Feature Extraction and Classification of Medication-Induced Hyperkinesia During Treatment of Parkinson´s Disease

Erik Liljeroth

Master's thesis
2020:E76

Lund Institute of Technology
Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematical Statistics

Centrum Scientiarum Mathematicarum

Feature Extraction and Classification of Medication-Induced Hyperkinesia During Treatment of Parkinson´s Disease

Erik Liljeroth

Master's thesis
2020:E76

Lund Institute of Technology
Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematical Statistics

Centrum Scientiarum Mathematicarum

Master's Theses in Mathematical Sciences 2020:E76
ISSN 1404-6342

LUTFMS-3399-2020

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/

Master's Theses in Mathematical Sciences 2020:E76
ISSN 1404-6342

LUTFMS-3399-2020

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/

Abstract

The recent progress in wearable sensor technology, signal processing and machine learning enables novel applications in fields such as automatic disease symptom tracking and classification. In this project, data from an ongoing Parkinson's disease study at the Lund University Hospital is gathered and analyzed, and classification models of varying complexity for symptom severity are evaluated. The accelerometer and gyroscope data is provided by a single mobile phone fastened around each patient's trunk and is complemented by a certified specialist's score labels, in clinical dyskinesia rating scale (CDRS). Out of four available tasks performed by the patient during measurement, the two tasks "walking" and "sitting and describing a picture" were cut out and analyzed in-depth.

The most practically usable simulation scenario was found to be forming individual models for the patients, with the best performing model proving to be an unsupervised feature extracting autoencoder combined with a linear discriminant. We could with descent accuracy distinguish between whole test signals in a binary-class scenario for a majority of patients and perform skillfully in the multi-class scenario, although not well enough for practical usability. The total patient-average, mean macro f1-score and accuracy obtained for binary-class classification of short (2 second) signal segments from the "describe picture"-task were 0.74 and 0.81 respectively. The corresponding mean macro f1-score and accuracy for the multi-class classification case were 0.52 and 0.65 respectively.

Acknowledgements

I would like to thank Sofia Christiansson, Monica Scharfenort and Sotirios Grigoriou for assisting me with expertise in medicine. Thank you Pierre Nugues at the Department of Computer Science for valuable discussions and inspiration related to artificial neural networks. Thank you James Hakim at the Division of Mathematical Statistics for your valuable IT expertise.

Finally, I would like to especially thank Andreas Jakobsson at the Division of Mathematical Statistics for being an invaluable mentor and letting me into his problem solving mind.

Contents

1

Introduction

1.1 Introduction

Parkinson's Disease

Parkinson's disease (PD) is a progressive neurodegenerative disorder of the central nervous system with an annual incidence of approximately 2000 patients in Sweden [Odin et al., 2019]. It is called neurodegenerative because it causes dysfunction of nerve cells with eventual cell death. These nerve cells are of the kind that produce dopamine, which is a neurotransmitter that the brain uses to control movements in the body. The progression of the disease is slow, with eventual motor symptoms (for example tremor, freezing and rigidity) that are characteristic for PD. The medical term for the slow movement, rigidity and inability to move the body swiftly on command is *bradykinesia*. Figure 1.1 shows an illustration of the degeneration of dopamine production for a Parkinson's disease patient.

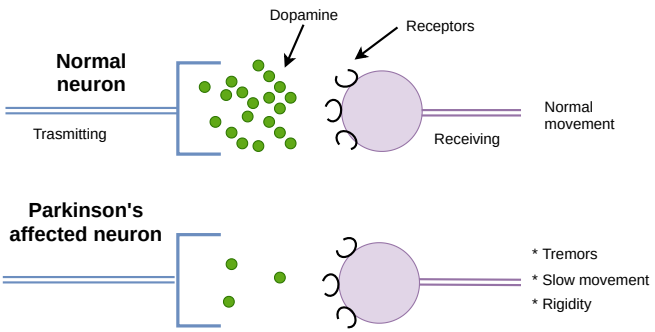

Figure 1.1 How Parkinson's disease affects dopamin producing neurons.

Treatment of PD is symptomatic, i.e. the treatment relieves the symptoms but does not affect the underlying disease. The standard treatment is dopamine replacement in tablet form. Most patients respond well to the treatment, but eventually

develop motor fluctuations and "dyskinesias" within 5-10 years of disease progression [Odin et al., 2019]. Dyskinesias are abnormal involuntary movements that are not part of the primary PD symptoms, but rather appear from long-time treatment with the tablet medication (interacting with the effects of the progressing disease). This means that increased rigidity during the disease progression results in higher doses of medication which might trigger more severe induced dyskinesias. The term dyskinesia can be further divided into the following groups:

- Dystonia: slow twisting movements and/or abnormal postures, which are most common in the trunk, legs and feet
- Hyperkinesia: chorea (fast movements that seem to rapidly flow across body parts), ballism (abrupt large movements of the upper limbs) and myoclonus (brief irregular muscle twitching)

Description of Parkinson's Disease Study

A research project with title "An exploratory study on the impact of D2/3 receptor agonists on the phenomenology of dyskinesias in patients with Parkinson's disease" is being conducted at the department of neurology at Lund University. In the study, the subjects are given two different medications on different occasions. These medications are both dopamine-based, but acting on different receptors. The main objective of the study is to compare localization, maximum severity and time frame in which hyperkinetic or dystonic abnormal involuntary movements occur after challenge doses of the different medications.

The symptoms are rated according to a severity scale, the "clinical dyskinesia rating scale" (CDRS) [Hagell and Widner, 1999]. Doctors can use this scale to give symptoms a "severity score" as an integer $\in \{0,1,2,3,4\}$, where 0 represents "no dyskinesia" and 4 represents "incapacitating dyskinesia which prohibits some postures and voluntary movements". As an exploratory extension to the research project, it is of interest to determine how the CDRS scores correlate with measurements of movements and acceleration as obtained using available accelerometer and gyroscope.

Project Goals

The goal of the project is to determine if it is possible to use annotated data to train classification models, and if possible, get a performance measure of how well this can be accomplished. The results will later, in collaboration with doctors, be used to plan further studies and possibly be implemented as an application. If the work is successful, it could lead to improved symptom monitoring and perhaps medication through feedback of the patient's current state.

1.2 Previous work

The area of automatic symptom classification for Parkinson's disease is currently very active. In the work by Lonini et al. (2018), the dataset consisted of 20 individuals with Parkinson's disease. The patients performed similar tasks as for example walking, or sitting down and drawing on paper while being recorded by accelerometer and gyroscope sensors. One key difference is that the subjects only took their ordinary medication dose in connection with the experiments and only 8% of the obtained dataset had annotated dyskinesias. In our dataset, we can expect a higher proportion of dyskinesias due to deliberately higher medication doses intended to trigger dyskinesias. The work by Lonini et al. (2018) continued to form patient-generic classification models only for bradykinesia (slowness of movement, rigidity) and tremors experienced in the "off"-medication state due to the low prevalence of dyskinesias. With a single sensor on the back of the hand, the group were able to detect bradykinesia and tremors in the upper extremity. The work concluded that PD symptoms can be detected using a single sensor and that it is best modeled by using a dataset incorporating many individuals.

Another approach is to use speech impairments from PD subjects as done in the work by Caliskan et al. (2017), where a deep neural network (DNN) classifier is evaluated on two standard datasets with the purpose to distinguish healthy people from people with PD. In their methodology, they filter out the noise from the speech signals and segment the data with time-windows. The hidden layers in the DNN are trained by repeatedly training autoencoders with only one encoding layer, to then stack the trained encoding layers in the DNN to form a "stacked autoencoder". The reported results are just barely better than a reference support vector machine (SVM) model. In this approach, specific PD symptoms are not targeted, instead the data is labeled as either PD patient or not. The noise-filtered speech data is fed directly to the autoencoders to perform automatic feature-extraction. In contrast to this approach, our work utilizes other preprocessing and feature extraction methods, such as computing spectral estimates. Our approach also specifically targets medication-induced hyperkinesia for a group of PD patients which is a key difference. However, the article might still provide some inspiration for approaches to try out since the underlying problem is to find PD characteristics in the data.

Another recent work utilizing accelerometer and gyroscope recordings for automatic PD symptom classification is the article by Goschenhofer et al. (2019). The measurement device was in this case a wrist-band collecting accelerometer and gyroscope data with a sampling frequency of 62.5 Hz, later downsampled to 20 Hz. The data was annotated by a medical doctor minute-wise and the segmentation window length was set to one minute. To increase the amount of data, they further segmented the minute-long segments with 80% overlap which is referred to as in-line with related work. The examined PD symptom is bradyskinesia, as in the work by Lonini et al. (2018), however the work by Goschenhofer et al. (2019) utilizes finer annotations by a doctor according to a bradyskinesia severity rating scale. The best

performing model for the scenario proved to be a regression convolutional neural network. In our work, we attempt to classify the medication-induced hyperkinesia according to the CDRS rating scale and we draw inspiration from this related work by evaluating a regression convolutional neural network as a comparison model.

1.3 Contribution from this work

Our work is restricted to treating the hyperkinesia symptom as a subgroup of dyskinesia symptoms. Compared to previous work, the dataset in this project can be expected to reflect a broader spectrum of symptom severity, which is due to deliberately triggering medication-induced dyskinesia with larger than usual patient-individual doses of dopamine-based medication. This makes the dataset currently unique as no existing similar scenario in other research has been found.

We treat different simulation scenarios with the gathered dataset. We first attempt to find patient-generic models to later attempt and see what can be accomplished with patient-individual models. The aim is to arrive at some qualitative results that can be used to plan further studies within the area. In the long run, the project could lead to increased life quality for PD patients.

1.4 Report outline

Chapters “Feature extraction”, “Machine learning” and “Artificial neural networks” focus on underlying theory for the methods used. Chapter 5 explains in detail how the project was conducted and the process from data gathering to modeling. In chapter 6 we present simulation results and discuss our findings. Finally in chapter 7 we draw conclusions from our findings and suggest some ideas for future experiments in chapter 8.

2

Feature extraction

The methods of feature extraction are in the following chapter divided into time domain and frequency domain. The purpose is to extract properties of the data that can later be used to discriminate by class.

2.1 Time domain

Signal variance

The variance is a common statistical tool and is typically estimated using equation 2.1, which is an unbiased estimate [Blom et al., 2017] with $\underline{\mathbf{x}} = [x_0, x_1, ..., x_{N-1}]$ being a signal and m_x being the mean of the signal. This is equivalent to the signal power if the mean is zero [Chaparro, 2019].

$$(\sigma^2)^* = \frac{1}{N-1} \sum_{t=0}^{N-1} (x_t - m_x)^2 \tag{2.1}$$

Autocovariance function

The autocovariance function (ACF) is a tool that can measure dependencies within and between stochastic processes that fulfill the requirement of being *wide sense stationary* (WSS), i.e., fulfilling the following conditions [Jakobsson, 2017]:

(i) The mean of the process is constant and finite.

(ii) The autocovariance depends only on the time lag and not on the time itself.

(iii) The variance of the process is finite.

The above conditions are assumed to approximately hold for the underlying process when using the method on data, which is treated like a realization of the process. Except from these conditions the method is said to be non-parametric as it does not impose some specific parametric model on the data. The ACF can be used as one tool to distinguish between auto-regressive (AR) and moving average

(MA) models. The autocovariance function is estimated using the *biased* estimate, which has the property that the variance of the estimate for high lags is more stable [Jakobsson, 2017]. In this project we restrict ourselves to real-valued signals, which gives the resulting formula for the estimate as equation 2.2

$$\hat{r}_x(\tau) = \frac{1}{N}\sum_{t=1}^{N-\tau}(x_t - \hat{m}_x)(x_{t+\tau} - \hat{m}_x) \tag{2.2}$$

AR modeling

Autoregressive (AR) models are used in modeling of human voiced speech, for example in the article by Berezina et al. (2010). For this reason it might be a suitable model to consider for other human phenomena. A stationary stochastic process $\{X_t\}$ is called an AR(p)-process if it satisfies equation 2.3 with $A(z) = 1 + a_1 z^{-1} + ... + a_p z{-}p$ being a so-called *generating polynomial* in the time-shift operator z and $\{e_t\}$ being a white-noise sequence.

$$A(z)X_t = X_t + a_1 X_{t-1} + ... + a_p X_{t-p} = e_t \tag{2.3}$$

For a given AR(p) - model, the spectral density can be calculated according to equation 2.4 where σ^2 is the variance of the white-noise sequence $\{e_t\}$. This enables the possibility of first estimating the model and then computing the spectral density which could be used for feature-extracting purposes.

$$R_x(f) = \frac{\sigma^2}{\left|\sum_{k=0}^{p} a_k e^{-i2\pi fk}\right|^2} \tag{2.4}$$

For an AR(p) process, the autocovariance function has a specific property in that it solves the Yule-Walker equations

$$\begin{cases} r_X(k) + a_1 r_X(k-1) + ... + a_p r_X(k-p) = 0 \quad \text{for } k = 1,2,.. \\ r_X(0) + a_1 r_X(1) + ... + a_p r_X(p) = \sigma^2 \quad \text{for } k = 0 \end{cases} \tag{2.5}$$

2.2 Frequency domain

Periodogram

A common way to extract frequency information from a temporal signal is to apply the Fourier transform, which in continuous time is defined as

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-i2\pi ft} \mathrm{d}t \tag{2.6}$$

The output of the Fourier transform $X(f)$ is in general complex, containing the information about power and phase of each frequency in the signal. In order to

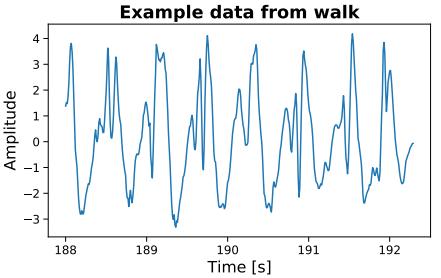

Figure 2.1 Accelerometer signal in the gravity's direction of a walking patient.

overview the power of different frequencies in the signal, the squared magnitude of the Fourier transform can be used. This happens to be proportional to the so-called *periodogram*, which is an estimate of the power spectral density (PSD) of the signal [Lindgren et al., 2014]. The periodogram is defined as equation 2.7 where we instead have the discrete Fourier transform (DFT) of the signal sequence $x(t)$.

$$\hat{R}_x(f) = \frac{1}{N}|X(f)|^2 = \frac{1}{N}\left|\sum_{t=0}^{N-1} x(t)e^{-12\pi ft}\right|^2 \tag{2.7}$$

For the methods based on the Fourier transform, the signal is assumed to be stationary, meaning that the frequency content should be roughly the same throughout the time signal. This seems to be the case for a subset of the data in this project, e.g. consider the example from the study where a patient is walking and the corresponding accelerometer signal in the gravity's direction is shown in Figure 2.1. The signal is repetitive where the individual steps can be seen. Removing the mean and computing the periodogram for the data example above, the signal appears to mostly consist out of two frequencies as is seen in Figure 2.2.

Windowing In practice, signals do not have infinite length, but rather consist of N samples. A consequence of this is that the DFT can be interpreted as performing the Fourier transform of an infinitely long signal multiplied by a window that segments the infinitely long signal in time. This corresponds to a convolution with the Fourier transform of said window in the frequency domain, which causes a phenomenon called *sidelobes* [Lindgren et al., 2014]. By using a window in the time domain that is not rectangular we can control the trade off between main lobe width and sidelobe height. Usually the window function is a design parameter chosen by the engineer, and many windows achieve similar behaviour for the main lobe and side lobes for the spectral estimate. One window that suppresses sidelobes is the *Hamming window* [Prabhu, 2013].

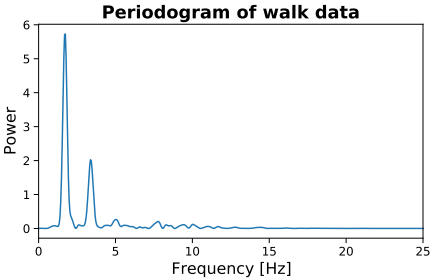

Figure 2.2 Periodogram of the data where a patient is walking. The periodogram is computed using a 1024 point fast Fourier transform(fft) and the data is windowed using a Hamming window.

Spectrogram

A non-parametric time-frequency transformation is the spectrogram [Sandsten, 2020]. The definition of the spectrogram utilizes the short-time Fourier transform (STFT) which in continuous time is defined as

$$X(t,f) = \int_{-\infty}^{\infty} x(t_1)h^*(t_1 - t)e^{-i2\pi f t_1}\mathrm{d}t_1 \tag{2.8}$$

where $h(t)$ is a window function centered at time t. The window function cuts the signal around time t and the Fourier transform is applied to the resulting segment. In a similar manner to the periodogram and ordinary Fourier transform, the spectrogram is then defined as

$$S_x(t,f) = |X(t,f)|^2 \quad -\infty < t,f < \infty \tag{2.9}$$

which can be a useful tool to analyze time-varying and non-stationary signals. Note that in order for the transformation to be useful, the original signal should have at least a decent length in order to be further subdivided into segments by the STFT.

3

Machine Learning

This chapter starts with discussing classification problems and how to evaluate the results, then introduces some simple classification models.

3.1 Classification problems and performance measures

In a classification problem we have a data array of training data $\underline{\mathbf{X}}^{\text{train}} = [\underline{\mathbf{x}}_0, \underline{\mathbf{x}}_1, ..., \underline{\mathbf{x}}_{N-1}]$, with the individual samples $\underline{\mathbf{x}}_i$ typically being arrays. They can e.g. be images or spectral representations. We have an accompanying array $\underline{\mathbf{Y}}^{\text{train}} = [y_0, y_1, ..., y_{N-1}]$ containing the labels for each sample in $\underline{\mathbf{X}}$, e.g. 'tulip', 'rose' and 'dandelion' for a flower classification problem. The classification problem consists of using the data in $\underline{\mathbf{X}}^{\text{train}}$ with the labels $\underline{\mathbf{Y}}^{\text{train}}$ to form a model that describes the data as well as possible. The model is then typically evaluated on a separate test set $\underline{\mathbf{X}}^{\text{test}}$ to evaluate how well the model generalises to data which has not been used during training.

When performing classification there are multiple performance measures to evaluate the result. In this section the used performance measures in this project are introduced and explained. The most simple classification problem consists of one-class classification. This means that samples will be labeled as either belonging to the class or not. The following quantities can be defined:

(i) True positives (TP): The samples that were correctly classified as positive.

(ii) False positives (FP): The samples that were wrongly classified as positive.

(iii) False negatives (FN): The samples that were wrongly classified as negative.

(iv) True negatives (TN): The samples that were correctly classified as negative.

Using the anove definitions, some performance measures are defined in table 3.1.

Table 3.1 Classification performance measures

Performance measure	Formula
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F1-score	$2\,\frac{\text{Precison}\cdot\text{Recall}}{\text{Precison}+\text{Recall}}$
Accuracy	$\frac{TP+TN}{TP+FP+FN+TN}$

The measures in table 3.1 are computed for every class in a classification problem, usually followed by computing a weighted measure as a final score. In this work, the "macro F1-score" is used as the weighted measure. For the K-class classification problem the macro average f1-score is defined as in equation 3.1, which simply takes the average of the individual f1-scores.

$$f1_{\text{macro}} = \frac{1}{K}\sum_{k=0}^{K-1} f1_k \tag{3.1}$$

The macro f1-score is a good complement to the overall prediction accuracy since it penalizes bad classification performance for small classes. Consider a simple binary classification example where 95% of the dataset belongs to class 0 and other 5% to class 1. Imagine that a terrible classifier is constructed which classifies all samples to class 0. The overall prediction accuracy will be 0.95 which is misleadingly high but the macro f1-score will only be 0.5, hence highlighting the poor classification performance for the smaller class.

The confusion matrix is defined as $C = \{C_{i,j}\}$, where $C_{i,j}$ is equal to the number of samples in group i that are predicted to be in group j. This is visualised by figure 3.1 for the "binary classification, two classes" case.

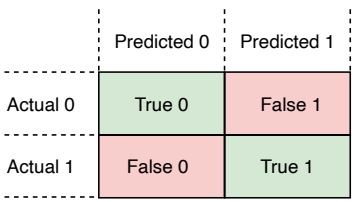

Figure 3.1 The structure of the confusion matrix. The figure shows the binary classification case.

3.2 The naive Bayesian classifier

A class of simple models are the naive Bayesian classifiers, described in for example [Norvig and Russel, 2010]. A Bayesian classifier is a conditional probability model. Given an input vector of n features $\underline{\mathbf{x}}_n = [x_0, x_1, ..., x_{n-1}]$, it is of interest to compute $P[C_k|\underline{\mathbf{x}}_n]$. This is the conditional probability of class C_k given the feature vector $\underline{\mathbf{x}}_n$. If this probability is known for every class C_k, we can then during prediction assign a class to a sample $\underline{\mathbf{x}}_n$ by picking the argument (class) that maximises $P[C_k|\underline{\mathbf{x}}_n]$, i.e.,

$$\hat{y} = \arg\max_{C_k} P[C_k|\underline{\mathbf{x}}_n] \tag{3.2}$$

Using the definition of conditional probability twice, the expression $P[C_k|\underline{\mathbf{x}}_n]$ can be rewritten on the form known as "Bayes rule":

$$P[C_k|\underline{\mathbf{x}}] = \frac{P[C_k, \underline{\mathbf{x}}]}{P[\underline{\mathbf{x}}]} = \frac{P[C_k]P[\underline{\mathbf{x}}|C_k]}{P[\underline{\mathbf{x}}]} \tag{3.3}$$

The expression for Bayes rule is not directly practically useful and needs to be rewritten. The expression $P[\underline{\mathbf{x}}|C_k]$ in the numerator of equation 3.3 can be rewritten using the definition of conditional probability multiple times.

$$\begin{aligned} P[x_0, x_1, ..., x_{n-1}|C_k] &= P[x_0|x_1, x_2, ..., x_{n-1}, C_k] \cdot P[x_1, x_2, ..., x_{n-1}, C_k] = \\ P[x_0|x_1, x_2, ..., x_{n-1}, C_k] &\cdot P[x_1|x_2, x_3, ..., x_{n-1}, C_k] \cdot P[x_2|x_3, ..., x_{n-1}, C_k] = \\ = ... = P[x_0|x_1, x_2, ..., x_{n-1}, C_k] &\cdot P[x_1|x_2, ..., x_{n-1}, C_k] \cdot ... \cdot P[x_{n-1}|C_k] \end{aligned} \tag{3.4}$$

At this stage the *naive* assumption is made that the features $x_i, i \in \{0, 1, 2, ...n-1\}$ are conditionally independent given the class C_k. As a consequence, equation 3.4 can be rewritten as

$$P[x_0, x_1, ..., x_{n-1}|C_k] = \prod_{i=0}^{n-1} P[x_i|C_k] \tag{3.5}$$

The results are summarized as equation 3.6, arriving at the naive Bayesian classifier.

$$P[C_k|\underline{\mathbf{x}}] = \frac{P[C_k]\prod_{i=0}^{n-1} P[x_i|C_k]}{P[\underline{\mathbf{x}}]} \tag{3.6}$$

Given an input feature vector $\underline{\mathbf{x}}$, $P[\underline{\mathbf{x}}]$ is a constant when evaluating $P[C_k|\underline{\mathbf{x}}]$ for different classes C_k. Therefore it can be omitted in the classifying algorithm. The resulting implemented classifier is given by equation 3.7.

$$P[C_k|\underline{\mathbf{x}}] \propto P[C_k] \prod_{i=0}^{n-1} P[x_i|C_k] \tag{3.7}$$

Dealing with continuous data

Typically the naive Bayesian classifier is trained by counting the occurrences of the feature values and storing them in a dictionary known as a *frequency dictionary*. From this dictionary is is then possible to estimate $P[x_i|C_k]$. This causes problems for unseen feature values since the estimate $\hat{P}[x_i|C_k] = 0$ if x_i has not been observed. The problem is typically rectified by smoothing the feature dictionary with feature-individual distributions. This technique is useful when the feature value space is big and discrete or if the feature value space is continuous.

A common distribution that is widely applied is the Gaussian probability density function (pdf). Essentially the imposed assumption is that $P[x_i|C_k] \sim N(\mu_{k,i}, \sigma_{k,i})$, meaning that we assume that the individual features are conditionally normal distributed given class C_k.

Naive Bayesian classifier using kernel density estimation

Instead of imposing the Gaussian pdf, we can use a non-parametric way of estimating the feature pdfs, and use the estimated pdfs to form predictions. The pdfs can be estimated by using the non-parametric "kernel denisty estimation" method [Scott, 2015]. For a data vector $\underline{\mathbf{x}} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}$, the pdf kernel density estimator is given by equation 3.8 where K is the used kernel (typically Gaussian), n is the number of values in the data vector and h is the kernel bandwidth which is chosen as a design parameter.

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - x_i}{h}) \tag{3.8}$$

The kernel bandwidth h can be chosen according to Scott's rule given by equation 3.9 [Scott, 2015], where $\hat{\sigma}$ is the standard deviation estimate. This is a good rule of thumb selection of h.

$$h = 1.06 \hat{\sigma} n^{-\frac{1}{5}} \tag{3.9}$$

The model fitting stage estimates the pdfs $f_{C_k,i}$ for class C_k and feature i using the above approach. Prediction is performed by using the Bayesian classifier presented earlier as equation 3.7.

$$\hat{y_i} \approx \underset{C_k}{\arg\max} \left(P[C_k] P[x_i|C_k] \right) \tag{3.10}$$

3.3 Linear discriminant analysis

Linear discriminant analysis (LDA) is a category of methods treating functions that project the feature space of an input vector to a lower-dimensional space using conditions that enhance class-separation. It is similar to principal component analysis

(PCA) in the sense of dimensionality-reduction but with fundamental differences as PCA is an unsupervised learning technique that finds the directions of maximum variance in the original feature space [Martinez and Kak, 2001].

The most simple case of LDA where the projection is done to one dimension is described by the following equation, where y is the output value in a one-dimensional space, $\underline{\mathbf{x}}$ is an input sample with feature values, $\underline{\mathbf{w}}$ is a vector of weights and w_0 is a constant bias.

$$y(\underline{\mathbf{x}}) = \underline{\mathbf{w}}^T \underline{\mathbf{x}} + w_0 \tag{3.11}$$

This highlights the linearity of the method as the output value is a linear combination of the features. In general the $\underline{\mathbf{w}}$-matrix can be of dimension D'xD where the projection is taken from the original feature space dimension D to a space of smaller dimension D', resulting in a vector output $\underline{\mathbf{y}}$.

Fisher linear discriminant (FLDA) is a submethod of LDA where the projection matrix is found by maximizing the between-class distance and minimizing the within-class distance, i.e., the aim is for the classes to be internally focused but well separated. This is implemented by first computing two covariance matrices:

(i) The total within-class covariance matrix:

$$S_w = \sum_{k=1}^{K} \sum_{n \in C_k} (\underline{\mathbf{x}}_n - \underline{\mathbf{m}}_{C_k})(\underline{\mathbf{x}}_n - \underline{\mathbf{m}}_{C_k})^T \tag{3.12}$$

where K is the number of classes, $\underline{\mathbf{m}}_{C_k}$ is the classwise mean for class C_k and $\underline{\mathbf{x}}_n$ is the n'th sample of class C_k.

(ii) The total between class covariance matrix:

$$S_b = \sum_{k=1}^{K} N_k (\underline{\mathbf{m}}_{C_k} - \underline{\mathbf{m}})(\underline{\mathbf{m}}_{C_k} - \underline{\mathbf{m}})^T \tag{3.13}$$

where N_k is the amount of samples in class C_k and $\underline{\mathbf{m}}$ is the overall sample mean.

The are many choices of the Fisher optimization criterion [Fukunaga, 1990] and one way is to formulate it as equation 3.14. This criterion can be shown to be equal to solving an eigenvalueproblem of the matrix $S_w^{-1} S_b$, where the projection weight vectors $W = w_{d'}$ are chosen as the eigenvectors corresponding to the D' largest eigenvalues [Bishop, 2006].

$$J(W) = \mathrm{Tr}\left\{ (W S_w W^T)^{-1} (W S_b W^T) \right\} \tag{3.14}$$

As an example, the method is applied to the handwritten digits dataset [Dua and Graff, 2017]. This dataset consists of handwritten digits that are 8x8 pixels. Each pixel is interpreted as a feature resulting in 64 features. Figure 3.2 shows a scatter plot of 2D projections obtained when applying Fisher's linear discriminant to this example.

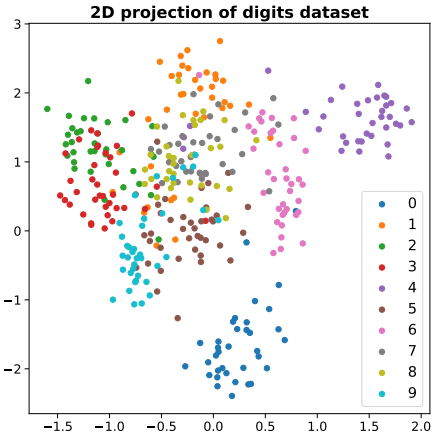

Figure 3.2 Scatter plot of 2D-projection of digits dataset using Fisher LDA method. Example by Erik Liljeroth.

According to this algorithm, for example the numbers 3 and 9 seem to be similar as they have a small euclidean distance in the 2D projection space. The numbers 0 and 4 are the most separated from the crowd.

4

Artificial Neural Networks

Artificial neural networks are a popular class of methods that can be used in, e.g., regression, classification and compression problems. The technology has been around for several decades with, for example, the pattern recognition properties of the convolutional neural network being discussed in the paper by LeCun et al. (1999). However it is only lately, with the rise in cheap computational power, that the popularity of the techniques has skyrocketed. This chapter discusses the basic functionality of neural network models and introduces concepts that are used in the methods of the project.

4.1 General

A way to model a single neuron is equation 4.1, where h_j is the output of the neuron, D is the number of inputs, b_j is a bias, w_{ji} is the weight that multiplies with input x_i and $a(\cdot)$ is a non-linear activation function [Bishop, 2006]. The bias can be incorporated by defining an additional input variable $x_0 \triangleq 1$ resulting in the second equality, note that the summation index starts from zero instead.

$$h_j = a\left(\sum_{i=1}^{D} w_{ji}x_i + b_j\right) = a\left(\sum_{i=0}^{D} w_{ji}x_i\right) \tag{4.1}$$

Placing a couple of neurons in a network with one input layer, one hidden layer and one output layer produces the architecture shown in Figure 4.1, where the incorporation of the bias is seen as the uppermost neurons in the input layer and hidden layer. Another name for a hidden layer is *dense* layer.

From the above network the k'th output y_k as a function of the input $\underline{\mathbf{x}}$ and the weights $\underline{\mathbf{w}}$ can be written as equation 4.2 with $J = 4$ neurons in the hidden layer and $D = 3$ inputs.

$$y_k(\underline{\mathbf{x}}, \underline{\mathbf{w}}) = a_{\text{output}}\left(\sum_{j=0}^{J=4} w_{kj}^{(2)} a_1\left(\sum_{i=0}^{D=3} w_{ji}^{(1)} x_i\right)\right) \tag{4.2}$$

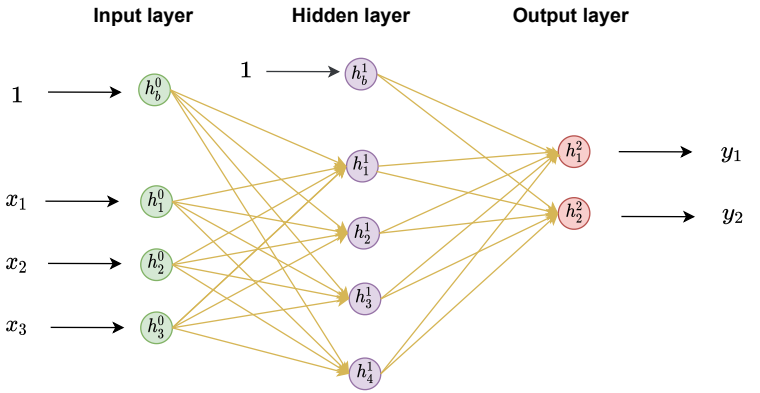

Figure 4.1 Example of architecture for a multi-layer perceptron.

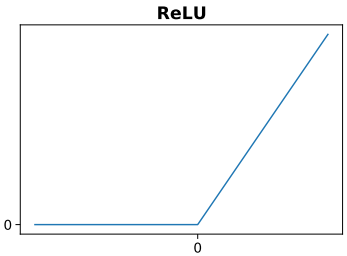

Figure 4.2 ReLU activation function.

If the activation functions a_1 and a_{output} are removed, then the map from the input to the output is simply linear and can be represented with chained matrix multiplications. The activation functions are added in order to introduce non-linearity in the network, which allows the network to learn non-linear relations [Chollet, 2017]. A popular choice of activation function is the ReLU (Rectified Linear Unit) shown in Figure 4.2. Applying this function to the output of a neuron yields a non-linear transformation, yet very close to linear in the sense that the function is a piece-wise linear function of two pieces. The almost-linearity of the ReLU-activation preserves many properties which makes gradient-descent based optimization easier [Goodfellow et al., 2016].

Loss functions

A loss function is defined in order to evaluate the performance of the network, and is a function of the predicted output $\hat{\underline{\mathbf{y}}}$ and the true output label $\underline{\mathbf{y}}$. Typically, the loss function is chosen depending on the application. Here, two common loss functions are described.

Binary cross-entropy Binary cross-entropy (BCE), computed according to equation 4.3, is a common choice for the binary classification problem, and is derived from the maximum likelihood principle [Bishop, 2006].

$$BCE(\underline{\mathbf{y}}, \underline{\hat{\mathbf{y}}}) = -\sum_k (y_k \log(\hat{y}_k) + (1 - y_k) \log(1 - \hat{y}_k)) \tag{4.3}$$

If, for example, there is one output and the true label is $y_k = 0$, then the formula boils down to "$-\log(1-\hat{y}_k)$" which is minimized and equal to zero for $\hat{y}_k = 0$ (note that the binary cross-entropy loss is used in conjunction with the sigmoid-activation function on the final layer resulting in $\hat{y}_k$ only taking values in the interval $[0,1]$ [Chollet, 2017]). Other properties are that the binary cross-entropy is always minimized for $\underline{\hat{\mathbf{y}}} = \underline{\mathbf{y}}$ but only equal to zero if $y_k = 0$ or $y_k = 1$, which can be verified by the formula. The loss function can be extended to multi-class problems.

Mean squared error The mean squared error (MSE) loss is typically used in regression problems or as the reconstruction loss in autoencoders, and is constructed as follows for K outputs:

$$\text{MSE}(\underline{\mathbf{y}}, \underline{\hat{\mathbf{y}}}) = \frac{1}{K} \sum_{k=1}^{K} (y_k - \hat{y}_k)^2. \tag{4.4}$$

Unlike the binary cross-entropy, the MSE loss is usually accompanied with a linear output activation function (equivalent to none), which allows $\underline{\hat{\mathbf{y}}}$ to take values outside the interval $[0,1]$.

Backpropagation

When training the network the objective is to minimize the loss function L. In ordinary convex optimization problems we are guaranteed to reach optimality, however the non-linear activations of the neural network cause the loss function to become a non-convex complicated surface. Gradient descent can be used, however we are not guaranteed to reach a global minima [Goodfellow et al., 2016]. We wish to update the weights $w_{j,i}$ at layer j multiplying with input i of the network by walking in the negative direction of the gradient with a learning rate l.

$$w_{j,i} \leftarrow w_{j,i} - l \frac{\partial L}{\partial w_{j,i}} \tag{4.5}$$

It is not trivial how the gradient $\partial L / \partial w_{j,i}$ is computed, but this is a solved technology since the introduction of *Backpropagation*. Backpropagation allows computing the loss at the end of the network and then propagating the error backwards through the network to update the weights [Sathyanarayana, 2014]. The algorithm can be summarized in the following steps.

(i) Initialize the weights in the network to small random values

(ii) Choose an input t from the training set and set the input layer neurons

$$h_j^0 = x_j(t), \quad \forall j \tag{4.6}$$

(iii) Propagate the signal through the network, for the m'th layer do

$$h_j^m = a_j^m \left(\sum_i w_{i,j} h_i^{m-1} \right), \quad \forall j,m \tag{4.7}$$

(iv) Calculate the "deltas" also called sensitivity components for the final layer

$$\delta_j^M = a_{\text{output}} \left(\sum_i w_{i,j}^M h_i^{M-1} \right) (y_j(t) - h_j^M) \tag{4.8}$$

(v) Calculate the deltas for the remaining layers by propagating the error backwards through the network (for $m = M, M-1 \ldots, 2$)

$$\delta_j^{m-1} = a_j^{m-1} \left(\sum_i w_{i,j}^{m-1} h_i^{m-2} \right) \sum_j w_{j,i}^m \delta_i^m \tag{4.9}$$

(vi) Update the weights

$$w_{j,i}(k+1) = w_{j,i}(k) - l \delta_j^m h_i^{m-1} \tag{4.10}$$

Useful deep learning terminology

The following useful terminology is referred to later in the report.

- **Batch** - The batch size is a hyperparameter that defines the number of samples that the neural network sees before updating the internal weights.
- **Epoch** - During an epoch the neural network will see the entire dataset, i.e., training for 500 epochs means that the entire dataset flows through the network 500 times.

4.2 Convolutional layers

In the previous section the discussed network was a "multi layer perceptron" consisting of multiple layer of neurons. In modern application different kinds of feature extracting layers are incorporated in the neural networks such as the convolutional layer, which is explained in this section. The convolutional layer has proven to be

excellent in finding translation-invariant patterns in images and became popular after the convolutional neural network "AlexNet" won the ImageNet competition in 2011 [Krizhevsky et al., 2012]. AlexNet consisted of only five convolutional layers and three dense, classifying layers.

Convolutional arithmetic

The convolutional layers in neural networks are based off the discrete-time convolution. Equation 4.11 shows the expression for the two-dimensional case which is most often encountered due to the popularity in image analysis.

$$O(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n) \tag{4.11}$$

The discrete time convolution is commutative meaning that $I * K = K * I$. This is a useful property in proofs but not necessarily in neural networks. In most neural network libraries the cross-correlation is implemented instead, which is equivalent with the convolution except of not flipping the kernel [Goodfellow et al., 2016]. In the literature about convolutional neural networks usually the cross-correlation is referred to as "convolution" which is also the case in this report.

$$O(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{4.12}$$

Figure 4.3 shows an example of performing a convolution according to equation 4.12 between a kernel and an input matrix and producing an output.

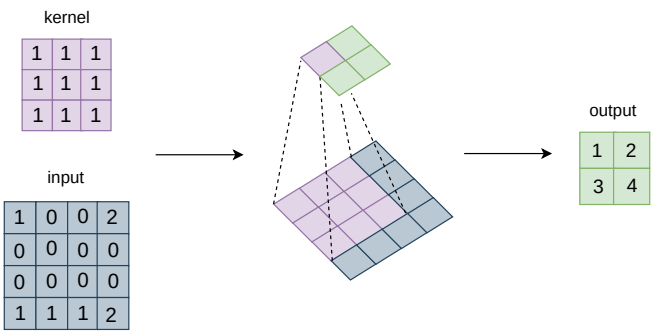

Figure 4.3 Example of the convolution operation with an input matrix, convolution kernel and output matrix.

The operation is intuitively understood by sliding the kernel over the input matrix and at each valid location performing a dot product, thus elementwise multiplication and then summation. If the input matrix is not extended by, i.e., zero

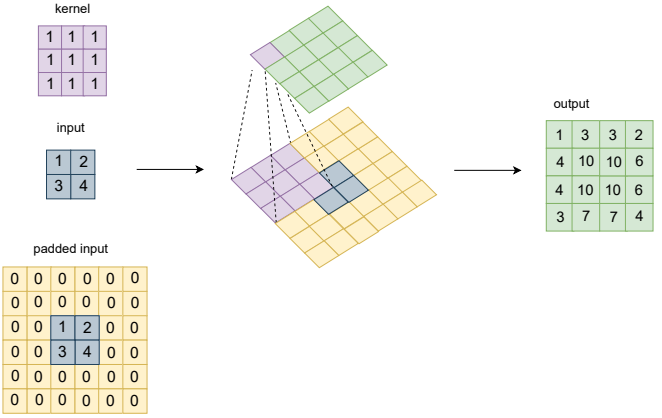

Figure 4.4 Example of the transposed convolution operation with an input matrix, padded input matrix, convolution kernel and output matrix.

padding, the output shape of the feature map will in general be smaller (in the example going from 4x4 to 2x2). When a neural network is built off of convolutional layers, the shape of the input must be monitored as it passes through the convolutional layers.

Transposed convolution

Sometimes it is of interest to "go in the other direction", i.e., increase the size of the produced feature map through the convolution of the input while maintaining the connectivity pattern that is compatible with the considered convolution. This is desirable with e.g. autoencoders that are introduced in the upcoming section. For the example in Figure 4.3 this can be shown to be analogous to performing a convolution with a zero-padded input and is called *transposed convolution* [Dumoulin and Visin, 2016]. Figure 4.4 shows the corresponding transposed convolution to the example in Figure 4.3.

In the Keras neural network library the transposed convolution is not implemented in the 1D-case, instead the equivalent behaviour is accomplished by combining a layer that upsamples its input in combination with an ordinary convolutional layer [Chollet, 2016].

4.3 Autoencoders

Autoencoders are neural networks where the aim is to reconstruct the input as the network output. Typically the autoencoder has an architecture similar to what is

shown in Figure 4.5 where the signal's size through the network is "hourglass"-like. This allows the autoencoder to learn how to compress the input information to a lower-dimensional *latent space* [Chollet, 2017]. Applications include image compression [Alexandre et al., 2019], image denoising [Ali et al., 2018] and feature extraction. For example if the autoencoder is fed with noisy images and the target images are without noise, the autoencoder can learn to denoise unseen images.

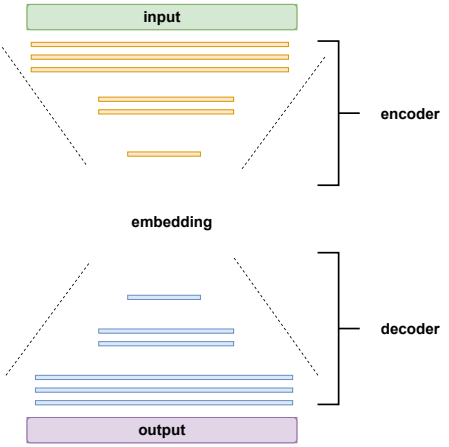

Figure 4.5 Sketch over autoencoder architecture.

In this project the autoencoder serves as an unsupervised feature extractor. This works by training the autoencoder to reconstruct its input and then utilizing the embeddings created in the lower-dimensional latent space and feed them to a classifier. The class labels associated with the input are not used in the reconstruction process making the method unsupervised. Instead the assumption is made that if the input data differs, then the encoded representations should separate well in the latent space. The encoder and decoder depth affects the abstraction level of the captured features [Goodfellow et al., 2016].

Visualizing what neural networks learn

A problem with neural network models are that they are often considered as black-box models, i.e., it is not well understood what patterns the network actually finds in the input data. Lately there has been progress in the area of understanding convolutional neural network decisions. One of these methods is the *GRAD-CAM* method [Selvaraju et al., 2017].

For a given classified input map, the method extracts the feature map of the final convolutional feature extracting layer and then computes the gradient of the class decision with respect to said feature map. By this method it is possible to produce a heatmap of the regions in the input that affected the class decision more.

5

Methods

5.1 Experimental setup and data gathering

Sensor measurements and video footage was available for the project. The clock between the video and sensors was not synched and therefore manual inspection of video was necessary in order to connect movements to sensor data in time.

Each patient was tested on two different occasions (in this report referred to as *measurement days*), one day for each of the two testdoses of different dopamine-based medication. Prior to the measurement day, the patient stopped taking his/her ordinary dopamine medication in order to be as "off" medication as possible when given the testdose prior to measurements.

On each measurement day the patient was first given the testdose, then measurements occured every 30 minutes for up to 5 hours. After 5 hours the medication should no longer be active and the patients' states should have returned to "off". During the measurements each patient performed the following tasks:

(i) Describe a picture

(ii) Drink from a cup

(iii) Take on/off a lab coat

(iv) Walk 2 x 4.5 m

Sensor measurements were performed by using the "Medoclinic App" for mobile phones [MedoTemic, released 2018]. The mobile phone was placed in a pouch that was fastened around the patient's trunk with a belt. Figure 5.1 shows how the mobile phone was attached to the body.

The raw sensor output contained 6-dimensional data; accelerations $(\text{acc}_x, \text{acc}_y, \text{acc}_z)$ and gyroscope data $(\text{gyro}_x, \text{gyro}_y, \text{gyro}_z)$ which were sampled with a sampling frequency of 100 Hz. A problem from the engineering point of view is that the phone was not consistently placed in the same way every time when attached. Figure 5.2 shows two possible orientations how the phone could be positioned in the belt

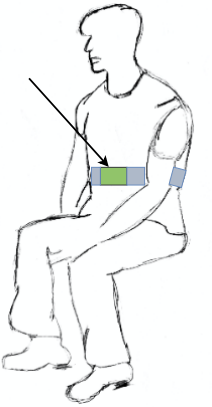

Figure 5.1 Sensor placement on body.

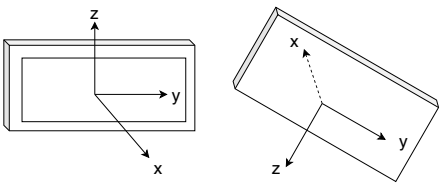

Figure 5.2 Examples of ways the phone's coordinate system could be oriented during measurements.

pouch. This creates a problem since the coordinate system of the sensors would not be consistently oriented in the same way between measurement occasions.

The accelerometer and gyroscope of the sensor can be used together to estimate the coordinate systems orientation relative to the direction of gravity. The different sensors are fused together by applying a low-pass filter to the accelerometer (due to high-frequency noise) and a high-pass filter to the gyroscope (due to the tendency of drifting), as explained in e.g. Mathworks article [Mathworks, 2020]. The coordinate system of the mobile phone can then be rotated to align one coordinate axis in the direction of gravity. This gives one direction for which the data can be compared in a consistent way between different measurement occasions and independently of how the mobile phone is placed. The trade-off is that the other two coordinate axes' orientations in the perpendicular plane are unknown.

For the rest of the work, all original measured data was rotated according to above and the acceleration in the z-direction was extracted and used in the engineered methods.

Dividing data per task

Figure 5.3 shows the data for a single measurement occasion in the study. The different tasks that the patient performed have been highlighted by color in the plot. These were identified by viewing the corresponding video footage.

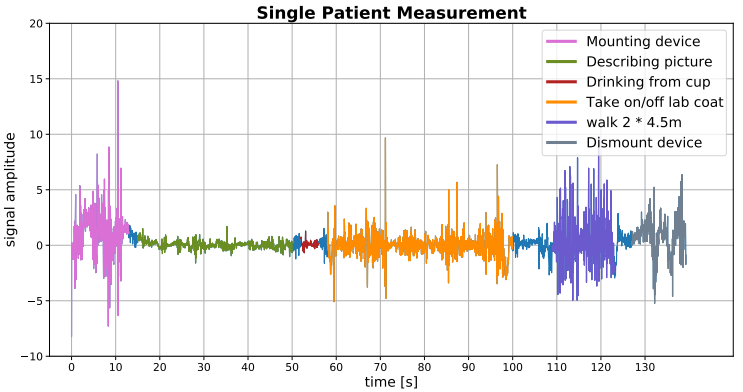

Figure 5.3 z-direction accelerometer measurement example for a patient in the study.

Tasks were cut out from the data and analyzed separately. At this stage we selected the tasks that were as consistently performed as possible between measurement occasions and different patients. We did this to give a consistent basis from which the disease symptoms could be analyzed. Not all of the tasks were deemed useful to analyze. The “drink from cup”-task was typically short (2-4 seconds, resulting in 200-400 data points) and not performed in a consistent way between patients. The cup was empty during experiments which allowed for some unique interpretations of how the task was performed. The “take on/off lab coat” was also performed very uniquely for each measurement occasion. Sometimes the patients performed the task standing and sometimes sitting.

Under the assumption, the most promising task was deemed to be “describe picture”. The task was typically consistently performed and had a long duration ($\sim$20-40 seconds, resulting in 2000-4000 data points per occasion). Except for this task, also the “walk”-task was analyzed, since it was the only other task which was similarly performed for all patients and also not too short in data length. For the rest of the work, these two tasks were cut out and analyzed.

Convention for file layout during data storage

The data for the tasks “describe picture” and “walk” were cut out and stored in corresponding task-specific folders. Together with the data, an info string in JSON format was appended at the top of each file with necessary information to uniquely identify the data segment and to provide the doctor’s CDRS score as class label [ISO-21778:2017(E), 2017]. The patients identity was anonymized using the corresponding patient number from the study. A small excerpt from a data file is shown in listing 5.1. A code library was written is Python to load and process the collected data.

Listing 5.1 Generated task-specific data file.

```
# {'pt':12,'time':'1024','medicine':'no agonist','task':'describe pic','label':{'B':3}}
index,accX,accY,accZ,gyroX,gyroY,gyroZ,timeAcc,timeGyro
1800,0.889,2.761,0.065,0.085,-0.173,-0.012,18.0,18.0
1801,0.875,2.715,0.071,0.093,-0.176,-0.020,18.01,18.01
1802,0.858,2.696,0.072,0.086,-0.176,-0.034,18.02,18.02
1803,0.822,2.673,0.0695,0.084,-0.181,-0.045,18.03,18.03
1804,0.772,2.648,0.061,0.089,-0.180,-0.054,18.04,18.04
1805,0.733,2.587,0.071,0.093,-0.175,-0.060,18.05,18.05
1806,0.688,2.553,0.065,0.096,-0.173,-0.0671,18.06,18.06
```

Summary of data library

From the gathered data-library we obtained an overview of the task-specific data with information about class excitation, i.e., too what extent the different labels have sample support from different patients. Listing 5.2 and 5.3 show the class counts for tasks “describe picture” and “walk” respectively.

Listing 5.2 Class counts shown in JSON format according to the convention “class:counts” for task: “describe picture”

```
-------------------------------
Summary of data in library, task: describe picture
-------------------------------
Patient 1: Class counts: {0: 6, 1: 4, 2: 7, 3: 0}
Patient 3: Class counts: {0: 6, 1: 8, 2: 2, 3: 0}
Patient 4: Class counts: {0: 4, 1: 9, 2: 4, 3: 0}
Patient 5: Class counts: {0: 4, 1: 3, 2: 1, 3: 0}
Patient 6: Class counts: {0: 1, 1: 1, 2: 3, 3: 13}
Patient 7: Class counts: {0: 9, 1: 5, 2: 0, 3: 0}
Patient 9: Class counts: {0: 12, 1: 0, 2: 0, 3: 0}
Patient 10: Class counts: {0: 2, 1: 8, 2: 5, 3: 0}
Patient 11: Class counts: {0: 5, 1: 10, 2: 4, 3: 0}
Patient 12: Class counts: {0: 3, 1: 2, 2: 6, 3: 3}
Patient 13: Class counts: {0: 6, 1: 2, 2: 3, 3: 1}
Patient 14: Class counts: {0: 8, 1: 3, 2: 0, 3: 0}
-------------------------------
```

Listing 5.3 Class counts shown in JSON format according to the convention "class:counts" for task: "walk"

```
------------------------------
Summary of data in library, task: walk
------------------------------
Patient 1: Class counts: {0: 4, 1: 8, 2: 1}
Patient 3: Class counts: {0: 6, 1: 10, 2: 0}
Patient 4: Class counts: {0: 16, 1: 1, 2: 0}
Patient 5: Class counts: {0: 6, 1: 2, 2: 0}
Patient 6: Class counts: {0: 2, 1: 3, 2: 13}
Patient 7: Class counts: {0: 8, 1: 6, 2: 0}
Patient 9: Class counts: {0: 12, 1: 0, 2: 0}
Patient 10: Class counts: {0: 9, 1: 5, 2: 0}
Patient 11: Class counts: {0: 9, 1: 10, 2: 0}
Patient 12: Class counts: {0: 3, 1: 5, 2: 6}
Patient 13: Class counts: {0: 8, 1: 1, 2: 3}
Patient 14: Class counts: {0: 10, 1: 1, 2: 0}
------------------------------
```

Starting with the "describe picture"-data, we note that the label CDRS = 3 only occurs for patients 6,12 and 13 which most likely affects models to be more non-generic and biased towards these three patients. This label is also not seen in the "walk"-data even though the "walk" task was performed only shortly after the "describe picture"-task. We proceeded with preprocessing and modeling being aware of the dataset's limitations.

5.2 Preprocessing

The preprocessing section covers the steps of loading data from the libraries created in the previous section, and the steps taken before the processed data was fed into classifying models. The classifiers take sets of training, validation samples to learn from and are then evaluated on the test set. In order to create more data from the files in the library, shorter segments referred to as *snippets* were sampled from the task segments. This was done by sliding a window over the signal and at each viable window location copy out the currently selected "snippet". The window was controlled by two parameters:

(i) `window_size`: The length of the window.

(ii) `slide_interval`: The length of the window jumps.

Figure 5.4 shows the single-task data "describe picture" which has been cut out from the measurement in figure 5.3, with illustration of how a window is slided over the data.

A "describe picture" data segment is typically around 20-40 seconds long which corresponds to $2000-4000$ samples with the sampling frequency of 100 Hz. This is

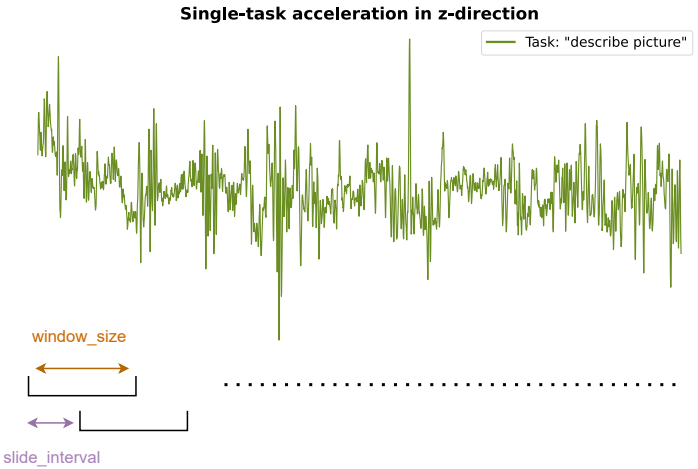

Figure 5.4 z-direction acceleration data for the task “describe picture”, cut out from a patient measurement occasion.

plenty of data to perform transformations to the frequency domain. The size of the sliding window and the slide interval for the window jumps are design parameters that were initially chosen as `window_size` = 800 samples, `slide_interval` = 100 samples. This results in overlapping windows that could be interpreted as the data being *augmented* which could still be beneficial in e.g. neural network models. Later in the project we simulated the window size as described in section 5.4, which resulted in a smaller window size `window_size` = 200 which was selected for the final simulations.

Figure 5.5 shows a random set of snippets for the task “describe picture” from the dataset which have been extracted with the above method. At this stage the mean of each snippet is subtracted to make them zero-mean in order to Every snippet is associated with a measurement occasion by the information in the string above each subfigure.

The extracted snippets are processed by the feature extraction algorithms. When examining the data from the task “walk”, the acceleration typically looks like the example in Figure 5.6. It is notable how the patient first performs several steps, then turns in the middle of the data segment and walks back to the starting position. In order to apply the same methodology as for the describe picture task, the middle segment with the turn is cut out. The result is data which only contains the “steps” which are of interest to analyze.

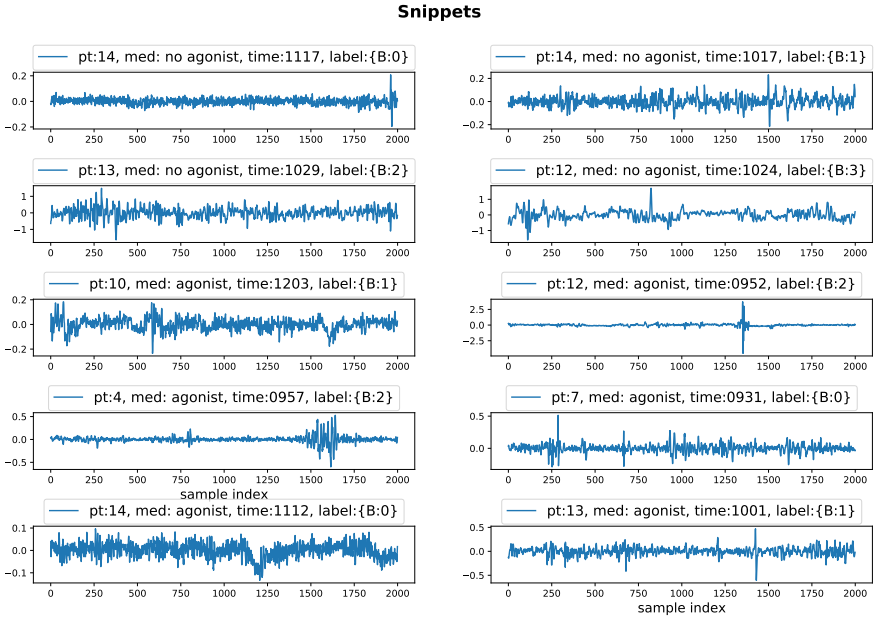

Figure 5.5 A random collection of snippets that are samples from the "describe picture"-task. They are uniquely identified to a a measurement occasion with attached string which also includes the label for the body "B".

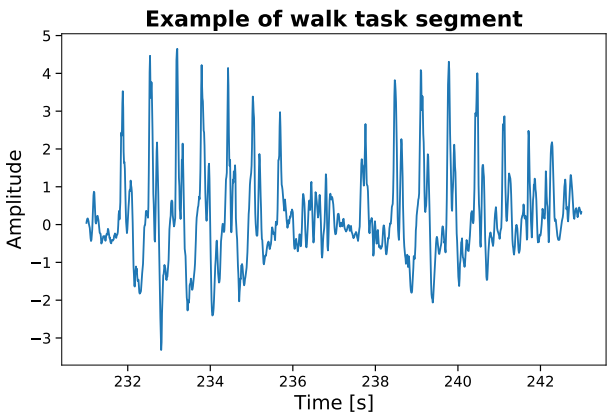

Figure 5.6 Example of accelerometer data from a patient performing the "walk" task.

In the same way as for the “describe picture” task, snippets are copied out from the original “walk” - task and the methods are applied to them. Since the task segments for this task are typically much shorter in time, smaller window sizes ought to be used.

Energy measure

As an energy measure of the time signals we used the variance. Figures 5.7 and 5.8 show the time signal variance as histograms separated by class for two patients in the study. This indicates that the classes are somewhat separated by the time signal variance and therefore this measure can be useful as a feature in the classification problem. Large outliers are mapped to a lower threshold which results in the bins to the far right of the plots, in order to improve visibility. After eyeballing similar plots for more patients in the study, this feature showed good potential and is used by simple reference models and the algorithm in section 5.4.

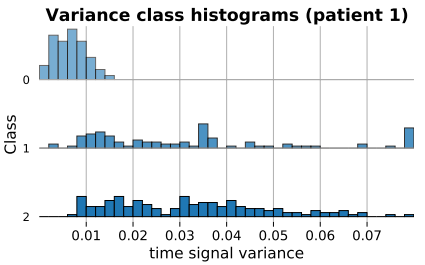

Figure 5.7 Histograms over time signal variance computed for the dataset from patient 1.

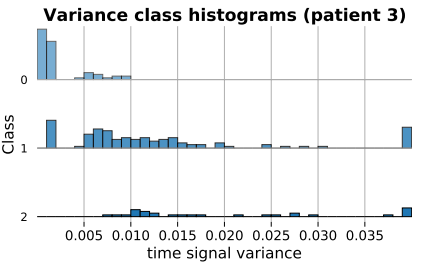

Figure 5.8 Histograms over time signal variance computed for the dataset from patient 3.

Task time duration

Heuristically it is tempting to investigate the task time duration, i.e., the time it takes for a patient to complete a task. This could possibly relate to disease symptoms. However after examining this feature it was found to be more patient specific than class specific, and was deemed as not useful in the modeling.

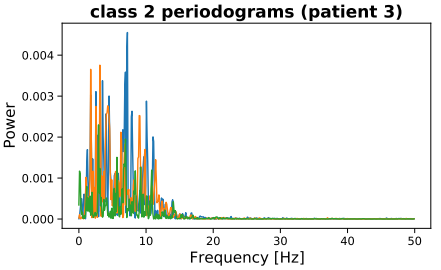

Figure 5.9 A couple of periodograms labeled as class 2 from patient 3.

Periodogram

We used the periodogram as a one-dimenional, non-parametric spectral estimate for the task-specific snippets. The periodograms were computed using a 1024-point fft, with zero-padding applied to shorter signals. The signals were windowed using a "Hamming" window. The signal energy which relates to the time signal variance is seen in the periodogram estimates as the magnitude of the power. Figure 5.9 shows a couple of overlapping periodograms computed from patient 3, class 2. The sampling frequency of 100 Hz results in a Nyquist frequency of 50 Hz, meaning that this is the highest frequency we can capture given the sampling frequency. However, eyeballing countless of periodograms belonging to different patients and classes, there is very little to none activity in the frequency interval $[25, 50]$ Hz as seem in e.g. Figure 5.9. It is also discussed in the work by Goschenhofer et al. (2019) that PD related patterns do not exceed 20 Hz. For these reasons the content in the interval $[25, 50]$ Hz was discarded since it would significantly reduce training time for the artificial neural network models.

Periodograms from different patients and classes were eyeballed in order to find spectral patterns, which could potentially be used in a parametric modeling approach. Figure 5.10 shows periodograms computed for a set of snippets from class 0 (left column) and class 2 (right column) sampled from patient 3 for the task "describe picture". In order to visually compare the spectral patterns, the periodograms were individually normalized.

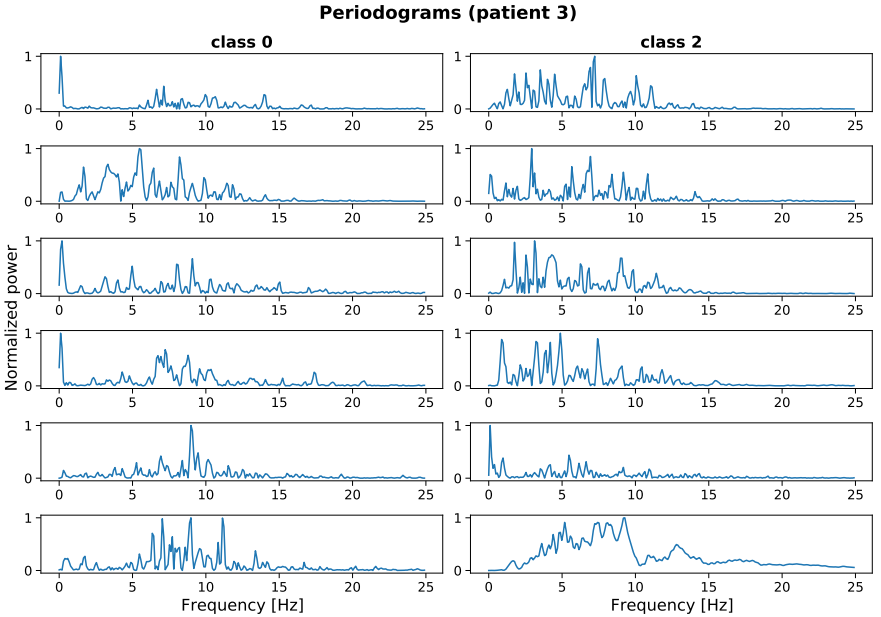

Figure 5.10 A set of periodograms computed on snippets from patient 3 in categories class 0 (left column) and snippets of class 2 (right column).

Figure 5.10 shows periodograms from snippets of length 800, meaning that 224 zeros are padded at the end of each signal. By viewing several sets of periodograms no apparent pattern related to one certain frequency could be identified. In some class 2 periodograms, there appears to be some peaks in the interval [0,5] Hz that stand out from the class 0 periodograms. In order to investigate the occurance of peaks in different classes, we examined two features which are:

(i) The amount of peaks above a certain threshold.

(ii) The amount of spectrum points, i.e., values of the periodogram, above a certain threshold.

Counting peaks

We developed a simple algorithm for counting peaks, with the short Python implementation shown in listing 5.4.

Listing 5.4 Peak counting algorithm as implemented in Python

```
def count_peaks(data, threshold):
    '''
    Counts peaks in "data", which typically is a spectral estimate.
    Only peaks above or equal to "threshold" are accounted for.
    '''
    counts = 0
    for i, point in enumerate(data):
        if point < threshold:
            pass
        # left border condition
        elif i == 0 and point < data[i+1]:
            pass
        # right border conditions
        elif i >= len(data) or i == len(data)-1 and point < data[i-1]:
            pass
        elif i == len(data)-1 and point > data[i-1]:
            counts = counts + 1
        # if we are climbing or descending a peak
        elif point < data[i-1] or point < data[i+1]:
            pass
        else:
            counts = counts + 1
    return counts
```

In a similar manner as earlier, class-wise histograms of the peak counts as well as the number of points above the threshold were generated for the dataset. Using a threshold of 0.8, the histograms for patients 1 and 3 are shown in Figures 5.11, 5.12 and 5.13, 5.14 respectively.

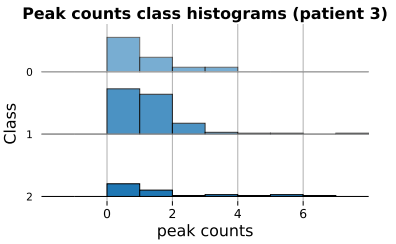

Figure 5.11 Histograms over peak counts in periodograms from patient 3.

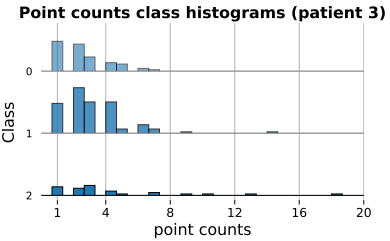

Figure 5.12 Histograms over the number of spectrum-points over a certain threshold in periodograms from patient 3.

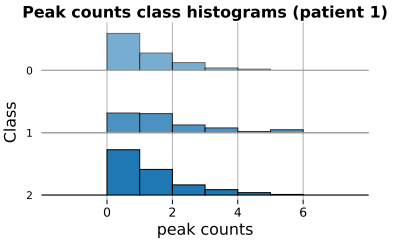

Figure 5.13 Histograms over peak counts in periodograms from patient 1.

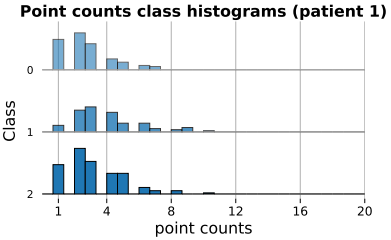

Figure 5.14 Histograms over the number of spectrum-points over a certain threshold in periodograms from patient 1.

We see that these features do not separate the classes as well as the time signal variance, however by viewing the figures for patient 3 there seems to be a few high counts only in classes 1 and 2. These features are used in the algorithm in section 5.4.

Spectrogram

Examining spectrograms computed from signals of varying length did not visually present anything useful, and were therefore omitted in the report. However, we did use spectrograms as input to one of the models (2D convolutional neural network) that is evaluated in the section about patient-generic modeling in the results chapter. When computing the spectrograms, the short-term Fourier transform windows were selected to have length = 200 and use a 512 point fft.

5.3 CNN classifier

We used a method consisting of a convolutional neural network with spectral estimates as input. A flowchart illustrating the method is shown in Figure 5.15. The input time signal is first segmented as described earlier and the segments are transformed to periodograms. All periodograms in the dataset are normalized to the interval $[0, 1]$ by the largest measured value, which maintains the difference in power between different periodograms. The normalization was done to enhance the convergence speed of the optimization algorithm when using binary cross-entropy loss function and sigmoid activation [Chollet, 2017].

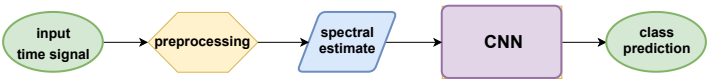

Figure 5.15 Flowchart over the CNN classifier method, where the method takes a signal in the time domain and maps it to a class.

We tried multiple varieties of layer architectures for the CNN, both concerning hyperparameters but also dimension and loss functions. The general architecture we developed and later used as basis is shown in listing 5.5, consisting of three convolutional layers and 2 dense layers. A novel regularization technique called *Batch Normalization* developed by Ioffe and Szegedy (2015) was tested but did not yield significantly better results.

Listing 5.5 CNN 1D architecture

```
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 250, 16)           128
_________________________________________________________________
max_pooling1d_1 (MaxPool1D)  (None, 125, 16)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 119, 32)           3616
_________________________________________________________________
max_pooling1d_2 (MaxPool1D)  (None, 59, 32)            0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 53, 64)            14400
_________________________________________________________________
max_pooling1d_3 (MaxPool1D)  (None, 26, 64)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1664)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               426240
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 444,641, Trainable params: 444,641
```

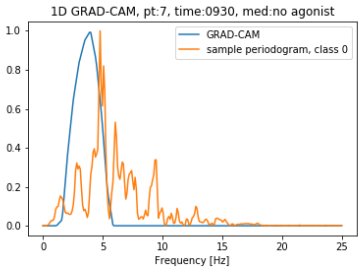

Figure 5.16 Periodogram from class 0 overlayed by a GRAD-CAM heatmap indicating important regions that had more impact on the class decision.

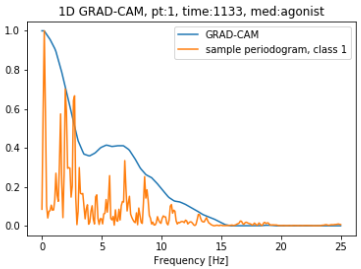

Figure 5.17 Periodogram from class 1 overlayed by a GRAD-CAM heatmap indicating important regions that had more impact on the class decision.

GRAD-CAM validation

Apart from loss function validation using the validation set, we also used GRAD-CAM validation. By this method, we were able to produce 1D-plots of the areas in the input that were more important for the class decision. We could then plot the input periodogram with the GRAD-CAM as overlay. Figure 5.16 and 5.17 show GRAD-CAMs for one periodogram classified as 0 and one periodogram classified as 1 respectively. We were able to see that low frequency content in general was more important invariant of class, which also strengthens the decision to toss away frequency content in the interval $[25,50]$ Hz. We were not able to draw any specific conclusions on specific important frequencies by this method. This is partly due to the computed GRAD-CAMs resolution being to low. In future work we suggest to use this method for deeper neural networks which might increase heat map resolution.

5.4 Feature extracting autoencoder with discriminant

In order to use a convolutional neural network in an unsupervised fashion, we propose a scheme depicted in Figure 5.18. In this case the convolutional neural network part of the algorithm will be focused entirely on reconstruction of its input and during this process produce lower-dimensional embeddings. The idea is that if the input data from different classes differs, then this should be reflected in the latent space (embedding space) of the autoencoder, where class separation should be achieved in an unsupervised fashion.

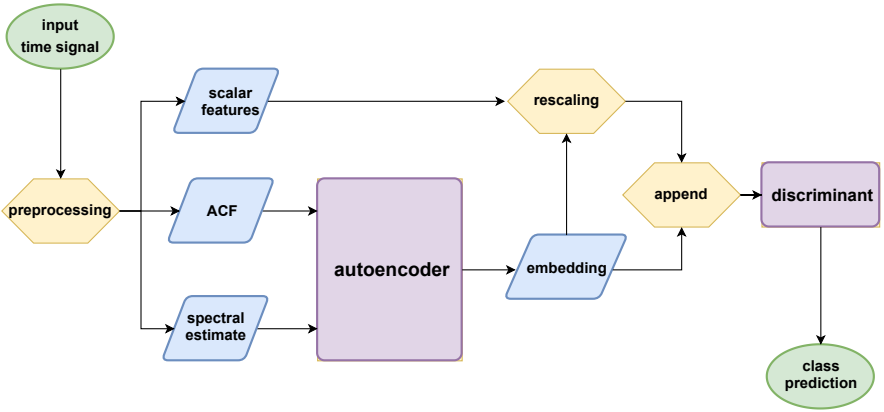

Figure 5.18 Flowchart over the proposed algorithm which utilizes a feature extracting autoencoder together with a linear discriminant.

The method works as follow:

(i) The input data is fed through a preprocessing step where a spectral estimate, the ACF and scalar features are computed. The spectral estimate is computed by e.g. the periodogram or the spectrogram.

(ii) The neural network part, given by the autoencoder, trains on finding patterns in the ACF and spectral estimate of the training set.

(iii) After training of the autoencoder, it produces embeddings by encoding input samples to latent space using the encoder part of the autoencoder.

(iv) The embeddings are concatenated with the scalar features to append additional information and the result is fed into a discriminant which has to be individually trained with embeddings from the training set.

(v) The discriminant performs class predictions.

In the following subsections, we explain how different segments in the algorithm are optimized.

Preprocessing

The slide window size was selected through grid-search simulating the method for a set of window sizes and evaluating the classification score in each iteration. Each window size was simulated 100 times where the training, validation and test sets were randomly selected as usual. Due to this procedure being time consuming it was only conducted for data belonging to two patients, with the assumption that

the result generalizes to the whole dataset. The parameter `slide_interval` was chosen as the same value as `window_size` for window sizes up to 400 samples (no window overlap) and reduced to `slide_interval` = 200 for the remaining window sizes. This was needed to make sure that there was a sufficient amount of snippets sampled, which becomes a problem for the larger window sizes.

The result of the simulation for patients 1 and 3 are shown in Figures 5.19 and 5.20, where histograms of the macro f1-score (for the binary classification problem, where CDRS scores 1, 2, 3 are grouped together) are formed for each window size.

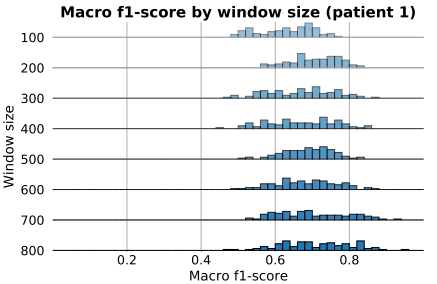

Figure 5.19 Histograms over the macro f1-score obtained in simulations for different window sizes when using the dataset from patient 1.

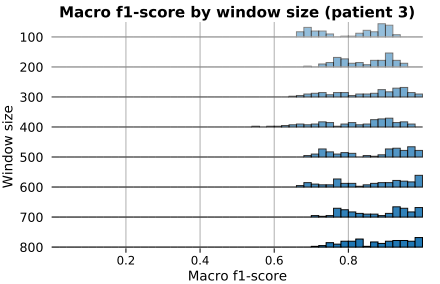

Figure 5.20 Histograms over the macro f1-score obtained in simulations for different window sizes when using the dataset from patient 3.

We selected suitable window parameters by summarizing the simulation results in a combined score, which for each distribution is the mean divided by the standard deviation. This score reflects that we want a high mean together with a low standard deviation for the macro f1-score distributions. The individual scores as well as the combined mean are shown in table 5.1. According to this metric, the highest score is achieved for `window_size` = 200.

Table 5.1 Combined score results

window size	100	200	300	400	500	600	700	800
patient 1	8.5	11.0	7.43	7.38	10.49	8.23	7.38	7.37
patient 3	8.89	12.34	9.04	7.99	8.75	8.59	10.48	10.63
mean	8.70	**11.67**	8.24	7.69	9.62	8.41	8.93	9

Autoencoder

The autoencoder is trained separately in the algorithm. Its task is to reconstruct its input, being the spectral estimate and ACF, as well as possible. This is achieved by minimizing the MSE (Mean Squared Error) between the input and output which is selected as the loss function for the autoencoder. A multiple input multiple output (MIMO) model is used for the implementation. A conceptual sketch is shown in Figure 5.21, with the final optimized architectures for the ecnoder and decoder including all layer shapes shown in Appendix figures B.1 and B.2.

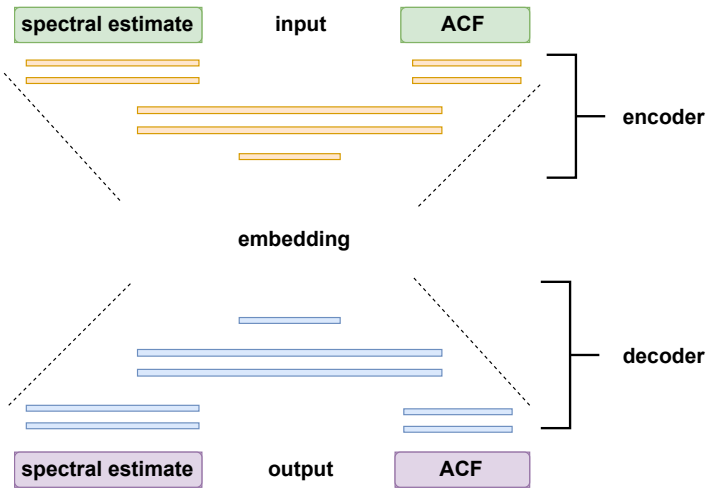

Figure 5.21 Sketch over our MIMO autoencoder architecture, which shows how the two input channels are combined into a common latent space and then decoded into separate pipelines.

The inputs are normalized to the interval $[0, 1]$. When having big power difference in the not-normalized periodograms as input, the autoencoder seemed to be biased towards periodograms with big power and reconstructed all samples as such. Instead, the important feature of power difference is reflected by the time signal variance, which is included in the method as a scalar feature.

The different pipelines for the spectral estimate and ACF are optimized individually by simulating different models and evaluating test set MSE. In these simulations the number of convolutional layers, the kernel sizes and the use of special layers such as batch normalization are compared by the MSE metric resulting in model choices. Apart from the validation and test set MSEs, another used validation metric was to eyeball the reconstructions created by the autoencoder. Figures 5.22 and 5.23 show reconstructions of periodograms (selected as spectral estimate) and ACFs respectively from the test set for one simulation of data gathered from pa-

tient 3. In this simulation the autoencoder was trained until the validation set MSE did not decrease for 100 epochs.

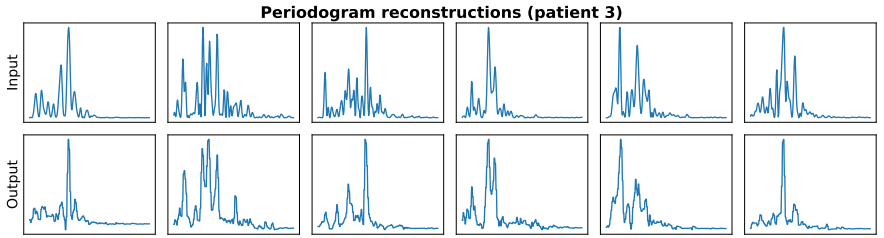

Figure 5.22 Periodogram reconstructions. The first row shows input periodograms and the second row shows the reconstruction as output.

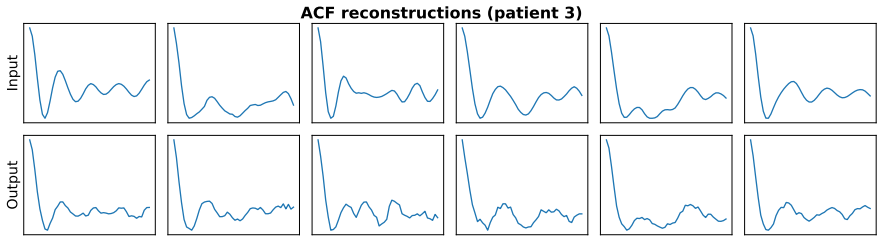

Figure 5.23 ACF reconstructions. The first row shows input ACFs and the second row shows the reconstruction as output.

Although the reconstructions are not perfect, we see that the main behaviour with important peaks is captured in the periodogram reconstructions. It is worth to comment that the autoencoder has not trained on these samples as they are drawn from the test set.

In order to select an appropriate latent space size, we simulated how the reconstruction error from the autoencoder depends on the latent space size. This was done by fixing a dataset for one patient and a set of encoding dimension to test, in this case even numbers between 2 and 32. For every value of encoding dimension, 100 autoencoders were trained to account for the stochasticity in the autoencoder optimization algorithm. The result was averaged for every encoding dim value, and the resulting plot is shown in Figure 5.24.

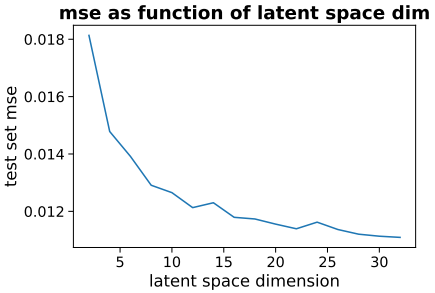

Figure 5.24 The mean squared error of the output reconstruction for the autoencoder as a function of the latent space dimension. These results are obtained by simulation.

We opted for an encoding dimension of 16 which is a tradeoff between having a low reconstruction MSE and keeping the resulting embedding small in dimension to later make the discrimant possible to fit. In the case of fitting Fisher's linear discriminant, the covariance matrix requires more data to obtain full rank for a higher dimensional embedding.

Rescaling features

The feature values in the embeddings produced by the autoencoder are by default on another magnitude scale compared to the scalar features such as the time signal variance. This can have negative impact on certain algorithms in the final block of our method such as principal component analysis (PCA). Features on different magnitude scales can make the algorithms biased towards features with large magnitude, which are given more importance. This problem is discussed in e.g. the paper by [Leznik and Tofallis, 2005] and on Scikit-learn's website [*Importance of Feature Scaling* 2020]. To deal with this issue and make our method susceptible to different final-block clusering/classifying algorithms, the scalar features that are appended to the autoencoder-produced embeddings are scaled to be in the same magnitude as the embedding features. This is implemented by computing the average embedding feature range and then scaling the features accordingly.

Discriminant

Having the autoencoder-produced embeddings concatenated with scalar features, we end up with 19-dimensional embeddings to be clustered. The first approach we attempted for clustering was to compute the angle between the 19-dimensional vectors, and mapping the resulting value to the interval [-1, 1] by applying the cosine function (also known as *cosine similarity*). However a much more efficient way for clustering and separating classes was found to be Fisher's linear discriminant, projecting the 19-dimensional data onto one or two dimensions.

Figures 5.25 shows smoothed histograms of 1D projections sampled from patient 3. Figure 5.26 shows a scatter plot of the same embeddings projected to 2D instead. The embeddings originate from 2 measurement occasions labeled class 0, 2 occasions labeled as class 1 and one occasion labeled as class 2 with a total test set size of 73 embeddings.

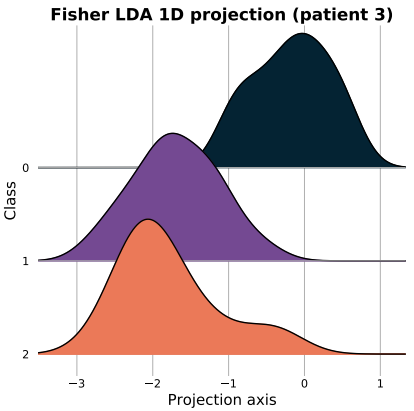

Figure 5.25 Smoothed histograms of 1D embedding projections using Fisher LDA method.

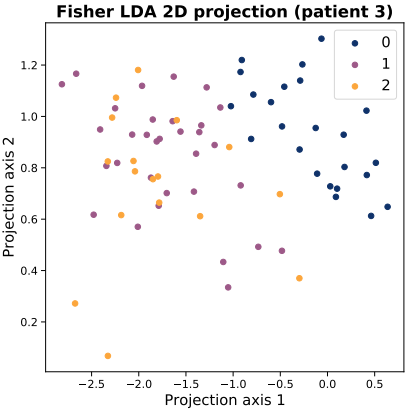

Figure 5.26 Scatter plot of 2D embedding projections using Fisher LDA method.

In this case it is seen that class 0 separates well but classes 1 and 2 have substantial overlap. The resulting projections are approximated to be normal distributed in order to use a Gaussian Naive Bayesian Classifier to form class predictions.

5.5 Simulation

During the course of the project different scenarios were simulated. At the project start, there was no annotated data available. As a preliminary study, we annotated the data from the “describe picture”-task ourselves taking out only the most extreme cases of the Hyperkinesia symptom and dividing these into two classes, generic for all patients. The simulation results for this binary classification problem are found in section 6.1.

After the preliminary study, the data set for the task “describe picture” was annotated by a certified doctor. At this stage we opted for a generic model for all patients and evaluated different models on the new annotated dataset. The simulation results for this scenario are found in section 6.2.

In the third study we tried to form individual models for each patient as well as developing the method explained earlier in section 5.4. The new method was simulated against the best performing model from the previous study. The simulation results from the individual modeling are shown in section 6.3.

Simulation strategy

The simulations have all been subject to the same simulation strategy. In each simulation iteration, the training, validation and test sets were randomly selected and the classification score was reported by evaluation of the test set, similar to an n-fold cross validation approach. By having a constant set of random seeds in the dataset-selection phase, each different model could have iteration-wise the same dataset which makes model-comparison more consistent. Further 70% of the data was used for training, 10% for validation and 20% for testing. The simulations where run using five 8-core Intel Core i7-9700 CPU nodes.

6

Results and discussion

In this chapter the results from the project are presented and discussed. Sections 6.1, 6.2 and 6.3 treat different simulation cases where the last section about individual modeling represents the case most likely suitable for an application. Section 6.4 treats how the algorithm would work in practice.

6.1 Results from preliminary study using own annotations for the data

The preliminary study used the "describe picture"- dataset, which we annotated ourselves into classes "big hyperkinesia symptoms" and "small hyperkinesia symptoms". The following patient-generic models were evaluated:

1. **1D_CNN**: A one-dimenional convolutional neural network with binary cross-entropy loss function taking periodograms as input.
2. **var_GNBC**: Naive Baysian Gaussian classifier taking the time signal variance as input.

The models were trained until the score on the validation set was not improved for 20 epochs. Histograms of the macro f1-score after 150 simulations for the two models are shown in Figure 6.1.

These preliminary results show that the time signal variance is an important feature as the single-feature classifier performs descent with a mean macro f1-score around 0.8. The 1D CNN model seems to perform significantly better (as the distribution is more shifted towards higher scores) which would imply that there is additional information to gain from the periodograms. It is seen that the variance of the score distributions is high, i.e., "how lucky" we are with the selection of training, validation and test set is important. This is most likely affected by the small size of the dataset, which was further decreased due to removal of ambiguous data, i.e., only the extreme cases were kept in the dataset. The results were used as motivation to properly annotate the dataset using an expert.

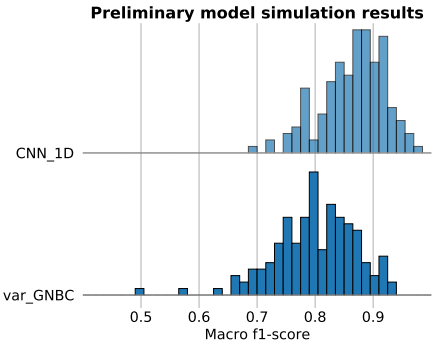

Figure 6.1 Histograms over the macro f1-score obtained during simulation for the two tested models.

6.2 Generic modeling using doctor-annotated data

At this stage of the project the doctor-annotated data for the "describe picture"-task was available as CDRS scores. We attempted to find a model that was generic for all patients. The best binary classification result was achieved for combining CDRS scores 0 and 1 to the first class and CDRS scores 2 and 3 to the second class. The models were simulated until the score on the validation set was not improved for 20 epochs. The following models were simulated:

1. **CNN1D_bin**: Binary 1D CNN using the binary cross-entropy loss function and having periodograms as input.
2. **CNN1D_reg_bin**: Binary 1D CNN regression model using the mean squared error loss function and periodograms as input. The model was trained on 4 classes but evaluated on 2 classes.
3. **CNN2D_bin**: Binary 2D CNN using binary cross-entropy loss function and spectrograms as input.
4. **Gaussian KDE NBC**: Naive Bayesian Classifier having the time signal variance as input feature. The feature class-wise pdfs were estimated using kernel density estimation with a gaussian kernel.
5. **Simple threshold classifier**: The input to the classifier is the time signal variance. The classes are separated by having the middle point between the class-means as a threshold.

Figures 6.2 and 6.3 show smoothed histograms of the macro f1-score and accuracy respectively for the different models after 150 simulations of the binary classification problem.

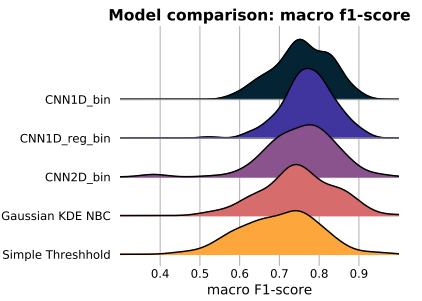

Figure 6.2 Smoothed histograms over the macro f1-score obtained during binary-class simulation of the different tested models.

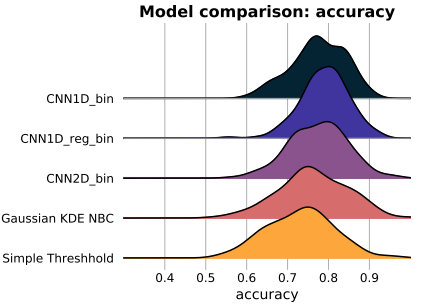

Figure 6.3 Smoothed histograms over the classification accuracy obtained during binary-class simulation of the models.

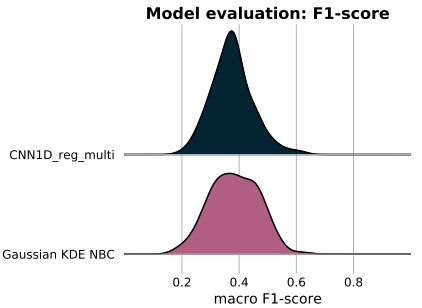

Figure 6.4 Smoothed histograms over the macro f1-score obtained during multi-class simulation for two models.

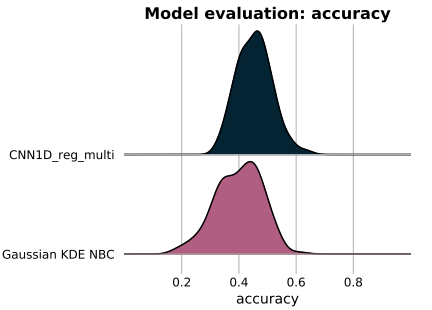

Figure 6.5 Smoothed histograms over the classification accuracy obtained during multi-class simulation for two models.

Figures 6.4 and 6.5 show the corresponding simulation results for the multi-class classification problem. Here only the best performing model and the bayesian classifier from the binary classification case were simulated due to time constraints.

The confusion matrix obtained from one of the simulations of the CNN 1D regression model is shown in equation 6.1.

$$C = \begin{bmatrix} 3 & 143 & 5 & 0 \\ 0 & 150 & 70 & 13 \\ 0 & 28 & 87 & 29 \\ 0 & 3 & 57 & 39 \end{bmatrix} \tag{6.1}$$

We see a descent classification result for the binary classification problem, al-

though the variance of the score-distributions appears to be quite large. For the multi-class case, the performance is worse which is also indicated by the confusion matrix from one of the simulations of the CNN 1D regression model. The confusion matrix shows that class 0 is most often predicted as class 1 and that the model could not is a satisfactory way distinguish between classes 2 and 3. This is likely partly affected by class-excitation where, e.g., class 3 only has sample support from two patients and would therefore probably not generalize well to the crowd.

According to the doctors, for an application it would make more sense to use CDRS score 0 as one class and put the other scores in another class due to CDRS score 0 representing the patient being "off medication". A satisfactory result for this class division would probably not be obtained with the current methodology, and therefore the project proceeded by modeling the patients *individually*.

6.3 Individual modeling using doctor-annotated data

We attempted individual modeling of each patient. The models were allowed to train until the validation score was not improved for 50 epochs, which typically happened before 500 epochs for all models. Like before, the same random seeds shuffling the data were used in order to reproduce same dataset in each iteration for all models. The following models were simulated:

1. **Feature extracting autoencoder** (using all features)
2. **Feature extracting autoencoder** (using everything but the ACF)
3. **Feature extracting autoencoder** (using only the periodogram)
4. **CNN_1D_reg_bin**: Binary 1D CNN regression model using the mean squared error loss function and periodograms as input.
5. **Gaussian NBC**: Naive Bayesian classifier having the time signal variance as input. The class-wise feature pdfs are assumed to be normal distributed.
6. **Simple guess classifier**: this classifier uses the "most-frequent" strategy where the most frequent label is predicted.

Describe picture dataset

In the following section in Figure 6.6 and 6.7 the simulation results from the best performing model (Model 2) are shown (blue color), together with the result from Model 6 (orange color) for reference. The corresponding simulation results from the other models are shown in Appendix C.1. The reference model acts as a tool to evaluate whether the simulated model is skillful or not. For the multi-class simulations only the results from the patients that have labels in more classes than two are shown.

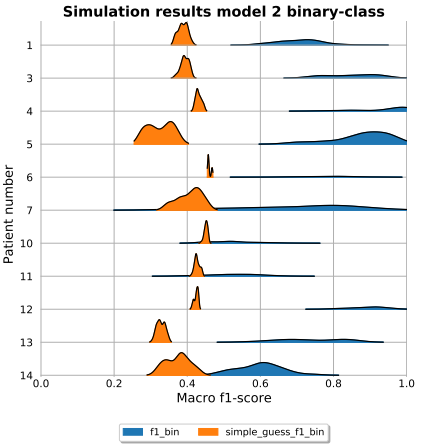

Figure 6.6 Smoothed histograms of the macro f1-score obtained during binary-class simulation of the "describe picture" dataset when evaluating model 2.

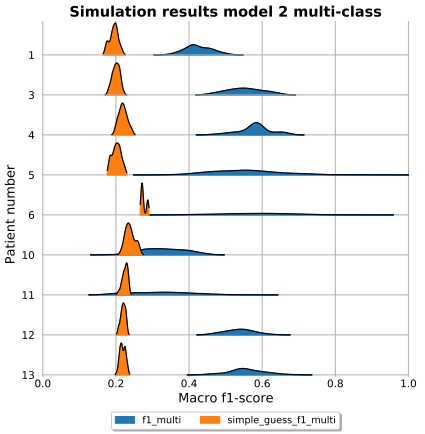

Figure 6.7 Smoothed histograms of the macro f1-score obtained during multi-class simulation of the "describe picture" dataset when evaluating model 2.

Figure 6.6 highlights that the performance differs between patients. For some patients the macro f1-score distributions are significantly higher (better) than others. The score distributions in general have a big variance as was seen in earlier simulation cases as well.

Classification score summary The model simulations are summarized in a total score which is defined for each distribution as the mean divided by the standard deviation. This is because we want a high mean together with a low standard deviation for each distribution. Tables 6.1 and 6.2 show the total score for each patient for the different models for the binary-class scenario and multi-class scenario respectively. At the very bottom the median of each model computed, as well as the mean when the highest and lowest values are omitted (in the table referred to as *adj. mean*). The reason for omitting the highest and lowest value is to be able to discard extreme outliers as were found in some examples (for example model 5, patient 6 in the binary-class scenario where the standard deviation proved to be very small).

Table 6.1 Total score results binary-class

Model	1	2	3	4	5
patient 1	12.7	11.9	9.9	4.0	14.4
patient 3	12.8	12.4	11.9	4.7	8.5
patient 4	20.2	13.4	17.5	6.1	7.8
patient 5	12.7	11.5	16.2	5.3	4.2
patient 6	9.8	10.5	10.4	4.4	320.9
patient 7	4.3	4.3	5.6	3.3	2.6
patient 10	6.5	8.3	6.4	47.9	23.2
patient 11	6.8	7.6	8.2	5.0	8.2
patient 12	22.0	18.5	14.4	5.3	10.4
patient 13	8.5	8.5	9.5	11.8	7.9
patient 14	9.7	9.8	10.3	5.6	4.1
adj. mean	**11.1**	10.4	10.8	5.8	9.9
median	9.8	**10.5**	10.3	5.3	8.2

Table 6.2 Total score results multi-class

Model	1	2	3	4	5
patient 1	11.9	12.0	7.1	3.8	12.7
patient 3	11.0	11.6	11.8	3.3	6.1
patient 4	15.9	13.7	13.8	4.8	8.2
patient 5	5.7	6.0	10.9	2.8	6.7
patient 6	5.7	6.3	6.2	3.3	11.1
patient 7	4.3	4.3	5.6	2.7	2.6
patient 10	5.2	5.9	5.4	3.5	3.7
patient 11	4.5	4.0	4.4	3.8	6.7
patient 12	14.6	13.8	12.9	3.7	6.3
patient 13	11.0	12.0	7.2	5.6	7.1
patient 14	9.7	9.8	10.3	3.9	4.1
adj. mean	8.8	**9.1**	8.6	3.7	6.7
median	9.7	**9.8**	7.2	3.7	6.7

It is apparent that models 1 and 2 seem to perform better when weighing together the total scores from simulation for binary-class and multi-class. Additionally when looking at the distribution plots it is hard to distinguish a difference between performance of the two models. For this reason model 2 is chosen as the best model since it is simpler. Taking this model and computing the overall mean macro f1-score and accuracy the following values were obtained:

- **binary-class overall mean macro f1-score**: 0.74
- **binary-class overall mean accuracy**: 0.81
- **multi-class overall mean macro f1-score**: 0.52
- **multi-class overall mean accuracy**: 0.65

However, it should be noted that there is an overall big variance in the results judging by the distributions in Figures 6.6 and 6.7.

Walk dataset

In this section the results are shown when evaluating the best model from the previous section (Model 2) on the "walk"-dataset. Figures 6.8 and 6.9 show the macro f1-score for simulating the binary-class and multi-class scenarios respectively. Note that only the patients that have multi-class labeled data are shown for the multi-class case. The corresponding total scores for this simulation were:

(i) **total score adj mean**: 7.8

(i) **total score median**: 7.1

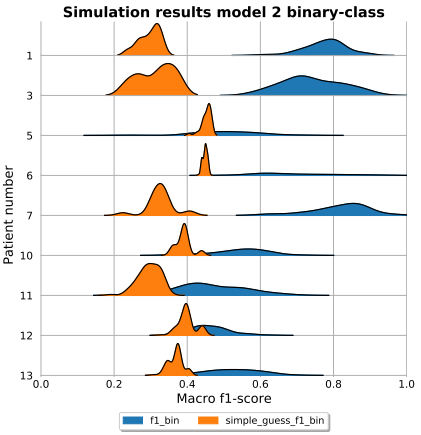

Figure 6.8 Smoothed histograms of the macro f1-score obtained during binary-class simulation of the "walk" dataset when evaluating model 2.

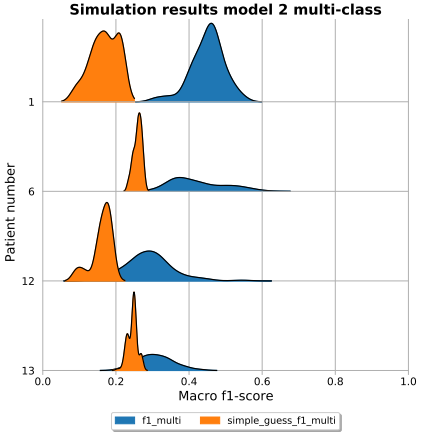

Figure 6.9 Smoothed histograms of the macro f1-score obtained during multi-class simulation of the "walk" dataset when evaluating model 2.

(i) **binary-class overall mean macro f1-score**: 0.61

(i) **binary-class overall mean accuracy**: 0.68

(i) **multi-class overall mean macro f1-score**: 0.5

(i) **multi-class overall mean accuracy**: 0.61

Here, all the results imply that a better classification performance was achieved when using the "describe picture"-task data compared to the "walk"-task data.

6.4 Use of algorithm in practice

In practice the algorithm could run in real time and would give a verdict every 2 seconds with the best model. The most natural way for the total prediction would then be to pick the prediction with most votes for a given occasion. The classification score for the model directly affects how successful the algorithm will work in practice.

Figures 6.10 and 6.11 show the time domain signal (lower subfigure) together with the algorithm score over time (upper subfigure) for two test-occasions for patient 12 in the multi-class scenario for the "describe picture"-task. In these cases the most votes would result in the correct total label prediction, however, for the

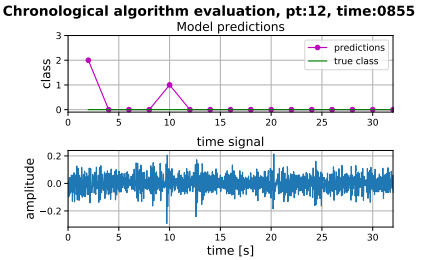

Figure 6.10 Algorithm CDRS predictions (upper subfigure) on unseen data with true class CDRS = 0 (lower subfigure) for patient 12.

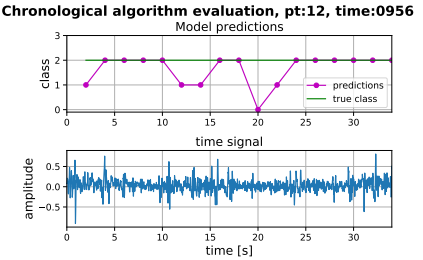

Figure 6.11 Algorithm CDRS predictions (upper subfigure) on unseen data with true class CDRS = 2 (lower subfigure) for patient 12.

multi-class scenario this is not usually the case. Most commonly there is confusion between classes 1 and 2. Class 0 is usually well separated resulting in good performance for the binary class scenario. <Z

Figures 6.12 and 6.13 show two out of five test set measurement occasions and model predictions for one selection of dataset for patient 1 for the "describe picture"-task. In this case the model is trained for the binary class scenario. Even though the classification macro f1-score is 0.72, all the test set measurement occasions are correctly classified when the prediction votes are counted, highlighting the usability even though the classification score is not perfect. It should also be noted that judging from Figure 6.6 several other patients have higher macro f1-score distributions than patient 1 indicating that the algorithm should perform better for these patients. However the algorithm's performance is still not perfect and could be further developed and evaluated.

6.5 Optimizations and limitations

Algorithm improvements

During the development of the *autoencoder with discriminant* method, we simulated some hyperparameters such as autoencoder layer architecture, size of the latent space dimension and the size of the window segmenting the time-domain signals. This is only a few of the hyperparameters and there are still endless of possibilities for model variations. One approach attempt deeper architectures for the autoencoder that enables learning of more abstract features. In our design, we increased depth until the autoencoder reconstruction was satisfactory, however, perhaps there are more abstract patterns to be learnt from the data that can be used to improve classification performance.

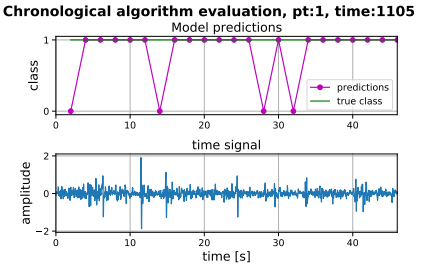

Figure 6.12 Algorithm CDRS predictions (upper subfigure) on unseen data with true class CDRS = 1 (lower subfigure) for patient 1.

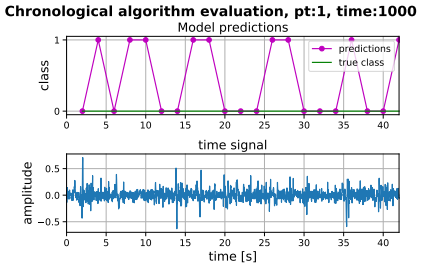

Figure 6.13 Algorithm CDRS predictions (upper subfigure) on unseen data with true class CDRS = 0 (lower subfigure) for patient 1.

The spectral estimate could perhaps be computed differently by applying some type of parametric or semi-parametric approach such as modeling the time domain signal as an AR(p)-process. AR-models have been successful in modeling speech and perhaps could be attempted for other human phenomena such as movement signals.

It might also be possible to improve the discriminant part of the method. Currently a linear discriminant is applied but a reasonable alternative would be to use a small multi-layer perceptron network that can learn non-linear relations. However, due to the current small size of the dataset this might not be a feasable alternative, but rather something that could be attempted as the size of the dataset is increased.

Limitations

A big limitation with the project is the current small size of the dataset. This is affected by the covid-19 virus which has delayed experiments and caused potential study subjects to decline participation. For this reason this Master's thesis only processes data from 12 out of ideally over 30 patients. The results obtained in this work should be validated on a larger dataset in the future.

In discussion with the expert who annotated the data it became known that some parts of the dataset, especially for the "walk"-task, were difficult to annotate meaning that the CDRS score was not easy to determine. For this reason it can be assumed that there is noise in the expert's labels which affects all classifying models negatively.

7

Conclusion

We have investigated several feature extraction methods and classification methods on different versions on the new Parkinson's disease dataset with single-channel accelerometer and gyroscope data acquired at the Lund University Hospital. Repeated simulation of different models allowed for comparison and conclusions on feasible model architectures for the problem. We found that individual modeling of the patients is necessary for good performance in a practical implementation, creating a tailored model for the patient.

The model classification performance is in general significantly better than simply guessing the most frequent label in the dataset and the results point towards the algorithm being practically usable for the binary classification case, i.e., when grouping CDRS score 0 in the first class and CDRS scores 1, 2, 3 in the other class. However, the dataset that the methods are evaluated on is still small and the results should be further validated on a larger dataset. In the multi-class case the classification performance, allthough skillful, appeared to not be accurate enough for practical usability with a few exceptions discussed in section 6.4. Here at least the following two factors are affecting; firstly that the size of the dataset is further decreased when dividing it into more classes and secondly that there is more noise in the expert's labels. It is important to note that the trained models are task-specific for the analyzed tasks, meaning that in practice the patient would need to perform a similar task in order for the model to be valid.

Our use of relatively short 2-second data windows (200 samples) found by grid-simulation indicate that 2D time-frequency transformations might not be feasible as features in the current method framework, due to this signal length being relatively short for 2D transformations.

The best performing model was found to be a feature extracting autoencoder acting on periodogram input, combined with additional "simple" features such as the time signal variance. The concatenated features are fed into a Fisher linear discriminant. According to the simulation results, use of patterns in the auto-covariance function (ACF) as features did not yield a significant improvement in the classification score.

In our work we have concluded that it is possible to use annotated data in order to train classifying models to detect medication-induced hyperkinesia.

8

Future ideas

For future work it might be of interest to try more advanced versions of the models from our methodology. However, this is heavily reliant on computational power.

An interesting approach which could be investigated is to treat the whole problem as unsupervised. Starting from the *autoencoder with discriminant*-method the discriminant could be replaced by an unsupervised clustering algorithm such as principal component analysis (PCA). Under the assumption that the patient shows a broad spectra of symptom states during data gathering, the data corresponding to different states should be able to be clustered in a feature space. The different clusters learnt during this process could then be used as reference when monitoring the patient. If this could be possible it solves two major problems:

(i) **The cost**: A doctor would not need to annotate the training data of the patient, which in practice would result in a lower cost.

(ii) **The label noise**: The doctor's labels would not be used, hence there is no problem of noise in the labels.

For the supervised approach used in this master's thesis, the labeling of the data could possibly be improved by having multiple experts annotate the same dataset and weigh together their annotations. This could possibly help to reduce label noise. Another idea is to make a finer annotation, i.e., divide the original data into smaller parts that are individually annotated. This might increase the accuracy in annotations.

Bibliography

Alexandre, D., C. Chang, W. Peng, and H. Hang (2019). "An autoencoder-based learned image compressor: description of challenge proposal by NCTU". *CoRR* **abs/1902.07385**. arXiv: 1902.07385. URL: http://arxiv.org/abs/1902.07385.

Ali, I., H. Lashari, S. Hassan, A. Maitlo, and B. Qureshi (2018). "Image denoising with color scheme by using autoencoders". **18**, pp. 158–161.

Berezina, M. A., D. Rudoy, and P. J. Wolfe (2010). "Autoregressive modeling of voiced speech". In: *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 5042–5045.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Blom, G. et al. (2017). *Sannolikhetsteori och statistikteori med tillämpningar (6th edition)*. Studentlitteratur AB.

Caliskan, A., H. Badem, A. Basturk, and M. E. Yuksel (2017). "Diagnosis of the parkinson disease by using deep neural network classifier". *Istanbul University - Journal of Electrical and Electronics Engineering* **17**, pp. 3311–3318.

Chaparro, L. (2019). "Signals and systems using matlab: third edition", pp. 1–855.

Chollet, F. (2016). *Building autoencoders in keras*. https://blog.keras.io/building-autoencoders-in-keras.html.

Chollet, F. (2017). *Deep Learning with Python*. Manning.

Dua, D. and C. Graff (2017). *UCI machine learning repository: handwritten digits data set*. URL: http://archive.ics.uci.edu/ml.

Dumoulin, V. and F. Visin (2016). *A guide to convolution arithmetic for deep learning*. cite arxiv:1603.07285. URL: http://arxiv.org/abs/1603.07285.

Fukunaga, K. (1990). *Introduction to Statistical Pattern Recognition (2nd Ed.)* Academic Press Professional, Inc., USA. ISBN: 0122698517.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Goschenhofer, J., F. Pfister, K. Yuksel, B. Bischl, U. Fietzek, and J. Thomas (2019). "Wearable-based parkinson's disease severity monitoring using deep learning".

Hagell, P. and H. Widner (1999). "Clinical rating of dyskinesias in parkinson's disease: use and reliability of a new rating scale". eng. Movement Disorders **14**:3, pp. 448–455. ISSN: 0885-3185. DOI: 10.1002/1531-8257(199905)14:3<448::AID-MDS1010>3.0.CO;2-0. URL: http://dx.doi.org/10.1002/1531-8257(199905)14:3<448::AID-MDS1010>3.0.CO;2-0.

Importance of Feature Scaling (2020). https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html. Accessed: 2020-08-13.

Ioffe, S. and C. Szegedy (2015). "Batch normalization: accelerating deep network training by reducing internal covariate shift".

ISO-21778:2017(E) (2017). *Information technology — The JSON data interchange syntax*. Standard. International Organization for Standardization, Geneva, CH.

Jakobsson, A. (2017). *An introduction to time series modeling*. Studentlitteratur AB.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Curran Associates Inc., Lake Tahoe, Nevada, 1097–1105.

LeCun, Y., P. Haffner, L. Bottou, and Y. Bengio (1999). "Object recognition with gradient-based learning". In: Springer-Verlag, Berlin, Heidelberg, p. 319. ISBN: 3540667229.

Leznik, M. and C. Tofallis (2005). *Estimating Invariant Principal Components Using Diagonal Regression*. English. WorkingPaper. University of Hertfordshire.

Lindgren, G., H. Rootzén, and M. Sandsten (2014). *Stationary Stochastic Processes for Scientists and Engineers*. CRC Press.

Lonini, L., A. Dai, N. Shawen, T. Simuni, C. Poon, L. Shimanovich, M. Daeschler, R. Ghaffari, J. Rogers, and A. Jayaraman (2018). "Wearable sensors for parkinson's disease: which data are worth collecting for training symptom detection models". *npj Digital Medicine* **1**. DOI: 10.1038/s41746-018-0071-z.

Martinez, A. M. and A. C. Kak (2001). "Pca versus lda". *IEEE Transactions on Pattern Analysis and Machine Intelligence* **23**:2, pp. 228–233.

Mathworks (2020). *Imufilter*. https://www.mathworks.com/help/fusion/ref/imufilter-system-object.html. Accessed: 2020-04-26.

MedoTemic (released 2018). *Medoclinic app*. https://www.medotemic.com/medoclinic.

Norvig, P. and S. Russel (2010). *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall.

Odin, P. et al. (2019). *Study protocol: an exploratory study on the impact of d2/3 receptor agonists on the phenomenology of dyskinesias in patients with parkinson's disease*.

Prabhu, K. (2013). *Window Functions and Their Applications in Signal Processing*. CRC Press.

Sandsten, M. (2020). *Time-Frequency Analysis of Time-Varying Signals and Non-Stationary Processes - An Introduction*. Centre for Mathematical Sciences, Lund.

Sathyanarayana, S. (2014). "A gentle introduction to backpropagation". *Numeric Insight, Inc Whitepaper*.

Scott, D. W. (2015). *Multivariate Density Estimation. Theory, Practice, and Visualization 2nd edition*. Wiley.

Selvaraju, R. R., M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra (2017). "Grad-cam: visual explanations from deep networks via gradient-based localization". In: *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 618–626.

A

Appendix A

A.1 Brain-stimulating picture

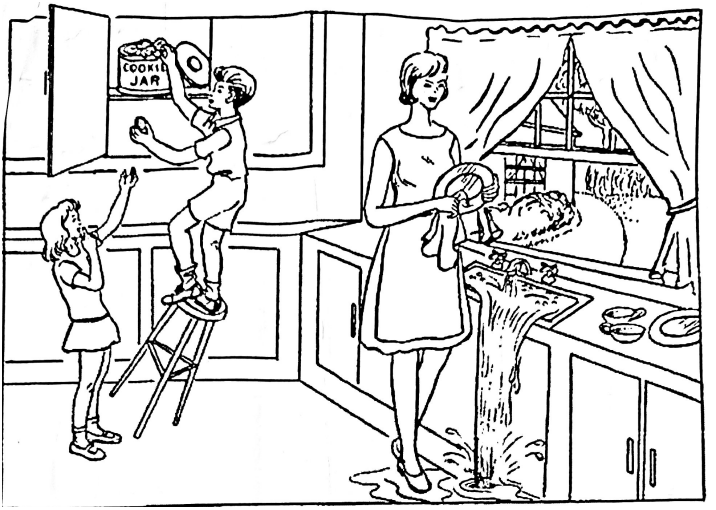

Figure A.1 The picture described during the “describe picture” task of the study.

B

Appendix B

B.1 Encoder architecture

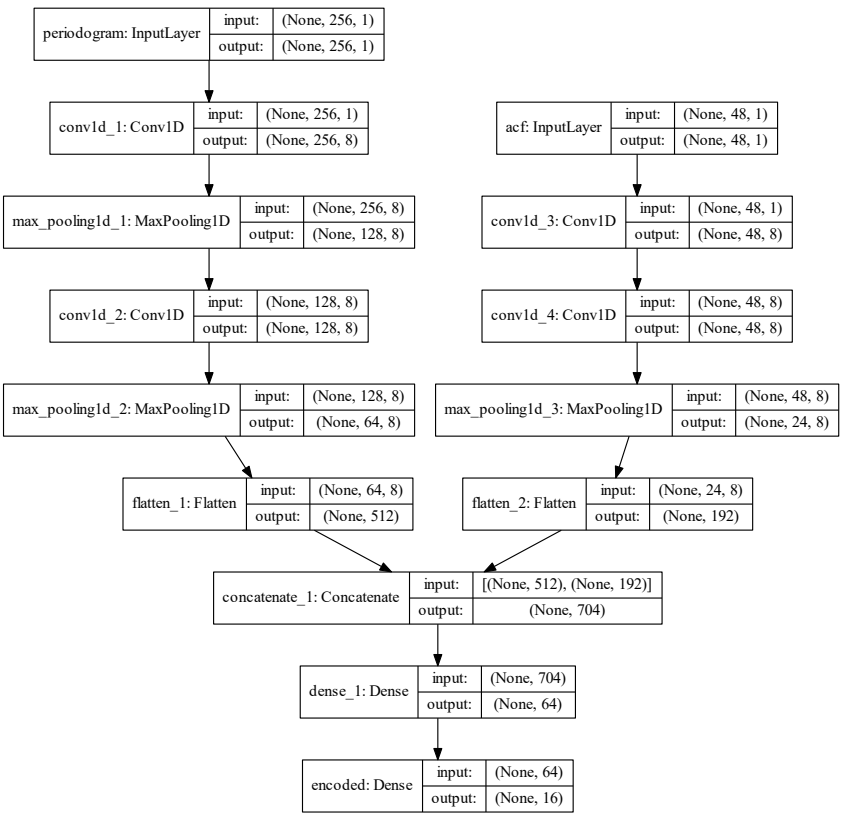

Figure B.1 Encoder architecture used in the autoencoder method.

B.2 Decoder architecture

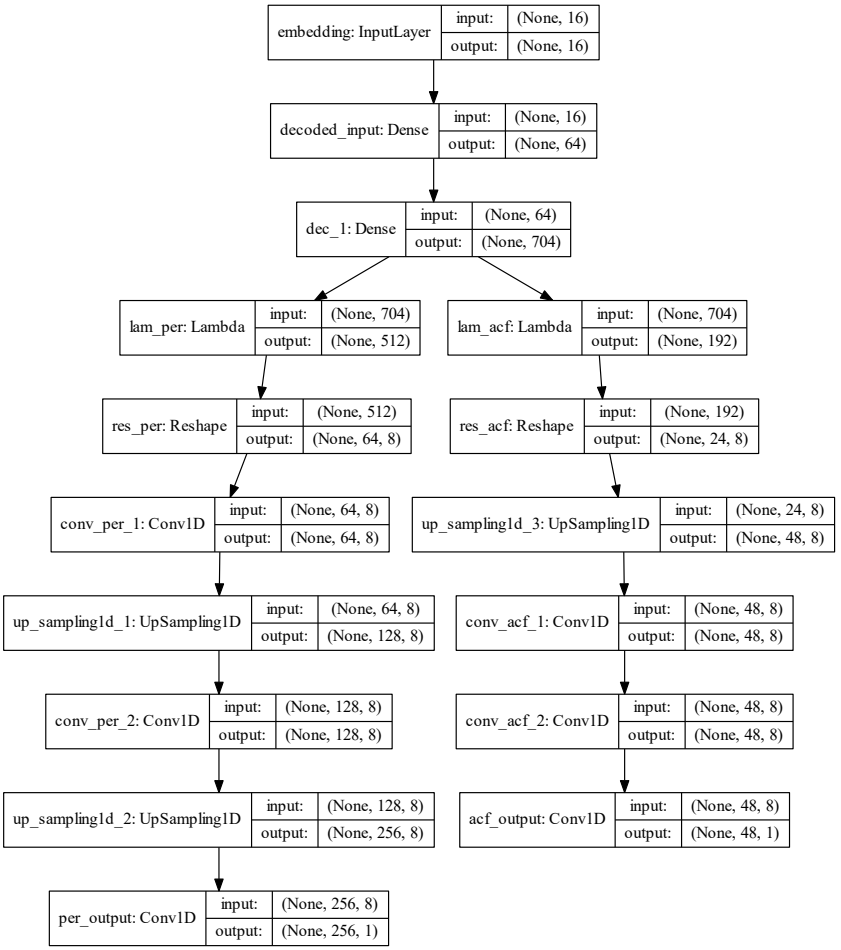

Figure B.2 Decoder architecture used in the autoencoder method.

C

Appendix C

C.1 Individual model results - comparison models

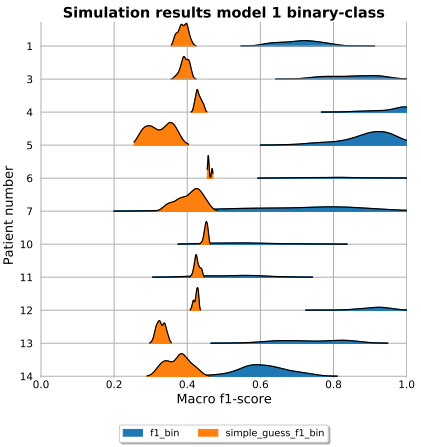

Figure C.1 Smoothed histograms of the macro f1-score obtained during binary-class simulation of the "describe picture" dataset when evaluating model 1.

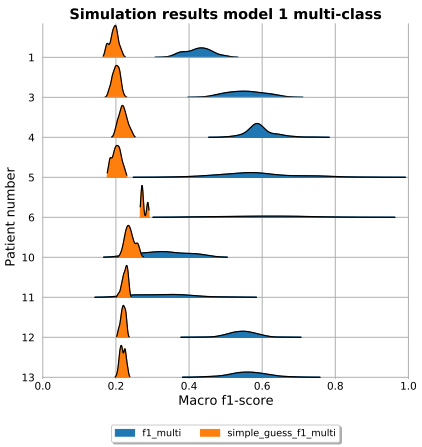

Figure C.2 Smoothed histograms of the macro f1-score obtained during multi-class simulation of the "describe picture" dataset when evaluating model 1.

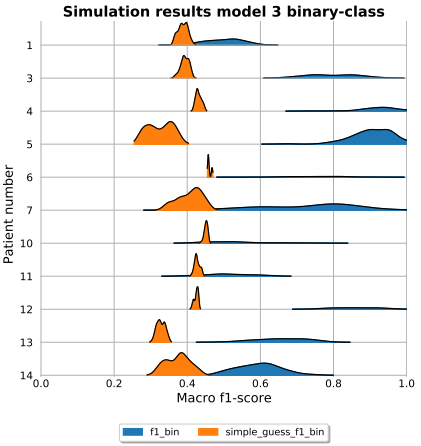

Figure C.3 Smoothed histograms of the macro f1-score obtained during binary-class simulation of the "describe picture" dataset when evaluating model 3.

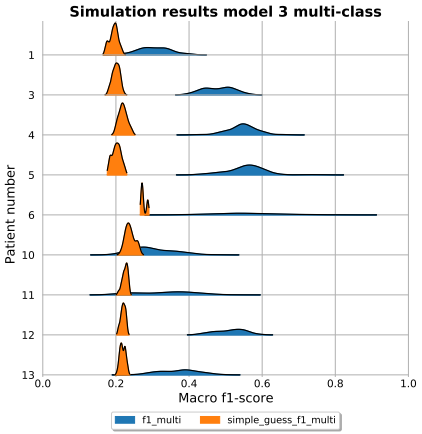

Figure C.4 Smoothed histograms of the macro f1-score obtained during multi-class simulation of the "describe picture" dataset when evaluating model 3.

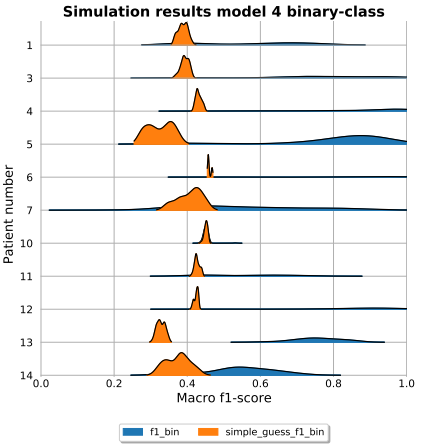

Figure C.5 Smoothed histograms of the macro f1-score obtained during binary-class simulation of the "describe picture" dataset when evaluating model 4.

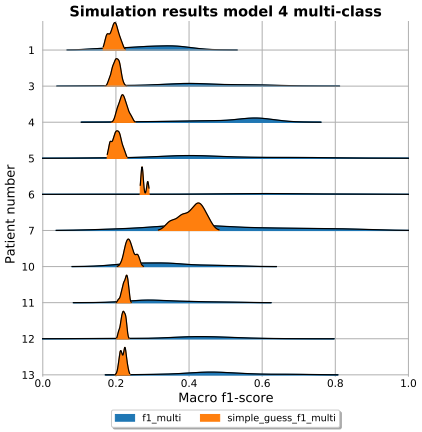

Figure C.6 Smoothed histograms of the macro f1-score obtained during multi-class simulation of the "describe picture" dataset when evaluating model 4.

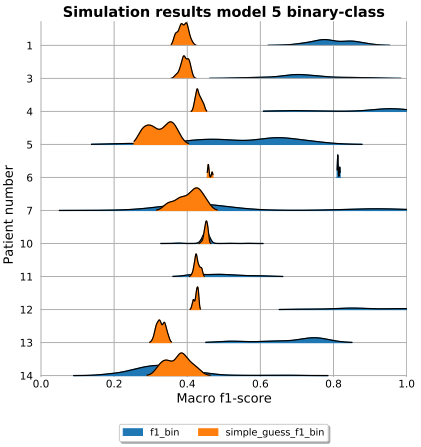

Figure C.7 Smoothed histograms of the macro f1-score obtained during binary-class simulation of the "describe picture" dataset when evaluating model 5.

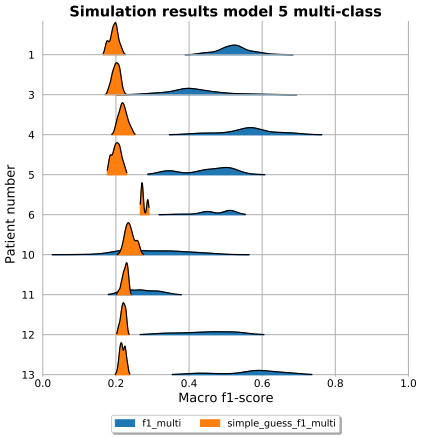

Figure C.8 Smoothed histograms of the macro f1-score obtained during multi-class simulation of the "describe picture" dataset when evaluating model 5.