or exploding gradients during training through SGD [52]. In practice, large gradients can be avoided by clipping the gradient [53]. Vanishing gradients however, remain a challenge of deep architectures and prevent the model from learning correlations between distant events.

An RNN can simultaneously take a sequence of inputs and produce a sequence of outputs (see Figure 2.5, left network). For example, this type of network is useful for predicting time series such as stock prices: you feed it the prices over the last N days, and it must output the prices shifted by one day into the future (i.e., from N-1 days ago to tomorrow). Alternatively, you could feed the network a sequence of inputs, and ignore all outputs except for the last one (see the right network). In other words, this is a sequence-to-vector network. For example, in STEADY use case, the network is fed with a sequence of days (7 days), and the network outputs a depression prediction for the next day.

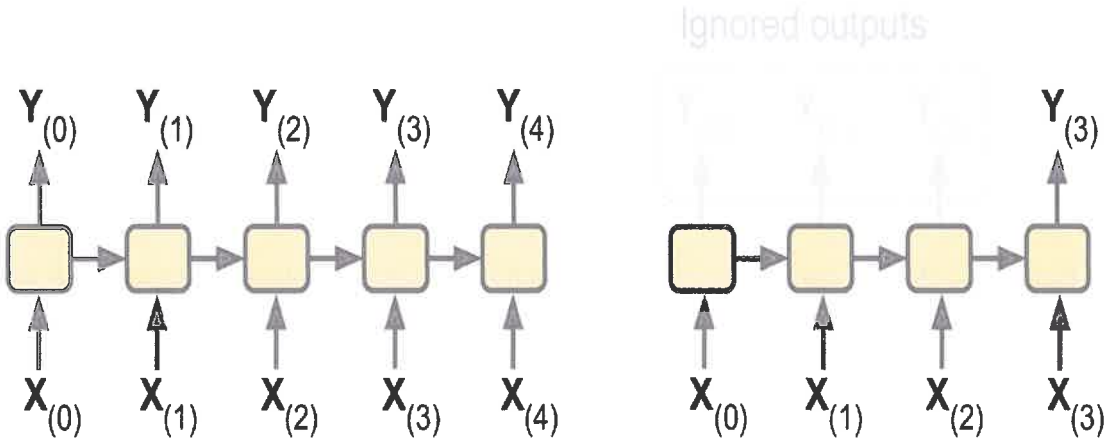


Figure 2.5.: Seq-to-seq (left) and Seq-to-vector (right) [12, p. 502]

Long short term memory (LSTMs)

Due to the transformations that the data goes through when traversing an RNN, certain patterns and information are lost at each time step. After a while, the RNN's state contains

almost no trace of the first inputs. This behaviour is very problematic for modelling long sequences, not to mention the problem of vanishing/ exploding gradients mentioned in the previous paragraph. To tackle both problems, various types of cells with a sort of memory was developed. One of them is the LSTM cells. They proven so successful that the basic cells are not used much anymore [12, p. 63]. LSTM cells solves this short-term dependency problem

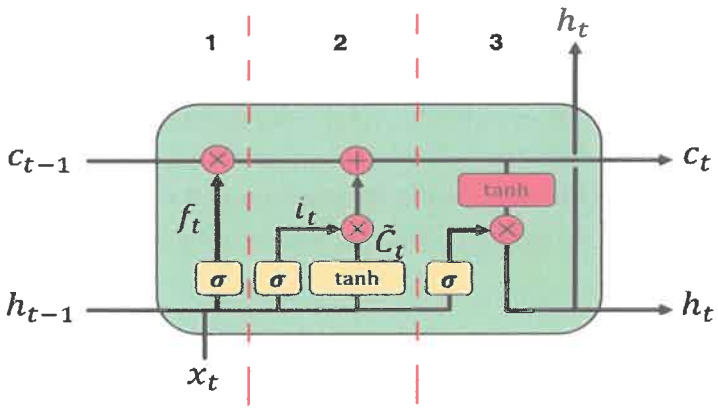


Figure 2.6.: LSTM cell

and make the network to remember longer dependencies. In addition to RNN structure, LSTM has a cell state, which transfers the information through LSTM cells with a minor change of the information. Consequently, the network's default behaviour is to remember long-term dependencies. LSTM cell behaviour can be summarised into three main gates, namely, Forget, Update, Output.

Forget gate uses the previous cell output h_{t-1} and input x_t to output a value between 0 and 1, varying between "completely forgetting" and "completely keep". This is achieved through the sigmoid function in the left third of the cell.

$$f_t = W_f \cdot \sigma[h_{t-1}, x_t] + b_f \tag{2.4}$$

Update gate decides which values to store in the cell state and then execute this update. This operation takes place in the second third of the LSTM cell through two steps, firstly, sigmoid activation to decide which values to update, again using the cell output h_{t-1} and input x_t.

$$i_t = \sigma(W_f[h_{t-1}, x_t] + b_i) \tag{2.5}$$

and simultaneously generate new vector of "candidate values" that could be added to the cell state.

$$\bar{C}_t = tanh(W_c[h_{t-1}, x_t] + b_c) \tag{2.6}$$

Then the update of the cell is executed through multiplying f_t with C_{t-1}, which translates to "forgetting" what needs to be forgotten, then adding the new candidate values C_t scaled by the input gate i_t to selectively update the state value.

$$C_t = f_t \otimes C_{t-1} + i_t \otimes \bar{C}_t \tag{2.7}$$

Finally, the ***Output gate*** outputs a filtered version of the cell state through.

$$h_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \otimes tanh(C_t) \tag{2.8}$$

The $tanh(C_t)$ acts as the filtration part in the equation as it outputs a value between -1 and 1, which is then multiplied by sigmoid activation layer to produce the output. [12, p. 514-517]

3. Depression Taxonomy and Steady project

As the field of clinical behavior analysis grows, it will benefit from analyses of increasingly complex and common clinical phenomena, especially those with significant public health implications. One such phenomenon is clinical depression, considered to be the "common cold" of outpatient populations. [5]

However, before a clinician can successfully treat a depression, he or she must first accurately diagnose the patient. Yet, there are different kinds of depression, which respond to different types of treatments. What works for one sort of depression may not work for another, or might even prove harmful. Only by first making an accurate diagnosis can a doctor figure out what treatments are likely to help.

The current subtyping of depression is based on the Diagnostic and Statistical Manual of Mental Disorders, 4th ed, Text Revision (DSM-IV-TR) categorical division of bipolar and depressive disorders [6]. Current evidence, however, supports a dimensional approach to depression, as a continuum/spectrum of overlapping disorders, ranging from bipolar I depression to major depressive disorder. There are several case-finding instruments for detecting depression in primary care, ranging from 2 to 28 items in length [7]. Typically these can be recorded as continuous measures of depression severity and also have established cut-points above which the probability of major depression is substantially increased. Scores on these various measures tend to be highly correlated [8], and it is not evident that any measure is superior to the others [7, 8].

In this following section, the PHQ method is being discussed, as it is the one used in project STEADY.

3.1. Patient Health Questionnaire (PHQ)

The PHQ is a ubiquitous instrument for making criteria-based diagnoses of depressive and other mental disorders commonly encountered in primary care. The diagnostic validity of the PHQ has been established in 2 studies involving 3000 patients in 7 obstetrics-gynaecology clinics [9]. At 9 items, the PQH depression scale, also called (PHQ-9), is half the length of many other depression measures, has comparable sensitivity and specificity, and consists of the actual 9 criterion upon which the diagnosis of DSM-IV depressive order is based [10].

Brief Patient Health Questionnaire

This questionnaire is an important part of providing you with the best health care possible. Your answers will help in understanding problems that you may have.

Name_____ Age____ Sex: ☐ Female ☐ Male Today's Date _____

1. Over the last 2 weeks, how often have you been bothered by any of the following problems?	Not at all	Several days	More than half the days	Nearly every day
a. Little interest or pleasure in doing things	☐	☐	☐	☐
b. Feeling down, depressed, or hopeless	☐	☐	☐	☐
c. Trouble falling or staying asleep, or sleeping too much	☐	☐	☐	☐
d. Feeling tired or having little energy	☐	☐	☐	☐
e. Poor appetite or overeating	☐	☐	☐	☐
f. Feeling bad about yourself — or that you are a failure or have let yourself or your family down	☐	☐	☐	☐
g. Trouble concentrating on things, such as reading the newspaper or watching television	☐	☐	☐	☐
h. Moving or speaking so slowly that other people could have noticed? Or the opposite — being so fidgety or restless that you have been moving around a lot more than usual	☐	☐	☐	☐
i. Thoughts that you would be better off dead or of hurting yourself in some way	☐	☐	☐	☐

Figure 3.1.: PHQ-9 Questionnaire.[9]

However, a briefer version of this questionnaire is used in the evening protocols (see section 4.1.2). The so-called (PHQ-2) confines itself to only two questions, the degree of the *loss of interest*, and *down mood*. Nevertheless, the scale is extended to 11 possible values (from 0 to 10).

The PHQ-2 has good sensitivity and specificity for detecting major depression. These proper-

ties, coupled with the brief nature of the instrument, make this tool promising as a first step for screening for adolescent depression in primary care [11].

3.2. Steady project

In this section, the details of the STEADY project are discussed and presented as the base for the following acts and endeavors. Launched by the federal ministry for education and research, and partnered with the Institute of applied computer science, german depression institute (Stiftung Deutsche Depressionshilfe), and adesso SE as the central part of action for the digital part, STEADY project aimed at establishing a digital platform that allows patients to collect data through the course of their ongoing disorders using smartphones and biomedical sensors. These data can then be used to help self-manage those patients and increase the efficiency of medical care. Affective disorders are particularly suitable for this approach because of their recurrent or chronic course, their frequency and severity, and the major deficits and bottlenecks in medical care that exist in this area.

The platform allows these data to be shared between the patients and their treating physicians and psychotherapists so that it could be used in the treatment process. The sharing part happens under two norms, First and by default, all the data gets pseudonymized, and each participating patient gets a numeric ID, so developers and scientists can see and analyze the actual data without knowing to whom it belongs. Secondly, if the patient allowed restricted access to his data from his physician to gain further insights.

The project aim is to inform the patient about changes in his/her symptoms (e.g., incoming depressive episodes) more precisely than through mere self-perception. If certain patterns of the biodata precede affective changes and point to factors that cause them, this can be used for prophylactic interventions and the optimization of self-management.

The duration of the STEADY project was three years, combining three phases of measurements and with slightly different types of sensor systems. In general, relevant data is collected in two forms. First, raw sensor data that comes form smartphones and smartwatches e.g. heart rate, skin conductivity, activity levels, location data. Secondly, morning and evening surveys or as called "protocols" that are filled on a daily basis. These data form the basis for the bachelor thesis (see [4] for more information).

4. Data Preprocessing

In this chapter, the data preprocessing procedures are illustrated. These procedures come in essence as the data collected from the sensors - whether mobile, or smartwatch sensors - are in a format that is not directly readable, nor usable by machine learning algorithms. Also, the raw sensor data and their respective protocols "labels" are collected with two different file structures, this requires an additional final step of merging the two pieces to form the classic dataset format with a set of features and a set of corresponding labels.

4.1. Dataset Overview

The dataset is divided into two main categories; the two have different file structure and a different method of how they are collected. In the following sections, the properties of each category and how it is related to goal of the thesis are discussed.

4.1.1. Sensor Data

Data captured by the different sensors are written to a specific file format, called a "fluff" file. A fluff file contains a specific number of measurements for different sensors e.g., GPS, acceleration, or gyroscope, within a specific time window, e.g., 1 or 2 hours. This unique format was developed by the Institute of applied computer science. These files come with its special java Implementation for basic reading and writing operations.

Listing 4.1: Structure of the fluff file

```
1 public  class FluffMetaData {
2     private final String fluffName;
3     public String[] dataTypes;
4     public int nSensors;
5     public long uxStartTime;
6     public String[] sensorSpecs;
7     public Object[][] data;
8     ..... // Remainder omitted for the sake of brevity
```

No official source of documentation for the .fluff file complete structure could be found, hence, all needed information for this work has to be inferred from the FluffMetaData class. Listing 4.1 shows the bare bone of a .fluff file. It contains a name for the record, this name is the MAC address of the used device concatenated with a Unix timestamp marking the start time for that specific window of measurements. Line 4, "sensorSpecs", is an array that contains information about the sensor types, names, and the units used in each measure. Lastly, "data" contains the actual values of the measurements. For example, figure 4.1 depicts the structure of the

```
sensorSepecs
    sg2_acc :
            unit1=us        tag1=SystemClock
            unit2=m/s^2     tag2=X_Acceleration
            unit3=m/s^2     tag3=Y_Acceleration
            unit4=m/s^2     tag4=Z_Acceleration

data
    sg2_acc : [874449329, 874489289, 874529337, 874569363, 874609391, 874649417]
    sg2_acc : [3.696911, 3.696911, 0.4570291, -1.1916257, 0.22731815, 0.21774687]
    sg2_acc : [-10.863414, -10.863414, -8.573483, -10.408779, -9.521042, -9.497113]
    sg2_acc : [-0.51445687, -0.51445687, 2.2875385, -2.1990042, -0.49292147, -1.806581]
```

Figure 4.1.: Sensor specification

acceleration sensor. It has four components, with each has a name and a measurement unit, and the data array contains the actual measurement values, respectively.

The first array of entries are offset values for each measurement. The actual time that corresponds to each measurement is calculated through adding the respective offset to the Unix start time of the fluff file. Note that offsets are in microseconds, and Unix start time is in seconds. This structure is pretty much the same for each sensor, except for the GPS sensor, its values are recorded without offsets, only Unix timestamps.

4.1.2. Daily Protocols

Every patient had to fill two protocols daily, one in the morning and a comprehensive one at night. These protocols measure different and multiple things; in particular, it contains the PHQ-2 result. Morning protocols are not considered for this work as they contain only encoded entries such as patient Id and timestamp, which is also contained in the evening protocols yet with many more decisive features.

```
"PHQ2_1": "6",   // from 0 to 10
"PHQ2_2": "3",   // from 0 to 10
"alc": "0",      // from 0 to 10
"mood": "6",     // from 0 to 10
"tired": "1",    // from 0 to 10
"period": "false",
"rumination": "6", // from 0 to 10
"socialize": "72", // from 0 to 100
"context": [
    {
        "name": "Einkaufen",
        "context_n_d": "105",
        "hours": ["12"]
    }
],
"daySleep": {
    "DS_TIB": "4", // total sleep in bed
    "hours": []
```

Figure 4.2.: A snippet of an evening protocol

The features set has been extended with key entries found in the evening protocols, so the features set is not confined to only sensor data but expanded to comprise information such as daily alcohol/ cigarettes consumption, mood, and tense levels.

4.2. Dataset Exploration - Processing challenges

The extended dataset comprises the daily values of sensor data, along with key information from the evening protocols, both parts serve as the feature set, and the PHQ-2 results as the labels. One of the most critical aspects of the dataset is that the sensor data are captured at very high frequencies, ranging between 100 kHz-250 kHz. Assuming an average of 200 kHz (5us), nearly 17.10^9 values per day - or 7.10^9 per hour- could be reached. Thinking logically, no depression level will alter throughout some finite number of microseconds. Hence, sensor data are to be compressed into one minute window, meaning one value per sensor each minute. That allows high precision while not losing much information.

The feature set is extended even more with the standard deviation of each compressed range. The main purpose of a standard deviation is to understand how spread out the compressed range is, i.e., high standard deviation would mean a potential loss of precision and low standard

deviation would assure the accuracy of the compression as not much data are lost. This gives even more insights into the compression process and helps increase the quality of the features.

Another issue is the irregularity of the time component. Expected is to have 1440 sensor values per day, that is 24 hours multiplied by 60 minutes per hour. However, the data exhibited many interruptions and missing values that clipped that number to 400-600 values per day. Moreover, there are time gaps that reaches up to 10 days, that means 10 days of no data entirely. These time gaps and interruptions were, due to empty batteries of the devices, broken devices, or the user took of the watch.

Finally, both fluff files and evening protocols are matched with each other depending on the timestamp of each on them. Consequently, adding the results of the PHQ-2 as labels for the complete dataset.

4.3. Preprocessing Pipeline

In essence, there are two main pipelines. The first one is responsible for transforming sensor data files from fluff files to a CSV file. The second one is responsible for reading and converting evening protocols into a CSV file. Transformation processes illustration follows in the next two sections.

4.3.1. Sensor data pipeline

Each fluff file contains all sensor measurements through a given time window. This window varies from half an hour to full two hours. That dictated, firstly, separating the whole fluff files list, then aggregating each sensor values into a single file per sensor to allow further processing. Each patient had on average 2000 fluff files, these files spanned up to 10 months in some cases. After the separating step, four to five files - depending on how many sensors to be extracted - for each patient are generated.
Secondly, each sensor file gets compressed into a one minute window as mentioned earlier (see section 4.2). At this step, the standard deviation of each compressed range is added as a new feature.

Thirdly, each compressed sensor file gets merged with other compressed sensor files to form one CSV file that contains all sensor data. It is very important to know that each measurement in each file has its own specific timestamp, specifically, a Unix Timestamp. This merging

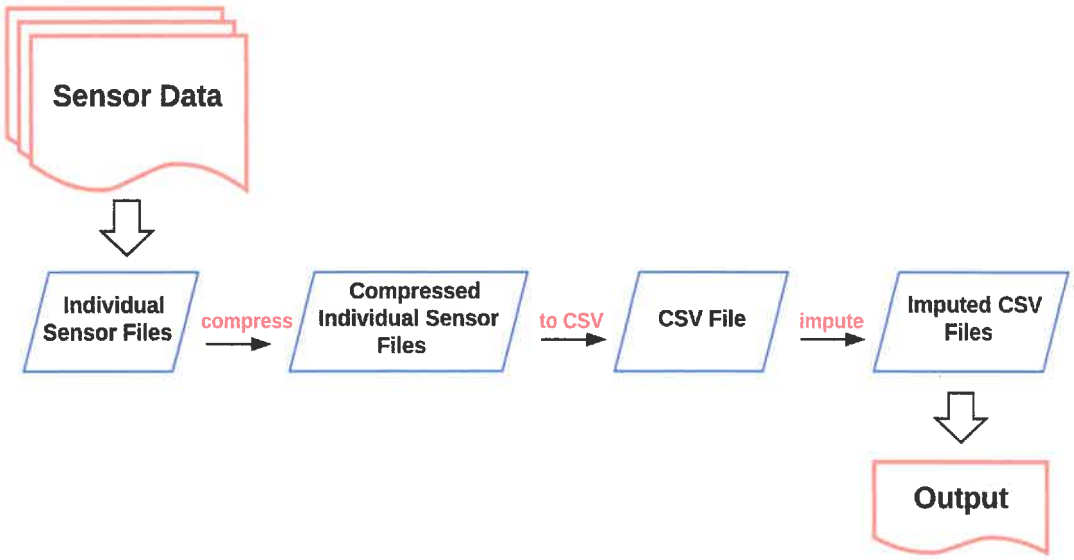


Figure 4.3.: Sensor data processing pipeline

process builds upon those timestamps to match measurements from different sensor files. The exact process goes as follows; the earlier timestamp of all readings is picked form all compressed files, then all readings with equal timestamps - or almost equal with a window of +60 seconds - are also picked and merged together forming one complete measurements line. This loop continues until there is no more data in any of the compressed files.

Finally, an imputation with 'median' strategy is performed on the CSV file to replace any missing values, as machine learning models, particularly NNs, suffer from missing values. Median values of the dataset are saved separately, as any future dataset that will use these models will have to be imputed in the same way as the training data to ensure the same behaviour [12, p. 63].

4.3.2. Evening protocols pipeline

Evening protocols are in JSON format and they are daily filled, that means no compression nor separating is needed here. As shown in figure 4.4, only parsing the evening protocols together with imputing missed values is required.

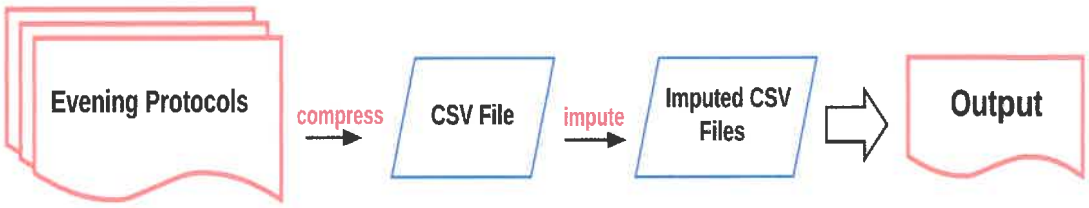


Figure 4.4.: Evening protocols processing pipeline

4.3.3. Sensor data and Evening protocols

Sensor data and Evening protocols are merged for convenient usage by machine learning models. Worth noting here is the time difference between the two files, as in sensor data, the values are between 400-600 value per day, while only one value per day from the evening protocols. That means that the values of the evening protocols had to be replicated along the corresponding days from the sensor data. This transformation happens in this last step, as depicted in figure 4.5.

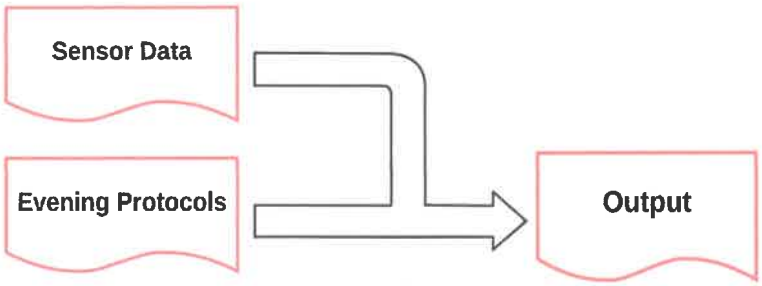


Figure 4.5.: Merging Sensor data and Evening protocols

After this step, a complete imputed dataset is ready for usage by different machine learning algorithms.

5. Machine Learning Approaches

Broadly speaking, supervised machine learning is the computational task of learning correlations between variables in annotated data (the training set), and using this information to create a predictive model capable of inferring annotations for new data, whose annotations are not known (the test set) [54]. Supervised machine learning consists of methods for automatically building a predictive function $f(x) \rightarrow Y$ that maps x (the predictor attributes of an instance), to a prediction Y (the target variable of an instance), given a set of training instances (the training set) represented by tuples (x_i, y_i) where y_i is the target variable and x_i is the vector (typically containing numerical and/or categorical values) encoding the predictor attributes (features) associated with the i-th instance [55].

For the problem in hand, a number of supervised machine learning models are considered and implemented. The models are chosen depending on two criteria:

- Traceability of the model decisions.
- Attention to the temporal dimension of the dataset.

Traceability quality roots back to the thesis' use case being a medical application which requires the used algorithms to be justifiable and explainable, i.e. when the model says that a given patient is depressed with a certain value, it is important to be understood how the model reaches its decisions. The second criterion tries to exploit the idea that there is a relationship between depressive episodes and the changes occur to patients over time. To better illustrate this criterion, it must be explained how data is consumed by classic NNs and most of other machine learning algorithms.

Classic NNs architectures and machine learning algorithms do not expect the dataset to have any order. These algorithms treat the dataset as discrete points of information. Hence they mostly produce better models when the dataset is shuffled [56]. To this models, a depression phase that comes to a given patient after a failed relationship or loosing a job does not mean that there is a relation between it and the actual loss that the patient has suffered. They can

not apprehend any relations between the data points, which in this given case, seems to be pivotal information. A solution to this problem is presented by the models deploying recurrent NNs architectures. These models treat the time sequence as a central feature of the data. This behaviour comes from the inherent way of how RNN models make predictions of a new data point (see section 2.2).

Models in the following sections are implemented and investigated on six patients of a whole of twenty patients for the complete study. The reason for this small number stems from data consideration, i.e. some patients have enough amounts of sensor data but not enough evening protocols (labels), some other patients have enough amounts of evening protocols but not enough sensor data; lastly, some patients have a minimal amount of data for both sensor data and evening protocols. Patients belonging to those three categories have to be ignored and consider only patients with enough data for both sensor data and evening protocols. Consequently, since data from only six patients are available, the work for this thesis confines itself to the evaluation of *personalised* models for each patient. The current amount of data is not sufficient to consider a generalised model that can be trained on multiple patients and predict depression levels on patients that it has not seen before.

5.1. Classical approaches

5.1.1. Random Forest model

As the name suggests, Random Forest (RF) destroys any temporal patterns in the data, as it interacts with the data in an inherently random way. This happens by shuffling the dataset before feeding it to the model to be fitted. Since RF are essentially a bunch of decision trees (called estimators) that are trained on a random subset of the dataset and then averaged to produce a single prediction per forest, the structure of each decision tree (estimator) can be visualised and investigated, and feature importance can be examined. Scikit-Learn offers a very intuitive API for classic machine learning algorithms such as Random Forest. Listing 5.1 shows an example of the implementation of the RF algorithm.

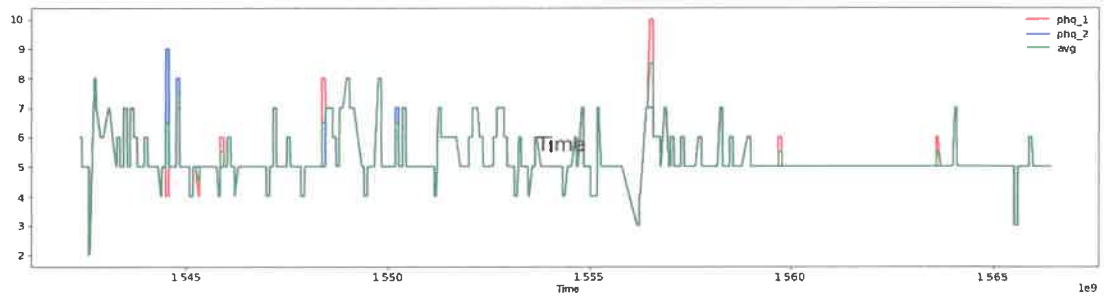
Listing 5.1: Random Forest model

```
from sklearn.ensemble import RandomForestRegressor
rnd_reg = RandomForestRegressor(n_estimators=500,
                                max_leaf_nodes=35, n_jobs=-1,
                                oob_score=True)
rnd_reg.fit(X_train, y_train_1)
```

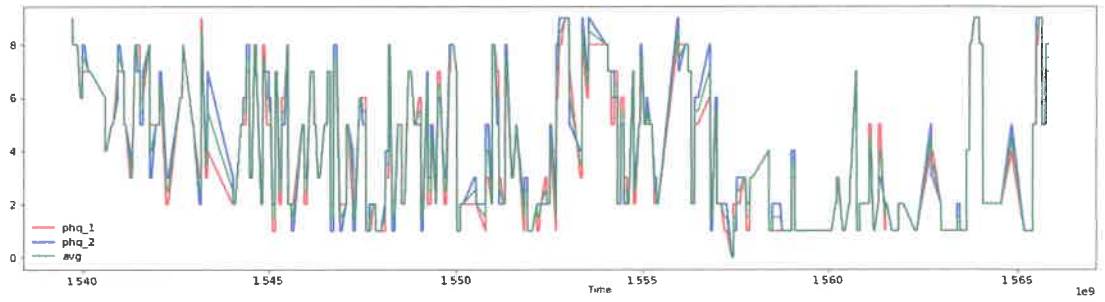
This forest contains 500 separate decision trees, each with up to 35 leaf nodes. "*max-leaf-nodes*" acts a regularisation for the model, as it restricts the decision tree freedom during training. Optimising the hyper-parameter values to increase the performance is done. As a result, with increasing the number of estimators and leaf nodes, the MAE of the models decreases, however, this decrease comes on the cost of longer training time. A tradeoff between the former two aspects has to be done reaching these values. "*y-train-1*" represents the labels of the PHQ-1 question, that means that to predict the PHQ-2 question, another model that is trained on the PHQ-2 labels has to be used. Plots in figure 5.1 shows the values of the PHQ-1 and PHQ-2 over the complete time window for two patients. In such cases, the models' performance on PHQ-1 labels is very similar, almost identical, to the models' performance on PHQ-2. For the sake of brevity, only results from PHQ-1 models are considered.

Estimator structure

Figure 5.2 depicts a part of a single decision tree from patient ST-1814523348. Individual decision trees can be interpreted easily by simply visualizing the tree structure (see appendix A for the full tree).



(a) patient ST-1814523348



(b) patient ST-1871742707

Figure 5.1.: PHQ-1 and PHQ-2 values

Scikit-learn uses a Regression Tree algorithm to train such decision trees. The algorithm works by first splitting the training set into two subsets using a single feature j and a threshold t_m (e.g. "day-sleep" ≤ 235.0). It chooses this value based on a search of the pair $\theta = (j, t_m)$ that produces the purest subsets based on their weighted size (see Scikit-learn docs for the complete mathematical proof [19]).

Feature Importance

Random Forest can also estimate how the features are affecting the result of the prediction. This estimate is very helpful in the feature engineering part of the data preprocessing step as it provides information of which features contribute more to the model performance and which features could be ignored or removed. Table 5.1 comprises the average of the each feature's importance of the patients' models.

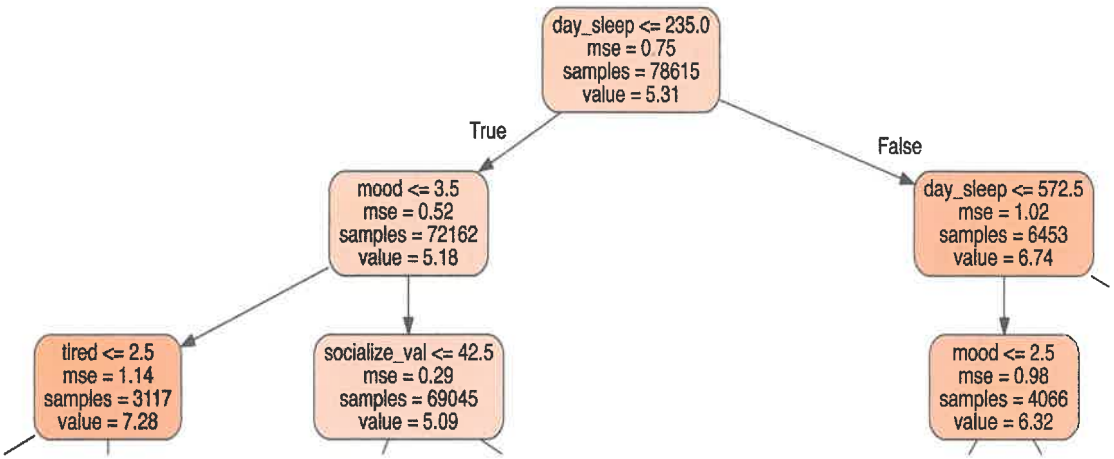


Figure 5.2.: Snippet of estimator 5 of patient ST-1814523348

accelerationX	0.0001	alcohol	0.0049
accelerationX-std-dev	0.0	cigarates	0.0100
accelerationY	0.0001	mood	0.2225
accelerationY-std-dev	0.0	tense	0.0796
accelerationZ	0.0	tired	0.0634
accelerationZ-std-dev	0.0	period	0.0019
gyroscopeX	0.0001	rumination	0.3712
gyroscopeX-std-dev	0.0	socialize	0.1205
gyroscopeY	0.0	socialize-value	0.1331
gyroscopeY-std-dev	0.0	sport-time	0.0150
gyroscopeZ	0.0	work-time	0.0576
gyroscopeZ-std-dev	0.0	heart-rate	0.0010
heart-rate-std-dev	0.0	PlethysmogramGreen	0.0017
ple-std	0.0025	AirTemperature	0.0105
AT-std	0.0016	AirPressure	0.0051
AP-std	0.0001		

Table 5.1.: Features' importance

Clearly, the sensors information contributes very poorly to the model prediction capability (see figure 5.3), on the other hand, features like mood, rumination, and socialize, seem to be quite decisive towards the prediction of the depression.

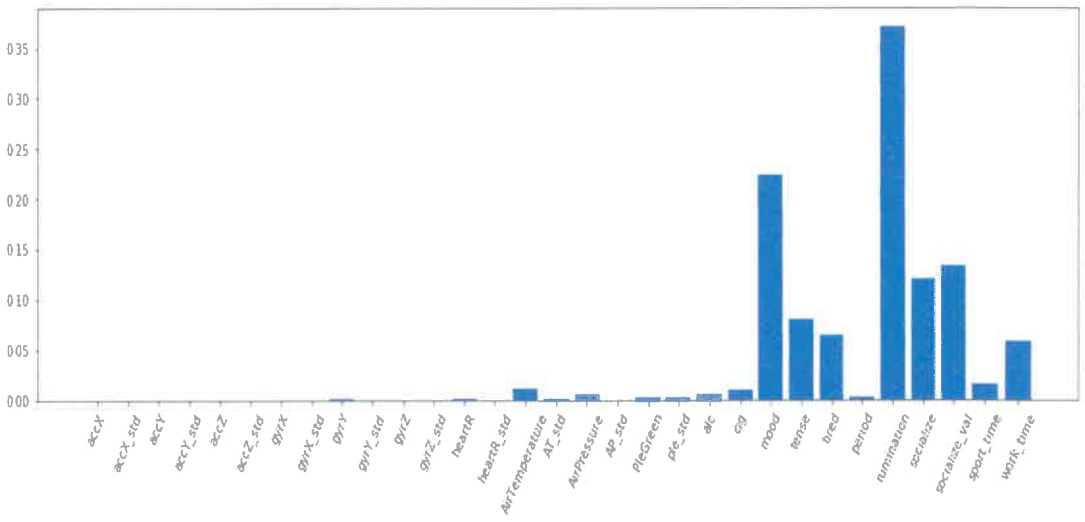


Figure 5.3.: Random Forest features' importance

Prediction results

Table 5.2 contains the results of the Random Forest trained on the PHQ-1 labels for the six patients. The used metric is the mean absolute error (MAE).

Patient	Train-mae	Test-mae	cross-validation average	cross-validation std-dev
ST-1505558269	0.5592	0.5567	0.5600	0.0039
ST-1814523348	0.1968	0.1958	0.1972	0.0023
ST-1233329802	0.4892	0.4920	0.4883	0.0059
ST-1441993385	0.9697	0.9690	0.9614	0.0106
ST-1871742707	0.3479	0.3465	0.3466	0.0053
ST-1946093440	0.3805	0.3768	0.3817	0.0071

Table 5.2.: Random Forest results

One way to evaluate the Random Forest models is through cross-validation, this splits the training set into 10 distinct subsets (called folds) then it trains and evaluates the model 10 times, picking a different fold for evaluation every time and training on the other 9 folds. The result is an array containing 10 evaluation scores. This is done to make sure there is no overfitting of the models. Results are attached to table 5.2.

5.1.2. Support Vector Regression model

It is well known that SVM generalization performance depends on a good setting of meta-parameters parameters C, ε and the kernel parameters [17]. For this reason, a search for the best fitting meta-parameters for each patient's model is implemented by Scikit-learn "GridSerachCV" utility (see listing 5.2).

Listing 5.2: SVR grid search with cross validation

```
param_grid = {'epsilon':[0,0.1,0.2,0.5,1],
              'tol':[1e-1, 1e-3,1e-5],
              'C':[1, 1.5, 10, 50]
              }
svr = LinearSVR(random_state=42)
regressor_st = GridSearchCV(svr, param_grid, n_jobs=-1,
               cv=3, scoring='neg_mean_absolute_error')
regressor_st.fit(X_train_st_scaled, y_train_st_1)
regressor_st.best_params
# example answer => {'C': xx, 'epsilon': xx, 'tol': xxx}
best_model = regressor_st.best_estimator_
best_model.coef_
# example answer => [-0.02477377,... ,0.19853483, -0.3067323 ]
```

This "*param_grid*" variable tells Scikit-learn to evaluate $5 \times 3 \times 4 = 60$ combinations of ε, "tol", and C values, then train those 60 models 3 times each (cv=3). At the end, variable "*best_params_*" contains the best parameters chosen based on the mean absolute error (MAE) score. For parameters C, and ε meaning, see section 2.2. "tol" parameter specifies a tolerance for the stopping criteria in the optimisation equation in 2.3 [18, 20].

Models' weights

The set of the learned weights w (regression coefficients) is available under the attribute of "*coef_*". These weights give more insights into how the model reaches its decisions and how important is each feature in the depression detection. Table 5.3 contains the absolute average of each feature's weight in the patients' models. Figure 5.4 depicts a bar chart of the values in table 5.3 to better illustrate the results.

accelerationX	0.0140	alcohol	0.0312
accelerationX-std-dev	0.0192	cigarates	0.0299
accelerationY	0.0078	mood	0.3431
accelerationY-std-dev	0.0010	tense	0.2113
accelerationZ	0.0196	tired	0.0118
accelerationZ-std-dev	0.0010	period	0.0235
gyroscopeX	0.0099	rumination	0.8003
gyroscopeX-std-dev	0.0223	socialize	0.1351
gyroscopeY	0.0052	socialize-value	0.1712
gyroscopeY-std-dev	0.0179	sport-time	0.0974
gyroscopeZ	0.0046	work-time	0.0165
gyroscopeZ-std-dev	0.0092	heart-rate	0.0035
heart-rate-std-dev	0.0032	PlethysmogramGreen	0.0132
ple-std	0.0215	AirTemperature	0.0429
AT-std	0.0526	AirPressure	0.2565
AP-std	0.1138		

Table 5.3.: SVR models' absolute average weights

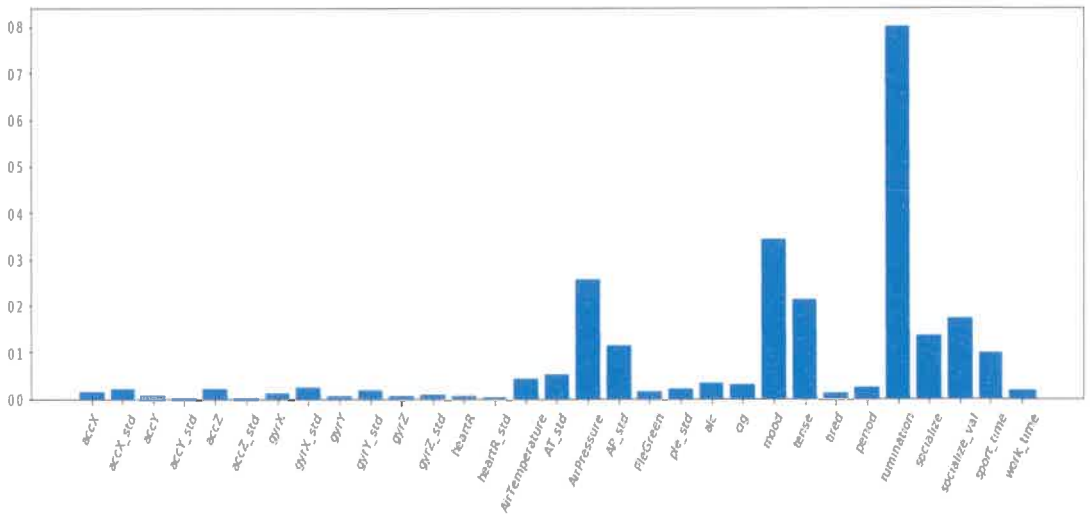


Figure 5.4.: SVR models' weights

Although the average weights do not present a specific behavioural information per se, as it does not correspond to a distinct model, it serves as a general indication of the significance that is given by the LinearSVR algorithm to each feature in the dataset.

Prediction results

Table 5.4 contains the results of the LinearSVR model trained on the PHQ-1 labels for the six patients. The used metric is the mean absolute error (MAE).

Patient	Train-mae	Test-mae
ST-1505558269	0.9949	0.9987
ST-1814523348	0.3795	0.3810
ST-1233329802	1.0545	1.0559
ST-1441993385	1.4546	1.4592
ST-1871742707	0.6953	0.6924
ST-1946093440	0.8008	0.8022

Table 5.4.: Linear support vector regressor results

5.2. Neural Networks

5.2.1. Time series data preparation

To get started with RNNs training, the dataset has to be preprocessed to be in a format that is compatible with such networks, and thus:

- Since each time series in the data is on a different scale (e.g. mood, tense, tired takes a value between 0 and 10, while socialize and socialize-val takes a value form 0 to 100). These values have to be normalized to a similar scale.
- Creating a generator that takes the current dataset and yields batches of data from recent past, along with a target value of the PHQ-1/ PHQ-2.

Whenever time matters in the dataset, it is recommended to store it in a 3D array (samples, time, features) with an explicit time axis. Each sample can be encoded as a sequence of vectors (2D array) [51, p. 35]. This generator yields batches of size (time, features), where each batch represents a day and "time" are the data points in this day with different values of the "features". To abide by the previous structure, data points has to be divided into equidistant samples, i.e. all

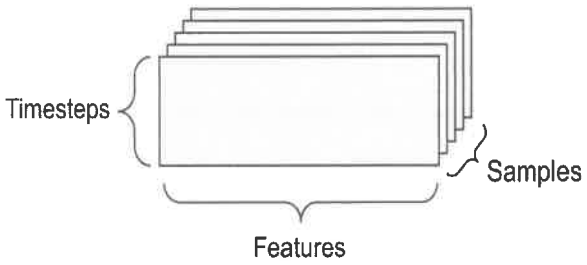


Figure 5.5.: A 3D timeseries data array [51, p. 35]

batches (days) have to have the same number of data points. Since the datasets are completely irregularly spaced .i.e. a typical day comprises between 400-600 sample, a simplification has to be made on the dataset. This simplification makes the median of samples per day the default value for each day in the dataset, then divide the whole length of the dataset by this number to get the number of batches (days) for the dataset. This implementation is shown in listing 5.3.

Listing 5.3: Evenly spacing function

```
def to_evenly_spaced(df):
    date = df['time'].apply(lambda x: get_readable_date(x))
    df.insert(loc=1, column='date', value=date)
    median = np.trunc(df.groupby('date').size().median())
    df.drop(['date'], axis=1, inplace=True)
    day = np.trunc(len(df.index) / median)
    new_last_index = int(median * day)
    df.drop(['time'], axis=1, inplace=True)
    df = df.iloc[:new_last_index]
    return df.drop(['phq_2'], axis=1),
           df.drop(['phq_1'], axis=1),
           int(median)
```

Function in listing 5.3 returns a copy of the data with labels as the PHQ-1 and the another copy with labels as PHQ-2. This makes training separate models easier and more straight-forward. After this step, the previous generator can be used to yield batches for training, validating, and testing. The exact formulation of the task for RNNs is as follows: given data as far back as "*look-back*" timesteps and sampled every "*steps*" timesteps, "*delay*" timesteps can be predicted (see section 2.2). Following list comprises all parameters passed to the generator:

- *data* normalized data.
- *look-back* how many timesteps back the observations should go.
- *delay* how many timesteps in the future the target should go.
- *min-index* and *max-index* Indices in the *data* array that delimit which timesteps to draw from, this is how data is segmented into train, test, validate sets.
- *shuffle* whether to shuffle the data or not.
- *batch-size* number of samples per batch.
- *step* the period, in timesteps, at which the data is sampled.

Listing 5.4 depicts an example of the the generator usage.

Listing 5.4: Generator example of patient ST-1505558269

```
# for phq-1
train_gen_1 = generator(float_data_1,
                        look_b=look_back,
                        delay=delay,
                        min_index=0,
                        max_index=142884,
                        shuffle=False,
                        step=step,
                        batch_size=batch_size)
val_gen_1   = generator(float_data_1,
                        look_b=look_back,
                        delay=delay,
                        min_index=142885,
                        max_index=173124,
                        step=step,
                        batch_size=batch_size)
test_gen_1  = generator(float_data_1,
                        look_b=look_back,
                        delay=delay,
                        min_index=173125,
                        max_index=195804,
                        step=step,
                        batch_size=batch_size)

train_steps = 142884//batch_size                    # = 189
val_steps   = (173124 - 142884)//batch_size         # = 40
test_steps  = (195804 - 173124)//batch_size         # = 30
```

In this case (listing 5.4), the dataset length is 195804 timesteps with 756 timesteps per batch (that is the number returned by the function "*to_evenly_spaced()*"). This results in 259 batches (days) of data. This 259 is then divided into train, validation, and test data by specifying the corresponding index. The "*train_gen_1*" takes from index 0 to index 142884, with each batch (day) 756 timesteps, these are the first 189 days for training. In the same manner, "*val_gen_1*"

takes next 40 days for validation and "*test_gen_1*" takes the last 30 days as test data to evaluate the model upon.

The numbers "*train_steps*", "*val_steps*", and "*test_steps*" specifies the how many steps to draw from the corresponding generators, respectively, to see the entire sets. Finally, the generators yield a tuple (sample, targets), where *samples* is one batch of input data and *targets* is the array of PHQ-1/PHQ-2 labels. These tuples are fed directly after to the NNs models to start training.

5.2.2. Baseline approach

Before starting with NNs, it is recommended to try a simple, common-sense approach that serves as a sanity check for the NNs' models [51, p. 212]. The performance of the NNs is then compared to such metric (also called naive model) to evaluate their results. This common-sense approach predicts the depression level for a given day to be the same as the day before and calculates the mean absolute error (MAE) of the result.

Listing 5.5 depicts the naive model method [51, p. 213]. It takes an instance of a generator, in this case, it is an instance of the *val_gen_1*, which is the validation set generator with PHQ-1 as the targets (labels), then, it loops over it for *val_steps* and calculates the MAE for each loop and, finally, prints the mean value of the MAE vector.

Listing 5.5: Naive model method

```
def evaluate_naive_model(v_gen):
    batch_maes = []
    for step in range(val_steps):
        samples, targets = next(v_gen)
        preds = samples[:, -1, 1]
        mae = np.mean(np.abs(preds - targets))
        batch_maes.append(mae)
    print(np.mean(batch_maes))
```

Table 5.5 lists the naive model MAE scores with PHQ-1 and PHQ-2.

Patient	PHQ-1	PHQ-2
ST-1505558269	1.3140	1.2567
ST-1814523348	1.0044	1.0003
ST-1233329802	1.1344	0.9302
ST-1441993385	1.4569	1.6850
ST-1871742707	0.8324	0.8298
ST-1946093440	1.2485	1.1235

Table 5.5.: Naive model MAE scores

The similarity of PHQ-1 and PHQ-2 results of the naive model (see table 5.5) comes from the fact that PHQ-1 and PHQ-2 values are almost identical in the patients' data. This correlation (see figure 5.1) supports the decision of considering only results of PHQ-1 question, and in the meantime, produce accredited results and conclusions.

5.2.3. GRU Layer model

Gated recurrent unit (GRU) is considered to be a variation of the LSTM. Both are designed to solve the vanishing gradient problem, which allowed modeling long sequences of data and is considered now as standard for the RNNs (see section 2.2). Models in listing 5.6, 5.7 are inspired from the models introduced by Chollet in his book "Deep learning with Python" solving similar time series problems, and they showed excellent learning capabilities with very low losses[1]. The model in listing 5.6 consists of one GRU layer as input, followed by a one neurone layer (regression problem). RMSprop as an optimizer and mean absolute error (MAE) as the loss function that is to be minimized.

Listing 5.6: GRU layer model

```
model = Sequential()
model.add(layers.GRU(32, input_shape=(None, float_data_1.shape[-1])))
model.add(layers.Dense(1))
model.compile(optimzer=RMSprop(), loss='mae')
history = model.fit(train_gen_1,
                    steps_per_epoch=train_steps,
                    epochs=20,
                    validation_data=val_gen_1,
                    validation_steps=val_steps)
```

[1]See Francois Chollet, "Deep learning with Python" chapter 6 [51, p. 215].

The model takes an instance of "*train-gen-1*" which is an instance of the training generator (see section 5.2.1) with labels of the PHQ-1, together with validation generator "*val-gen-1*", and trains the model for 20 epochs.

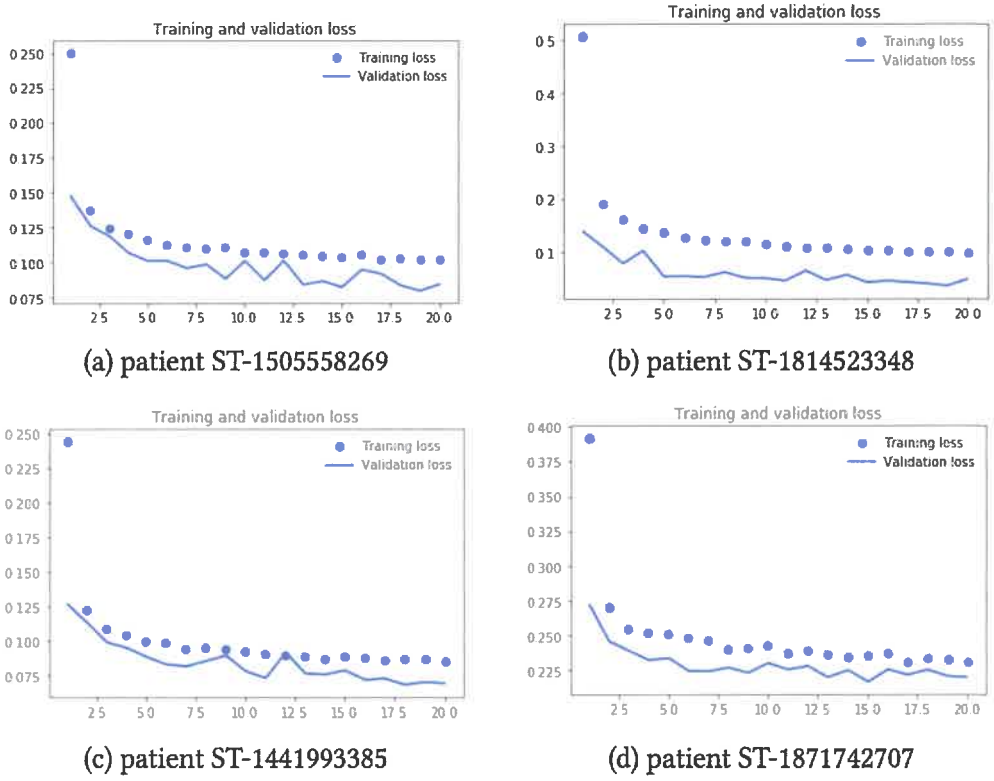
Table 5.6 presents the scores of the PHQ-1 models on the test-sets.

Patient	Test-mae	Time-per-epoch (avg) in s
ST-1505558269	0.0590	110
ST-1814523348	0.0528	130
ST-1233329802	0.1709	60
ST-1441993385	0.0609	85
ST-1871742707	0.1781	45
ST-1946093440	0.2563	50

Table 5.6.: GRU layer model results

Plots in figure 5.6 depict the training and validation losses through 20 epochs for four patients. It is clear how the training and testing losses decrease steadily each epoch, which is a sign for the stability of the model during training time. However, the models exhibit a certain degree of underfitting, this is indicated by the validation loss being lower than the training loss. Underfitting occurs when the model is too simple to learn the underlying structure of the data. A remedy to this problem could be selecting a more powerful model, with more parameters, or reduce the constraints on the model (e.g. reduce the regularization hyperparameters) [12, p. 29].

On the other hand, underfitting could be alleviated by simply training for more epochs, meaning that maybe the point where the two losses lines intersect and the validation loss increases lies ahead a certain number of epochs. Therefore, each model is trained with additional 20 epochs. Nevertheless, that did not solve the underfitting problem. This behaviour is clear in the case of patients ST-1505558269 and ST-1441993385 (see subplots (a) and (c) in figure 5.6) as the losses already reached nearly 0 MAE, yet, still the validation loss lower than the training.



(a) patient ST-1505558269

(b) patient ST-1814523348

(c) patient ST-1441993385

(d) patient ST-1871742707

Figure 5.6.: Training and validation loss (GRU layer model)

5.2.4. 1D Convolutional with GRU Layer model

As the previous model (see section 5.2.3) suffered from underfitting, a more complex model with more parameters is needed to fit the data better. Thus, the model in listing 5.7 is realized. It consists of the five following layers:

- *Conv1D* 1D convolutional layer as input layer.
- *MaxPooling1D* this is the equivalent of 2D pooling operation for images, it extracts 1D patches (subsequences) from the input and outputs the maximum value. This is used to capture strong patterns in the data and also help prevent overfitting by reducing the length of the 1D input (down-sampling).
- *Conv1D* 1D convolutional layer.
- *GRU* GRU layer with 0.2 dropout and 0.2 recurrent-dropout.
- *Dense* one neurone dense layer as output layer.

Listing 5.7: 1D Convolutional with GRU Layer model

```
model = Sequential()
model.add(layers.Conv1D(32, 5, activation='relu',
                        input_shape=(None, float_data_1.shape[-1])))
model.add(layers.MaxPooling1D(3))
model.add(layers.Conv1D(32, 5, activation='relu'))
model.add(layers.GRU(32, dropout=0.2, recurrent_dropout=0.2))
model.add(layers.Dense(1))
model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit(train_gen_1,
                    steps_per_epoch=train_steps,
                    epochs=10,
                    validation_data=val_gen_1,
                    validation_steps=val_steps)
```

This model (see listing 5.7) uses RMSprop as an optimizer and mean absolute error (mae) as the loss function. However, it runs for only 10 epochs, as it is evident that it overfits the dateset if the epochs number increased (see figure 5.7). To address the problem of overfitting, two kinds of dropout are added to the GRU layer. Dropout refers to a technique where input units are randomly zeroed out. This technique helps the model resist overfitting and be more robust. Firstly, "*dropout*" parameter is a float specifying the dropout rate for input units of the layer,

secondly, "*recurrent-dropout*" specifies the dropout rate of the recurrent units of the network.

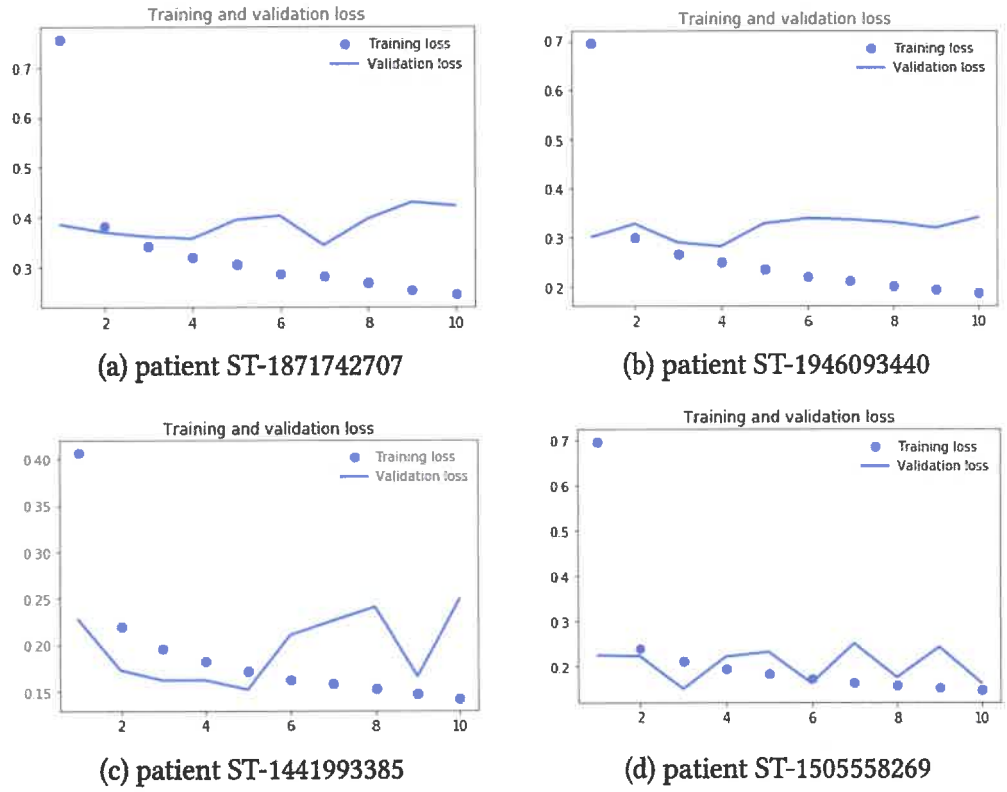
The 1D convent acts as a preprocessing step before the RNN; this technique combines the speed of the convnets with the order-sensitivity of RNNs and helps in dealing with long sequences with thousands of steps. The convnet will turn the long sequence into much shorter sequences of higher-level features that will then be the input to the RNN part of the network.

Table 5.6 presents the scores of the PHQ-1 models on the test-sets.

Patient	Test-mae	Time-per-epoch (avg) in s
ST-1505558269	0.2180	70
ST-1814523348	0.0570	52
ST-1233329802	0.3772	42
ST-1441993385	0.1921	60
ST-1871742707	0.1921	33
ST-1946093440	0.3404	32

Table 5.7.: 1D Convolutional with GRU Layer model results

Plots in figure 5.7 depict the training and validation losses through 20 epochs for four patients. Through epochs, it is noticeable that at epochs 4-5 the models start to overfit and the losses lines begin to diverge. There are many techniques that could be used to extract the best models (i.e. models with lowest MAE) before actually overfitting the data. One of which is the "Early stopping" technique, and it means stop the training as soon as the validation loss reaches a minimum [12, p. 141].



(a) patient ST-1871742707

(b) patient ST-1946093440

(c) patient ST-1441993385

(d) patient ST-1505558269

Figure 5.7.: Training and validation loss (1D Convolutional with GRU Layer model)

6. Evaluation of the results

6.1. Traceability and Explainability

Linear SVR and Random Forest models could be categorized as *White-box* machine learning approaches. As with the information from the feature importance (see section 5.1.1) and the models' weights (see section 5.1.2), it can be explained how the models behave and how they produce predictions and what are the influencing variables. On the other hand, NNs models are often criticized to be non-transparent and their predictions are not traceable by humans [57]. For example, the SVR models have 32 trainable weights. Those are the weights that control the behaviour of the model (see equation 2.1), and they correspond to one weight per feature. Conversely, the 1D-convnet GRU model (see section 5.2.4) and the GRU model (see section 5.2.3) have 16.673, and 6.369 trainable weights, respectively. The 1D-convnet GRU model parameters number corresponds to the model's architecture consisting of the following layers:

- *Conv1D* 32 filters and 32 input width (features) and a kernel_size of 5. The formula is $((m \times n) + 1) \times k$, with m as the filter's width, n as the filter's height, and k as the number of fllters. This results in $((32 \times 5) + 1) \times 32 = 5152$ parameters
- *MaxPooling1D* no parameters.
- *Conv1D* same 5152 parameters as previous conv1D layer.
- *GRU* 32 neurone (units) and 32 input weights. The formula is $3(n^2 + mn + 2n)$ with m as input dimension and n as output dimension. This results in $3(32^2 + 32^2 + 64) = 6336$
- *Dense* 32 wights as the previous layer output and one bias term resulting in 33 weights.

These parameters adds up to $5152 + 5152 + 6336 + 33 = 16.673$. Adding only parameters from the last two layers results in 6.369 weights for the GRU layer model. These weights acts as knobs that control the behaviour of an NN and its performance. While NNs models produce far better results -in terms of the MAE scores- than Random Forest and Support Vector machines (see table 6.3), this comes at the price of the models being untraceable and unexplainable.

6.2. Relations between features and depression prediction

Investigating the results of sections 5.1.1 and 5.1.2 shows how the models' predictions are affected by certain features in the dataset. Features like rumination, mood, tired, and tense have a direct influence on the model predictions, while features like acceleration and gyroscope hardly affect the model behavior. It can be safely said that the sensor data does not play a valuable role in the depression detection context. However, this conclusion does not exclusively mean the unusefulness of sensor data per se. Instead, it should point to the issues surrounding the sensor data acquisition process. In such a process, the sensor data may be distorted or have a substantial tolerance. Also, many sensors need to be checked and calibrated regularly, and there is no way to make sure that this happens. Conversely, the process for acquiring other features like alcohol and cigarette consumption or mood and tense levels is very clear and unambiguous and thus provide more reliable information.

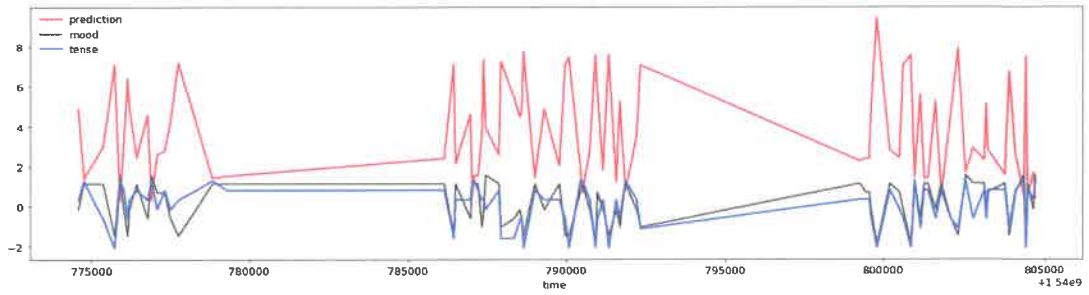
Figure 6.1 shows a clear polarized relation between depression levels and mood or tense values, and colour maps in figure 6.2 shows the relation between depression prediction levels and rumination and tired values. With tired value decreasing and rumination value increasing, depression levels reache nearly maximum values. Note that colour bars in figure 6.2 have different ranges due to pandas[1] drawing the colour index depending on the values present in the predictions vector. So for subplot (a), the predictions vector does not include any values exceeding the 8 range. Therefore, the index is capped at that value.

Such information about the correlation between different features and their influence on the depression levels could prove helpful in the hands of the treating psychiatrists as it would draw their attention to clear points that directly affect the patient's depression levels. For instance, patients seem to have low depression levels when their tired levels are high, and rumination values are low (see figure 6.2), which could suggest that doing a daily exercise or practicing a sport would help them prevent ruminating and curb depression levels.

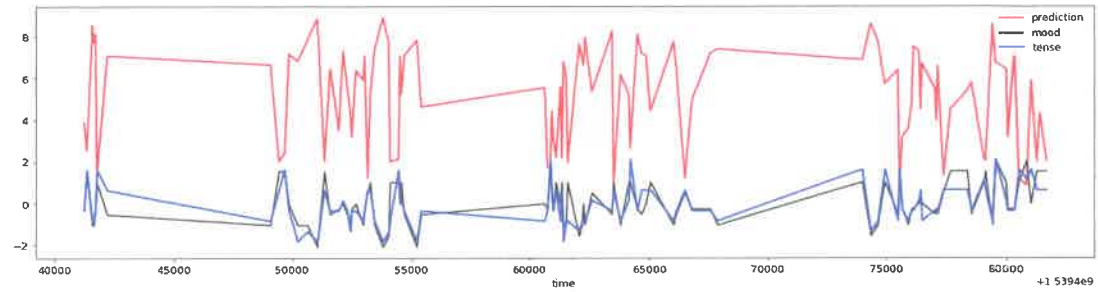
6.3. Mean Absolute Error scores

Mean Absolute Error (MAE) is an indication of the average deviation of the predicted values from the corresponding observed values and can present information on long term performance of the models; the lower MAE the better is the long term model prediction [59]. The scores in table 6.1 are evaluated on the test-sets on their respective patients. Test-sets are the last 30 days

[1] pandas is a software library written for the Python programming language for data manipulation and analysis.

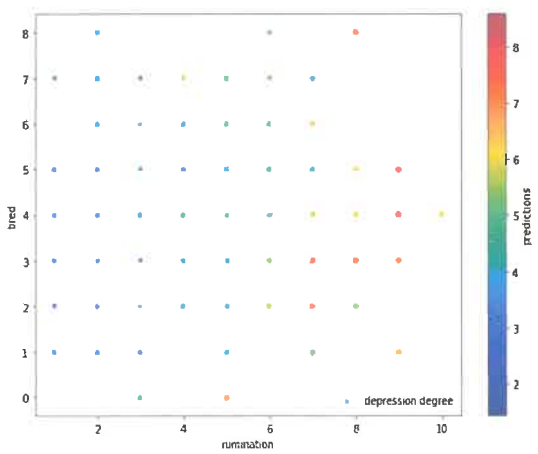


(a) patient ST-1871742707

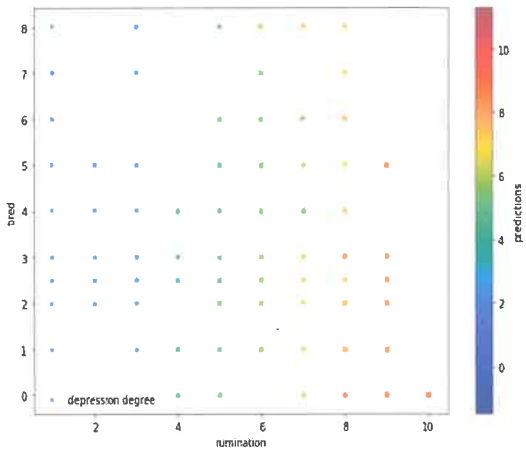


(b) patient ST-1946093440

Figure 6.1.: Depression prediction levels vs mood and tense values



(a) patient ST-1505558269



(b) patient ST-1946093440

Figure 6.2.: Depression prediction levels, blue is not depressed, red is depressed

of the of patients records in the case of the NNs models, as the dataset is grouped into days for the NNs, and 25% of the records for the Random Forest and the SVR models, as the data is regarded as discrete entries. GRU models clearly outperform other models (see figure 6.3), with MAE scores reaches down to nearly 0.05 loss in the case of patients ST-..269 and ST-..348, directly after come the 1D-convnet-GRU models with slightly higher error scores. NN-baseline approach (left-most column in table 6.1) produces error scores that are an order of magnitude higher than their NN counterparts, which supports the credibility and the robustness of the NN models. In the third place comes the Random Forest models with higher error scores, ending with the SVR as the model with the highest error scores across all patients.

Patient	RF-mae	SVR-mae	GRU-mae	1D-CN-GRU-mae	NN-baseline
ST-1505558269	0.5592	0.9987	0.0590	0.2180	1.3140
ST-1814523348	0.1968	0.3810	0.0528	0.0570	1.0044
ST-1233329802	0.4892	1.0559	0.1709	0.3772	1.1344
ST-1441993385	0.9697	1.4592	0.0609	0.1921	1.4569
ST-1871742707	0.3479	0.6924	0.1781	0.1921	0.8324
ST-1946093440	0.3805	0.8022	0.2563	0.3404	1.2485

Table 6.1.: MEA of all models

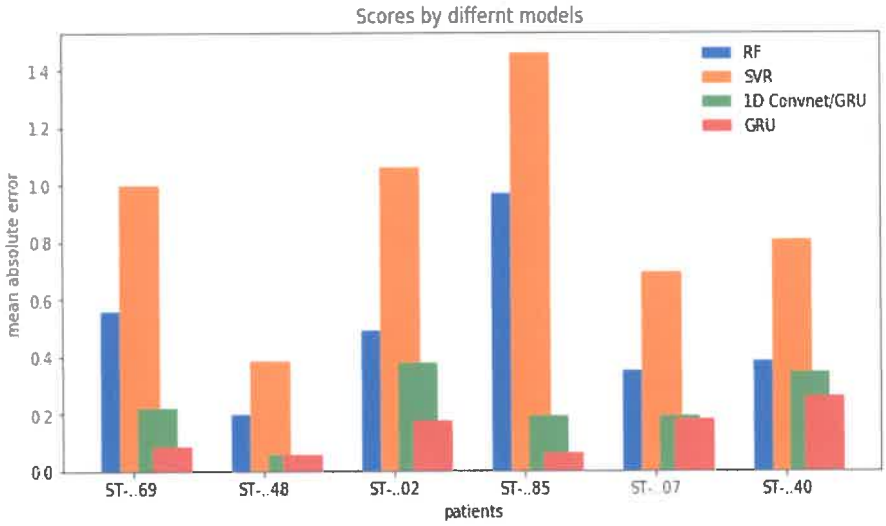


Figure 6.3.: MEA of all models

6.4. Neural Networks Training Time

Random Forest and linear SVR models training time varies linearly with the size of the dataset reaching a maximum of 3 minutes. On the other hand, NN models take longer times reaching up to 43 minutes in the case of the GRU layer model for patient ST-..348. Table 6.2 contains average training time per epoch for NN models. Although the 1D-convnet-GRU model has a more complex architecture than the GRU layer, it takes nearly half the time for training. This is due to the 1D convolutional part that downsamples the input before feeding it to the GRU layer, enabling the network to consume large inputs efficiently. However, lower training time comes with higher error scores (see figure 6.3). Such trade-offs are inevitable and have to be considered thoroughly before opting for a decision.

Patient	GRU	1D-convnet-GRU
ST-1505558269	110	70
ST-1814523348	130	52
ST-1233329802	60	42
ST-1441993385	85	60
ST-1871742707	45	33
ST-1946093440	50	32

Table 6.2.: NNs average training time per epoch in s

7. Conclusion

In this thesis, several prominent machine learning methods are investigated for depression prediction on the basis of patients own data. That means each patient has a dedicated model for the prediction of his/her depressive episodes over time. Depressed individuals who wait for three months to get an appointment with psychiatrists, not only suffer from constant psychological damage but also bear multiple risks such as losing a job or a partner or suicide in some cases. To those individuals, a predictive model with well-being metrics capabilities has paramount importance.

7.1. Achievements and Overall Discussion

The empirical findings are presented thoroughly in chapter 6. In this section some observational notes are synthesized to address the thesis's objectives.

Machine Learning (ML) is a very suitable technique for depression detection applications. Depending on the data structure and acquisition process, multiple approaches could be adopted. The latter assertion is underpinned with literary research conducted in the form of state of the art chapter (see chapter 2). However, to be able to apply ML approaches, the input data has to be in proper shape and structure. Therefore, chapter 4 presents a complete pipeline for merging sensor data and evening protocols and transforming them into a ready-to-use dataset. The objective of this thesis was to analyze the personal health data of a number of patients and develop an ML model that can predict their respective depression levels. For that objective, four ML approaches are implemented and evaluated, with two RNNs models being the front-runners with error scores reaching down to 0.05 MAE. Due to the thesis's use-case being medical application, Random Forest and linear SVR models are considered for their nature being *White-box* models that can be explained and interpreted. Moreover, information about specific features directly influencing depression levels is discovered, giving the treating psychiatrists specific points form which they can approach the patient's depression treatment process and providing them with valuable patient's historical data.

7.2. Future Work

Based on the results made in this thesis, further work in the following directions could be of interest. Regarding data quality, a more consolidated analysis of the data acquisition process could be useful to improve data quality and integrity. Moreover, the work of David C Mohr [60] provides promising approaches to detect correlations between depressive symptoms and mobile sensor data. Since the data preprocessing contributed to an essential part of the success in depression prediction, it is worthwhile to repeat the experiments by permuting the preprocessing methods and their parameters (e.g., try out different variance thresholds or other missing value imputation techniques). Regarding machine learning approaches, several different models and architectures could be exploited. The work of Yasin and Emre [61] provides approaches to employ LSTM networks and mobile sensors to predict stress levels. The work on GRU layers in sections 5.2.3, 5.2.4 could be extended to try different hyperparameter optimization techniques to push the network performance further, and try other GRU based architectures.

Appendices

A. Estimator Structure

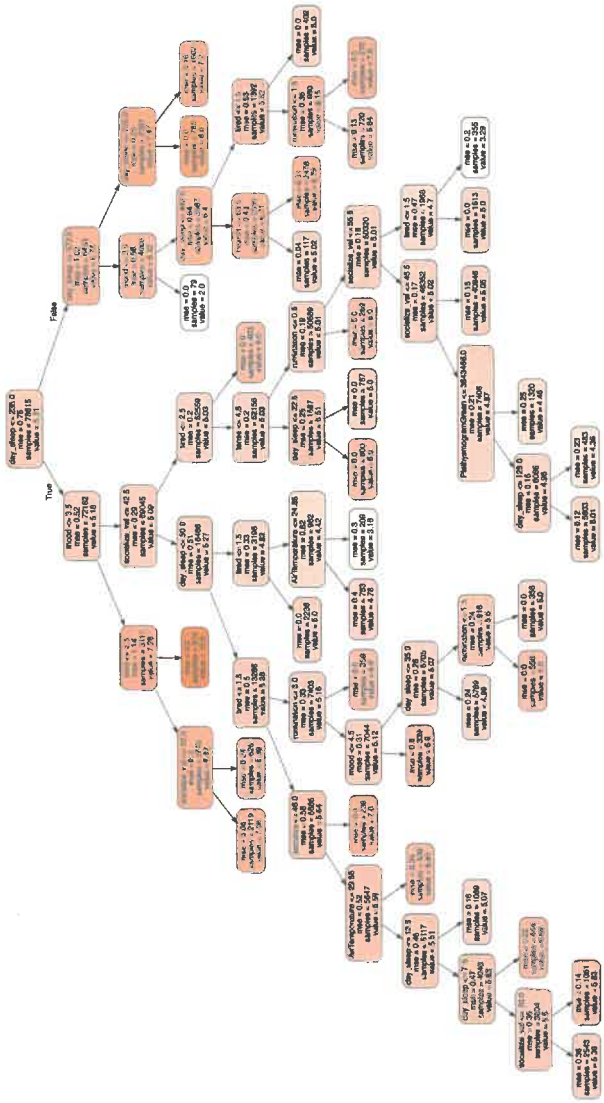


Figure A.1.: Estimator 5 from Random Forest of patient ST-181452334

Bibliography

[1] https://www.dw.com/en/53-million-germans-suffer-from-depression-each-year/a-46506088, available on 16.03.2020.

[2] https://www.euractiv.com/section/health-consumers/news/german-economy-burdened-by-growing-rate-of-depression/3/, available on 16.03.2020.

[3] https://www.morgenpost.de/web-wissen/gesundheit/article210125769/So-funktionieren-die-Sprechstunden-der-Psychotherapeuten.html. available on 03.03.2020

[4] https://www.steady-projekt.de/overview/, available on 16.03.2020.

[5] Keller, M. B. (1994). Depression: A long-term illness. *British Journal of Psychiatry*

[6] Benazzi F,. (2006). Various forms of depression

[7] Mulrow CD, Williams JW, Gerety MB, Ramirez G, Montiel OM, Kerber C (1995). Case-finding instruments for depression in primary care settings. Ann Intern Med. 122:913-21

[8] Keller M. B., Kocsis JH, Thase ME, et a1 (1998). Maintenance phase efficacy of sertraline for chromic depression: a randomised controlled trial. JAMA. 280:1665-72.

[9] Spitzer RL, Kroemke K, Williams JBW(1999). Patient Health Questionnaire Study Group. Validity and utility of a self-report version of PRIME-MD, the PHQ Primary Care Study. JAMA.282:1737-44.

[10] Kroenke K, MD, Spitzer RL, Williams J. B. W., DSW (2001). Validity of a Brief Depression Severity Measure.

[11] Richardson LP, et al (2010). Evaluation of the PHQ-2 as a brief screen for detecting major depression among adolescents.

[12] Aurelien G (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.

[13] Vapnik V, Golowich S and Smola A (1997). "Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing", in M. Mozer, M. Jordan, and T. Petsche (eds.), Neural Information Processing Systems, Vol. 9. MIT Press, Cambridge, MA.

[14] Debasish B, Srimanta P, Dipak C P (2017). Support Vector Regression

[15] Alex J. Smoal, Bernhard S (2003). A tutorial on support vector regression

[16] Vapnik V (1995). The Nature of Statistical Learning Theory. Springer, New York.

[17] http://kernelsvm.tripod.com/. available at 02.04.2020

[18] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.htmlsklearn.svm.LinearSVR. available at 02.04.2020

[19] https://scikit-learn.org/stable/modules/tree.htmlregression. available at 02.04.2020

[20] Andrew Ng. CS229 Part 5 Support Vector Machines.

[21] Devakunchari R, Vaibhav S, Priyanka Z (2019). Study of Depression Analysis using Machine Learning Techniques

[22] Le Ming H, Tran D H, Tran Van L(2018) Automatic heart disease prediction using feature selection and data mining technique

[23] Asma G, Szymon F et al. (2017). Objective Assessment of Depressive Symptoms with Machine Learning and Wearable Sensors Data

[24] Cohn, Jeffrey F., Tomas S. Kruez, Iain Matthews, Ying Yang, Minh H. Nguyen, Margara T. Padilla, Feng Zhou, and Fernando D. Torre (2009). "Detecting Depression from Facial Actions and Vocal Prosdy." 3rd International Conference on Affective Computing and Intelligent Interaction, IEEE: 1-7.

[25] Jarrold, William L., Bart Peintner, Eric Yeh, Ruth Krasnow, Harold S. Javitz, and Gary E. Swan (2010). "Language Analytics for Assessing Brain Health: Cognitive Impairment, Depression and Pre-symptomatic Alzheimer's Disease." Brain Informatics, Lecture Notes in Computer Science 6334: 299-307.

[26] Taeheon Jeong, Diego Klabjanm, Justin Starren. Predictive Analytics Using Smartphone Sensors for Depressive Episodes

[27] Burns MN, Begale M, Duffecy J, Gergle D, Karr CJ, Giangrande E, et al (2011). Harnessing context sensing to develop a mobile intervention for depression. J Med Internet Res;13(3):e55

[28] Doryab A, Min J, Wiese J, Zimmerman J, Hong J (2014). Detection of behavior change in people with depression. : AAAI; Presented at: AAAI Workshop on Modern Artificial Intelligence for Health Analytics; QuÃ©bec City, QuÃ©bec, Canada.

[29] Rabbi M, Ali S, Choudhury T, Berke E (2011). Passive and in-situ assessment of mental and physical well-being using mobile sensors. ACM Presented at: International Conference on Ubiquitous Computing (UbiComp); Beijing, China p. 385-394.

[30] Wang R, Chen F, Chen Z, Li T, Harari G, Tignor S (2014). StudentLife: assessing mental health, academic performance and behavioral trends of college students using smartphones. ACM Presented at: International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp); Seattle, Washington,USA p. 3-14.

[31] Moses P (1954). The voice of neurosis. Grune and Stratton, New York, USA

[32] Darby J (1984). Speech and voice parameters in depression: a pilot study. J of Comm Disorders 17: 87-94.

[33] Scherer K (1987). Vocal assessment of affective disorders, in Depression and expressive behavior. Maser J Ed Lawrence Erlbaum Associates pp: 57-83.

[34] Dietrich M, Verdolini Abbott K, Gartner-Schmidt J, Rosen CA (2008). The frequency of perceived stress, anxiety, and depression in patients with common pathologies affecting voice. J Voice 22: 472-488.

[35] Quatieri T, Malyska N (2012). Vocal-Source Biomarkers for Depression: A Link to Psychomotor Activity in Proc. Interspeech Portland Oregon pp. 1059-1062.

[36] Mundt C (2007). Voice acoustic measures of depression severity and treatment response collected via interactive voice response technology. Journal of Neurolinguistics 20: 50-64.

[37] Cummins N, Epps J, Ambikairajah E (2013). Spectro Temporal Analysis of Speech Affected by Depression and Psychomotor Retardation. IEEE Int Conf on Acoustics Speech and Signal Processing

[38] Ang L, Dongdong J, Tingshao Z, "Detecting depression stigma on social media: A linguistic analysis"; Journal of Affective Disorders; Volume 232; pp. 358-362; 2018.

[39] Patricia A. Cavazos-Rehg, Melissa J. Krauss, ShainaSowles, Sarah Connolly, Carlos Rosas,MeghanaBharadwaj, and Laura J. Bierut (2016). "A content analysis of depression related Tweets"; Computers in Human Behavior; Volume 54; pp. 351-357.

[40] Brian A. Primack, ArielShensa, CÃ©sar G.Escobar-Viera, Erica L. Barrett, Jaime E.Sidani, Jason B. Colditz, A. EveretteJames (2016)."Use of multiple social media platforms and symptoms of depression and anxiety: A nationallyrepresentative study among U.S. young adults"; Depression and Anxiety; Volume 33; pp. 323-331.

[41] Thin N, Dinh P, Bo D, SvethaVenkatesh, MichaelBerk (2014). "Affective and Content Analysis of Online Depression Communities"; IEEE Transactions on Affective Computing; Volume 5; pp. 217-226.

[42] Morganti E, Angelini L, Adami A, Lalanne D, Lorenzelli L, Mugellini E (2012). A smart watch with embedded sensors to recognise objects, grasps and forearm gestures. Procedia Eng, 41, 1169-1175.

[43] Sanchez-Riera J, Srinivasan K, Hua K, Cheng W, Hossain M.A., Alhamid M.F (2017). Robust RGB-D Hand Tracking Using Deep Learning Priors. IEEE Trans. Circuits Syst. Video Technol. 28, 2289-2301

[44] Stetco A, Dinmohammadi F, Zhao X, Robu V, Flynn D, Barnes M, Keane J, Nenadic G (2019). Machine learning methods for wind turbine condition monitoring: A review. Renew. Energy 2019, 133, 620-635.

[45] Papakostas G.I, Petersen T, Mahal Y, Mischoulon D, Nierenberg A.A, Fava M (2004). Quality of life assessments in major depressive disorder: A review of the literature. Gen. Hosp. Psychiatry, 26, 13-17.

[46] Mahendran N, Vincent D.R (2019). Effective Classification of Major Depressive Disorder Patients Using Machine Learning Techniques. Recent Pat. Comput. Sci.12, 41-48.

[47] Dietterich T.G (2002). Machine learning for sequential data: A review. In Proceedings of the Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR),Windsor, ON, Canada, 6-9 August 2002; Springer: Berlin/Heidelberg, Germany; pp. 15-30.

[48] Ham J, Chen Y, Crawford M.M, Ghosh J (2005). Investigation of the random forest framework for classification of hyperspectral data. IEEE Trans. Geosci. Remote Sens.43, 492-501.

[49] Breiman L (2001). Random forests. Mach. Learn. 2001, 45, 5-32.

[50] https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f.

[51] Francois Chollet (2017). Deep learning with Python.

[52] Hochreiter S., Bengio Y., Frasconi P., and Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[53] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In International Conference on Machine Learning (ICML), pages 1310-1318.

[54] Fabio F, Joao P M, Alex A (2017). A review of supervised machine learning applied to ageing research

[55] Witten IH, Frank E, Hall MA (2011). Data mining: practical machine learning tools and techniques, 3rd edn. MorganKaufmann Publishers, Cambridge

[56] Qi M, Wei C, Yue W, Zhi M M, Ti Y L (2017). Convergence Analysis of Distributed Stochastic Gradient Descent with Shuffling

[57] Vanessa B, David M, Michael A (2019).Analysis of Explainers of Black Box Deep Neural Networks for Computer Vision: A Survey

[58] https://www.coursera.org/lecture/convolutional-neural-networks/cnn-example-uRYL1. available at 05.04.2020

[59] Doreswamy and Chanabasayya .M. Vastrad (2013). Performance Analysis of Neural Networks Models for Oxazolines and Oxazoles Derivatives Descriptor Dataset.

[60] David C et al (2015).PhD Mobile Phone Sensor Correlates of Depressive Symptom Severity in Daily-Life Behavior: An Exploratory Study

[61] Yasin A, Emre A (2019).Prediction of stress levels with LSTM and passive mobile sensors

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 9. April 2020 Hesham Hussen