Master Thesis (732A64)

Human Age Prediction Based on Real and Simulated RR Intervals using Temporal Convolutional Neural Networks and Gaussian Processes

Division of Statistics and Machine Learning
Department of Computer and Information Science
Linköping University

Maximilian Pfundstein (maxpf364)

June 4, 2020

Supervisor: Krzysztof Bartoszek
Examiner: Oleg Sysoev

ISRN: LIU-IDA/STAT-A--20/004--SE

Upphovsrätt

Copyright

Abstract

Electrocardiography (ECG) is a non-invasive method used in medicine to track the electrical pulses sent by the heart. The time between two subsequent electrical impulses and hence the heartbeat of a subject, is referred to as an RR interval. Previous studies show that RR intervals can be used for identifying sleep patterns and cardiovascular diseases. Additional research indicates that RR intervals can be used to predict the cardiovascular age of a subject. This thesis investigates, if this assumption is true, based on two different datasets as well as simulated data based on Gaussian Processes. The datasets used are Holter recordings provided by the University of Gdańsk as well as a dataset provided by Physionet. The former represents a balanced dataset of recordings during nocturnal sleep of healthy subjects whereas the latter one describes an imbalanced dataset of records of a whole day of subjects that suffered from myocardial infarction. Feature-based models as well as a deep learning architecture called DeepSleep, based on a paper for sleep stage detection, are trained. The results show, that the prediction of a subject's age, only based in RR intervals, is difficult. For the first dataset, the highest obtained test accuracy is 37.84 per cent, with a baseline of 18.23 per cent. For the second dataset, the highest obtained accuracy is 42.58 per cent with a baseline of 39.14 per cent. Furthermore, data is simulated by fitting Gaussian Processes to the first dataset and following a Bayesian approach by assuming a distribution for all hyperparameters of the kernel function in use. The distributions for the hyperparameters are continuously updated by fitting a Gaussian Process to a slices of around 2.5 minutes. Then, samples from the fitted Gaussian Process are taken as simulated data, handling impurity and padding. The results show that the highest accuracy achieved is 31.12 per cent with a baseline of 18.23 per cent. Concludingly, cardiovascular age prediction based on RR intervals is a difficult problem and complex handling of impurity does not necessarily improve the results.

Acknowledgements

I want to thank my supervisor Krzysztof Bartoszek for continuously supporting me throughout the writing of this thesis. Also, I want to say thank you to Anna and Julia for proofreading and giving advice.

Preamble

This thesis, the source code and all conducted analysis can be found on GitHub at `https://github.com/flennic/master-thesis/`.

In some occasions throughout the thesis, a function is used on a matrix. This is mostly the case for activation functions being applied to a linear combination embedded into a matrix. To keep the notation short, the indexes are omitted. Mathematically, the activation function is applied on each linear combination of the feature vector and the weights of the model.

Contents

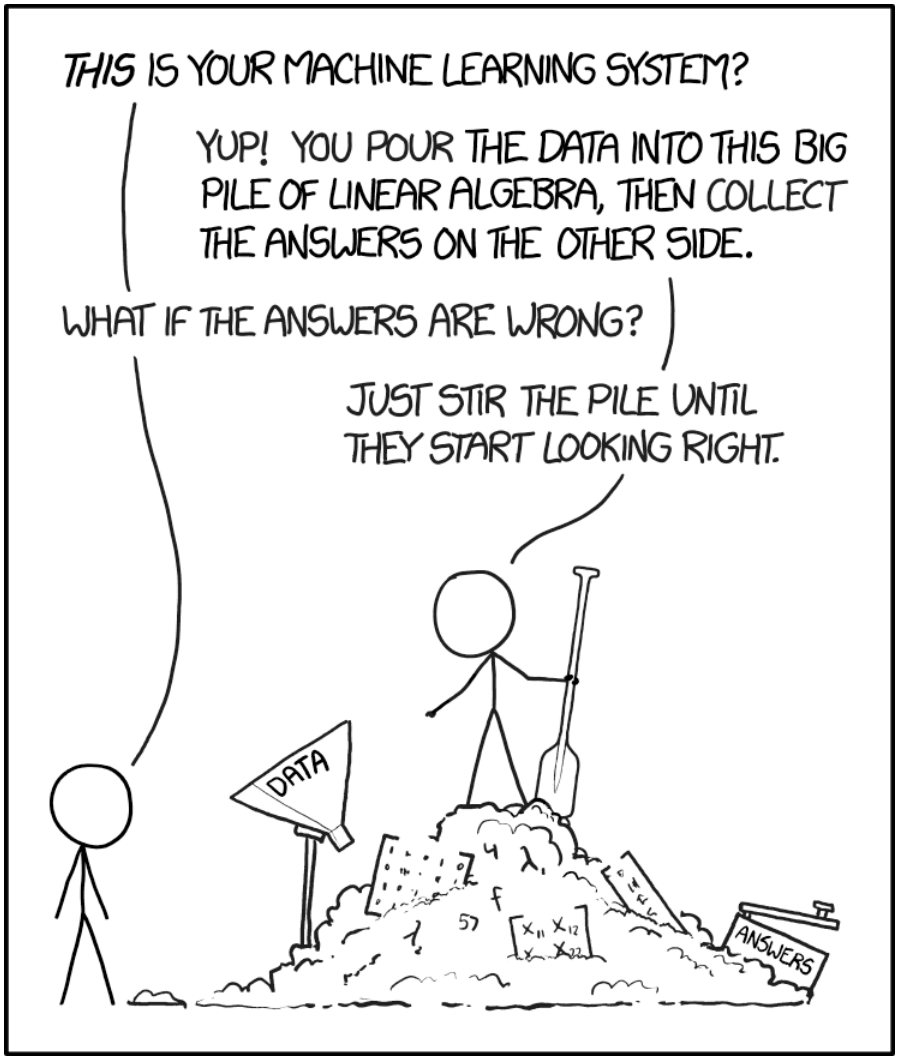


Figure 1: By Randall Munroe. *Source:* `https://xkcd.com/1838/`

1 Introduction

1.1 Background

In the field of medicine, electrocardiography (ECG) is a procedure of measuring the electrical potential and thereby the voltage between different potential levels of the heart. These measurements can be used to monitor the electrical activity of the heart which are fundamentally responsible for heart contractions and the heart rhythm. The analysis of these impulses can be used to study the health of a patient in a non-invasive way, inferentially ECG is used amongst other methods for detecting and analysing cardiovascular diseases [Tison et al. 2019], myocardial infarction [FM and TL 1988] or acute coronary syndromes [Birnbaum et al. 2014]. An example of such a measurement can be seen in figure 2. The data displayed is a signal used for testing[1] ECG devices and was obtained using the *WFDB Software Package*[2], issuing the command `rdsamp -r aami3a -p > out.txt` and then slicing the values between indices 10000 and 12000. The signal was recorded using a sampling rate of 720Hz and a resolution of 12 bit. Figure 2 thereby shows the period of 2.78 seconds including two RR spikes, thus one RR interval. The QRS complex can be seen as well, having a lower voltage before (Q) and after (S) the R spike. The remaining indices are used in the field of medicine, but are not considered for this thesis, as the data of interest consists of only RR intervals.

Earlier studies analysed the correlation between different features of RR signals and the age of the subject [Makowiec and Wdowczyk 2019] or focus on sleep stage classification using deep learning [Y. Zhang et al. 2019]. This thesis will emphasise on the prediction of the age of a subject given the RR intervals. The interest of this task is two-fold: Firstly, it seems that the process of collecting and archiving data is not always standardised, resulting in inconsistently formatted data, especially unlabelled RR intervals in the field of medicine. These labels are a necessity for researching this domain [e.g. Berg et al. 2018], therefore, the prediction of age with a high accuracy is of importance. Secondly, having a model available that predicts the age of healthy humans can be useful for estimating the cardiovascular age of unhealthy individuals. Assuming that the actual age and cardiovascular age of a healthy individual is the same, the model can be used to estimate the cardiovascular age of an unhealthy individual, as the cardiovascular age can be higher compared to the actual age, especially if the individual, for example, smokes.

[Makowiec and Wdowczyk 2019] focus on constructing and analysing 33 features from a given RR interval time series, used for traditional machine learning models like Support Vector Machines (SVMs). These features are well known in the analysis of RR intervals [Shaffer and Ginsberg 2017], some of them being correlated though and thereby do not add too much information for a feature-based classifier. Each subject's age decade (e.g. from 20 to 29) resembles one class, yielding in 7 classes (20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 70+) for the dataset used[3]. The paper reports very high

[1] `https://physionet.org/content/aami-ec13/1.0.0/#files-panel`
[2] `https://archive.physionet.org/physiotools/wfdb.shtml`
[3] Which is also one of the datasets used in this thesis, provided by the University of Gdańsk.

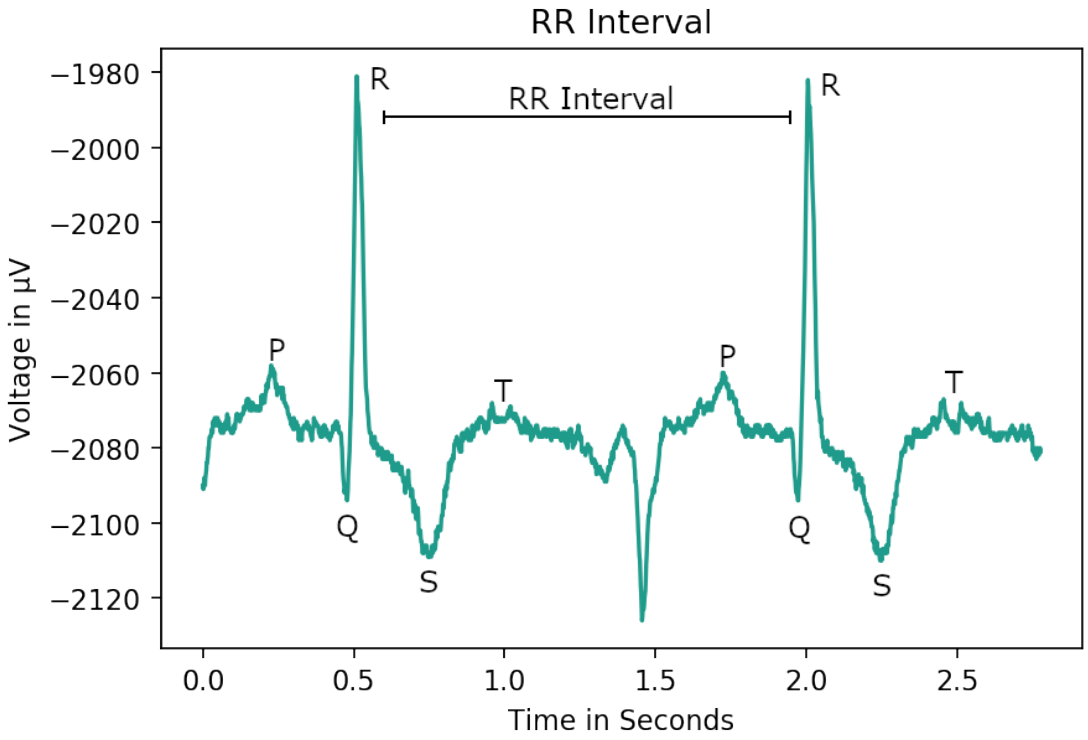


Figure 2: **Sinus Rhythm** with labels.

accuracies for this task of age classification. An accuracy of 94 per cent for a non-linear SVM using all features and an accuracy of 98 per cent using a majority vote for the classification of 5 minutes chunks using only *minimal stdRR* and *minimal HR* [Makowiec and Wdowczyk 2019, p. 16 and 17] as features.

These accuracies seem to be very high, considering the only validation which is being conducted was to shuffle the time signal and comparing a model trained on the shuffled data, which results in lower accuracies. By this, according to the paper, the distribution of values is preserved, but the order of the time series is destroyed. This is supposed to be sufficient for validating the obtained results. The absence of an actual validation or test set raises the question, if the classifiers learned to generalise or whether they overfitted the data that is used for training and prediction. They also state: *One can see that restriction of the set of features limited the classification quality, namely the score was significantly smaller than in the case when all features were taken into account* [ibid., p. 16]. This seems to be another indicator that the trained classifiers might be overfitted, as more features can lead to an overfit.

Another study [Poddar, Kumar, and Sharma 2015] also tries to classify subjects by age, based on given RR intervals. They also utilise feature-based classifiers and define three classes:

- **Y**oung (18-30)
- **M**iddle (30-45)
- **O**ld (45-60)

A test set was used in this analysis [Poddar, Kumar, and Sharma 2015, p. 4] and the results show an accuracy of around 70 per cent. This lower accuracy is contrary compared to having fewer classes (3 instead of 7), further raising the question, if the previous high accuracies might be the result of an overfit to the training data. Additionally, the reported classes in [ibid.] seem to overlap, at least the way they are defined in the paper. Both papers do not consider the order of classes. The order might allow conclusions about how much a prediction is off from the real class label as it makes a considerable difference if a subject of 20 years of age was misclassified as 30 or 70. The way the accuracies and thus the models are presented in both papers, automatically yield in a higher accuracy the less classes are defined. The actual predictive power of a model in terms of general predictability (e.g. regression) does not depend on the number of classes, but rather on prediction intervals and the respective error distributions. Of course, for the downstream task of classification, such an accuracy as a metric can be reported, but it makes the models incomparable. Furthermore, it seems that both analysis lack a real baseline model, hence they do not account for probable inequality in class distributions or the mentioned fact that the classes are ordered. More precisely, they do not state the loss function used, as e.g. cross-entropy loss does not account for the order of classes.

The domain of deep learning provides tools for time series prediction as well, namely Long Short-Term Memories (LSTMs) as they do not suffer that much from the vanishing gradient problem [Hochreiter and Schmidhuber 1997, abstract] compared to traditional Recurrent Neural Networks (RNNs) [Pascanu, Mikolov, and Bengio 2012] or Convolutional Neural Networks (CNN). CNNs use convolutions and thus moving averages in the discrete case [Ismail Fawaz et al. 2019, p. 7]). [ibid.] researched time series classification, specifically looking into Time Convolutional Neural Networks (TCNNs) which were first proposed by [Zhao et al. 2017]. The main difference to traditional CNNs is that instead of a cross-entropy loss the mean squared error (MSE) is being taken and that instead of *max* pooling, *average* pooling is being used [Ismail Fawaz et al. 2019]. Therefore, this thesis will also investigate, how well deep learning models work for this task of age classification compared to traditional feature-based models.

1.2 Objectives and Research Questions

The main research question this thesis aims to answer:

- Can the (cardiovascular) age of a person be predicted using recorded RR intervals and if yes, to which extend?

- Can RR intervals be simulated to generate training data, especially for deep learning models? How well does this work and does it improve the accuracy of the models?

For answering these research question, the following steps are conducted:

- Use feature-based models and investigate to which extend the previous results can be replicated.
- Building a deep learning to analyse if this approach works better compared to feature-based models.
- Find a better way of handling the impurity of the data by creating a statistical model for the data.
- Use the obtained statistical model to simulate training data for the deep learning model to increase the variability of the data and examine if that increases the predictive strength of both kinds of models.

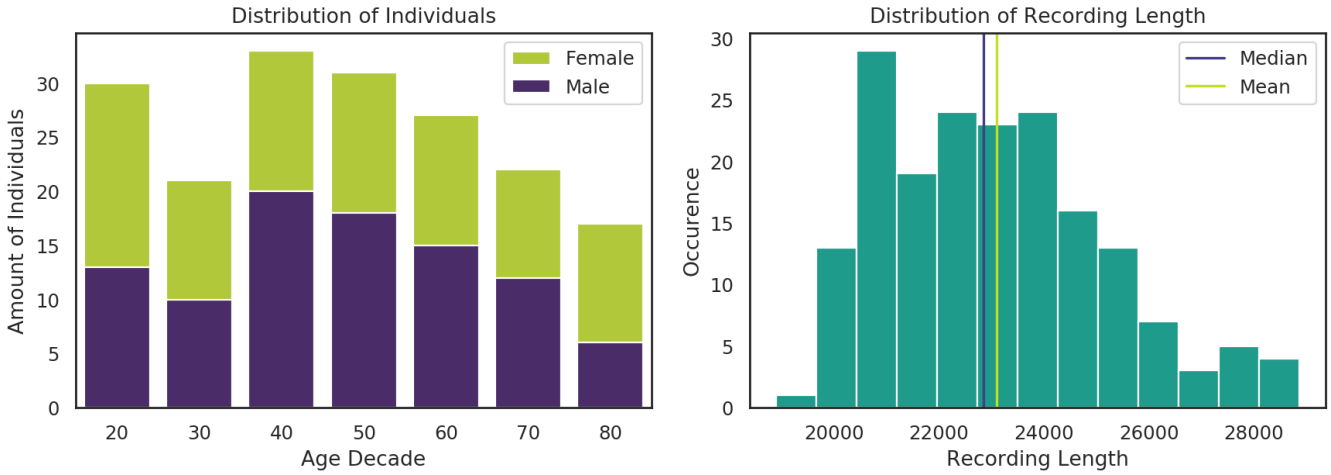
A more detailed overview of the different models and planned approaches is given in section 3.1. The following chapter will present the two datasets used in this thesis.

2 Data

The objective of age prediction is carried out on two datasets. For data simulation, only the first dataset, provided by the University of Gdańsk, will be used.

2.1 Holter Recordings from the University of Gdańsk

The first dataset was acquired during the analysis of *gdaHeart Rate Dynamics in Healthy Aging Population: Insights from Machine Learning Methods* [Makowiec and Wdowczyk 2019] and consists of 181 Holter recordings of healthy individuals. For each individual, gender, age decade and beginning of the recording are available as meta-information. An age decade, which is for example denoted by 20, means that the subject's age lies between 20 and 29 years. The distribution of age decades, number of classes and respective amounts of males and females are displayed in figure 3(a). The dataset is not completely balanced, still the amount of samples from each class label lies within a factor of two.



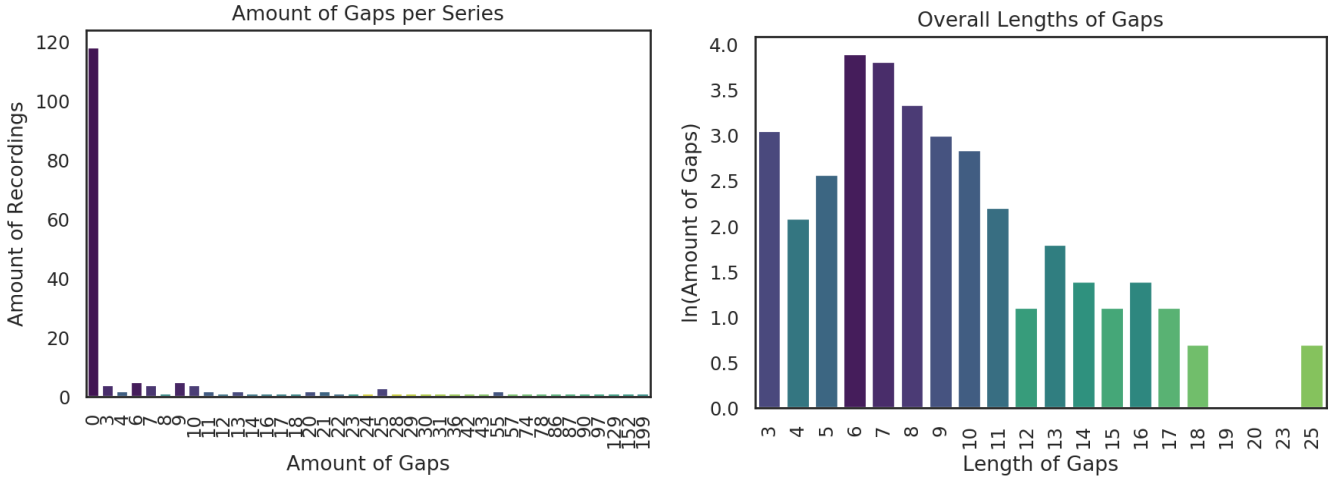
(a) Distribution of Individuals

(b) Distribution of Recording Lengths

Figure 3: Distribution of individuals, split by **male** and **female**, and **recording lengths** (Gdańsk) with **mean** and **median**.

The dataset is provided by the *Institute of Theoretical Physics and Astrophysics, University of Gdańsk* and is already preprocessed to some degree. Firstly, the recordings include only the RR intervals and no further information like the QRS-complex are available. Secondly, the dataset has been truncated to only include the first four hours of nocturnal sleep for each subject. This process was conducted by manual inspection of the signal. In the case the beginning of an individual's sleep was not evident, the signal starts at 00:00. Thirdly, gaps of size 1 and 2 were filled by the median value of the adjacent signals and each δRR was capped to $\pm$300ms [ibid., p.4].

The distribution of the recording's lengths with its mean and median can be seen in figure 3(b). The 95th percentile of the recording's lengths lies at 26959.



(a) Amount of Gaps for each Series

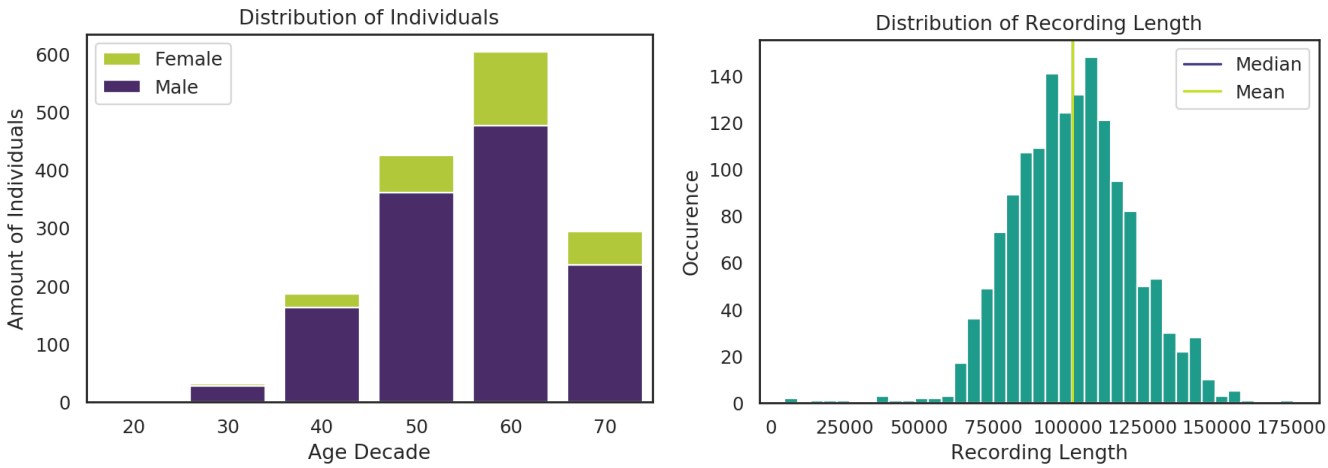
(b) Logged Amount of Gaps

Figure 4: Distribution of amount of gaps and length of gaps (Gdańsk).

Figure 4(a) shows that approximately 60 per cent of the series do not contain any gaps. For the remaining series, the distribution of the gap's lengths can be seen in figure 4(b). In the following, this dataset will be referred to as the *Gdańsk dataset*.

2.2 CAST RR Interval Sub-Study Database

The second dataset is available at PhysioNet [Goldberger et al. 2000] and is called *CAST RR Interval Sub-Study Database*. It consists of 1543 recordings of individuals that survived a myocardial infarction. Each subject was treated with a different medication: 285 with *Encainide*, 229 with *Flecainide* and 294 with *Moricizine*. Recordings before the treatment are available for most subjects, explaining the overall amount of recordings. For each subject, full 24 hours of recording are available compared to the first dataset, which provides only 4 hours of nocturnal sleep. The purpose of using this additional dataset is to ascertain that the obtained generalised predictabilities of the models also hold roughly for another dataset as well as analysing different methods of handling imbalance in the data. In theory, a vast amount of data from different sources is ideal, but the acquisition and amount of computational resources are out of scope for this thesis. One must be aware that models trained on this second dataset might be biased, as RR intervals with diseases are not labelled according to the subjects cardiovascular, but real age. As this thesis focuses mainly on the question if any information about the age can be extracted from the RR intervals in the first place, this bias does not pose a problem.



(a) Distribution of Individuals

(b) Distribution of Recording Lengths

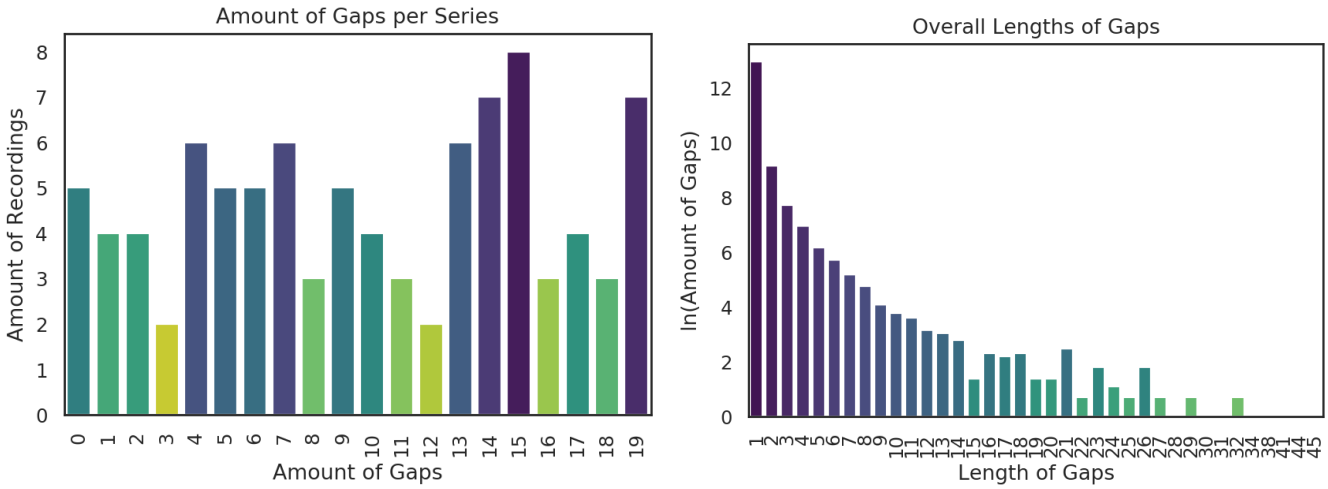
Figure 5: Distribution of individuals, split by **male** and **female**, and **recording lengths** (PhysioNet) with **mean** and **median**, which are covering each other due to their closeness.

For this second dataset, the distribution of age decades is shown in figure 5(a). It reveals that this dataset is more imbalanced compared to the first one. There are only 2 individuals for the age decade $20 - 29$, also males are more represented compared to females. As the dataset contains recordings for the whole day, the length of each series is around six times longer. The distribution, the respective mean and median, which are almost the same for this dataset, can be seen in figure 5(b).

The RR intervals are extracted from the raw data recordings using the *WFDB Software Package*[4]. The obtained signals contain outliers and therefore are preprocessed using the Python package `hrvanalysis`[5] with its function `hrvanalysis.preprocessing.remove_outliers(...)`, based on the following previous studies: [Inbar et al. 1994], [Miller, Wallace, and Eggert 1993], [Tanaka, Monahan, and Seals 2001] and [Gulati et al. 2010]. Outliers will be treated as missing data points.

Figure 6(a) shows the ordered amount of gaps per series capped at a length of 20. This dataset includes some very large gaps, excelling at 4437 missing data points in a row. For preprocessing, these large gaps provide a challenge, as simple spline interpolations might not result in an appropriate fit, especially when considering the trigonometric behaviour of the signals. Figure 7(a) and figure 7(b) show an excerpt of the overall time series of a young, healthy male. Figure 7(a) illustrates, that in the beginning of the recording, the subject's sleep phase started, yielding in slower heart contractions and thus larger RR intervals. The interval itself indicates the trigonometric pattern, which is more evident in figure 7(b) and shows the RR intervals during the end of the sleep

[4] `https://archive.physionet.org/physiotools/wfdb.shtml`
[5] `https://github.com/Aura-healthcare/hrvanalysis`



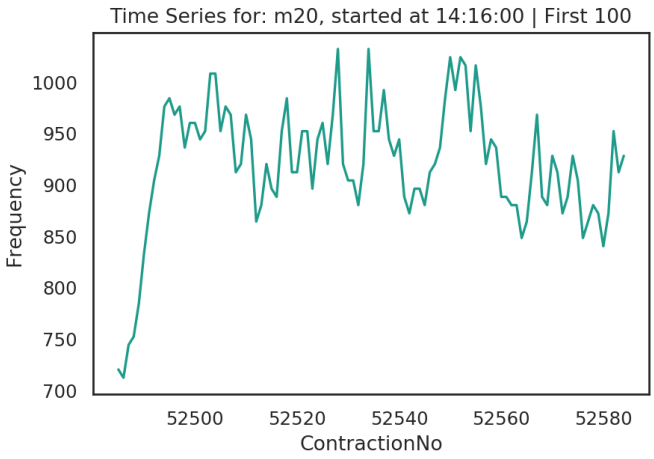
(a) Amount of Gaps for each Series (first 20)

(b) Logged Amount of Gaps

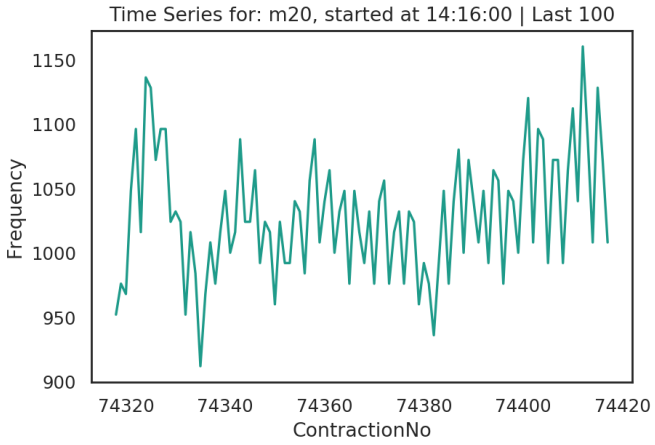
Figure 6: Distribution of amount of gaps and length of gaps (PhysioNet).

phase.

Looking at an older female, who is a survivor of a myocardial infarction, shown in figure 8(a), it can be seen that firstly, there are two spikes which seem out of the ordinary and secondly, that the trigonometric pattern is not as evident. Moreover, in figure 8(b) the mean of the series seems to change with time, whereas compared to 7(b) the mean seems more steady. The female subject might still have been awake during the end of the recording and is under the influence of medication, so the comparison might not provide too much information. Still, it seems that there is a difference between RR intervals of different subjects that can be investigated for predicting the age. This dataset will be referred to as the *Physionet dataset*.

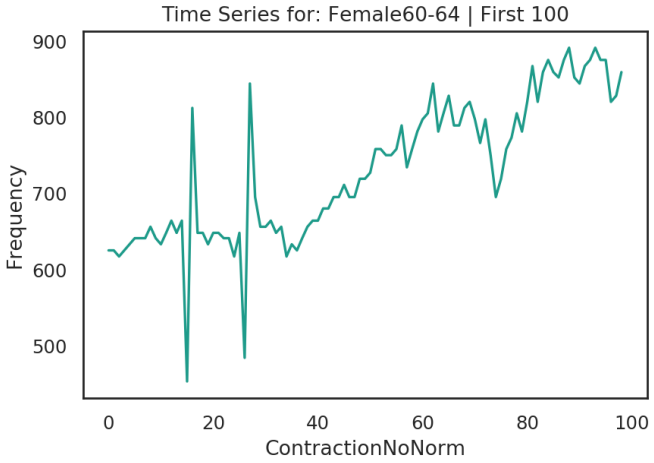


(a) First 100 RR intervals for a young, healthy male

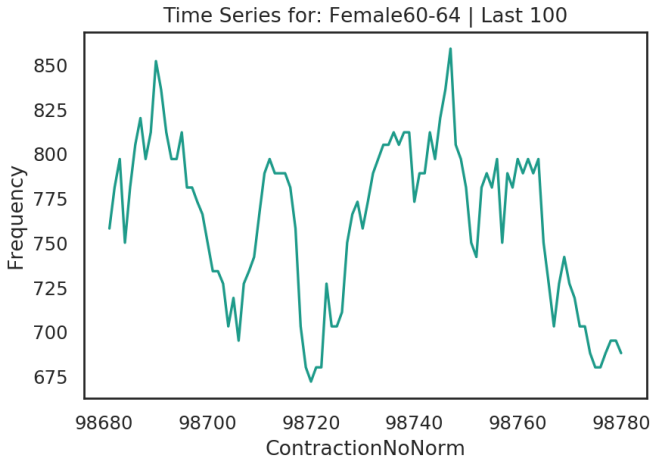


(b) Last 100 RR intervals for a young, healthy male

Figure 7: Example excerpt from the time series, showing the **first and last 100 RR intervals** of the series of a healthy, young male.



(a) First 100 RR intervals for an older female



(b) First 100 RR intervals for an older female

Figure 8: Example excerpt from the time series, showing the **first and last 100 RR intervals** of the series of an older female, who is a survivor of a myocardial infarction and under treatment of medication.

3 Theory

This chapter explains the different theoretical frameworks used in this thesis for answering the defined research questions. Firstly, a roadmap and thus an overview of the models is given. Secondly, the feature-based models and their choice of hyperparameters will be presented. Thirdly, the focus will shift towards deep learning and the modified version of a model proposed by another paper for a similar task is being analysed. Afterwards, different approaches for handling impurity and padding of the data are described, which includes interpolation (linear splines) and data simulation by using $\mathcal{GP}s$.

3.1 Roadmap

To answer the research questions, the first aim is to rebuild the SVM baseline model based the 33 features used in previous studies and to use cross-validation and hyperparameter search to understand the previous findings of [Makowiec and Wdowczyk 2019]. Afterwards, a more precise accuracy estimate is obtained by using a test set as well. Additionally, the following feature-based methods will be trained as well to investigate how well these models generalise: *XGBoost*, *Random Forests* and *Naive Bayes*.

For applying a deep learning approach, a model based on DeepSleep, proposed by [Supratak et al. 2017], will be trained. It consists of two CNNs heads, followed by an LSTM and a Multi-Layer Perceptron (MLP) for the downstream task of classification. It will be adjusted according to [Ismail Fawaz et al. 2019], which includes changing the pooling layer as well as the the loss function. Further reasons and explanations that led to choosing this model are explained in the theory chapter.

All models, feature-based and DeepSleep, will be trained and tested for classification and regression, whereas classification assumes no order of the classes and regression does. In the case of regression, the median of the age decade will be taken (e.g. 30-39 will result in 35). Also, two types of slicing will be applied to the data:

- **Complete** (the whole time series is being taken)
- **Constant** (the time series is split into equal chunks resembling a time frame of around 5 minutes)

For replicating the results by [Makowiec and Wdowczyk 2019] and to compare them to other feature-based models, spline interpolation will be used.

The second aim is two-folded: Firstly, the impurity of the data should be handled using a more sophisticated approach. Secondly, training data should be simulated to have more variability in the data, leading to less overfit, which is especially useful for the DeepSleep-based model. Gaussian Processes ($\mathcal{GP}s$) are particularly valuable in this scenario of time series modelling as they cover both aims [Roberts et al. 2013]. Firstly, $\mathcal{GP}s$ are defined as a distribution over functions, which means that the fitted $\mathcal{GP}$ can be evaluated at all desired predictive index points. This enables evaluating predictive index points which were originally not included in the observations. Therefore, missing

data points, as well as parts of the series that need to be padded for the DeepSleep model, are covered. Secondly, as we have a distribution over functions, any amount of samples can be taken, providing a larger amount of training data. Thirdly, a posterior distribution is being provided which can subsequently be used for sampling differently shaped time series for training.

This will be done by fitting $\mathcal{GP}s$ to each slice of around 5 minutes, as the memory consumption would be too high fitting a $\mathcal{GP}$ to the whole time series. Different kernel functions will be used. Afterwards, two different approaches will be investigated for optimising the kernel hyperparameters. The first approach is using gradient descend for obtaining point estimates, maximising the marginal log-likelihood of the $\mathcal{GP}$, following a more traditional way. The second approach uses Hamiltonian Monte Carlo (HMC) sampling to integrate out the prior parameters, following in a fully Bayesian approach, accounting for the uncertainty in the parameters and yielding in a full posterior predictive distribution. The resulting posterior distributions for the kernel parameters will then be used as priors for the next slice of the same time series. The obtained fitted $\mathcal{GP}s$ will serve to simulate training data, which will have no gaps and each slice will have the same length. All models will be re-trained on the simulated slices and will be compared to the models trained on the real data.

For handling the imbalance in both datasets, three different approaches have been investigated. Firstly, setting weights for the different classes (in case of classification), secondly, under-sampling and thirdly, over-sampling of the data. For classification, the approach of setting class weights works better compared to over- and under-sampling. In case of regression, applying no further adjustment works best. Therefore, the focus lies on those two approaches for the feature-based models to reduce the amount of overall models which have to be trained.

Summarising, the following steps will be conducted:

- Obtaining the 33 features, fitting different feature-based models and reevaluate them using a test set by applying hyperparameter search with cross-validation.
- Building a customised DeepSleep model including the time series adjustments.
- Simulating the data using $\mathcal{GP}s$ and reevaluate the best models.

The models will be fitted to both datasets where applicable to investigate if they show different behaviour and whether the results are stable. Due to computational constraints, the data simulation will only be based on the dataset provided by the university of Gdańsk, as it is smaller.

3.2 Feature-based Models

This section focuses on the feature-based methods. For all models, cross-validation is conducted with a manually selected grid of hyperparameters. For hyperparameters which are bound in both directions, the hyperparameters to consider are equally spaced

between their bounds and the computational resources available define their granularity. All other hyperparameters have a lower bound, hence only the upper bound must be determined. This is conducted by manually considering subsequent exponential values to find an upper bound after which the models continuously performs worse. It is to be assumed that higher values for the hyperparameters will not result in a lower global minimum of the loss function, as these continue to let the models overfit.

3.2.1 Models

The feature-based models are trained with the help of the `scikit-learn`[6] library. XGBoost is the only model implemented in its own library[7], which integrates neatly into `scikit-learn`. All feature-based models are:

- Naive Bayes
- Support Vector Machine
- Random Forest
- XGBoost

All models are trained using 3-fold cross-validation with hyperparameter search. The hyperparameters names for classification are subsetted by c and respectively by r for regression. Due to computational constraints, the maximum number of time series is limited for each model depending on the time it takes for training, fitting and prediction. The parameters are set in a way that the whole procedure does not exceed the time of 1h on an intel *i7 2600k*. The exact limits can be found in the metrics in appendix 9.2. The number of time series taken for the final fit of each model is double the amount for conducting the cross-validation.

3.2.1.1 Naive Bayes The Multinomial Naive Bayes implementation[8] only supports classification, therefore, the regression part will be skipped. The Naive Bayes model is based on Bayes Theorem, thereby assuming independence between all features. x_i denotes the ith set features x, where N is the number of given training points and M the dimensionality of the observations x. y denotes the corresponding label. Subsequently, the model is defined by [H. Zhang 2004, based on equation 3]:

$$p(y|x_i) \propto p(y) \prod_{i=1}^{M} p(x_i|y) \tag{1}$$

The rule for classification is then given by [Murty and Devi 2011, p. 96]:

$$\hat{y} = \arg\max_{y} p(y) \prod_{i=1}^{M} p(x_i|y) \tag{2}$$

[6] `https://scikit-learn.org/stable/`
[7] `https://xgboost.readthedocs.io/en/latest/`
[8] `https://scikit-learn.org/stable/modules/naive_bayes.html`

In the multinomial case, x is assumed to be a distribution represented by a histogram, where x_j represents the occurrence of the jth feature column. As the given features are on a continuous scale, *Laplace smoothing* (also referred to as *add-one smoothing*) is being applied. 1 is added to every occurrence of each x_j to circumvent the case of a given representation not appearing. Afterwards, the smoothing, hence the estimate for the occurrence, is defined as (based on [Manning, Raghavan, and Schütze 2008, eq. 13.7]):

$$\hat{\theta}_{yi} = \frac{x_i + \alpha}{N + \alpha M} \quad \text{with} \quad (i = 1, ..., M) \tag{3}$$

α is a hyperparameter that controls the smoothing. If $\alpha < 1$, it is called *Lidstone smoothing* whereas the case $\alpha = 1$ refers to the above mentioned *Laplace smoothing*.

During training, the following values for α are being considered:

$$\alpha_c = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \tag{4}$$

The distribution of the classes in the training datasets are taken as priors.

3.2.1.2 Support Vector Machine In its heart, the SVM is a linear classifier, trying to find a hyperplane in the feature space that maximises the margins to the nearest data points, also called support vectors. The hyperplane is simply expressed by [Evgeniou and Pontil 2001]:

$$0 = w \cdot x - b \tag{5}$$

w denotes the weights, b the bias and x the features. Assuming that the margin keeps the same distance towards, the hyperplanes incorporating the support vectors are given by:

$$-1 = w \cdot x - b \quad \text{and} \quad 1 = w \cdot x - b \tag{6}$$

This results in a hard margin. Most problems involve overlapping data points, thus a hard margin is not applicable. Therefore, the *Hinge loss* [Rosasco et al. 2004, p. 6] is introduced, which allows outliers while still penalising these:

$$\text{loss} = \max(0, 1 - y_i(w \cdot x - b)) \quad \text{and} \quad \text{loss} = \max(0, -1 - y_i(w \cdot x - b)) \tag{7}$$

Here, y_i denotes the label of the ith data point. Extending the loss to all given data points in the training set, as well as adding a regularisation them, the subjective function to minimise is:

$$\frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i(w \cdot x - b)) + C\|w\|^2 \tag{8}$$

C denotes the regularisation parameter and defines how much the model adapts to the training dataset. In this case, this is equivalent to L2 regularisation. Many

approaches exist for optimising this function, e.g. gradient descent or formulating a quadratic programming problem. When performing regression, an additional hyperparameter ϵ is introduced in the loss function, which defines the absolute amount that the regression can be off without being penalised. In this case, the *Hinge loss* is updated as following (exemplary for one label):

$$\text{loss} = \begin{cases} 0 & \text{if} \quad \|1 - y_i(w \cdot x - b)\| \leq \epsilon \\ \|1 - y_i(w \cdot x - b)\| - \epsilon & \text{otherwise} \end{cases} \tag{9}$$

SVMs use the *kernel trick* [Schölkopf 2000], where the dot product is calculated in a higher dimensional space by applying a kernel function. By this, the originally linear model can handle more complex problems. Mostly, the radial basis function kernel is used (which we will refer to as the *smoothing kernel* onwards). Equation 62 shows the definition of the kernel. The length scale parameter ℓ is referred to as γ by `sci-kit learn`. The sets of hyperparameters for classification are:

$$\begin{aligned} C_c &= \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \\ \gamma_c &= \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \end{aligned} \tag{10}$$

And for regression:

$$\begin{aligned} C_r &= \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \\ \gamma_r &= \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \\ \epsilon_r &= \{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0\} \end{aligned} \tag{11}$$

The class weights are set according to the inverse of the distribution of the classes.

3.2.1.3 Random Forest A Random Forest combines multiple decision trees which are trained on feature subsets of the data. A decision tree looks at all features and then decides at which feature and which split a cut would improve the predictability the most. To add randomness to the inputs of the decision trees, which prevents overfitting, two techniques are used. The first one is called *bagging*, where a random sample with replacement from the training data is being taken. Secondly, each time a new split has to be made, the decision tree only sees a subset of the feature, thus further adding randomness and preventing overfitting. In this way, multiple decision trees are fitted; thereafter, for inference, a majority vote is being made. The amount of decision trees is a hyperparameter of the Random Forest and denoted by `n_estimators`. The maximum depth of a tree is defined by the `max_depth` hyperparameter, thus deciding how deep a tree can be. The number of data points needed to consider an additional split is defined by `min_samples_split` and the minimum number of data points to create another leaf node is defined by `min_samples_leaf`. For the task of regression, the mean squared

error (MSE), see equation 26, is taken and for the task of classification, the *Gini Impurity* is being chosen. The *Gini Impurity* H is defined as[9]:

$$H(X_m) = \sum_k p_{mk}(1 - p_{mk}) \tag{12}$$

X_m denotes the features of the training data points in node m where the split is being considered and k the class label. The value p_{mk} is the proportion of of observations in node m belonging to class k and defined as:

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \tag{13}$$

N_m is the amount of data points in node m and is used for normalising p_{mk}. The data points itself are referred to as x_i and belong to the subset R_m which is the set of data points belonging to node m. $I(y_i = k)$ equals 1 when y, hence the class label, is equal to k which is the currently considered class.

The complete set of hyperparameters for the Random Forest model is given by:

$$\begin{aligned} \text{n\_estimators}_c &= \{5, 15, 20, 35, 40\} \\ \text{max\_depth}_c &= \{10, 15, 20, 25\} \\ \text{min\_samples\_split}_c &= \{2, 3, 4, 5\} \\ \text{min\_samples\_leaf}_c &= \{1, 2, 3, 4, 5\} \end{aligned} \tag{14}$$

$$\begin{aligned} \text{n\_estimators}_r &= \{5, 10, 15, 20, 25, 30\} \\ \text{max\_depth}_r &= \{5, 10, 15, 20, 25, 30\} \\ \text{min\_samples\_split}_r &= \{5, 15, 20, 35, 40\} \\ \text{min\_samples\_leaf}_r &= \{5, 15, 20, 35, 40\} \end{aligned} \tag{15}$$

The class weights are set to according to the inverse of the distribution of the classes.

3.2.1.4 XGBoost XGBoost also trains a set of decision trees, but in addition to the previous techniques, *gradient boosting* is applied. Gradient boosting is built on top of the same principle as AdaBoosting. When a new tree is being fitted, the errors made by the previous tree are taken into consideration. This concept is called *additive training*. Assuming each decision tree is a function $f(x_i)$, mapping the input x_i to the output y_i, t is the current number of iteration and i denoting the data point, the final model can be written as [Chen 2014, p. 20]:

$$f(x) = \hat{y_i}^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y_i}^{t-1} + f_t(x_i) \tag{16}$$

[9]Equation 12 and 13 are taken from section 1.10.7 in `https://scikit-learn.org/stable/modules/tree.html`.

It now remains to decide in each time step t which estimator to choose. This decision is based on a loss function. Normally, a loss function consists of the loss itself as well as a regularisation parameter. In case of the MSE loss function the derivate is easy to calculate. The aim of XGBoost is to derive a general analytical solution for the optimisation function, thus the Taylor expansion of the loss function up to the second-order is being calculated as an approximation. Omitting all constants, the final function to optimise if given by [Chen 2014, p. 23]:

$$\sum_{i=1}^{n} = \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t) \tag{17}$$

Where

$$g_i = \partial_{\hat{y}^{t-1}} l(y_i, \hat{y}^{t-1}) \tag{18}$$

and

$$h_i = \partial^2_{\hat{y}^{t-1}} l(y_i, \hat{y}^{t-1}) \tag{19}$$

Investigating equation 17, the left part in square brackets represents the Taylor approximated loss, generally denoted by $L(\Theta)$, while the right part $\Omega(f_t)$ denotes the regularisation, e.g. L1 or L2.

The hyperparameters considered for XGBoost are `n_estimators` which defines how many estimators are used overall. `max_depth` defines the maximum amount of splits for each decision (the depth of the tree). The learning rate η defines to which extend the calculated gradients are being applied and λ adjusts the regularisation according to:

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \tag{20}$$

Here, T is the number of leaves and γ is another (unused) hyperparameter that controls the minimum loss reduction which is required to further split a leaf node. By default $\gamma = 0$, thereby the left part of the regularisation term is not used. Additionally, the `booster` itself is another hyperparameter. Options used are `gbtree`, which uses trees as estimators, `gblinear` which uses linear estimators (and thus not utilising all hyperparameters mentioned here) and `dart` which is a booster applying dropout to trees to prevent overfitting [Rashmi and Gilad-Bachrach 2015].

The full set of hyperparameters considered is given by:

$$\begin{aligned} \text{n\_estimators}_{cr} &= \{1, 2, , 4, 8, 10\} \\ \text{max\_depth}_{cr} &= \{1, 2, 4, 8, 10\} \\ \text{booster}_{cr} &= \{\text{gbtree}, \text{gblinear}, \text{dart}\} \\ \eta_{cr} &= \{0.3, 0.6, 0.9\} \\ \lambda_{cr} &= (0.3, 0.6, 0.9) \end{aligned} \tag{21}$$

3.2.2 Cross-validation

Cross-validation is suitable for obtaining a better estimate of the model's generalisation, which is particularly useful when evaluating the best set of hyperparameters. The training set is split into K-folds, then one fold is being taken as an evaluation set and the model is trained on the remaining folds. The number of sets of hyperparameters considered is defined by J.

Let l_k^j denote the model's normalised loss of the kth fold trained on the respective training data, let the current hyperparameter set be denoted by j and let Λ denote mean loss averaged over all folds k for each set. Then, the best model is found by selecting the hyperparameter set that has the lowest averaged loss:

$$\Lambda_{\text{best}} = \min\{\Lambda_1, \Lambda_2, ..., \Lambda_K\} \tag{22}$$

The averaged loss estimates how well the model generalises and is given by:

$$\Lambda_j = \sum_{k=1}^{K} l_k^j \tag{23}$$

By this, the mean of the loss of all folds i is evaluated for each set of hyperparameters. The lowest loss Λ and thus the best model, with its set of hyperparameters, is chosen and evaluated on a hold-out test dataset.

3.3 Feed-Forward Neural Network

The concept of how a neural network functions, is inspired by biological neural networks used by the brain of humans and animals. When a being tastes, smells, hears, sees or feels, this input is forwarded to the brain for further processing. Afterwards it decides, if an action is being taken and if yes, it chooses which one. For example, an input could be the touch on a hot plate, whereas the action or output would be the immediate withdrawal of the hand. Like the brain, neural networks consist of neurons that are tightly interconnected with each other. If a neuron is getting excited because it receives an input satisfying a specific threshold value, it forwards the input to adjacent neurons. They get excited if the sum of their inputs, gathered from multiple adjacent neurons, exceeds their threshold as well. In this way, information can flow through the network. A more formal way of describing a neuron is the *perceptron*. Statistically speaking, the perceptron is a linear and binary classifier. Figure 9 shows a perceptron with inputs on the left side.

Usually, the input is denoted by x and is a vector of size d. The input is then summed up by the perceptron. To enable the perceptron to learn, each input also gets attached a *weight*, denoted by w, which is multiplied by the input before summing them up. Also, a bias b is added. This enables the perceptron the learn the *intercept* of a linear function.

The next step in building a neural network is to combine multiple perceptrons to a network. The resulting model is mostly either called *Feed Forward Network* (FFN) or *Multi-Layer Perceptron* (MLP). Figure 10 shows a simple neural network, consisting of multiple layers with multiple perceptrons.

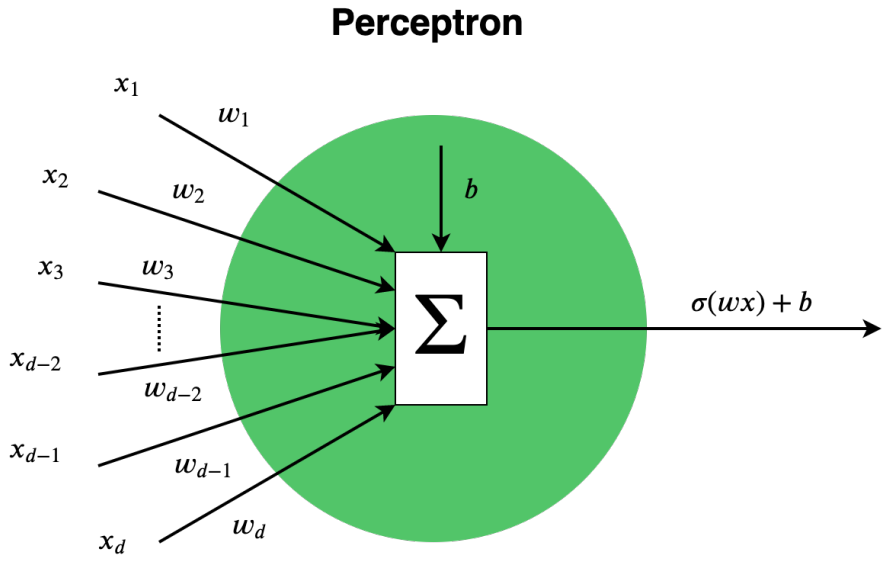


Figure 9: **Perceptron** with input x of dimension d, trainable weights w and the bias b. The output is a linear combination of the weights and the input plus the bias. Commonly, an activation function, here denoted by σ, is applied to the output to limit the range (usually σ refers to the *Sigmoid* activation function that squeezes the output between -1 and 1).

The first layer is called the *input* layer, the intermediate layers are called *hidden* layers and the last layer is called the *output* layer. The mathematical operations are conducted using a vectorised form, thus the dimensionality of x is usually $n \times d$ where n denotes the number of observations and d the dimensionality of the input. A *forward* pass then consists of calculating:

$$h(x) = \sigma(x^T w_1) \tag{24}$$

and

$$y(h) = \sigma(h^T w_2) \tag{25}$$

In these equations, w_1 and w_2 hold the weights of the layers, including the bias, making it easier using vectorised operations. σ defines the *Sigmoid* activation function which maps the output between $(-1, 1)$, see equation 34. Other activation functions can applied as well. The next step is to define a *loss* function and then use *backpropagation* the update the weights of the neural network.

The loss function defines the error of the neural network and depends heavily on the use case, for example, regression or classification. In the case of regression, the error is usually assumed to be normally distributed around the prediction y, therefore, the maximum likelihood estimate for the normal distribution is being optimised. This is

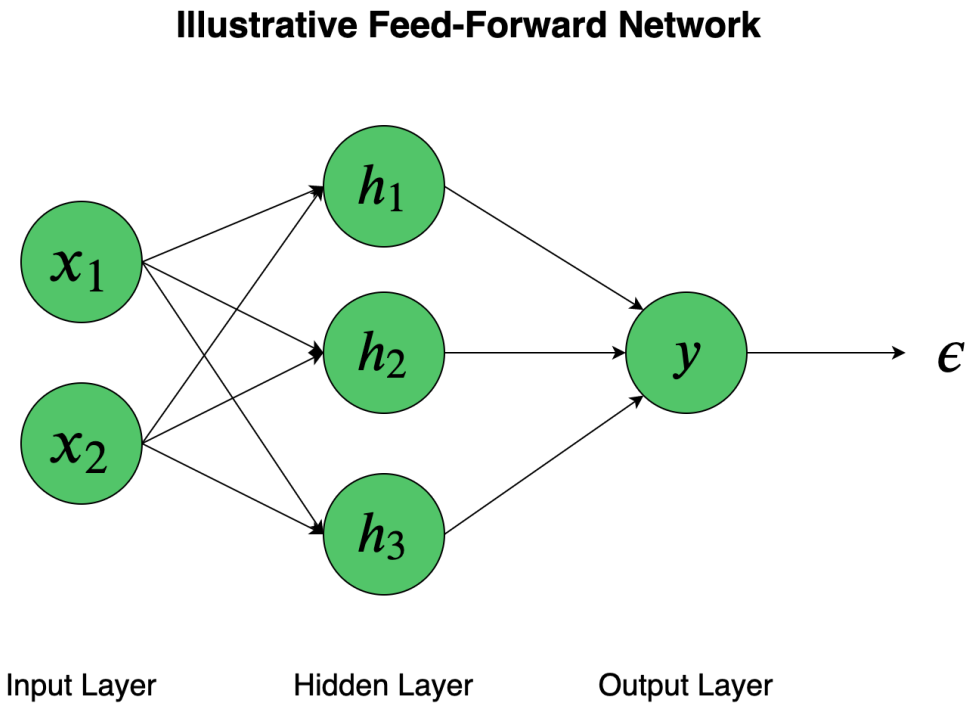


Figure 10: Illustrative **feed-forward neural network**.

done by maximising the *mean squared error* (MSE), where y denoted the true value and $\hat{y}$ the estimate of the network:

$$\epsilon(\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y)^2 \tag{26}$$

Having the loss defined, it is possible to calculate the gradient with respect to each weight and then update it accordingly. During each update, the learning rate α defines how much the weights are being updated. A neural network might see the training dataset multiple times during its training process. How often it sees the training dataset is referred to as *iterations* or *epochs*. In practice, more advanced *optimisers* are used, also using *momentum*. For explanatory purposes, a simple gradient update is being assumed. We are interested in the gradient of the error function with respect to each weight:

$$\nabla\epsilon = \left(\frac{\delta\epsilon}{\delta w_n}, \frac{\delta\epsilon}{\delta w_{n-1}}, \dots, \frac{\delta\epsilon}{\delta w_1}\right) \tag{27}$$

Then, to compute the gradient, the *chain rule* is being applied. For w_2 the unfolded chain rule results in:

$$\frac{\delta\epsilon}{\delta w_2} = \frac{\delta\epsilon}{\delta\hat{y}}\frac{\delta\hat{y}}{\delta h}\frac{\delta h}{\delta w_2} \tag{28}$$

Finally, the respective weight is being updated according to:

$$w_2^{\text{updated}} = w_2 - \alpha \odot \frac{\delta \epsilon}{\delta w_2} \tag{29}$$

Here, $\odot$ defines the *Hadamard product* and α the *learning rate* which influences by how much the gradients are being updated. In practice, the gradients are calculated via numerical optimisation, thus allowing arbitrary structures of the neural network without requiring to derive an analytical solution each time the structure of the network is altered. The choice of the structure and therefore the amount of tuneable parameters heavily depends on the problem to solve. Neural networks with a lot of layers are prone to the vanishing gradient problem [Pascanu, Mikolov, and Bengio 2012], which is one of the reasons that different types of neural networks are used for more complex problems. Some of them, which are used in this thesis, will be investigated in the following chapters.

3.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are primarily used for extracting features from inputs where parts of the input are highly correlated to the parts next to it. Therefore CNNs are used, amongst others, for feature extraction in images which are then used for downstream tasks of classification or regression. Theoretically, a traditional FFN is also capable of processing images, but the amount of weights required is a magnitude higher compared to CNNs, as CNNs use the same set of weights for the whole image instead of having one weight for each pixel. Mathematically speaking, a CNN uses convolutions on a discrete input, by learning a set of weights for different filters. For the following examples, we will look at the feature extraction of a black and white image, thus a two-dimensional input.

The brightness of each pixel is usually mapped to a value between 0 and 1 for normalisation. An example of this can be seen in figure 11. The CNN then applies a filter to the input which can be imagined as a sliding window that is hovered over the input. During each step, the dot product between input and filter is being calculated. Note that in this case, the shapes of input and filter would be 1×9 and 9×1 respectively, thus resulting in an output value of shape 1×1, even if the original image is assumed to be two-dimensional. Intuitively, one could assume that the dot product between a 3×3 and a 3×3 matrix is being taken, which is not the case. Applying this filter to each sliding window of the input, the output, which is displayed on the right side, is being calculated. In this example, the filter is always moved by 1 cell, which is referred to as having a *stride* of 1. To reduce the output size further, the stride could be increased. It can be noticed that the output will always be smaller than the input. To prevent this, if not desired, the input can also be *padded*, usually with zeros, so that the moving window will also cover the outer regions if the stride is not a multiple of the dimensionality.

Mathematically, this operation is called a convolution and for the 2-dimensional space defined by the following equation, where x is the input at location i and j, $\mathcal{F}$

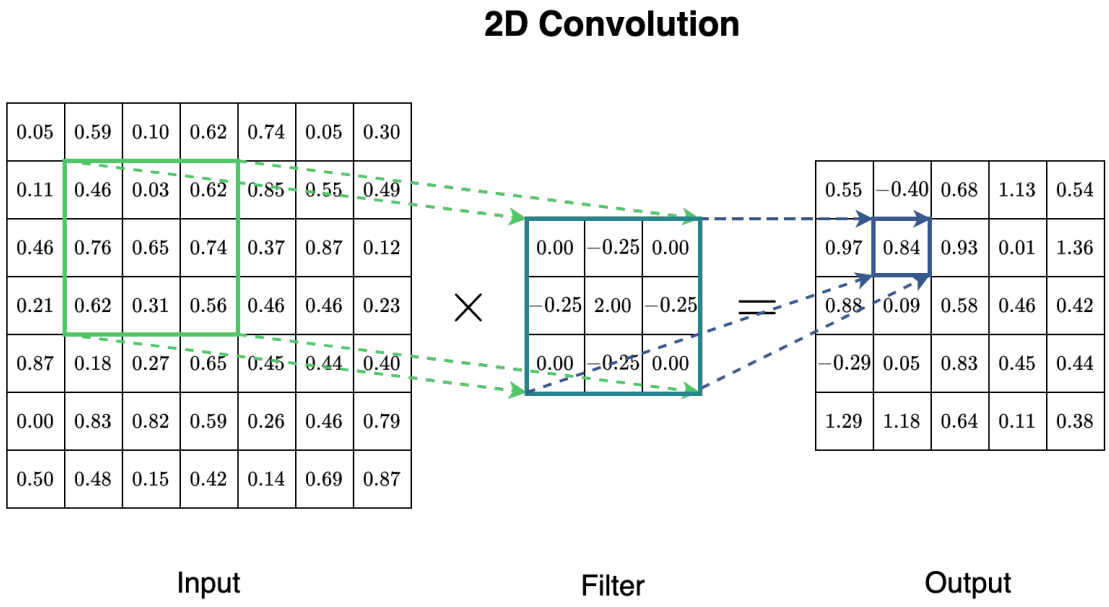


Figure 11: 2D convolution. The dot product between the **input** and the learnable **filter** is being calculated to produce the **output**.

the filter values, m the filter size and s the stride.

$$y_{ij} = \sum_{a=0}^{\lfloor m/s \rfloor} \sum_{b=0}^{\lfloor m/s \rfloor} x_{1+as,j+bs} \mathcal{F}_{a,b} \tag{30}$$

In the case of a 1-dimensional convolution, as applied for time series, the formula is reduced to:

$$y_i = \sum_{a=0}^{\lfloor m/s \rfloor} x_{i+as} \mathcal{F}_a \tag{31}$$

For extracting features from images, one is usually is interested in the edges of an image, thus a difference between the pixels next to each other (e.g. foreground to background, thus an *edge*). Generally, only these edges are of interest. As it is also of interest to reduce the size for faster processing, a concept called *pooling* is being applied in a second step. An example of *maxpooling* can be seen in figure 12.

Pooling can also be thought of as applying a sliding window with a mathematical operation applied to each window, *max* in this case. During pooling, there is usually no overlap between the selected regions of the slide, indicated by the dashed line. Mostly, the edge cases are ignored, but if desired, a padding can be applied as well. Both, the pooling and the filter size, are arbitrary and thus a hyperparameter of the network. One filter applies to the whole input, so for example in this case in figure 11, only 9 weights

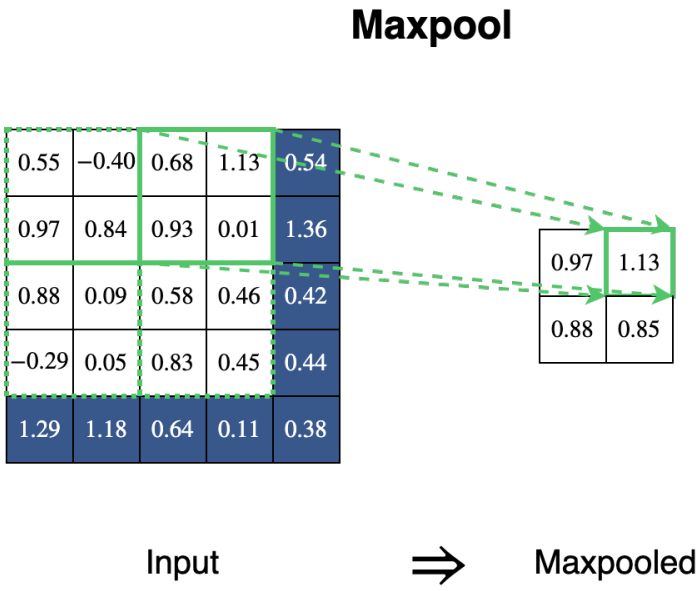


Figure 12: The **input** is being maxpooled by a window of 2×2, thus taking the maximum of each 2×2 window and constructing a new tensor. The **edge cases** are either ignored or padded with zeros, so that the *max* of a windows can still be taken.

must be trained compared to a traditional FFN, which would have 49 trainable weights. For large images or inputs, this is a huge advantage.

The filter shown in figure 11 is a high pass filter. CNNs would be quite limited if they could only learn one filter, which would mean that they could only focus on one property of the input. For example, a filter that detects *edges* where the colours shifts from black to white. Therefore, in each layer, multiple independent filter are applied and trained. For images, the first layer usually has 3 filters, one for each colour channel. To construct a whole CNN, filtering and pooling are concatenated multiple times, eventually providing features which embed information about more complex structures. These can then be used by a simpler model, for example, an FNN, for the downstream task of image classification or regression.

3.5 Temporal Convolutional Neural Networks

There exists no clear definition of what a Temporal Convolutional Neural Network (TCNN) exactly is. It can either be applied to multivariate time series forecasting [Wan et al. 2019] or classification of satellite images [Pelletier, Webb, and Petitjean 2018]. In the context of this thesis, the focus will lie mainly on two modifications compared to traditional CNNs. Firstly, for classification, the cross-entropy loss function is usually used, as it produces a distribution of classes that adds up to one. The cross-entropy loss

is defined as[10]:

$$\text{loss}(x, \text{class}) = -\log\left(\frac{\exp x[\text{class}]}{\exp \sum_i x[i]}\right) \tag{32}$$

$x[i]$ denotes the probability of the output of the model to belong to class i. Cross-entropy loss can also be defined in a way that it embeds weights for each class label. In this case, the loss is given by:

$$\text{loss}(x, \text{class}) = \text{weight}[\text{class}]\left(-x[\text{class}] + \log\left(\sum_i \exp x[i]\right)\right) \tag{33}$$

However, a presumed problem using cross-entropy loss is that it does not assume an order of classes. In our case of age prediction, this means that some information might be lost, as a miss-classification, which lies closer to the real age label, should be penalised less compared to one that is farther away. To take the order into account, the MSE loss, defined in equation 26, is being taken. This changes the model to a regressor. Mapping the regression output back to a class label is quite simple. For example, if the regression outputs the value 37.56, this output would map to the class label of individuals between 30-40.

The second difference is, that compared to taking maxpooling layers, which have been described earlier, *average pooling* layers are used. Contrary to detecting edges in images, time series analysis is not so much interested in the detection of edges but rather general information embedded in the time signals. Therefore, it is assumed that average pooling layers work better compared to maxpooling layers. [Zhao et al. 2017, p.4] also chose to use average pooling as it seemed to work best for their set of problems. One can not say that average pooling works better than maxpooling in general for time series classification, but there is also no indication that it works worse, hence the TCNN used in this thesis is implemented using average pooling.

3.6 Long Short-Term Memory

Traditional Recurrent Neural Networks (RNNs) were built to handle sequences of data, where a datapoint x_i is dependent on the previous data point x_{i-1}, thus not only on the current input of the sequence. The problem with this traditional approach is that calculating the gradients is prone to the *vanishing gradient problem* [Pascanu, Mikolov, and Bengio 2012]. The problem is two-folded: Firstly, the values of the gradients usually become smaller the deeper the network is, thus layers that are close to the output learn faster compared to layers closer to the input. This could be circumvented by longer training times, but raises issues with overfitting. Secondly, if the gradients become too small, it becomes computationally infeasible to represent these values in memory, as even double-precision floating-point numbers reach their limit quickly given long sequences. RNNs are specifically prone to longer input sequences.

[10] Based on `https://pytorch.org/docs/stable/nn.html?highlight=crossentropyloss#torch.nn.CrossEntropyLoss`

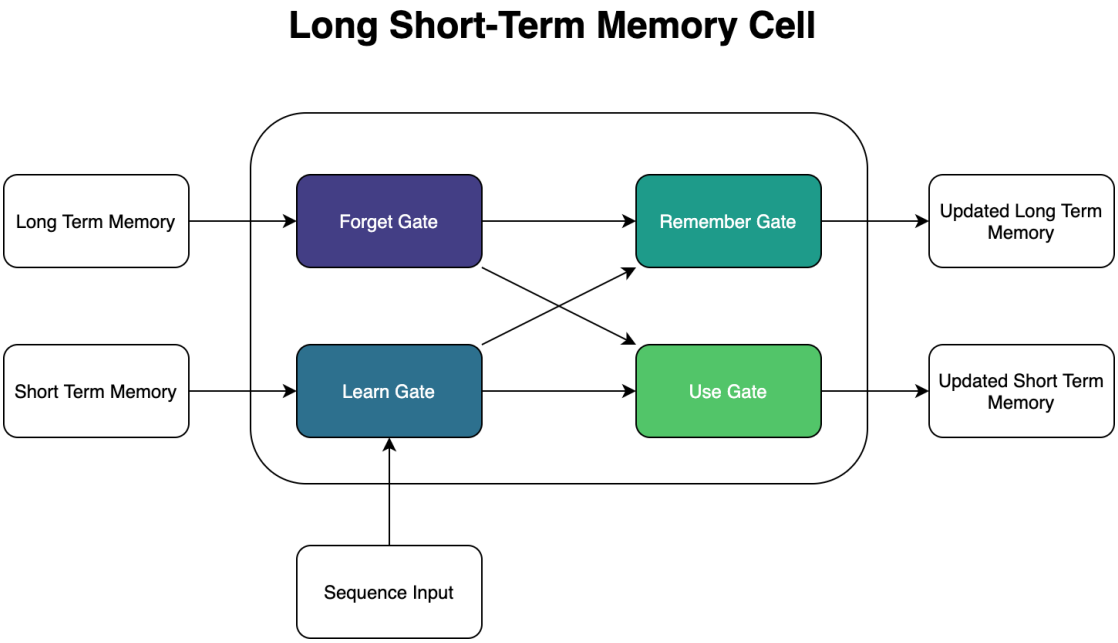


Figure 13: An LSTM consists of four gates. The **forget gate** takes the previous long-term memory and decides which information to discard. The **learn gate** looks at the previous short term memory and the input of the current sequence and decides which information is forwarded. The **remember gate** is responsible for updating the long-term memory based on the information given from the two previous gates. The **use gate** also looks at the output of the two previous cells, but focuses on updating the short term memory, thus the information which is used at the current point in time.

[Hochreiter and Schmidhuber 1997] proposed an architecture for an RNN which handles long sequences way better compared to their original counterparts. They proposed the so-called Long Short-Term Memory (LSTM) as it is particularly capable of handling long sequences. An LSTM consists of cells which have different functionalities for deciding which information to keep and which to discard. Figure 13 shows the high-level layout of such a cell. In each time step, thus for each element of the input sequence, the LSTM cell processes the new element in accordance to its internal state. This internal state consists of the Long-Term Memory (LTM) and the Short Term Memory (STM). Given the new element of the sequence, these internal status are updated in each step. The required calculations are carried out by four gates. The *forget gate* looks at the LTM and decides which information is being forgotten and thereby which information is preserved for further processing. Similarly, the *learn gate* looks at the STM, also often called *hidden state*, as well as the current element of the sequence and decides which information is relevant for further processing. In a next step, the *remember gate* sees both outputs of the previous gates and is responsible for saving long-term information

in the LTM, which is updated accordingly. The *use gate* filters the information which is important for usage at the given time step and saves that information in the STM.

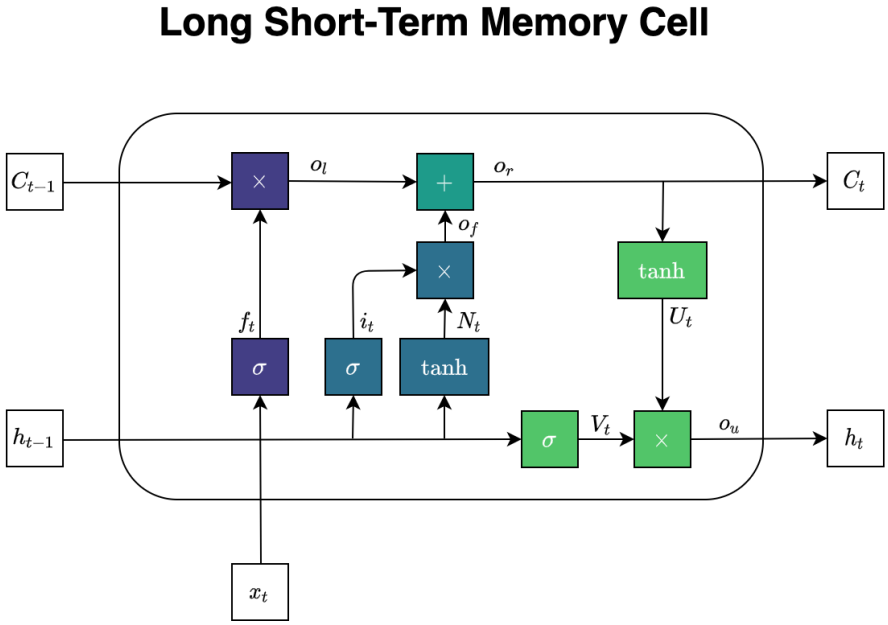


Figure 14: The four gates are coloured in the same way as in figure 13. The figure shows the mathematical operations applied to the inputs of the cell. The **forget gate** calculates the dot product between the LTM and the concatenation of the hidden state (STM) and the input x_t. The **learn gate** concatenates the short term memory h_{t-1} with the current input x_t as well and then calculates a state and a *forget factor*, which are then combined by multiplication. The **remember gate** simply concatenates the output of the **forget gate** and the **learn gate**. The **use gate** takes the output of the **remember gate** and calculates the product between with the concatenation of the STM h_1 and the current sequence input x_t. Every step is explained in more mathematical detail in the surrounding text.

Given this high-level picture of an LSTM, the next step is to mathematically describe the calculations which are carried out by the LSTM. Figure 14 shows a more detailed version of an LSTM cell, where each gate coloured according to figure 13. The previous LTM is denoted by C_{t-1} and the previous STM, or hidden state, is denoted by h_{t-1}, the updated states C_t and h_t respectively. The current element at time step t is denoted by x_t. Inside of the cell, mainly four calculations are conducted: Firstly, normalising the input by using an activation function which is either the *Sigmoid* (σ) or *tanh* (tanh) activation function; see equation 34 and 35. Secondly, the dot product between two tensors ($\times$). Thirdly, the concatenation of two vectors which is denoted by two merging arrows (an intersection) in the figure and by $\langle\rangle$ in the formulas. Fourthly, the sum of two vectors ($+$). Having all variables and operations defined, we take a look at the

different gates and how they work in detail.

$$\sigma(x) = \frac{1}{1+\exp(-x)} = \frac{\exp(x)}{\exp(x)+1} \tag{34}$$

$$\tanh(x) = \frac{2}{1+\exp(-2x)} - 1 \tag{35}$$

Learn Gate First, the current element x_t is being taken and concatenated with the current STM h_{t-1}. Then, the dot product between this resulting vector and the weights W_n of the learn gate is being calculated, before the tanh activation function is being applied. Let N_t denote this intermediate result. Furthermore, the *ignore factor* i_t is being calculated by applying the Sigmoid activation function to the dot product between additional weights W_i and the same concatenation of x_t and h_{t-1} as before. The output of the learn gate is then defined as the dot product $N_t i_t$.

$$N_t = \tanh(W_n \langle h_{t-1}, x_t \rangle) \tag{36}$$

$$i_t = \sigma(W_i \langle h_{t-1}, x_t \rangle) \tag{37}$$

$$o_l = N_t i_t \tag{38}$$

Forget Gate Similarly to the learn gate, the dot product between a set of weights W_f and the concatenation of x_t and h_{t-1} is passed to the Sigmoid activation function. Let this value, also called *forget factor*, be denoted by f_t. The output of the forget gate is then defined by the dot product between f_t and the LTM C_{t-1}.

$$f_t = \sigma(W_f \langle h_{t-1}, x_t \rangle) \tag{39}$$

$$o_f = C_{t-1} f_t \tag{40}$$

Remember Gate Simply, the output of the learn gate o_l is summed with the output of the dot product of the LTM C_{t-1} and the forget factor f_t of the forget gate.

$$o_r = o_l + o_f \tag{41}$$

Use Gate The tanh activation function of the dot product between a set of weights W_u and the output of the forget gate o_r is being calculated. Let this value be denoted by U_t. Additionally, the value V_t is being calculated by taking the Sigmoid activation of the dot product between a set of weights W_v and the concatenation of h_{t_1} and x_t again. Followingly, the output is defined as the product of U_t and V_t.

$$U_t = \tanh(W_u o_r) \tag{42}$$

$$V_t = \sigma(W_v \langle h_{t-1}, x_t \rangle) \tag{43}$$

$$o_u = U_t V_t \tag{44}$$

Inferring from the descriptions of the gates and figure 14, the new LTM C_t is defined by o_r and the new STM, or hidden state, h_t is defined by o_u. The whole LSTM cell then behaves like a traditional RNN, where each element of the sequence is being processed and the different states are updated accordingly. Figure 15 shows three cells of the unfolded sequence.

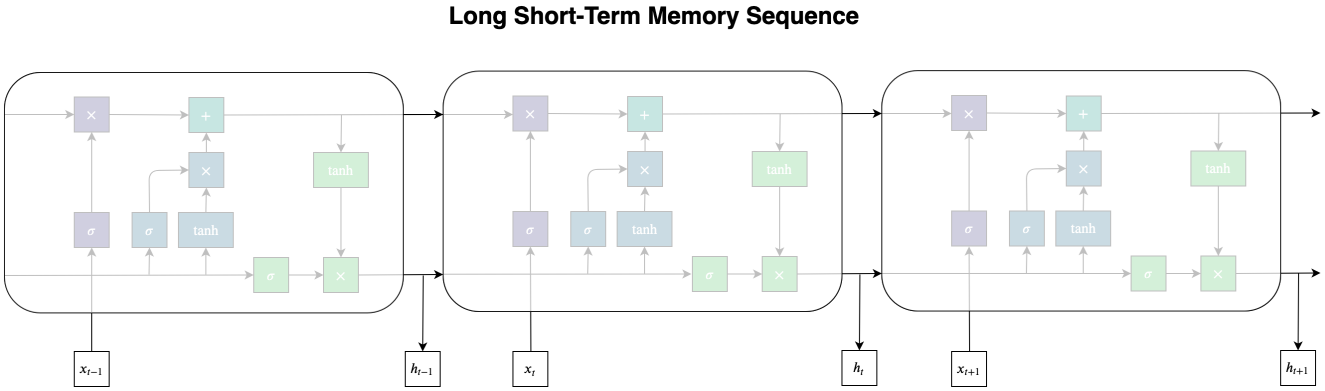


Figure 15: Multiple LSTM cells unfolded resemble a sequence, where each state is passed on to the next cell. This continues until the whole input sequence x_t is processed. This is related to the function of a classical RNN, but LSTMs do not suffer that much from vanishing or exploding gradient and are thus better in learning long-term dependencies in the input sequence.

Apart from the fact that it is easier for an LSTM to store long-term dependencies, the training history of an LSTM is not completely dependent on each previous operation. It only depends on the previous states, thus completely decoupling the calculations of the gradients for each input from its history. Therefore, LSTMs are less prone to the vanishing gradient problem. In practice, the gradients are automatically calculated. PyTorch[11] does this by applying *reverse mode automatic differentiation* [Paszke et al. 2019].

3.7 Linear Spline Interpolation

Missing data can be handled in multiple ways. A simplistic approach is the usage of spline interpolation. Splines define a piecewise polynomial function between two points of a series, also called nodes. Hence, missing data points can be calculated using the interpolated polynomial function. The most elementary polynomial function is of

[11] https://pytorch.org/

first order, thus only forcing the function values of adjacent nodes to be equal. Inferentially, defining a linear function. A second-order polynomial also defines the first derivative of the nodes to be equal and is hence called quadratic interpolation. Concludingly, speaking of a cubic interpolation, the second derivate has to be equal as well. The same applies for higher polynomials for interpolation.

As the computational costs of computing splines is quite high, this thesis will only focus on linear spline interpolation. Higher orders have been tested as well but take too much time, especially considering that the interpolation is still way too simple to account for the patterns in the time series. Figure 16 shows a slice of a time series which will also be used in further chapters as an example for data interpolation and simulation.

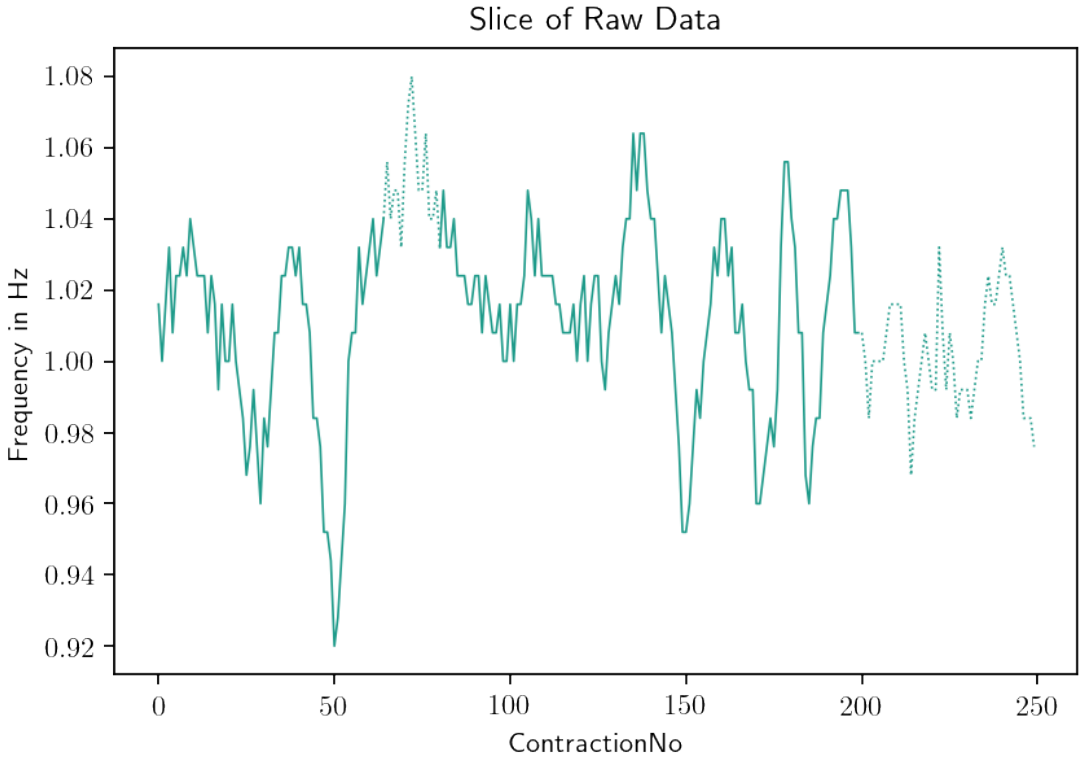


Figure 16: Example **slice of the data**. Dotted lines are missing or padded data points.

Let x_i denote the ith data point in the given series. Then, a linear spline between x_i and x_{i+1} is defined by[12]:

$$s(x) = f(x_i)\frac{x - x_{i+1}}{x_i - x_{i+1}} + f(x_{i+1})\frac{x - x_i}{x_{i+1} - x_i} \tag{45}$$

[12]**Source:** `https://www.math.uh.edu/~jingqiu/math4364/spline.pdf`

Given $s(x)$, the domain between x_i and x_{i+1} can be evaluated at any point. Figure 17 shows the linear spline interpolation applied to the exemplary slice. It can be seen that the fit is not very good, but enables us to pass the data to models for inference. It can also be seen, that for missing data at the end of a series, the last value is just being repeated as the right-hand values are missing due to padding. This arises from the fact that, for enabling batch processing, all input vectors must have the same length.

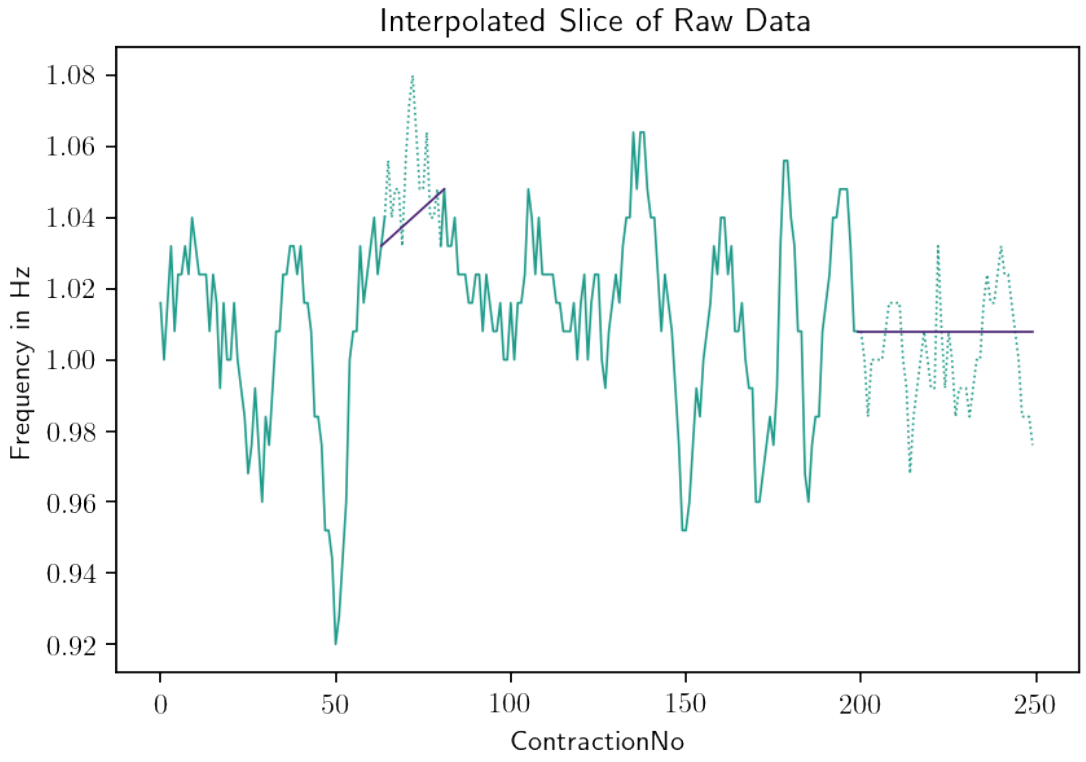


Figure 17: Example **slice of the data** used for **linear interpolation** and fitting a $\mathcal{GP}$. Dotted lines are missing data points not seen during interpolation or fitting of the $\mathcal{GP}$.

3.8 Gaussian Processes

Gaussian Processes ($\mathcal{GP}$s) are fully defined by a multivariate Normal distribution over a continuous input space, which is primarily defined by a finite number of observations [Rasmussen and Williams 2005, p. 13]. A $\mathcal{GP}$ can be evaluated at any set of index points, providing a comfortable way of evaluating points which were originally not present in the observations. Also, as the GP itself is a distribution, it can be used to obtain multiple samples from the fitted process. Therefore, $\mathcal{GP}$s can be used for tackling the following three challenges concerning RR intervals.

Firstly, recordings may have missing data points and while using polynomial functions helps to mitigate some of the arising issues, this approach fails to capture the underlying nature of the process. For example, using splines, a feature defined by the mean of the time series, will most likely be not that much affected whereas features that utilise the frequency domain are more prone to simple interpolation techniques. Secondly, deep learning models benefit strongly from the availability of a lot of training data. Having a defined distribution at hand, samples can be easily taken to artificially create more training data. The sampled data might be highly correlated, but still adds more variability compared to only using the original data while also accounting for the uncertainty and thereof the variability of the data. Another effect of this is, that less epochs are required for training the model which might lead to less overfit. CNNs for image classification work in a similar way by cropping and rotating the input images to enable the model to generalise better. Thirdly, to achieve a high throughput during the training of deep learning models, a fixed input shape is highly beneficial. Depending on the definition of the model, this is indispensable. While even perfectly recorded data has the issue, that due to the nature of a different pace of heart contractions, the recorded samples vary in length. Samples taken from a $\mathcal{GP}$ can be forced to always have the same length. For mitigating this third challenge, other approaches might be considered as well, but as $\mathcal{GP}$s incorporate the handling fo all three issues, they will be favoured and further investigated.

As a $\mathcal{GP}$ is a multivariate Normal distribution, it is fully defined by its mean μ_D and a covariance matrix Σ_D, where D defines the amount of observations and thus the discrete dimensionality. Let the set of observations be denoted by χ and let the corresponding $y = f(\chi)$ be assumed to be known for all observations. Then, the mapping function can be written as:

$$f(x) \sim \mathcal{GP}(\mu_D, \Sigma_D) \tag{46}$$

As μ_D is not compelled to be a constant value, it can be defined by a function itself, only depending on x. x is used instead of χ to underline that this function can be evaluated for any input at any time. In practice, x will be replaced by the predictive index points. We will let μ_D be defined by the following mean function:

$$\mu_D = m(x) \tag{47}$$

For constructing the covariance function, the *Kernel Trick* [Schölkopf 2000] is exploited to calculate the dot product in higher-dimensional space. Therefore, the covariance matrix Σ_D is defined by a *kernel function* which defines the similarity between different points in the input space. The sum or product of two kernels is still a kernel [Shawe-Taylor and Cristianini 2004, Proposition 3.22 (Closure Properties)]. This property can be used to combine different kernels to describe the given observations accurately and hence capturing the underlying natural process. Concludingly, the covariance matrix is given by the *gram matrix*, which defines the distances, based on the given kernel function, between all combinations of observations. This matrix will be further referred to

as the *kernel matrix* which must be symmetric and positive-definite [Rasmussen and Williams 2005, ch. 4]. The covariance matrix is defined by the following equation:

$$\Sigma_D = k(\chi, \chi) \tag{48}$$

Using definitions 47 and 48, we can refine 46 to:

$$f(\chi) \sim \mathcal{GP}\big(m(x), k(\chi, \chi)\big) \tag{49}$$

The whole statistical model is then defined by the following equation, where y denotes the output of the model:

$$y = \mathcal{N}\big(f(\chi), \sigma_{\text{noise}}\big) \tag{50}$$

where the observations are assumed to be normally distributed around the mapping function (defined by the $\mathcal{GP}$) and σ defines the noise of the observations.

By this definition, we acquired the desired distributions over functions which can be evaluated at any predictive point. Let χ^* denote this set of predictive points and let us refer to the evaluation of these points as *regression* or *prediction*. Let $k(...)$ denote the kernel function for obtaining the covariance matrix. To actually compute the predictions, first, the prior joint distribution between the observations $f(\chi)$ and $f(\chi^*)$ must be defined [ibid., eq. 2.21]:

$$\begin{bmatrix} y \\ f(\chi^*) \end{bmatrix} \sim \mathcal{N}\left(m(x), \begin{bmatrix} k(\chi, \chi) + \sigma^2_{\text{noise}} I & k(\chi, \chi^*) \\ k(\chi^*, \chi) & k(\chi^*, \chi^*) \end{bmatrix} \right) \tag{51}$$

Note that $y = f(\chi)$ as the noise of the observations is already included in the covariance matrix. Then, we condition on the observations [ibid., eq. 2.22, 2.23, 2.24] to obtain the predictive distribution:

$$f(\chi^*)|\chi^*, \chi, f(x) \sim \mathcal{N}\big(\bar{\chi_*}, \text{cov}(\chi_*)\big) \tag{52}$$

$$\bar{f_*} = k(\chi^*, \chi)\big(k(\chi, \chi) + \sigma^2_{\text{noise}} I\big)^{-1} y \tag{53}$$

$$\text{cov}(f_*) = k(\chi^*, \chi^*) - k(\chi^*, \chi)\big(k(\chi, \chi) + \sigma^2_{\text{noise}} I\big)^{-1} k(\chi, \chi^*) \tag{54}$$

Note that this predictive distribution assumes a zero mean, thus $m(x)$ must be subtracted first before applying the formula. Having the predictive distribution defined, the process of regression is straight forward[13]. The last step is to select an appropriate kernel function. Chapter 4.3.1 will explain in detail which kernel functions are used for the given specific problem set in this thesis.

For now, the focus will lie on the parameters that define the kernel functions, as most, if not all, kernel functions have parameters which can be tuned with respect to the given

[13] Either implementing algorithm 2.1 from Rasmussen and Williams 2005 or using a library.

set of observations. We will generally refer to these flexible parameters as *kernel hyperparameters* and denote them by θ, thus θ is a set of hyperparameters. The aim is to select the hyperparameters that most accurately describe the underlying process, in our case the RR intervals. When choosing an optimisation strategy for selecting appropriate hyperparameters, computational constraints must be taken into account. The approaches can be split into two categories: A traditional approach of finding point estimates ($\hat{\theta}$) for the hyperparameters and a Bayesian approach of defining a prior on each hyperparameter and then updating our beliefs according to the given observations. When optimising traditionally, a maximum likelihood point estimate for the marginal log-likelihood function is the object of interest. The marginal log-likelihood will be introduced shortly. One way is to conduct *grid search*, where the marginal log-likelihood of different combinations of hyperparameters is evaluated and the most likely set of parameters is being chosen. The drawback is that the search space might be tremendously large, yielding in very long run times. When limiting the search space, appropriate point estimates might not be found. To circumvent this issue, *gradient descent* [Ruder 2016] will be considered for retrieving point estimates for the set of hyperparameters. The Bayesian approach marginalises the hyperparameters, thus integrating out the parameters, to obtain a *posterior predictive distribution*, which can then be used to sample from. The benefit of the Bayesian approach is that the uncertainty will be taken into account. The following two sub-chapters will formalise and investigate both approaches.

3.8.1 Kernel Hyperparameters: Point Estimate Retrieval by Gradient Descent

The aim is to maximise the marginal log-likelihood of the $\mathcal{GP}$ with the set of hyperparameters θ. K_θ denotes the covariance matrix and θ is used as an index to emphasise its dependence on the kernel parameters. The log marginal likelihood is given by [Rasmussen and Williams 2005, eq. 2.30], which is extended by the dependency on θ.

$$\mathcal{L}(y|\chi,\theta) = -\frac{1}{2}y^T\left(K_\theta + \sigma^2_{\text{noise}}I\right)^{-1}y - \frac{1}{2}\log|K_\theta + \sigma^2_{\text{noise}}I| - \frac{n}{2}\log 2\pi \tag{55}$$

For optimisation, it is sufficient to optimise the proportion:

$$\mathcal{L}(y|\chi,\theta) \propto -\frac{1}{2}y^T\left(K_\theta + \sigma^2_{\text{noise}}I\right)^{-1}y - \frac{1}{2}\log|K_\theta + \sigma^2_{\text{noise}}I| = \mathcal{L}_{prop}(y|X,\theta) \tag{56}$$

For applying gradient descent, we are interested in the gradient of the negative proportional marginal log-likelihood. Let Θ denote the amount of optimisable parameters:

$$\nabla\theta = -\sum_{i=1}^{\Theta}\frac{\partial\mathcal{L}_{prop}(y|\chi,\theta_i)}{\partial\theta_i} = \left(-\frac{\partial\mathcal{L}_{prop}(y|\chi,\theta_1)}{\partial\theta_1},\ldots,-\frac{\partial\mathcal{L}_{prop}(y|\chi,\theta_\Theta)}{\partial\theta_\Theta}\right) \tag{57}$$

The exact form of this equation depends on the chosen kernel. In the case of a sum of kernels, the partial derivatives are quite easy to calculate, as the terms not concluding the current θ can simply be dropped. It becomes rather complex if a product of

kernels is being used. In practical usage, libraries such as PyTorch, use *reverse-mode automatic differentiation* [Paszke et al. 2019], as mentioned before. Therefore, the analytical solution will be omitted at this step.

3.8.2 Kernel Hyperparameters: Posterior Predictive Distribution

The posterior predictive distribution is given by

$$p(\tilde{y}|y) = \int_{\theta} p(\tilde{y}|\theta, y)p(\theta|y)d\theta \tag{58}$$

where $\tilde{y}$ is a future draw given the observed data. To be able to draw samples from this posterior predictive distribution, first, a posterior draw $\theta^{(1)}$ is being taken from the joint posterior distribution $p(\theta|y)$ and then a draw from the fitted $\mathcal{GP}$ (equation 52) given $\theta^{(1)}$ is taken. This process is repeated until the desired amount of samples is reached.

As it will be computationally infeasible to fit a $\mathcal{GP}$ to the complete time series, the simulation will be based on slices of the original time series and the beliefs about the parameters will be continuously being updated. This means that first of all, a $\mathcal{GP}$ is fitted to the first splice using initial priors defined for each parameter. Then, for the second slice, the priors are given according to the posterior parameter distributions of the previous slice. Eventually, the final posterior distributions will have considered all data points of the RR interval. Let's assume our observations are split into two slices χ_a and χ_b. After that, the sequential update of our beliefs can be formalised as [Murphy 2013, p. 75]:

$$p(\theta|\chi_a, \chi_b) \propto p(\chi_b|\theta)p(\theta|\chi_a) \tag{59}$$

This sequential update assumes exchangeability of our data [Bernardo 2001]. As we are dealing with time series data, this assumption is violated. Still, the resulting priors are assumed to be better, compared to setting them manually for each slice, as the hyperparameter space is quite high dimensional. Also, for some kernels, it is very hard to develop a good understanding of the priors, especially considering the final joint distribution and its likelihood.

3.8.3 Confidence and Prediction Intervals

As the error is assumed to be normally distributed, an approximation of $\sigma_\epsilon = 1.96$ is being taken for calculating the following 95 per cent intervals. The 95 per cent confidence interval for the mean can be evaluated for each predictive point χ^* and is given by:

$$f_*^{\text{upper/lower}} = m(\chi^*) \pm 1.96\sqrt{\text{tr}\,\text{cov}(f_*)} \tag{60}$$

For the 95 per cent prediction interval, the noise of the observations must be considered as well:

$$y_*^{\text{upper/lower}} = m(\chi^*) \pm 1.96\sqrt{\text{tr}\,\text{cov}(f_*) + \sigma^2_{\text{noise}}} \tag{61}$$

4 Methods

This chapter describes the different methods used for obtaining the results of this thesis. That includes a thorough description of the data pipeline.

4.1 Preprocessing

For conducting the task of age prediction, different approaches will be taken. The aim of these different approaches is, on the one hand, to make the previous results obtained by [Makowiec and Wdowczyk 2019] reproducible and on the other hand, to extract features for multiple slices of the time series as this might provide more information for the feature-based models. Therefore, each model will be trained once for the **complete** RR interval and once for **constant** slices of around 5 minutes. The expectation is, that for the DeepSleep-based model, the complete time series will lead to better results whereas the feature-based models profit from a set of features for each slice and hence more information. Additionally, the models will be trained for classification, not assuming an order of the age decades, as well as for regression, where an order of the class labels is being assumed. This leaves us with the following four combinations of preprocessed data for each dataset:

- complete classification
- constant classification
- complete regression
- constant regression

In case of constant slices, the models will, during inference, conduct one classification for each slice and perform a majority vote for the age decade label. In case of regression, the closest age decade will be assumed as the predicted class. Concluding from the limitation that the data can only be simulated for slices, the models run on the simulated data will only cover the type constant. Recall that outliers in the *CAST RR Interval Sub-Study Database* dataset (the Physionet dataset) are already dealt with during the reading process of the initial data.

Both datasets are split into training, validation and testing. For the Gdańsk dataset, the splits are set to $[0.6, 0.2, 0.2]$ as the dataset is rather small, whereas the splits for the PhysioNet dataset are set to $[0.8, 0.1, 0.1]$. The seed used for the random split is 42. The first step of data preprocessing includes linear spline interpolation, the functionality is provided by the function `interpolate()` by the `pandas` library[14]. In the case of slicing the time series, each RR interval is split into 48 slices for the Gdańsk dataset and into 240 slices for the Physionet dataset, using the function `array_split()` provided by the `numpy` library[15].

[14] `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.interpolate.html`

[15] `https://docs.scipy.org/doc/numpy/reference/generated/numpy.array_split.html`

For obtaining a closely related set of 33 features used by [Makowiec and Wdowczyk 2019], the library `hrvanalysis`[16] is used. The features are categorised as follows:

- **Time Domain Features:** Mean_NNI, SDNN, SDSD, NN50, pNN50, NN20, pNN20, RMSSD, Median_NN, Range_NN, CVSD, CV_NNI, Mean_HR, Max_HR, Min_HR, STD_HR
- **Geometrical Domain Features:** Triangular_index, TINN
- **Frequency Domain Features:** LF, HF, VLF, LH/HF ratio, LFnu, HFnu, Total_Power
- **Non-Linear Domain Features:** CSI, CVI, Modified_CSI, SD1, SD2, SD1/SD2 ratio, SampEn

Some of the features are not independent. For further information, please refer to either the python package or the methods chapter of [ibid., p. 3].

For the DeepSleep-based model, no features are necessary, but the time series must be padded for enabling batch processing. The 95th percentile of the respective lengths distributions has been chosen as a trade-off between including the greater part of the series and padded values. The percentiles have been rounded to the next value dividable by $1,000$. This results in a padding value of $27,000$ for the Gdańsk dataset and 135.000 for the Physionet dataset. In case of the type **constant**, the paddings are divided by the number of slices, thus resulting in 462 for the Gdańsk dataset and $2,812$ for the Physionet dataset.

4.2 DeepSleep Architecture

In addition to the feature-based models, a deep learning approach will be investigated to find out, how well this self-learning approach performs. The main difference between these types of models is that deep learning can extract features on its own and is independent of manually constructed features. Usually, deep learning models are not as interpretable as feature-based models, but can perform better under some circumstances.

[Supratak et al. 2017] proposed a deep learning architecture, consisting of a CNN for feature extraction, an LSTM for long-term dependencies and a simple FFN for the downstream task of classification and regression. Their architecture is called *DeepSleep* and works directly on raw single-channel EEG data. The obtained results for sleep stage detection seem to indicate that DeepSleep is suitable for long medical time series data, as they obtained significant F1 scores on two different datasets. Therefore, the proposed architecture serves as a basis where some modifications are being applied. The architecture used in this thesis can be seen in figure 18.

The first part of the model consists of a two-headed **CNN** with different kernel sizes, where one focuses more on longer patterns whereas the other one puts more emphasis on shorter patterns. The CNN consists of three layers. Contrary to CNNs working on

[16] `https://github.com/Aura-healthcare/hrvanalysis`

image data, the kernel size and amount of filters is not changed throughout the layers. After the CNN layers, the batches are normalised and average pooling is being applied to the CNN features, which has the same size as the filters. After concatenating both strains, another batch normalisation and activation is being applied to make sure that the output is standardised and the **LSTM** can process the input accordingly. The LSTM itself is bidirectional and consists of two layers. Dropout [Srivastava et al. 2014] is being applied to the input and the intermediate layers to prevent the model overfitting to the training data. Additionally to the LSTM, a **FNN** is directly working on the features provided by the CNN as well. *This enables our model to be able to add temporal information it learns from the previous input sequences into the feature extracted from the CNNs* [Supratak et al. 2017, p.3]. Afterwards, the output of the FNN and the output of the LSTM are concatenated. Then, depending on the downstream task, the final part deviates. For **Classification**, another linear layer maps the shape of the tensor to the desired amount of classes, followed by (log-)Softmax. For **Regression**, two linear layers are tied together, the first one reducing the shape further whereas the last linear layer maps to one regression output.

For the classifier, the negative log-likelihood is being taken as the loss function. The models are run using unweighted cross-entropy [32] as well as weighted cross-entropy [33]. For the regressor, the MSE loss function [26] is being taken. For optimising the loss functions, the Adam optimiser [Kingma and Ba 2014] is being used. Additionally, the models are run on oversampled data to test if this helps the model to generalise. Also, all DeepSleep-based models implement early stopping, choosing the model that performs best on the validation dataset during training. The metric for early stopping is the loss.

The hyperparameters can be seen in figure 18 as well. If possible, the largest number printed in the figure is being taken, but due to computational constraints, some adjustments are made. The exact settings of hyperparameters are contained in appendix 9 for each model.

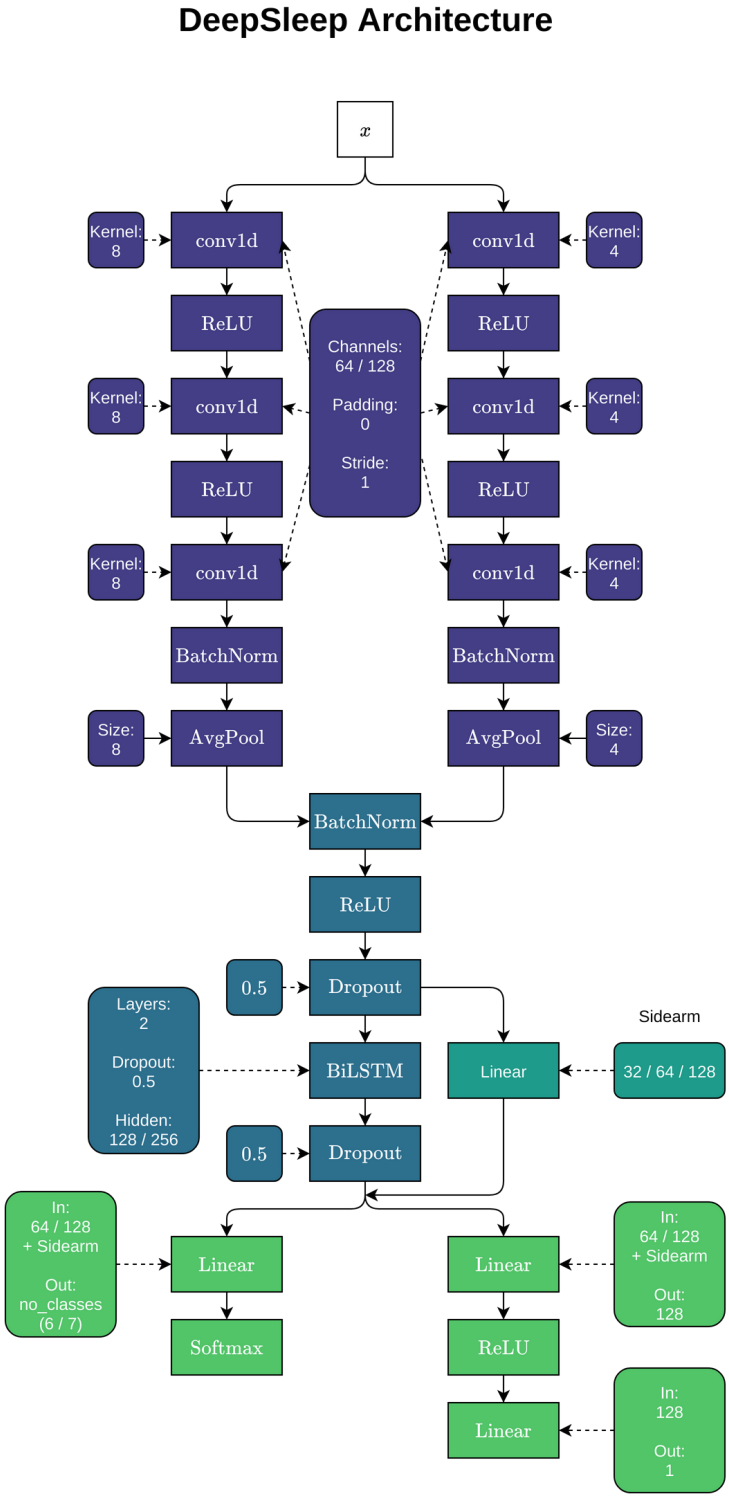


Figure 18: DeepSleep-based architecture. The first part consists of a **CNN** with two heads. The second part consists of an **BiLSTM** with a concurrent **FNN**, also denoted as *Sidearm*, and the **Downstream Classifier/Regressor** producing the output.

4.3 Gaussian Process Simulation

A $\mathcal{GP}$ is mostly defined by its covariance matrix. Therefore, defining a kernel that can capture the information embedded in the data is essential. As the sum or product of a kernel is a kernel [Shawe-Taylor and Cristianini 2004 and Bishop 2006 eq. 6.21 and 6.22], the sum of multiple kernels will be used for creating the covariance matrix. The covariance matrix grows rather quickly when adding new data points, as the shape of the covariance matrix will be defined by the square of the defined data points. Let N be the number of data points used for fitting a $\mathcal{GP}$, then the shape of the covariance matrix will be given by $[N, N]$. Due to memory constraints, the simulation for the data will only be investigated for the time series being split into chunks of around 2.5 minutes, which is twice the amount as for the original Gdańsk dataset.

4.3.1 Kernels and Hyperparameters

The assembly of the covariance function must be able to capture the underlying information embedded in the data. As seen in the data section, there is a high variability in the given time series. Therefore, it has been decided to consider the sum of multiple kernels such that the covariance function has the ability to adapt to all kinds of data. A smoothing kernel has been included to capture direct dependencies from adjacent points in the time series. Two periodic kernels have been chosen, one which is able to capture repeating patterns and one that is able to capture repeating but flattening periodic patterns. The irregular kernel is included to capture the long-term trend of the signal and the Matérn kernels have been included to embed information about the jagged behaviour of the time series.

The definition of the kernels and their hyperparameters are:

Smoothing Kernel:

$$k_s(x_a, x_b) = \sigma^2 \exp\left(- \frac{\|x_a - x_b\|^2}{2\ell^2} \right) \tag{62}$$

With trainable hyperparameters σ_s and ℓ_s.

Periodic Kernel:

$$k_p(x_a, x_b) = \sigma^2 \exp\left(- \frac{2}{\ell^2} \sin^2\left(\pi \frac{\|x_a - x_b\|}{p} \right) \right) \tag{63}$$

With trainable hyperparameters σ_p, ℓ_p and p_p.

Local Periodic Kernel:

$$k_{lp}(x_a, x_b) = \sigma_1^2 \exp\left(- \frac{2}{\ell_1^2} \sin^2\left(\pi \frac{\|x_a - x_b\|}{p} \right) \right) \sigma_2^2 \exp\left(- \frac{\|x_a - x_b\|^2}{2\ell_2^2} \right) \tag{64}$$

With trainable hyperparameters σ_{lp1}, ℓ_{lp1}, p_{lp}, σ_{lp2} and ℓ_{lp2}.

Irregular Kernel (Rational Quadratic):

$$k_{rq}(x_a, x_b) = \sigma^2 \left(1 + \frac{(x_a - x_b)^2}{2\alpha\ell}\right)^{-\alpha} \tag{65}$$

With trainable hyperparameters σ_{rq}, α_{rq} and ℓ_{rq}.

Matérn 1/2 Kernel:

$$k_{m1}(x_a, x_b) = \sigma^2 \exp\left(-\frac{\|x_a - x_b\|}{\ell}\right) \tag{66}$$

With trainable hyperparameters σ_{m1} and ℓ_{m1}

Matérn 3/2 Kernel:

$$k_{m3}(x_a, x_b) = \sigma^2(1+z)\exp(-z) \tag{67}$$

$$z = \sqrt{3}\frac{\|x_a - x_b\|}{\ell} \tag{68}$$

With trainable hyperparameters σ_{m3} and ℓ_{m3}.

Matérn 5/2 Kernel:

$$k_{m5}(x_a, x_b) = \sigma^2 \frac{1 + z + z^2}{3}\exp(-z) \tag{69}$$

$$z = \sqrt{5}\frac{\|x_a - x_b\|}{\ell} \tag{70}$$

With trainable hyperparameters σ_{m5} and ℓ_{m5}.

The sum of all kernels is the resulting kernel that will be used for fitting the $\mathcal{GP}$s and is defined by:

$$\begin{aligned} k_{gp}(x_a, x_b) = k_s(x_a, x_b) + k_p(x_a, x_b) + k_{lp}(x_a, x_b) + k_{rq}(x_a, x_b) \\ + k_{m1}(x_a, x_b) + k_{m3}(x_a, x_b) + k_{m5}(x_a, x_b) \end{aligned} \tag{71}$$

Additionally to these 19 parameters, σ_{noise} is also a trainable hyperparameter. Therefore, the whole set of trainable parameters is given by:

$$\theta_{gp} = \{\sigma_s, \ell_s, \sigma_p, \ell_p, p_p, \sigma_{lp1}, \ell_{lp1}, p_{lp1}, \sigma_{lp2}, \ell_{lp2}, \sigma_{rq}, \alpha_{rq}, \ell_{rq}, \sigma_{m1}, \ell_{m1}, \sigma_{m3}, \ell_{m3}, \sigma_{m5}, \ell_{m5}\sigma_{noise}\} \tag{72}$$

For optimising these hyperparameters, two approaches are being investigated: gradient descent for obtaining point estimates and a Bayesian approach where samples are taken from the posterior predictive distribution. The whole time series is sliced into 96 chunks, each spanning a time range of around 2.5 minutes with around 270 data points each. These observed data points are taken for sampling from the fitted $\mathcal{GP}$s. For each fitted $\mathcal{GP}$, 100 samples are being taken. The fitting represents some sort of stochastic transformation, which means that this transformation can be applied to the input data without modifying the labels. Therefore, this procedure will be applied to the training, validation and test dataset for the Gdańsk dataset. As the whole process is computationally very demanding, including the fitting of the models, the Physionet dataset will not be considered for this task.

In the following chapters, the procedure of fitting the $\mathcal{GP}$ will be explained based on one example slice, having some missing data points and some padding at the end. For the gradient descent, each slice is considered separately as no information flows between them. For the Bayesian approach, the whole procedure is simply repeated with the obtained posteriors of the previous slice as new priors for each subsequent slice.

4.3.2 Kernel Hyperparameter Optimisation Using Gradient Descent

One way to optimise the parameters of the kernel is to optimise the marginal log-likelihood function to obtain point estimates for the joint distribution of hyperparameters. This can be done by either performing grid search or gradient descent. The focus will lie on the latter. All parameters are initialised with the value 1, thus defining θ_{init}. The likelihood of the $\mathcal{GP}$ given the current parameters during learning can be seen in figure 19.

After 10.000 iterations, the following set of parameters is found and denotes the point estimate $\hat{\theta}$ for this example slice.

$$
\begin{aligned}
\hat{\theta} = \{ & \\
& \sigma_s = 0.021854665592977836, \quad \ell_s = 4.2588876554564825 \\
& \sigma_p = 0.05387198576409192, \quad p_p = 0.49980015957564133 \\
& \ell_p = 0.3679243361111636, \quad \sigma_{lp1} = 0.05387198576409192 \\
& \ell_{lp1} = 0.8774408675206453, \quad p_{lp1} = 2.7182818284590446 \\
& \sigma_{lp2} = 0.3678794411714424, \quad \ell_{lp2} = 0.37381285294546657 \\
& \sigma_{rq} = 0.01416396988294803, \quad \alpha_{rq} = 0.6015559752066347 \\
& \ell_{rq} = 4.146611585773073, \quad \sigma_{m1} = 0.004795200159066785 \\
& \ell_{m1} = 1.832663754018536, \quad \sigma_{m3} = 0.3678794411714424 \\
& \ell_{m3} = 0.3678794411714424, \quad \sigma_{m5} = 0.01162291185946003 \\
& \ell_{m5} = 5.030183381753336, \quad \sigma_{noise} = 8.441068346451243e^{-05} \\
\} &
\end{aligned}
\tag{73}
$$

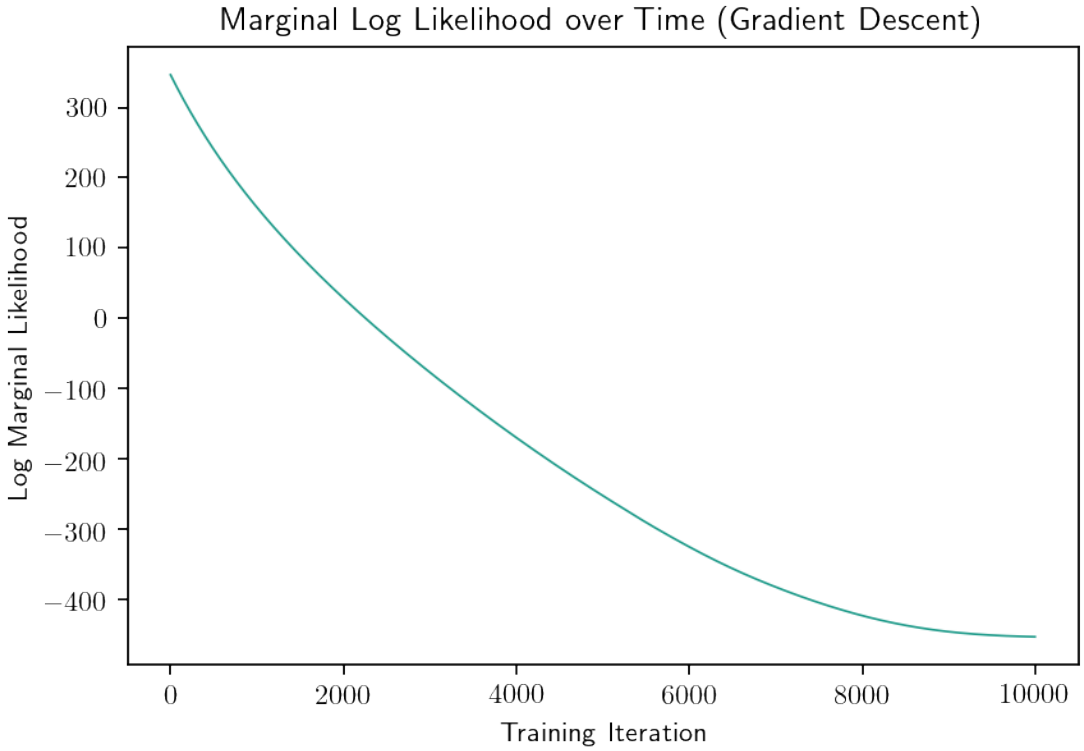


Figure 19: **Log-Likelihood** of the prior kernel parameters during optimisation.

Using these, a $\mathcal{GP}$ is fitted to the available data points. Having the fitted $\mathcal{GP}$ available, 50 draws are being taken for the shown visualisations, which are illustrated in figure 20. One drawing is printed in bold to visualise how a simulated slice will look like.

Figure 21 shows the mean of the $\mathcal{GP}$ together with its confidence bands and prediction bands. The variance at the indices of missing data is higher compared to the variance at known data. Both bands are quite similar in their uncertainty, as there is only one data point available for each point in time and point estimates do not account for the uncertainty in the parameters. Figure 22 shows the optimised kernel for this slice of data using gradient descent.

After careful consideration it has been decided that this approach will not be further investigated due to two reasons. Firstly, the fitting of $\hat{\theta}$ takes a lot of time. More precisely this means around 6 minutes for each slice, so having 96 slices for each time series and 181 series in total that results in 28 days of computational time. Secondly, the point estimates do not account for the uncertainty in the parameters of the kernel, making the second approach of sampling from the posterior predictive distribution more attractive.

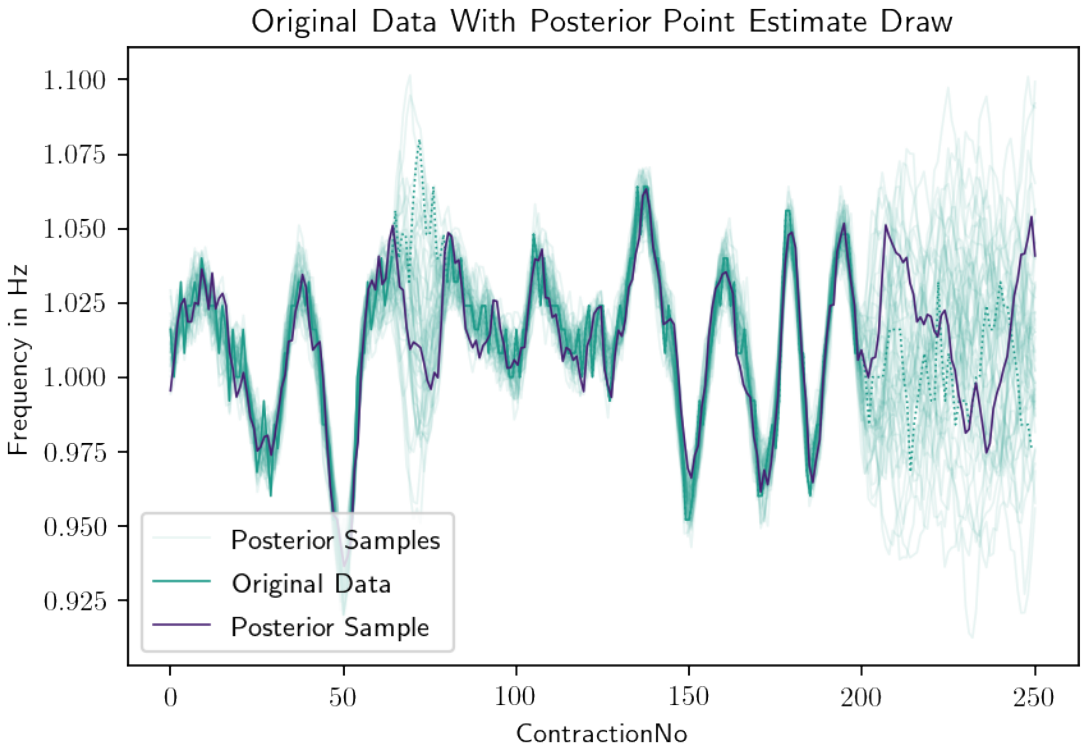


Figure 20: Draws from the **Gaussian Process** with point estimates for the optimal parameters.

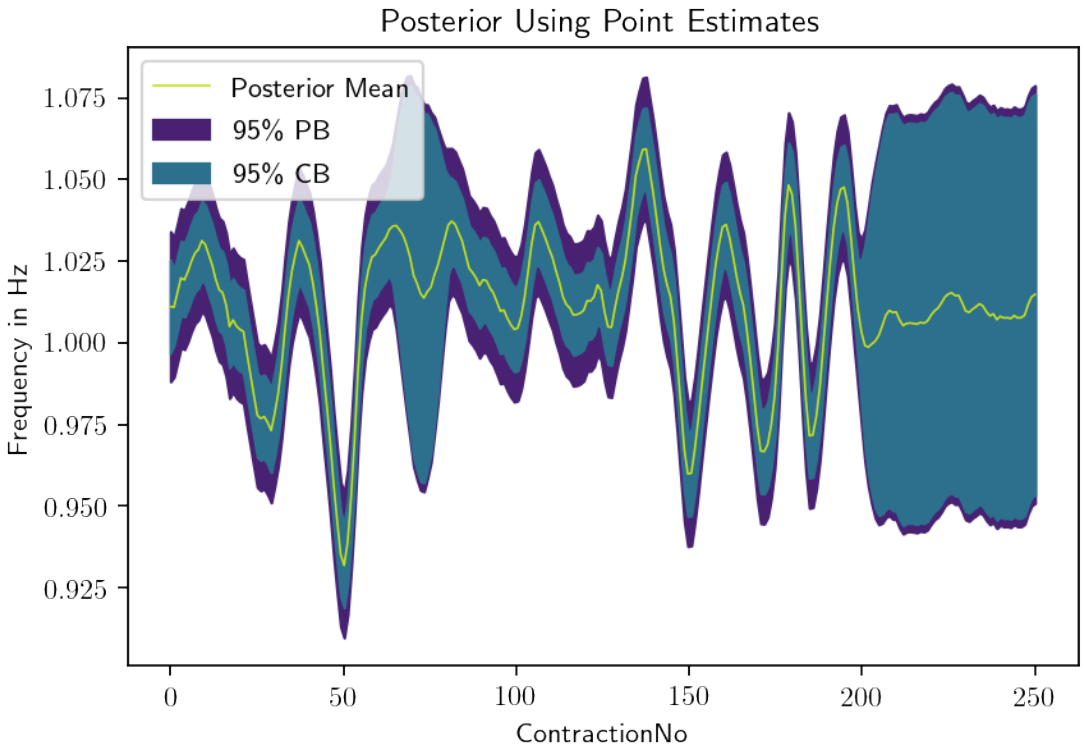


Figure 21: **Mean** with **Confidence Bands** and **Prediction Bands** with points estimates for the optimal parameters.

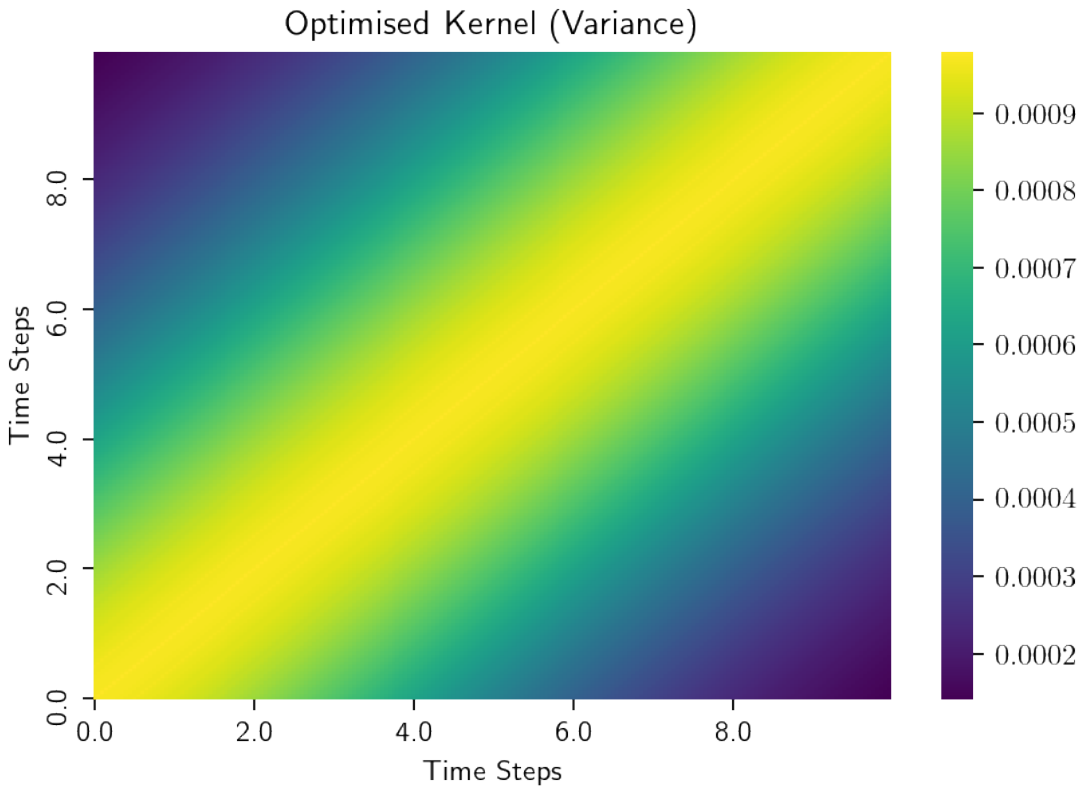


Figure 22: Optimised kernel using gradient descent.

4.3.3 Posterior Predictive Distribution

The implementation uses TensorFlow Probability[17]. The initial prior for all hyperparameters is given by:

$$\text{Lognormal}(x, \mu = 0, \sigma = 1) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(- \frac{\left(\ln(x) - \mu\right)^2}{2\sigma^2} \right) \tag{74}$$

This prior is considered uninformative in the given context and adds the property that the parameters are always positive (as length scale, period and variance are always positive). Hamiltonian Monte Carlo (HMC) [Neal 2012] is used for sampling from the posterior according to the log marginal likelihood 55 of the $\mathcal{GP}$. The step size is initially 0.1 and adaptive, aiming for an acceptance probability of 0.75. A larger step size leads to faster progress but also increases the probability of rejecting a draw. The implementation is based on [Hoffman and Gelman 2011, chapter 3.2]. The number of leapfrog steps is set to 8 and defines how many steps the leapfrog integrator is ran. 10 chains run concurrently with 200 burn-in steps each and the noise variance is assumed to be 0. It takes around 50 seconds for fitting one slice. Having 181 time series with 96 slices each, the computational time for this approach is around 4 days and thus faster compared to the point estimate approach.

Figure 23 shows the example slice, where the original slice is denoted by a dashed line. Multiple posterior draws are shown with a slight green line, where one posterior draw is presented with a bold blue line. These posterior draws will be used as the new, simulated data. It can be seen that around contraction number 60, the uncertainty in the posterior draws is higher, as well as towards the end of the slice where the time series is padded.

Figure 24 shows the 95 per cent confidence and prediction interval together with the mean of the posterior draws. It can be seen that, due to the fact that we have just one data point for each point in time, the confidence and prediction bands are quite closely tight together. The increased uncertainty can be seen as well.

For one of the kernels, namely the smoothing kernel, the posterior parameter distributions are displayed in figure 25. Note that the smoothed distribution includes areas below zero, but the histograms show that all draws are positive. The visualised kernel on the right-hand side represents the resulting kernel of one draw from the posterior distributions, thus the kernel actually changes for each draw. The distributions of all other hyperparameters and a respective kernel draws can be found in appendix 9.1.

[17] `https://www.tensorflow.org/probability`

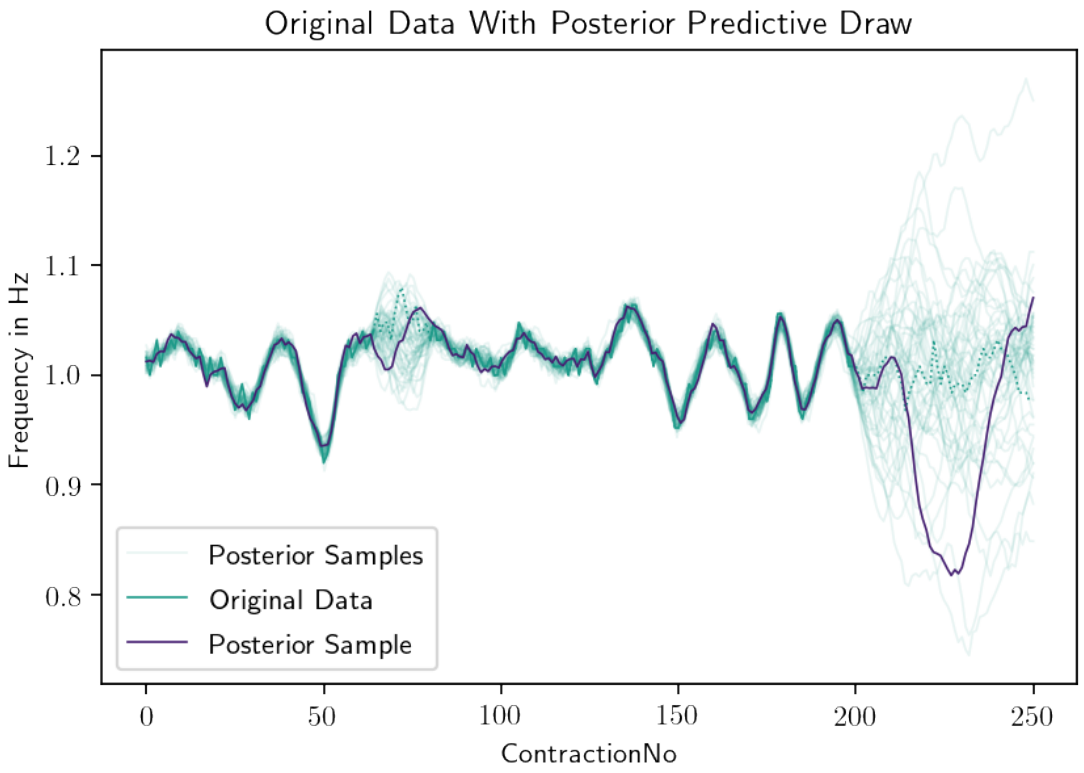


Figure 23: Draws from the **Gaussian Process** with posterior distributions for the parameters.

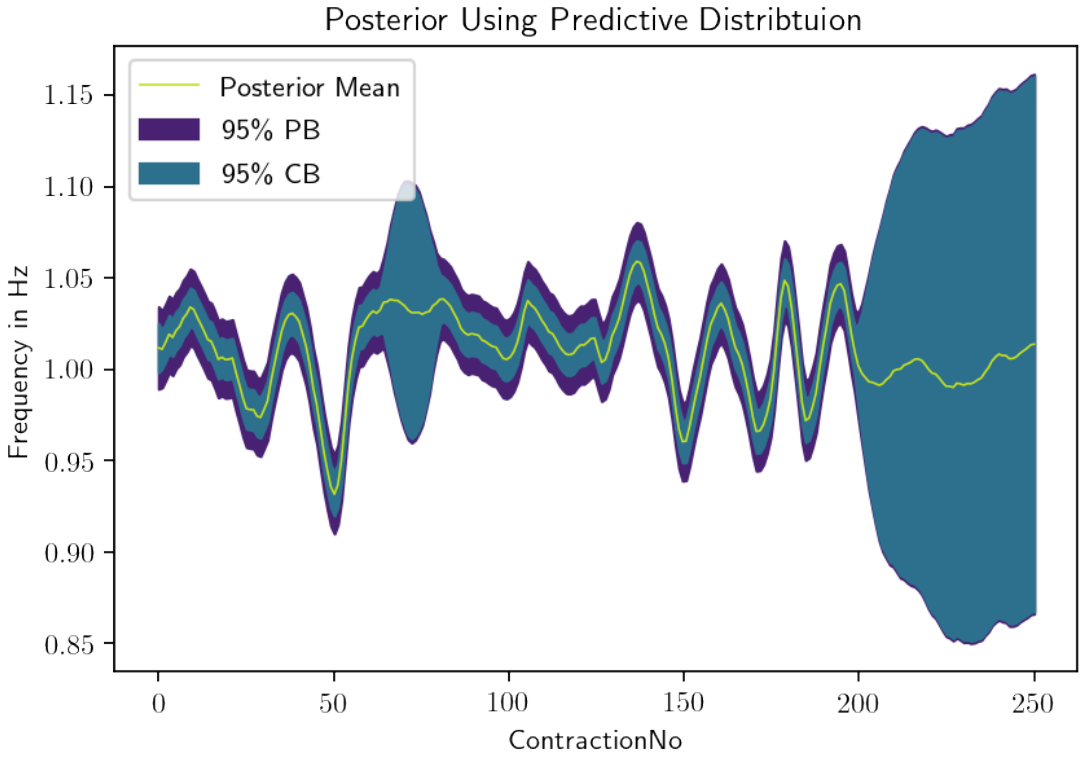


Figure 24: **Mean** with **Confidence Bands** and **Prediction Bands** with posterior distributions for the parameters.

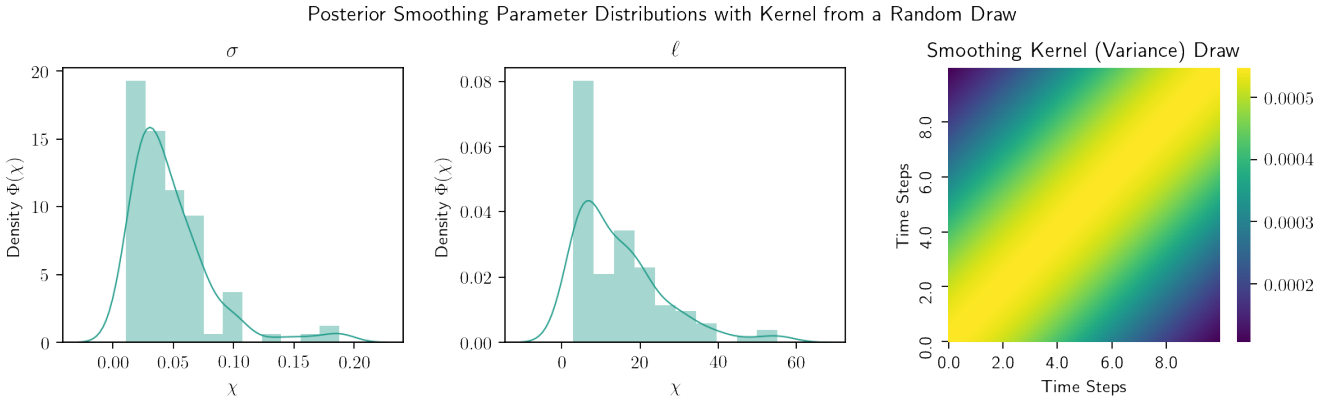


Figure 25: **Posterior Distributions** for the smoothing kernel with example draw for the kernel.

5 Results

This results chapter is divided into three parts. In the first part, the results for the Gdańsk dataset are presented. The second part shows the results for the dataset from Physionet and the third part investigates the results for the data simulated based on the Gdańsk dataset. The baseline models always refers to the naive approach of always predicting the class which is represented most in the training dataset. For the Gdańsk dataset, with seven categories, the baseline is 18.23 per cent and for the Physionet dataset, with six categories, the baseline is 39.14 per cent.

5.1 Gdańsk

Table 1: Accuracies of the feature-based models on the dataset provided by the University of Gdańsk. The baseline of always predicting the most represented class label in the training dataset is 18.23 per cent.

Gdańsk (Feature)

Accuracy	**Naive Bayes**	**Random Forest**	**SVM**	**XGBoost**
Regression / Complete				
Training		45.14%	19.44%	28.47%
Testing		**37.83%**	29.73%	21.62%
Regression / Constant				
Training		24.31%	22.92%	22.22%
Testing		24.32%	27.02%	29.73%
Classification / Complete				
Training	32.64%	73.61%	26.39%	68.06%
Testing	29.73%	24.32%	16.21%	21.62%
Classification / Constant				
Training	21.53%	100.00%	34.72%	39.58%
Testing	27.03%	**32.43%**	24.32%	24.32%

Table 1 shows the results for the feature-based methods on the Gdańsk dataset. Most models perform better compared to the baseline, the only exception being the SVM conducting complete classification. The remaining models have an accuracy between 20 and 30 per cent. The three best performing models on the Gdańsk dataset are highlighted in bold, two of them are feature-based and hence found in table 1. Overall, the best model combination consists of a Random Forest conducting constant classification and complete regression. It seems that the Random Forest had the tendency to overfit

to the training dataset when conducting classification, still, cross-validation has found that this combination of methodology and model generalises best, which is fortified by the high test accuracies. The Naive Bayes classifier performs good considering its very short training time and very simplistic approach. While SVM and XGBoost, apart from one exception, perform above the baseline. It seems that they work better on the task of regression compared to classification. In summary, most feature-based models seem to be able to beat the baseline. Nevertheless, due to the small test dataset with only 37 data points, the results are prone to fluctuation.

Table 2: Accuracies of the DeepSleep-based models on the dataset providing by the University of Gdańsk. The baseline of always predicting the most represented class label in the training set is 18.23 per cent.

Gdańsk (DeepSleep)

Accuracy	**unweighted**	**weighted**	**oversampled**
Regression / Complete			
Training	13.89%		18.52%
Validation	11.11%		8.33%
Testing	29.73%		16.22%
Regression / Constant			
Training	16.67%		14.81%
Validation	16.67%		11.11%
Testing	**32.43**%		18.92%
Classification / Complete			
Training	23.19%	22.22%	35.19%
Validation	11.11%	13.89%	16.67%
Testing	10.81%	10.81%	16.22%
Classification / Constant			
Training	22.22%	13.89%	23.15%
Validation	13.89%	11.11%	19.44%
Testing	10.81%	5.41%	13.51%

The results for the DeepSleep model can be found in table 2. DeepSleep seems to perform better when applying regression. The model working on the raw, unweighted data works best and yields in an overall accuracy of 32.43 per cent, which is the third-best model on this dataset. For the task of classification, independently of the methodology, the DeepSleep-based model performs worse than the baseline. When oversampling the data, the accuracies for classification increase, but still remain below the baseline accuracy. For DeepSleep-based models performing regression, the accuracies de-

crease when oversampling the data. Generally, it can be seen that the DeepSleep-based models performs worse compared to the feature-based methods on the Gdańsk dataset.

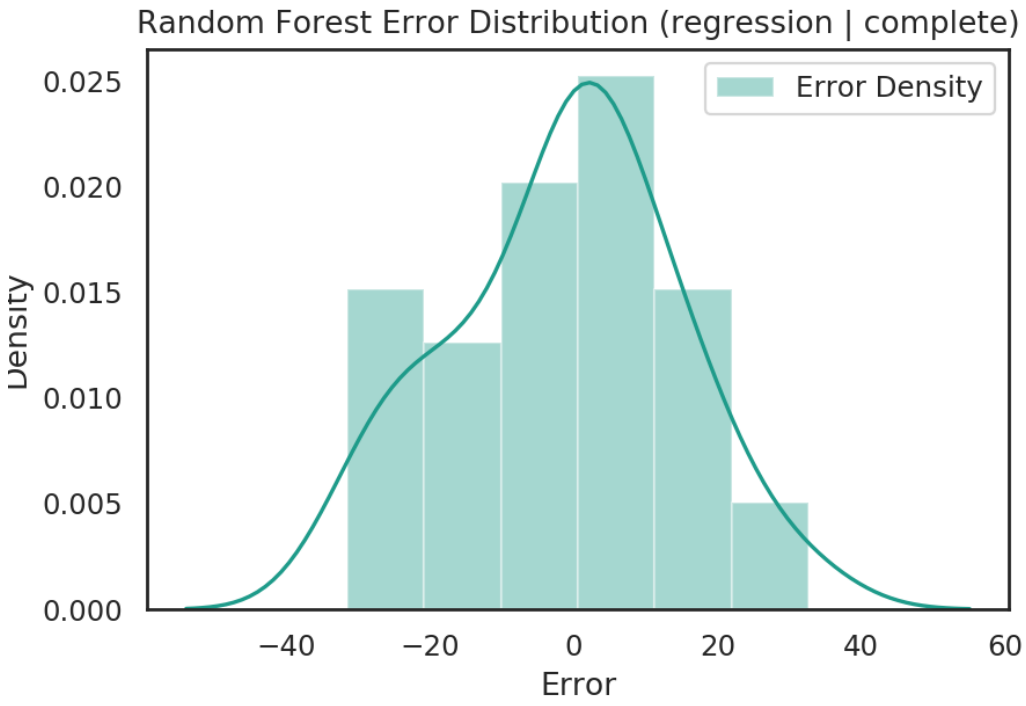


Figure 26: **Error distribution** for Random Forest performing complete regression.

Figure 26 shows the error distribution for the Random Forest performing complete regression. It can be seen that it is almost normal distributed with a mean close to 0. The variance is high, yielding in a 95 per cent prediction band spanning between -30.11 and 30.11, which is the whole age range included in the data. Nevertheless, the distribution does not show a misclassification that lies further away than 30 years from its real label. The Random Forest chosen by cross-validation consists of just 5 estimators, although having a depth of 30 splits.

5.2 Physionet

The results for the feature-based models on the Physionet dataset are shown in table 3. None of the models managed to reach a higher accuracy than the baseline with 39.14 per cent. The Naive Bayes classifiers result in very low accuracies, despite considering the prior class probabilities. Furthermore, tree-based models, like Random Forest and XGBoost, seem to perform best, although the SVM provides still quite reasonable performance. The same behaviour of seeing overfitted models generalising best can be observed again. Nevertheless, these overfitted tree-based models show the best results

Table 3: Accuracies of the feature-based models on the dataset provided by Physionet. The baseline of always predicting the most represented class label in the training dataset is 39.14 per cent.

Physionet (Feature)

Accuracy	Naive Bayes	Random Forest	SVM	XGBoost
Regression / Complete				
Training		60.45%	40.13%	42.44%
Testing		36.13%	30.32%	32.26%
Regression / Constant				
Training		46.53%	40.26%	44.93%
Testing		30.59%	29.29%	30.45%
Classification / Complete				
Training	15.35%	99.78%	28.53%	46.33%
Testing	12.90%	29.03%	25.81%	34.84%
Classification / Constant				
Training	13.46%	71.80%	32.05%	53.89%
Testing	12.39%	33.94%	24.65%	33.81%

on the test dataset.

Looking at the results for the DeepSleep-based models in table 4, only the DeepSleep-based model conducting constant classification generalises and outperforms the baseline. Using a weighted loss for classification does not seem to help for age classification, yielding in low accuracies for constant classification. Also, oversampling the training data does not seem to improve the accuracies.

Figure 27(a) and 27(b) show the learning curves for cross-entropy loss and accuracy during training of DeepSleep-based model performing constant classification. Note that the accuracy score during training is evaluated differently compared to the reported final predictions. During training, a label is considered correct, if it lies within a range of five years of the true label. The final prediction will automatically choose the lowest or highest label, even if the prediction lies further away. For example, during training, a prediction of 91 would be considered as wrong, whereas during the final classification the highest label, here from 80-90, would be chosen. The reason for this difference in accuracy evaluation enables to monitor the accuracy more precisely during training and to achieve better results during prediction. Also, the accuracy reported on the training dataset during the training of the model is evaluated while the model has dropout enabled. For the finally reported accuracy scores for all datasets, dropout is disabled. Analysing these plots, it can be noticed that both loss curves are dropping

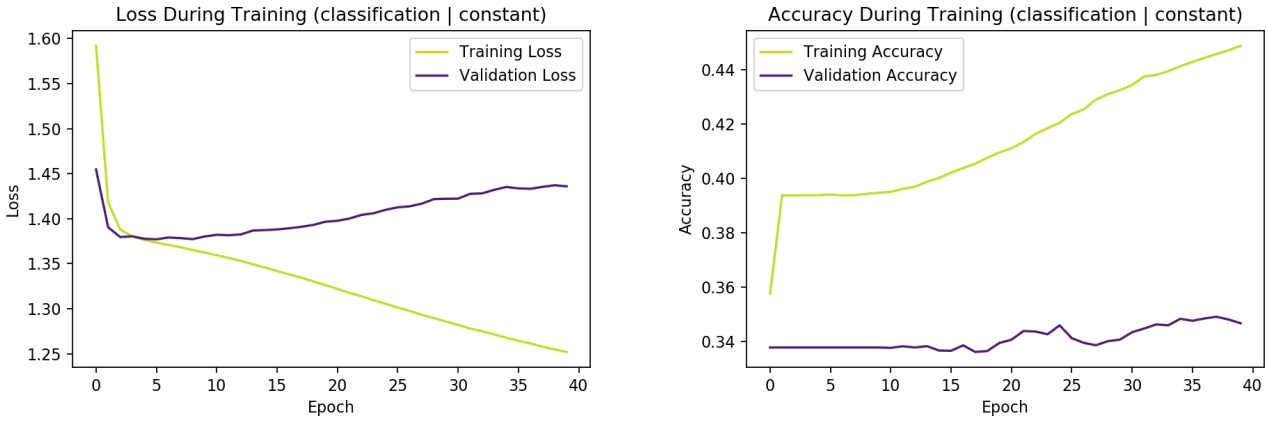
Table 4: Accuracies of the DeepSleep-based models on the dataset provided by Physionet. The baseline of always predicting the most represented class label in the training dataset is 39.14 per cent.

Physionet (DeepSleep)

Accuracy	**unweighted**	**weighted**	**oversampled**
Regression / Complete			
Training	24.07%		27.63%
Validation	28.57%		25.32%
Testing	25.16%		31.61%
Regression / Constant			
Training	38.82%		36.39%
Validation	43.51%		38.96%
Testing	33.55%		34.84%
Classification / Complete			
Training	39.47%	40.19%	29.01%
Validation	37.66%	33.77%	25.32%
Testing	38.06%	34.39%	23.87%
Classification / Constant			
Training	39.38%	21.23%	18.64%
Validation	33.77%	17.53%	19.48%
Testing	**42.58**%	16.13%	16.13%

quickly at the beginning. While the training loss continues to decrease, the validation loss starts growing again.

Overall, the model with the lowest validation loss is chosen by early stopping. Due to the larger size of the Physionet dataset, less epochs are required for a convergence of the model. This effect is even more noticeable on the oversampled dataset, where the validation accuracy shows that the model barely learns new information. Even though the accuracy increases towards the end of the training, the validation loss continuously becomes worse.



(a) Loss

(b) Accuracy

Figure 27: **Training** and **validation** loss and accuracy for DeepSleep performing standard constant classification.

5.3 Simulated Dataset

Table 5 shows the results for the different models ran on the simulated dataset.

It can be seen that the achieved accuracies for the simulated dataset are not higher compared to the original Gdańsk dataset. The models that performed below the baseline are the SVM conducting classification, the DeepSleep-based model conducting classification as well as the DeepSleep-based model running on oversampled data and conducting classification. For almost all models, the accuracies are higher when performing regression. The three highest test accuracies for the simulated dataset are marked in bold. Despite that fact that these three models still have a smaller accuracy than the best performing models on the Gdańsk dataset, the results seem to be more stable with less outliers in terms of very low accuracies.[18]

Even though an epoch of the DeepSleep-based model has a longer running time than the the epochs in the other models for the simulated data, it still results in an overall lower training time. This is due to the fact that less training epochs are needed while still achieving roughly the same results.

In regions of high variance, the data simulation, based on Gaussian processes, can sample negative values. Naive Bayes cannot handle these values, as it assumes them to be frequencies which are not allowed to be negative by definition. As the simulation returned at least one negative value, Naive Bayes is not included in the results for the simulated data.

[18] All metrics for the different models can be found in the appendix.

Table 5: Accuracies of all models except Naive Bayes on the simulated dataset. The baseline of always predicting the most represented class label in the training dataset is 18.23.

Simulated Dataset

Accuracy	**Regression / Constant**	**Classification / Constant**
Random Forest		
Training	59.89%	99.99%
Testing	21.82%	21.24%
Support Vector Machine		
Training	23.90%	33.28%
Testing	**31.16%**	15.46%
XGBoost		
Training	37.79%	99.04%
Testing	22.05%	23.94%
DeepSleep		
Training	20.94%	29.75%
Validation	18.01%	25.43%
Testing	**27.66%**	13.08%
DeepSleep (weighted)		
Training		29.26%
Validation		31.33%
Testing		19.41%
DeepSleep (oversample)		
Training	23.43%	30.75%
Validation	15.24%	26.42%
Testing	**29.51%**	17.65%

6 Discussion

This chapter will analyse and discuss the results presented in the previous chapter, with the aim to answer the originally defined research questions. Also, the limitations of the different approaches, along with the data, will be discussed. To put the results into perspective, we will compare them with [Poddar, Kumar, and Sharma 2015] and [Makowiec and Wdowczyk 2019].

6.1 Applicability of Cardiovascular Age Prediction

Comparing the results from both datasets, it seems that the models extract more information when working on the Gdańsk dataset. That follows from the difference between the best and average accuracies of the models compared to the baselines. This can be due to several reasons, though it can not be assessed which one is the actual cause. One reason could be, that the Gdańsk dataset only includes healthy patients which encodes more information about cardiovascular age. Furthermore, the dataset is close to balance compared to the dataset from Physionet. Another possible reason is, that it is easier to predict the cardiovascular age, if the RR intervals during nocturnal sleep are given. As the Physionet dataset provides recordings throughout the whole day, this could add impurity or noise to the information embedded in the nocturnal sleep stages. Additionally, it could be that the given cardiovascular diseases in the Physionet dataset have such a high impact on the models, that the embedded information about age becomes prone to the changes implied by those diseases. Further investigations would be necessary to evaluate the actual cause for the different predictable strengths of the models. Another takeaway is, that the original assumption, that regression provides more information for the model and thus provides better results, does not hold for the feature-based models. However, the DeepSleep-based model, based on gradient descent, seems to completely fail to learn on the task of classification and benefits from defining the problem as a regression problem.

When comparing the error distributions of the different models and methodologies, it can be seen that the error distributions do not differ much from each other, thus the variance seems to be roughly the same, also when comparing the distributions of both datasets with each other. This strengthens the choice of evaluating two datasets, as it seems that the underlying uncertainty remains the same, regardless of the chosen model methodology or dataset. Ignoring the fact that all these combinations represent just a sample from the real world. Inferentially, it can be that accurately predicting the cardiovascular age to a certain degree, only given the RR intervals of a subject, might not be possible. This statement still implies, that if the number of classes is considerable low (speaking of two or three), the different models and methodologies might be accurate enough to serve their purpose. Overall, that is dependent on which objectives the models are supposed to support. For the original research question, if these models can accurately predict the cardiovascular age within a range of ten years (thus age decades), the answer is, that they can not.

This leaves us with the comparison of the obtained results with other papers to en-

sure that we can be certain about the obtained results. Therefore, we will investigate and compare the obtained results of [Poddar, Kumar, and Sharma 2015] and [Makowiec and Wdowczyk 2019]. In [Poddar, Kumar, and Sharma 2015], 60 subjects, 20 for each age group (18-30, 30-45, 45-60), were classified with a maximum accuracy of 70 per cent. The obtained accuracy is quite high, but accounting for the larger age range as well as the lower amount of classes, this is in accordance with the obtained error distributions from this thesis. Moreover, out of these 60 subjects, 30 have been used for training and 30 for testing, thus representing just a small sample. [Makowiec and Wdowczyk 2019] conduct a more thorough analysis of the problem: The reported *very high* accuracies of around 96 per cent are for the training set only, as no validation or test set has been used. This becomes evident when investigating figure 8 in the paper, where the reported results are clearly stated for the whole dataset (the Gdańsk dataset), with 181 subjects and respective subsets for the different age decades, always covering all samples. This implies that the reported accuracies are training accuracies and might be severely overfitted to the training data. Delving deeper into their results, chapter 3.4 states that they used a SVM with a Gaussian kernel, where $\gamma = 0.2$ and $C = 1.0$. A high value of C enforces the SVM to choose the smallest error margins possible, trying to prevent outliers at all cost. This does not necessarily mean that a SVM with a high C overfits to its training data per se, but makes it more likely, as the decision boundary is more flexible to account for the training points. If we compare their value of C with the ones that we found using cross-validation, this problem becomes more eminent: $C = [0.3, 0.5, 0.5, 1.0, 0.6, 0.1, 0.5, 0.1]$. Out of 8 differently trained SVM models, only one chose $C = 1.0$, all of the others clearly chose lower values for C. The model that chose $C = 1.0$ is the SVM conducting constant classification on the Gdańsk dataset, with a test accuracy of 24.34 per cent. Additionally, the used metrics for validation, which consists of shuffling the time series, is questionable, especially as a simple method as hold-out data could have been used. We conclude from that, that the reported results provided by [ibid.] do not allow for any conclusions about the generalisation of their reported models. Taking the results from both papers into account, the obtained results seem to be reasonable.

The simulation of the data requires a lot of computational resources. The intention was to provide the models with more various data and thereby support the model to generalise better, which ultimately should yield in better results. The results for the models, ran on the simulated data, do not provide better accuracies, but seem to solve the problem that some methodologies resulted in accuracies way below the baseline. As already stated, the DeepSleep model takes less epochs and therefore also less time to train. Nevertheless, this does not seem to be worth the time required for simulating the data. It indeed seems that the effort of complex data simulation does not provide better results, at least for this problem. Considering that the accuracies are quite low in general, the room for improvements is small.

6.2 Theoretical Analysis

One interesting observation is that some feature-based models that have been selected by cross-validation seem to have a large gap between training and test accuracy. This applies specifically to the tree-based methods Random Forest and XGBoost. This finding is especially unusual for XGBoost as it, in addition to cross-validation, also uses regularisation as can be seen in equation 20. Considering this, there are to matter to think about. Firstly, do these models overfit? That depends foremost on the definition of an overfitted model. Is a model that generalises best compared to other models, but still has a large difference in train and test accuracy, considered overfitted? The answer most likely depends. As none of the models achieved a high accuracy, we can safely conclude that these models most likely overfitted, despite having the best set of hyperparameters determined by cross-validation.

Inspecting loss and accuracy of the DeepSleep based model performing complete classification on the Gdańsk dataset in figure 44(a) and figure 44(b) respectively, it can be observed that the loss immediately becomes worse with the second epoch while the accuracy remains roughly the same throughout the training of the model. Looking at the loss in figure 52(a) and the corresponding accuracy in figure 52(b) for the same model again conducting weighted, constant classification, it can be seen that the loss becomes worse as well but this time the accuracy seems so slightly increase each epoch. This is an odd finding and raises the question, if the loss function, in this case, cross-entropy, is appropriate as the accuracy becomes better while the loss becomes worse. If we assume the problem to be a regression, for example in figure 62(a) which shows the loss and figure 62(b) the accuracy, the curves make more sense. One things which remains interesting is the *plateau* around epoch 60 where the loss and the lowest and the accuracy the highest. In this setting, the loss function actually represents the accuracy well. The DeepSleep based model seems to perform better for a regression loss and the behaviour seems to be more natural. Nevertheless, this does not yet explain why this is not the case for the feature-based models.

It seems that choosing an LSTM does not only enable the model to learn long-term dependencies, but also enforces the usage of a loss that accurately represents the embedded information on a continuous spectrum. It might be of interest to investigate this further, based on an easier task such that is easier to track which settings cause which behaviour. In the current setting, it is not possible to spot the unique cause of the observed behaviours.

6.3 Deep Learning in Perspective

Opting for deep learning models is not always the correct path to follow, as they do not necessarily provide better results. On the other hand, feature-based models have the problem, that they do not scale well. Especially SVMs require a lot of training time, as they involve solving a quadratic programming problem, which adds constraints to an equation to solve. Also taking hyperparameter search and prediction into account, feature-based methods can be very slow. All feature-based models in this thesis were

run only on a small subset of the available data, withholding some of the information that would be available for the models. Deep learning models themselves also take their time for fitting, but usually deep learning models scale quite well in a way that they can be parallelised on GPUs, resulting in lower run times. It is not very common to do extensive grid hyperparameter search for deep learning models, hence some time must be spent on choosing the right hyperparameters as well as the right model architecture. Contrary to deep learning approaches, feature-based models require manually constructed features, which are not that straight forward to construct for time series data. In the end, the models to choose always depend on the problem.

6.4 Result Evaluation

We know from other sources, that there is some information embedded in the RR signals, which includes disease or nocturnal sleep detection. However, it seems like the age of a person is not embedded into the signal. The DeepSleep model is rather complex, still, it does not perform better compared to the simple feature-based models. Usually, complex deep learning models accomplish at least the same level of results for a given problem set, which is not the case here. Also, considering that weighting the classes, as well as oversampling the training dataset, yields in worse results, the question is raised, if the models that achieved high accuracies actually generalised or if they simply learned the class distribution. For some feature-based models, grid search for the hyperparameters as well as cross-validation, resulted in over 2,000 models fitted. With this high amount of fits, it could have quite likely happened that one of the higher test accuracies just happened by chance for the Gdańsk dataset, as the test dataset is rather small. Considering the results from the second datasets, where more data is available, this assumption could be quite likely, as there are no models that achieved a significantly higher accuracy compared to the baseline. Unfortunately, there is no simple way of proofing this assumption.

It is hard to confidently explain these low accuracies. Given all these different methodologies and models, it is quite unlikely that one mistake holds for all of these different approaches. Especially, as the obtained results are consistent with results provided by other papers. Therefore, it seems unlikely that a discrepancy remains uncovered. We conclude from this, that the desired information is just not available in RR intervals. Assuming this hypothesis to be true, it raises the questions if there is another way of improving the results. One way could be to simply provide more information to the models. The original problem was to be able to label data in the medical field, where the quality of the data is not assumed to be good. Presumably, we do not expect a lot more meta-information about the subjects. Nevertheless, what might be available is not the RR intervals, but the raw underlying data as it is shown in figure 2. If that is the case, further analysis could be conducted on this data to see if that improves the results. Having a better resolution on the labels available could potentially help the models to extract more information as the current labels are quite blurry as they span a time range of ten years.

Considering all this, the true purpose of labelling new data should not be forgot-

ten. Hospitals have unlabelled data, so when automating the labelling, a high accuracy should be ensured, since the data will most likely be, especially considering that this data will most likely be used for clinical research and erroneous labelled data could have severe impacts. It is up for debate, which accuracy or error distribution would be acceptable, but no matter how it would be defined, the results obtained in this thesis indicate, that they are still far from being applicable in a practical context. Even if one manages to increase the accuracies to 80 per cent for the given datasets, it is questionable if that would be enough for applying these models in the required context.

Concludingly, the results of this thesis show, that the accuracies for cardiovascular age prediction are not yet sufficient for any real-world application.

7 Conclusion

The main takeaway from this thesis is, that cardiovascular age prediction, based on RR intervals is a difficult problem. We have seen other research indicating high accuracies, which themselves do not provide a good estimate for how well a model generalises. The aim of developing models should always be, to make them generalise to unseen data. To get an estimate of its general predictive strength, at least held-out data should be used. A better estimate could be obtained when the actual test data is coming from a different context, to prevent a bias towards the given dataset. Therefore, to create a model which performs well on various types of RR intervals, one should also train this model on different sources of RR intervals. This can for example include recordings of people doing sports, sleeping or having diseases. However, acquiring a dataset that embeds all of this information, is a challenging task in itself.

We can also conclude, that for time series processing, linear spline interpolation works rather well compared to other methods, especially considering how simple this approach of handling impurity is. It remains unsolved to which extend this applies to other time series data. Considering the low accuracies obtained, it is also hard to tell how good $\mathcal{GP}$s would work in a context where the desired information is easier accessible for the models. Still, $\mathcal{GP}$s have the benefit of providing a distribution for sampling as well as the ability to be evaluated at any predictive point. In the end, they add variability to the existing data, but are unfortunately limited to smaller slices due to their computational complexity. In a Bayesian setting, some information can flow between the slices. Ultimately, it would be of interest to consider the complete time series as well as all of them at the same time for fitting the $\mathcal{GP}$. At the moment, this is computationally infeasible, but frameworks such as TensorFlow Probability enable GPU acceleration and thus provide a possibility to use these kinds of methods within new fields.

To continue the work on this task of age prediction, the most promising directions are to either gather more relevant meta-information or more detailed time series including QQ-intervals or the raw input signal of an ECG. Also, constructing new features could be a way to improve the existing models rather than trying to tweak the existing complex models.

8 References

FM, Fesmire and MacMath TL (1988). "The ECG in Acute Myocardial Infarction". In: *The Journal of Emergency Medicine*. DOI: 10.1016/0736-4679(88)90015-7.

Miller, Wayne, Janet Wallace, and Karen Eggert (1993). "Predicting max HR and the HRV Relationship for Exercise Prescription in Obesity". In: *Medicine and Science in Sports and Exercise*.

Inbar, Omri et al. (1994). "Normal Cardiopulmonary Responses During Incremental Exercise in 20- to 70-yr-old Men". In: *Medicine and science in sports and exercise*. DOI: 10.1249/00005768-199405000-00003.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Comput.* DOI: 10.1162/neco.1997.9.8.1735.

Goldberger, Ary L. et al. (2000). "PhysioBank, PhysioToolkit, and PhysioNet". In: *Circulation*. DOI: 10.1161/01.CIR.101.23.e215.

Schölkopf, Bernhard (2000). "The Kernel Trick for Distances". In: DOI: 10.5555/3008751.3008793.

Bernardo, José M. (2001). "The Concept of Exchangeability and its Applications". In: *Metrika*. DOI: 0.1007/s00184-016-0602-z.

Evgeniou, Theodoros and Massimiliano Pontil (2001). "Support Vector Machines: Theory and Applications". In: DOI: 10.1007/3-540-44673-7_12.

Tanaka, Hirofumi, Kevin D Monahan, and Douglas R Seals (2001). "Age-predicted Maximal Heart Rate Revisited". In: *Journal of the American College of Cardiology*. DOI: 10.1016/S0735-1097(00)01054-8.

Rosasco, Lorenzo et al. (2004). "Are Loss Functions All the Same?" In: *MIT Press*. DOI: 10.1162/089976604773135104.

Shawe-Taylor, John and Nello Cristianini (2004). "Properties of Kernels". In: *Cambridge University Press*. DOI: 10.1017/CBO9780511809682.004.

Zhang, Harry (2004). "The Optimality of Naive Bayes". In: DOI: 10.1134/S1054661814010088.

Rasmussen, Carl Edward and Christopher K. I. Williams (2005). "Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)". In: *The MIT Press*. DOI: 10.5555/1162254.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag. ISBN: 978-0387310732.

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze (2008). "Introduction to Information Retrieval". In: *Cambridge University Press*. DOI: 10.5555/1394399.

Gulati, Martha et al. (2010). "Heart Rate Response to Exercise Stress Testing in Asymptomatic Women". In: *Circulation*. DOI: 10.1161/CIRCULATIONAHA.110.939249.

Hoffman, Matthew D. and Andrew Gelman (2011). "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo". In: arXiv: 1111.4246 [stat.CO].

Murty, M. and V. Devi (2011). *Pattern Recognition. An Algorithmic Approach*. DOI: 10.1007/978-0-85729-495-1.

Neal, Radford M. (2012). "MCMC using Hamiltonian Dynamics". In: arXiv: 1206.1901 [stat.CO].

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2012). "On the Difficulty of Training Recurrent Neural Networks". In: arXiv: 1211.5063 [cs.LG].

Murphy, Kevin P. (2013). *Machine learning: A Probabilistic Perspective*. The MIT Press. ISBN: 978-0262018029.

Roberts, S. et al. (2013). "Gaussian Processes for Timeseries Modelling". In: *Philosophical Transactions of the Royal Society (Part A*. DOI: 10.1098/rsta.2011.0550.

Birnbaum, Yochai et al. (2014). "The Role of the ECG in Diagnosis, Risk Estimation, and Catheterization Laboratory Activation in Patients with Acute Coronary Syndromes: A Consensus Document". In: *Annals of Noninvasive Electrocardiology*. DOI: 10.1111/anec.12196.

Chen, Tianqi (2014). "Introduction to Boosted Trees". In: *Slidesheet*.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: arXiv: 1412.6980 [cs.LG].

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research*. DOI: 10.5555/2627435.2670313.

Poddar, Mohan, Vinod Kumar, and Yash Sharma (2015). "Heart Rate Variability: Analysis and Classification of Healthy Subjects for Different Age Groups". In: *Conference: INDIACom-2015, At New Delhi (INDIA)*.

Rashmi, K. V. and Ran Gilad-Bachrach (2015). "DART: Dropouts Meet Multiple Additive Regression Trees". In: arXiv: 1505.01866 [cs.LG].

Ruder, Sebastian (2016). "An Overview of Gradient Descent Optimization Algorithms". In: *CoRR*. arXiv: 1609.04747.

Shaffer, Fred and J. P. Ginsberg (2017). "An Overview of Heart Rate Variability Metrics and Norms". In: *Frontiers in Public Health*. DOI: 10.3389/fpubh.2017.00258.

Supratak, A. et al. (2017). "DeepSleepNet: A Model for Automatic Sleep Stage Scoring Based on Raw Single-Channel EEG". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering*. DOI: 10.1109/TNSRE.2017.2721116.

Zhao, B. et al. (2017). "Convolutional Neural Networks for Time Series Classification". In: *Journal of Systems Engineering and Electronics*. DOI: 10.21629/JSEE.2017.01.18.

Berg, Marten E. van den et al. (2018). "Normal Values of Corrected Heart-Rate Variability in 10-Second Electrocardiograms for All Ages". In: *Frontiers in Physiology*. DOI: 10.3389/fphys.2018.00424.

Pelletier, Charlotte, Geoffrey I. Webb, and Francois Petitjean (2018). *Temporal Convolutional Neural Network for the Classification of Satellite Image Time Series*. arXiv: 1811.10166 [cs.CV].

Ismail Fawaz, Hassan et al. (2019). "Deep Learning For Time Series Classification: A Review". In: *Data Mining and Knowledge Discovery*. DOI: 10.1007/s10618-019-00619-1.

Makowiec, Danuta and Joanna Wdowczyk (2019). "Patterns of Heart Rate Dynamics in Healthy Aging Population: Insights from Machine Learning Methods". In: *Entropy*. DOI: 10.3390/e21121206.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: arXiv: 1912.01703 [cs.LG].

Tison, Geoffrey H. et al. (2019). "Automated and Interpretable Patient ECG Profiles for Disease Detection, Tracking, and Discovery". In: *Circulation: Cardiovascular Quality and Outcomes*. DOI: 10.1161/CIRCOUTCOMES.118.005289.

Wan, Renzhuo et al. (2019). "Multivariate Temporal Convolutional Network: A Deep Neural Networks Approach for Multivariate Time Series Forecasting". In: *Electronics*. DOI: 10.3390/electronics8080876.

Zhang, Yuezhou et al. (2019). "Sleep Stage Classification Using Bidirectional LSTM in Wearable Multi-sensor Systems". In: *IEEE*. DOI: 10.1109/INFCOMW.2019.8845115.

9 Appendix

The appendix includes information about the Bayesian data simulation as well as all metrics and hyperparameters associated to all models found in this thesis.

9.1 Posterior Kernel Distributions with Heatmaps

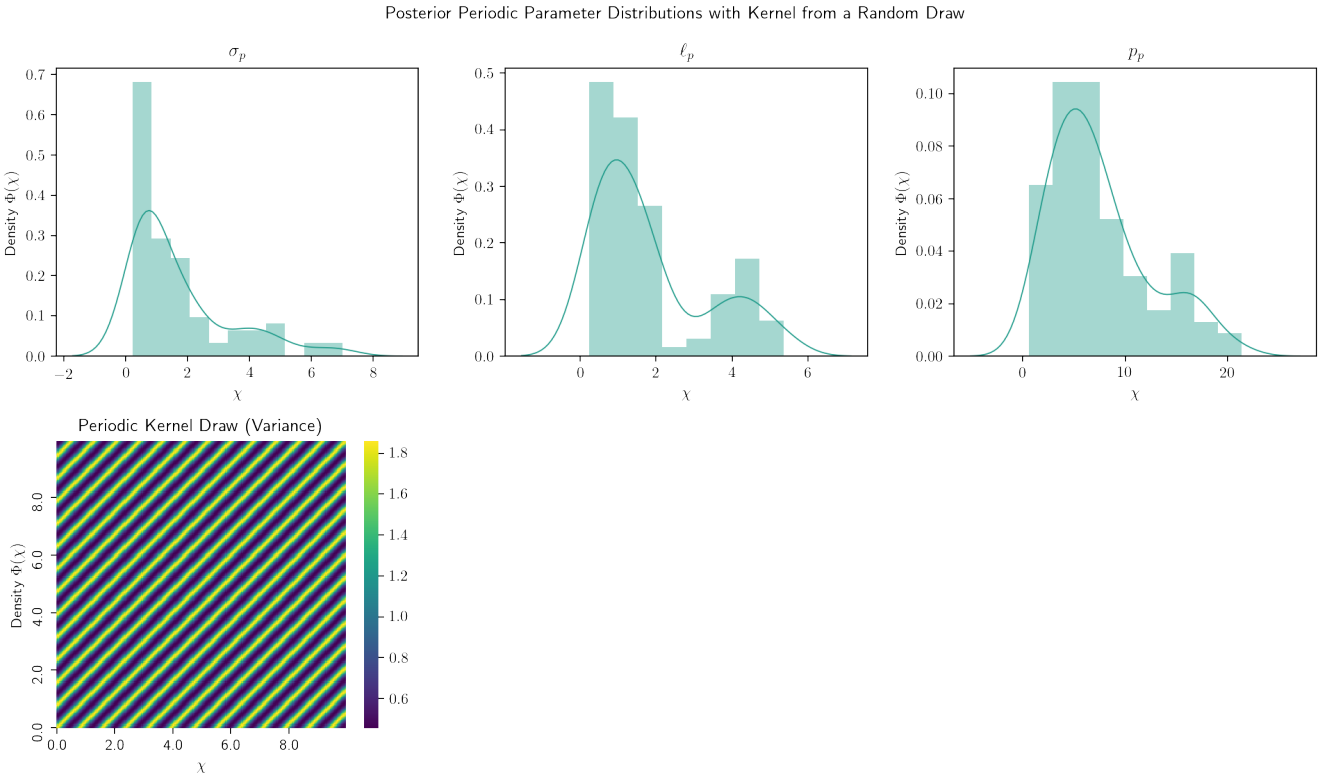


Figure 28: **Posterior Distributions** for the periodic kernel with example draw for the kernel.

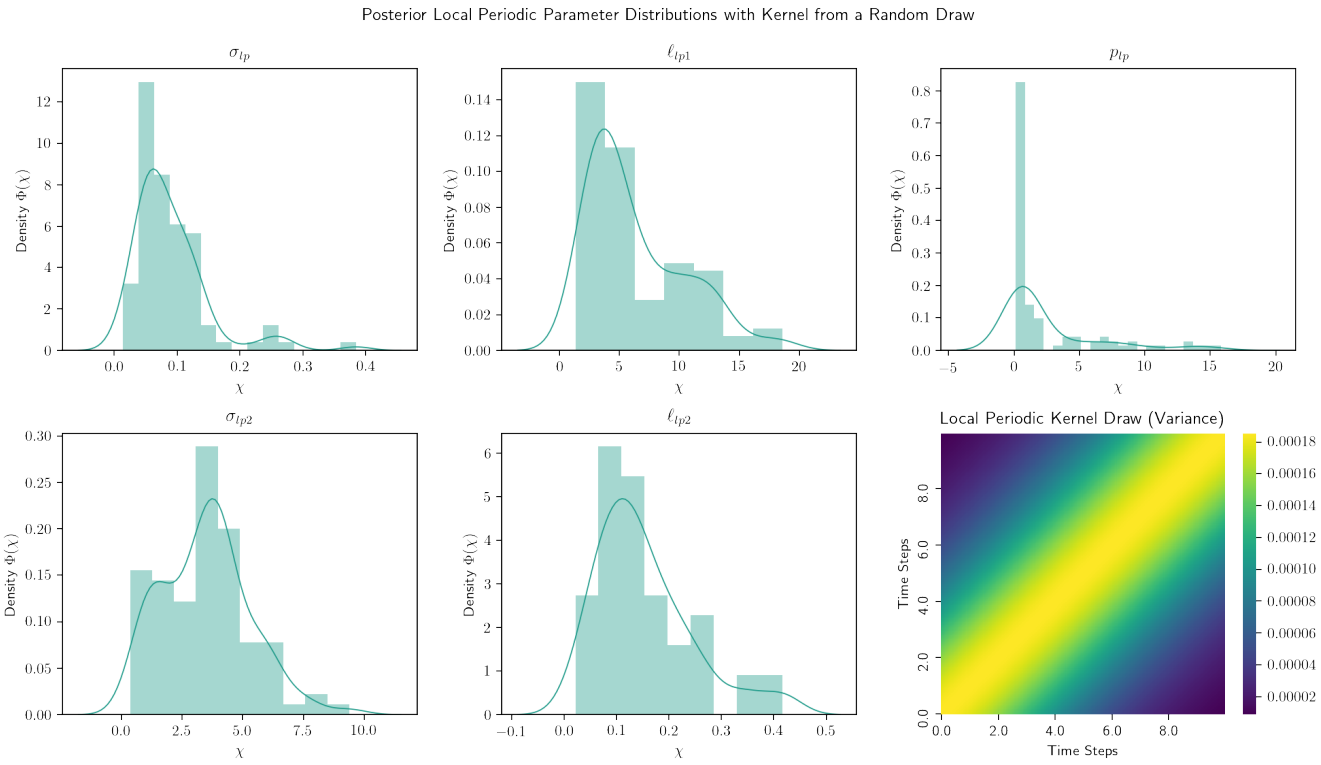


Figure 29: **Posterior Distributions** for the local periodic kernel with example draw for the kernel.

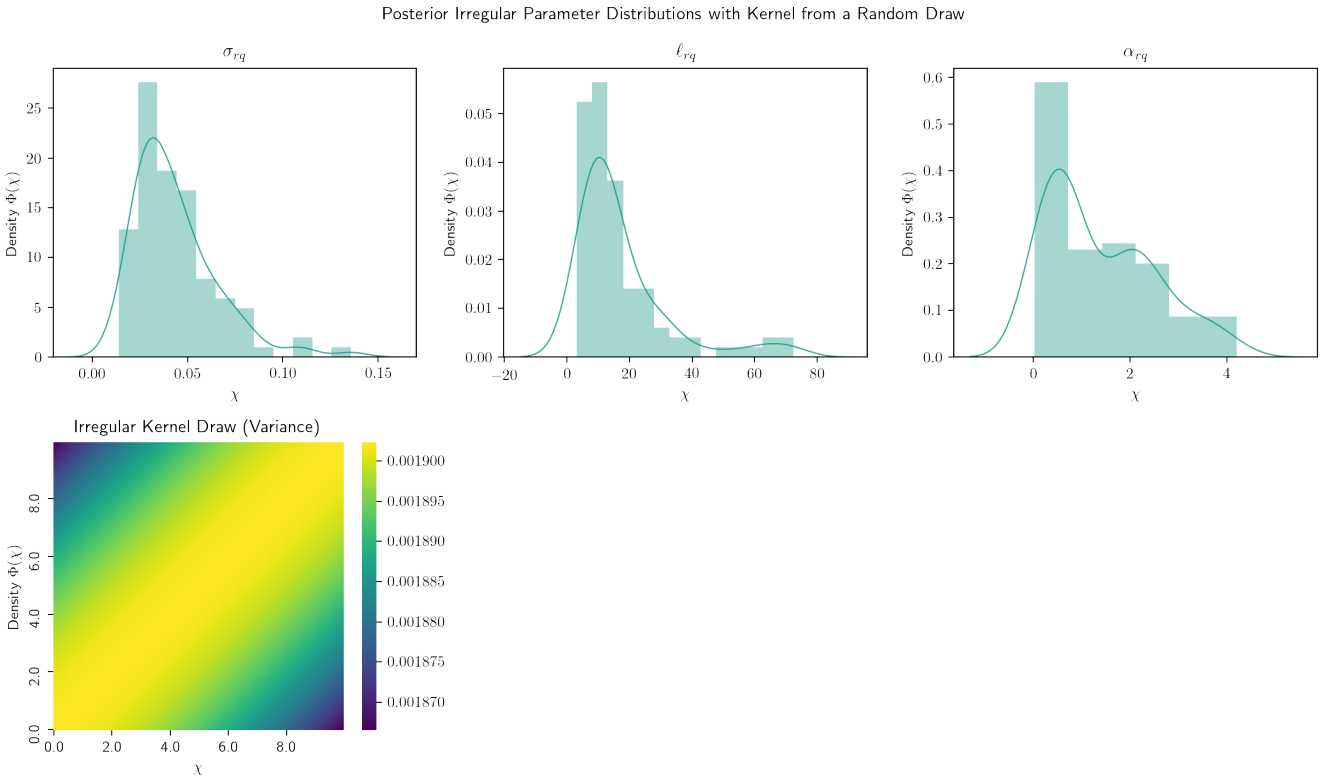


Figure 30: **Posterior Distributions** for the irregular kernel with example draw for the kernel.

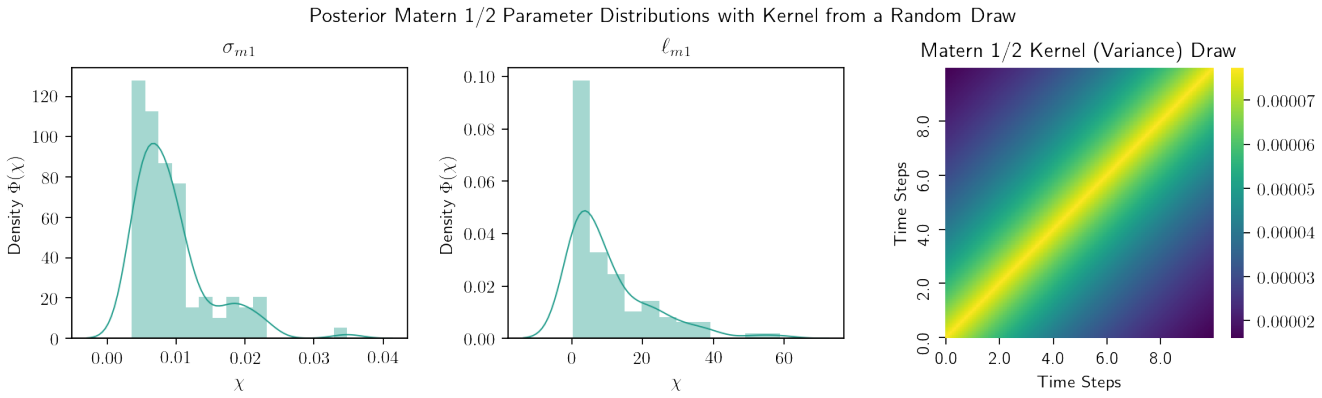


Figure 31: **Posterior Distributions** for the Matérn 1/2 kernel with example draw for the kernel.

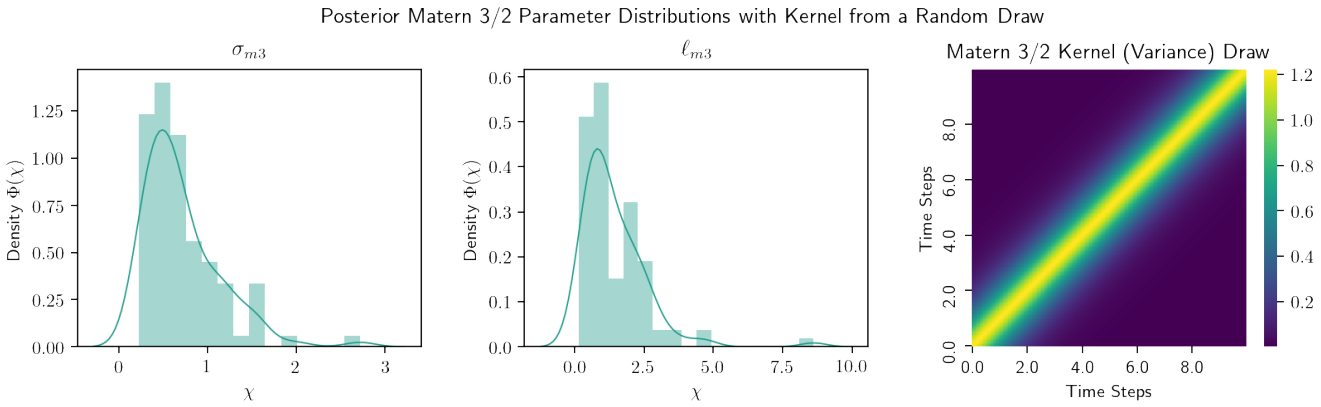


Figure 32: **Posterior Distributions** for the Matérn 3/2 kernel with example draw for the kernel.

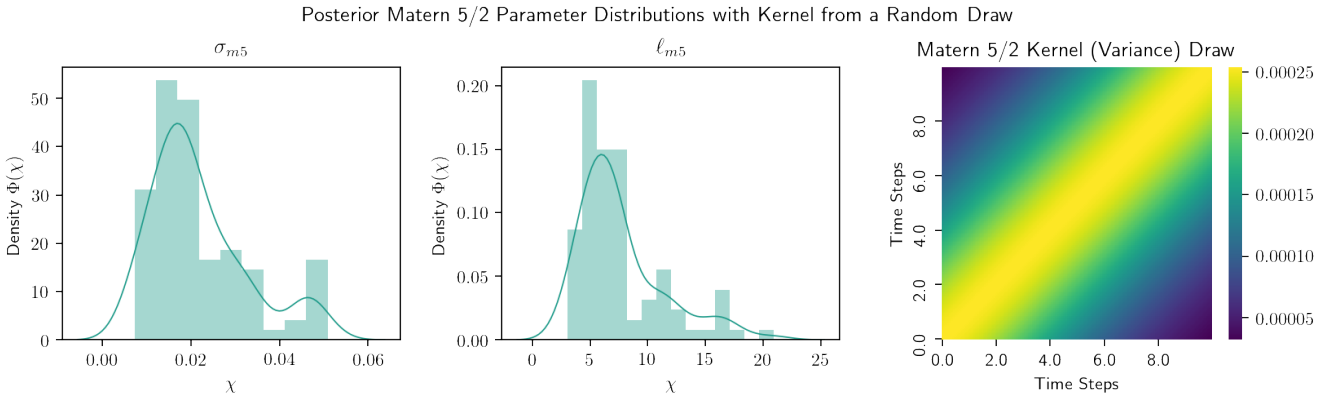
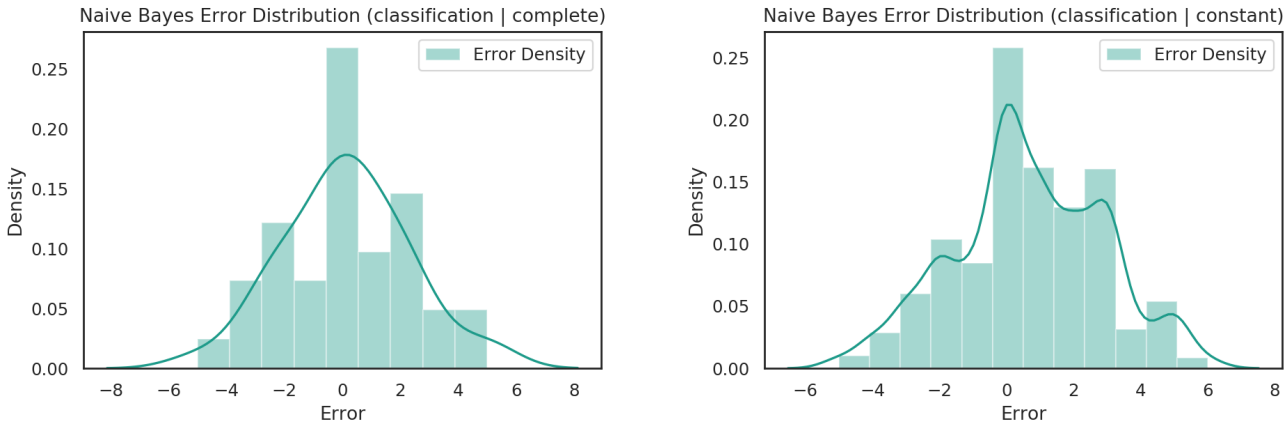


Figure 33: **Posterior Distributions** for the Matérn 5/2 kernel with example draw for the kernel.

9.2 Gdańsk: Naive Bayes

9.2.1 Error Distributions Classification



(a) Complete (b) Constant

Figure 34: **Error distributions** for Naive Bayes performing classification.

9.2.2 Classification Metrics (Complete)

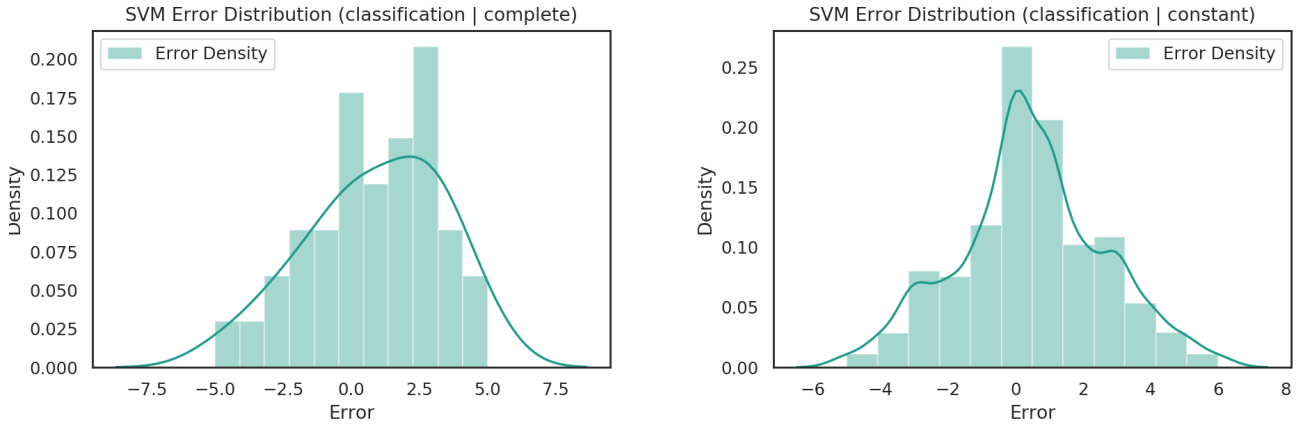
```
Accuracy Train: 0.3263888888888889
Accuracy Test: 0.2972972972972973

Max Size CV: Taken -> 144, cap -> 80000, available -> 144.
Max Size Fit: Taken -> 144, cap -> 160000, available -> 144.
Best Parameters: {'classifier__alpha': 0.1}
```

9.2.3 Classification Metrics (Constant)

```
Accuracy Train: 0.2152777777777778
Accuracy Test: 0.2702702702702703

Max Size CV: Taken -> 6912, cap -> 80000, available -> 6912.
Max Size Fit: Taken -> 6912, cap -> 160000, available -> 6912.
Best Parameters: {'classifier__alpha': 0.1}
```

9.3 Gdańsk: Support Vector Machine

9.3.1 Error Distribution Classification



(a) Complete (b) Constant

Figure 35: **Error distributions** for SVM performing classification.

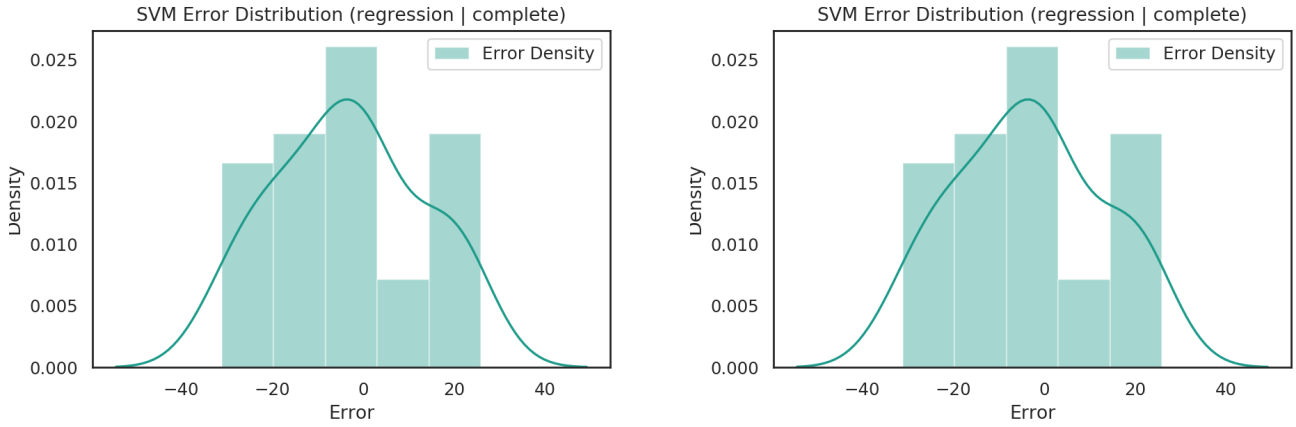
9.3.2 Classification Metrics (Complete)

```
Accuracy Train: 0.2638888888888889
Accuracy Test: 0.16216216216216217

Max Size CV: Taken -> 144, cap -> 20000, available -> 144.
Max Size Fit: Taken -> 144, cap -> 40000, available -> 144.
Best Parameter: {'classifier__C': 0.6}
```

9.3.3 Classification Metrics (Constant)

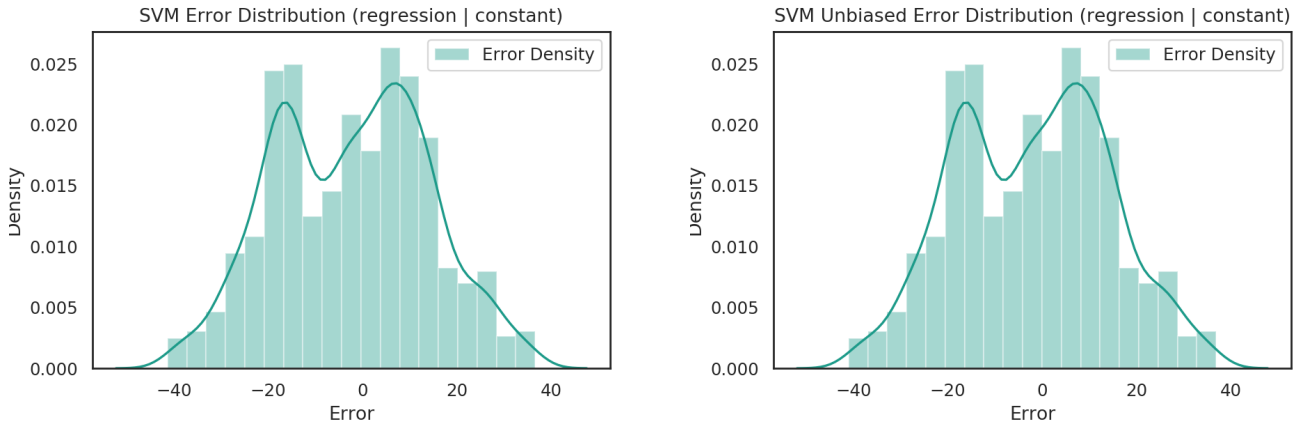
```
Accuracy Train: 0.3472222222222222
Accuracy Test: 0.24324324324324326

Max Size CV: Taken -> 6912, cap -> 20000, available -> 6912.
Max Size Fit: Taken -> 6912, cap -> 40000, available -> 6912.
Best Parameter: {'classifier__C': 1.0}
```

9.3.4 Error Distribution Regression



(a) Biased (b) Unbiased

Figure 36: **Error distributions** for SVM performing complete regression.



(a) Biased (b) Unbiased

Figure 37: **Error distributions** for SVM performing constant regression.

9.3.5 Metrics (Complete)

```
Error Train: 17.160711116087146
Error Test: 16.26820686484959
MSE Train: 294.49000600979696
MSE Test: 264.65455459753923
Accuracy Train: 0.1944444444444445
Accuracy Test: 0.2972972972972973
```

```
Estimated Bias: -1.5455638713158324

Unbiased Error Test: 16.022632086202712
Unbiased MSE Test: 256.72473896981273
Unbiased Accuracy Test: 0.2972972972972973

Max Size CV: Taken -> 144, cap -> 20000, available -> 144.
Max Size Fit: Taken -> 144, cap -> 40000, available -> 144.
Best Parameter: {'classifier__C': 0.5, 'classifier__epsilon':
   ↪ 1.0}



# Estimates
Unbiased Mean:   -1.7925651764531396
Standard Deviation:  15.922042866980991
95% Prediction Interval: -32.99976919573588 to
   ↪ 29.414638842829604

# Estimate from percentiles
95% Prediction Interval: -30.55170860972112 to
   ↪ 22.723823121338523

# Pure PI
95% Prediction Interval: -31.20720401928274 to
   ↪ 31.20720401928274
```

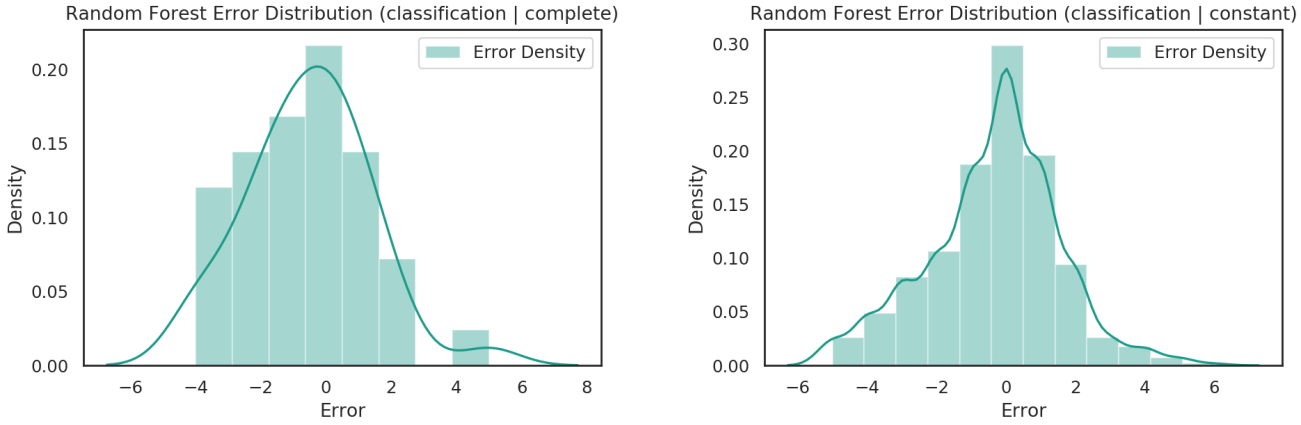
9.3.6 Metrics (Constant)

```
Error Train: 16.832966740617934
Error Test: 16.39323180283274
MSE Train: 283.34876929074954
MSE Test: 268.7380489414068
Accuracy Train: 0.22916666666666666
Accuracy Test: 0.2702702702702703

Estimated Bias: -0.3005000713840105

Unbiased Error Test: 16.35350795683985
Unbiased MSE Test: 267.4372224944242
Unbiased Accuracy Test: 0.24324324324324326

Max Size CV: Taken -> 6912, cap -> 20000, available -> 6912.
Max Size Fit: Taken -> 6912, cap -> 40000, available -> 6912.
```

```
Best Parameter: {'classifier__C': 0.5, 'classifier__epsilon':
    ↪ 10.0}


# Estimates
Unbiased Mean:   -2.0141861339758655
Standard Deviation:  16.228994938446544
95% Prediction Interval: -33.82301621333109 to
    ↪ 29.79464394537936

# Estimate from percentiles
95% Prediction Interval: -33.30652644698348 to
    ↪ 26.106562027594265

# Pure PI
95% Prediction Interval: -31.808830079355225 to
    ↪ 31.808830079355225
```

9.4 Gdańsk: Random Forest

9.4.1 Error Distribution Classification



(a) Complete

(b) Constant

Figure 38: **Error distributions** for Random Forest performing classification.

9.4.2 Classification Metrics (Complete)

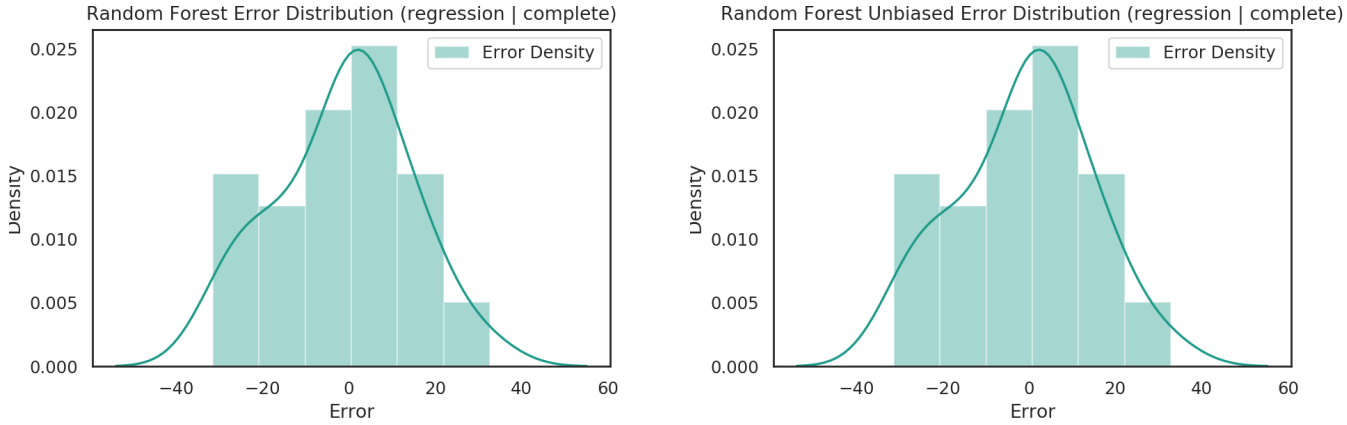
```
Accuracy Train: 0.7361111111111112
```

```
Accuracy Test: 0.24324324324324326

Max Size CV: Taken -> 144, cap -> 40000, available -> 144.
Max Size Fit: Taken -> 144, cap -> 80000, available -> 144.
Best Parameters: {'classifier__max_depth': 10, '
    ↪ classifier__min_samples_leaf': 3, '
    ↪ classifier__min_samples_split': 3, '
    ↪ classifier__n_estimators': 5}
```

9.4.3 Classification Metrics (Constant)

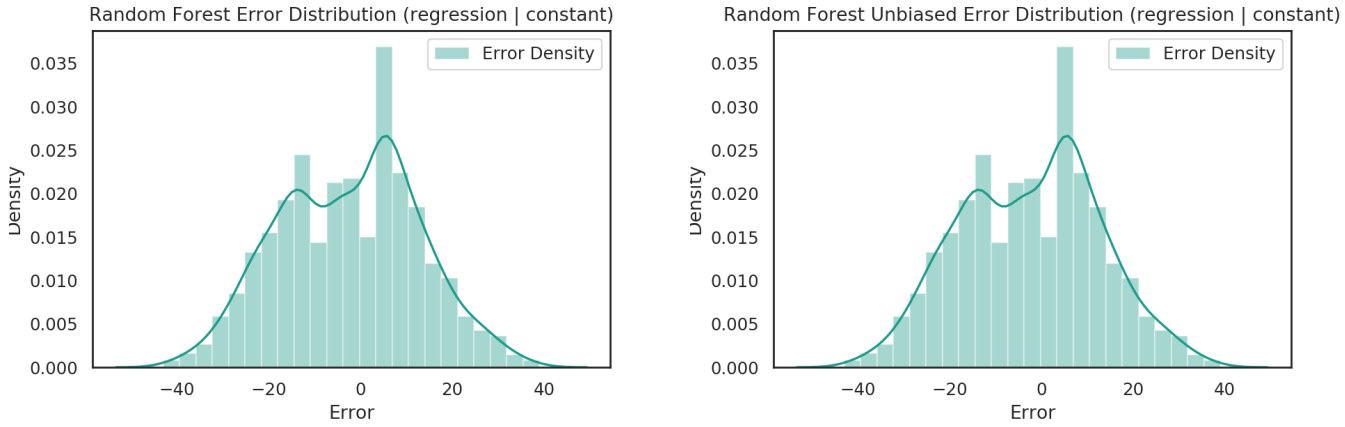
```
Accuracy Train: 1.0
Accuracy Test: 0.32432432432432434

Max Size CV: Taken -> 6912, cap -> 40000, available -> 6912.
Max Size Fit: Taken -> 6912, cap -> 80000, available -> 6912.
Best Parameters: {'classifier__max_depth': 20, '
    ↪ classifier__min_samples_leaf': 1, '
    ↪ classifier__min_samples_split': 2, '
    ↪ classifier__n_estimators': 40}
```

9.4.4 Error Distribution Regression



(a) Biased (b) Unbiased

Figure 39: **Error distributions** for Random Forest performing complete regression.



(a) Biased (b) Unbiased

Figure 40: **Error distributions** for Random Forest performing constant regression.

9.4.5 Regression Metrics (Complete)

```
Error Train: 10.472920110297297
Error Test: 15.4241929118593
MSE Train: 109.68205563666953
MSE Test: 237.90572698225068
Accuracy Train: 0.4513888888888889
Accuracy Test: 0.3783783783783784

Estimated Bias: -0.19053354053354127

Unbiased Error Test: 15.407965412447373
Unbiased MSE Test: 237.40539815117455
Unbiased Accuracy Test: 0.35135135135135137

Max Size CV: Taken -> 144, cap -> 40000, available -> 144.
Max Size Fit: Taken -> 144, cap -> 80000, available -> 144.
Best Parameters: {'classifier__max_depth': 30, '
    ↪ classifier__min_samples_leaf': 5, '
    ↪ classifier__min_samples_split': 5, '
    ↪ classifier__n_estimators': 5}



# Estimates
Unbiased Mean:    -1.217701092701095
```

```
Standard Deviation:  15.359772205342408
95% Prediction Interval: -31.32285461517221 to
   ↪ 28.887452429770022

# Estimate from percentiles
95% Prediction Interval: -30.63865106865107 to
   ↪ 24.57081696331696

# Pure PI
95% Prediction Interval: -30.105153522471117 to
   ↪ 30.105153522471117
```

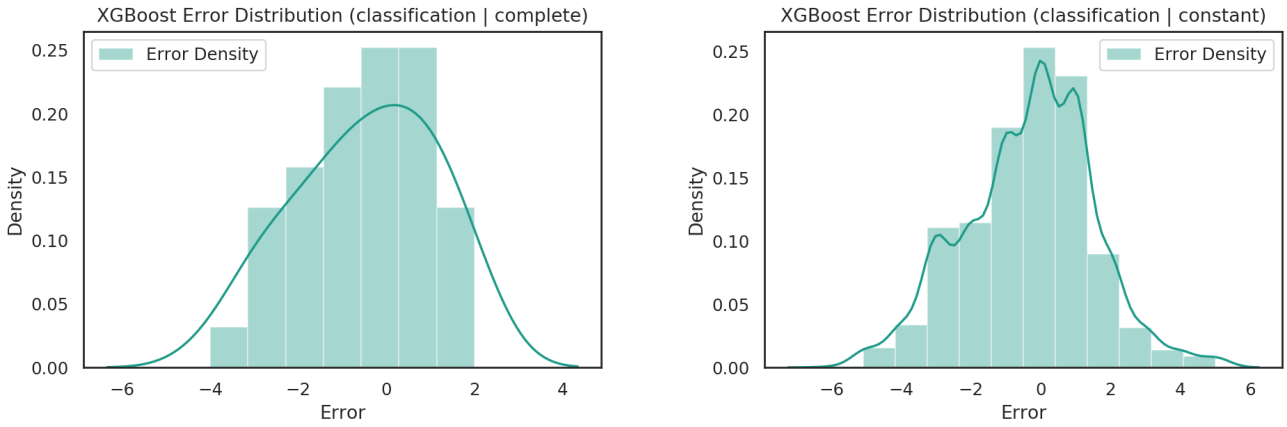
9.4.6 Regression Metrics (Constant)

```
Error Train: 14.54696989107066
Error Test: 15.471075050818163
MSE Train: 211.61433301171633
MSE Test: 239.35416322804824
Accuracy Train: 0.24305555555555555
Accuracy Test: 0.24324324324324326

Estimated Bias: -0.04382251073285072

Unbiased Error Test: 15.464305591282523
Unbiased MSE Test: 239.1447474205719
Unbiased Accuracy Test: 0.24324324324324326

Max Size CV: Taken -> 6912, cap -> 40000, available -> 6912.
Max Size Fit: Taken -> 6912, cap -> 80000, available -> 6912.
Best Parameters: {'classifier__max_depth': 5, '
   ↪ classifier__min_samples_leaf': 4, '
   ↪ classifier__min_samples_split': 5, '
   ↪ classifier__n_estimators': 15}



# Estimates
Unbiased Mean:   -2.3674521559746373
Standard Deviation:  15.28201288148073
95% Prediction Interval: -32.32019740367687 to
   ↪ 27.585293091727596

# Estimate from percentiles
95% Prediction Interval: -32.91633020706914 to
```

```
    ↪ 24.66245738661318

# Pure PI
95% Prediction Interval: -29.952745247702232 to
    ↪ 29.952745247702232
```

9.5 Gdańsk: XGBoost

9.5.1 Error Distribution Classification



(a) Complete (b) Constant

Figure 41: **Error distributions** for XGBoost performing classification.

9.5.2 Classification Metrics (Complete)

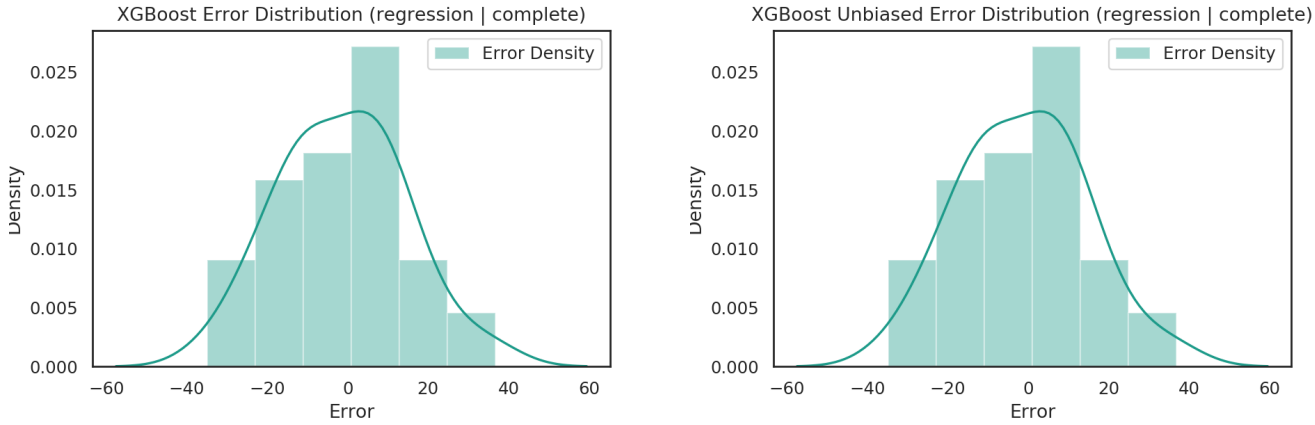
```
Accuracy Train: 0.6805555555555556
Accuracy Test: 0.21621621621621623

Max Size CV: Taken -> 144, cap -> 20000, available -> 144.
Max Size Fit: Taken -> 144, cap -> 40000, available -> 144.
Best Parameter: {'classifier__booster': 'gbtree', '
    ↪ classifier__learning_rate': 0.6, 'classifier__max_depth':
    ↪  1, 'classifier__n_estimators': 10, '
    ↪ classifier__reg_lambda': 0.6}
```

9.5.3 Classification Metrics (Constant)

```
Accuracy Train: 0.3958333333333333
Accuracy Test: 0.24324324324324326
```

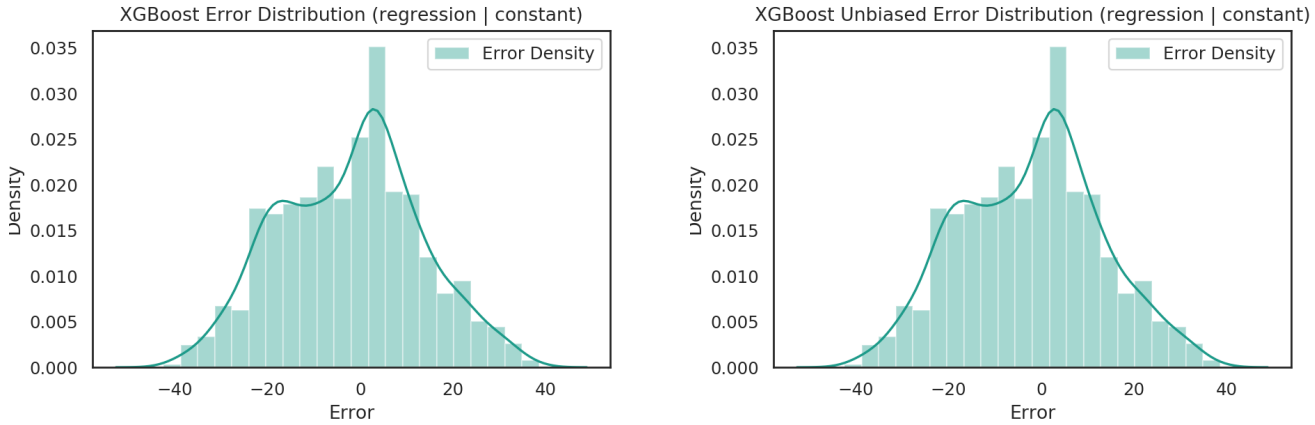
```
Max Size CV: Taken -> 6912, cap -> 20000, available -> 6912.
Max Size Fit: Taken -> 6912, cap -> 40000, available -> 6912.
Best Parameter: {'classifier__booster': 'gblinear', '
    ↪ classifier__learning_rate': 0.6, 'classifier__max_depth':
    ↪  1, 'classifier__n_estimators': 4, '
    ↪ classifier__reg_lambda': 0.3}
```

9.5.4 Error Distribution Regression



(a) Biased (b) Unbiased

Figure 42: **Error distributions** for XGBoost performing complete regression.



(a) Biased (b) Unbiased

Figure 43: **Error distributions** for XGBoost performing constant regression.

9.5.5 Regression Metrics (Complete)

```
Error Train: 12.345360484249067
Error Test: 15.653443745411101
MSE Train: 152.40792548605836
MSE Test: 245.03030109074993
Accuracy Train: 0.2847222222222222
Accuracy Test: 0.21621621621621623

Estimated Bias: -0.07732407251993816

Unbiased Error Test: 15.646663183262698
Unbiased MSE Test: 244.81806877046836
Unbiased Accuracy Test: 0.21621621621621623

Max Size CV: Taken -> 144, cap -> 20000, available -> 144.
Max Size Fit: Taken -> 144, cap -> 40000, available -> 144.
Best Parameter: {'classifier__booster': 'gblinear', '
   ↪ classifier__learning_rate': 0.6, 'classifier__max_depth':
   ↪  1, 'classifier__n_estimators': 10, '
   ↪ classifier__reg_lambda': 0.9}



# Estimates
Unbiased Mean:   -1.3336969324060388
Standard Deviation:  15.589718447199713
95% Prediction Interval: -31.889545088917476 to
   ↪ 29.2221512241054

# Estimate from percentiles
95% Prediction Interval: -28.070598024935336 to
   ↪ 27.683145337491435

# Pure PI
95% Prediction Interval: -30.555848156511438 to
   ↪ 30.555848156511438
```

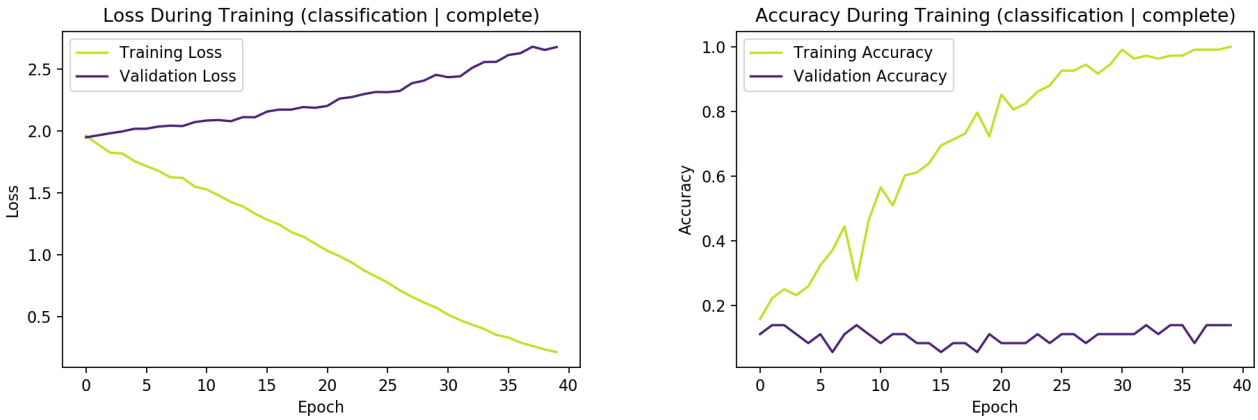
9.5.6 Regression Metrics (Constant)

```
Error Train: 15.396803789570408
Error Test: 15.39731780852079
MSE Train: 237.06156693452965
MSE Test: 237.07739569659145
Accuracy Train: 0.2222222222222222
```

```
Accuracy Test: 0.2972972972972973

Estimated Bias: -0.03635219015457012

Unbiased Error Test: 15.392136278029993
Unbiased MSE Test: 236.91785920144702
Unbiased Accuracy Test: 0.2972972972972973

Max Size CV: Taken -> 6912, cap -> 20000, available -> 6912.
Max Size Fit: Taken -> 6912, cap -> 40000, available -> 6912.
Best Parameter: {'classifier__booster': 'dart', '
   ↪ classifier__learning_rate': 0.6, 'classifier__max_depth':
   ↪  2, 'classifier__n_estimators': 8, '
   ↪ classifier__reg_lambda': 0.9}



# Estimates
Unbiased Mean:   -2.17602568596333
Standard Deviation:  15.23754479617615
95% Prediction Interval: -32.041613486468584 to
   ↪ 27.68956211454192

# Estimate from percentiles
95% Prediction Interval: -32.215483007130324 to
   ↪ 25.747036161723436

# Pure PI
95% Prediction Interval: -29.865587800505253 to
   ↪ 29.865587800505253
```

9.6 Gdańsk: DeepSleep

9.6.1 Complete Classification (standard)



(a) Loss

(b) Accuracy

Figure 44: **Training** and **validation** loss and accuracy for DeepSleep performing standard complete classification.

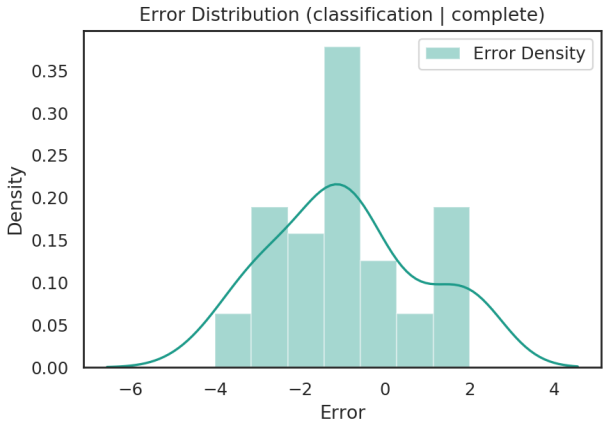
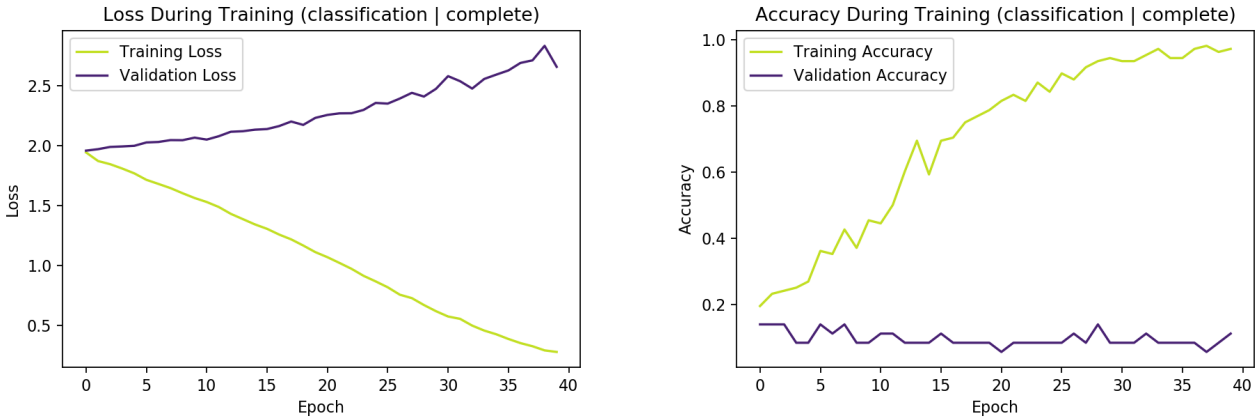


Figure 45: **Error distribution** for DeepSleep performing standard complete classification.

```
Training Accuracy: 0.23148148148148148
Validation Accuracy: 0.1111111111111111
Test Accuracy: 0.10810810810810811

Estimated Bias: -0.8333333333333334


Epochs: 40
Learning Rate: 1e-06
Batch Size: 1
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: none
```

9.6.2 Complete Classification (weighted)



(a) Loss

(b) Accuracy

Figure 46: **Training** and **validation** loss and accuracy for DeepSleep performing weighted complete classification.

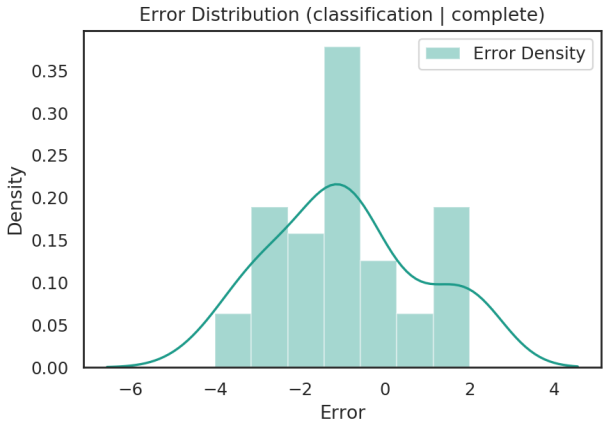
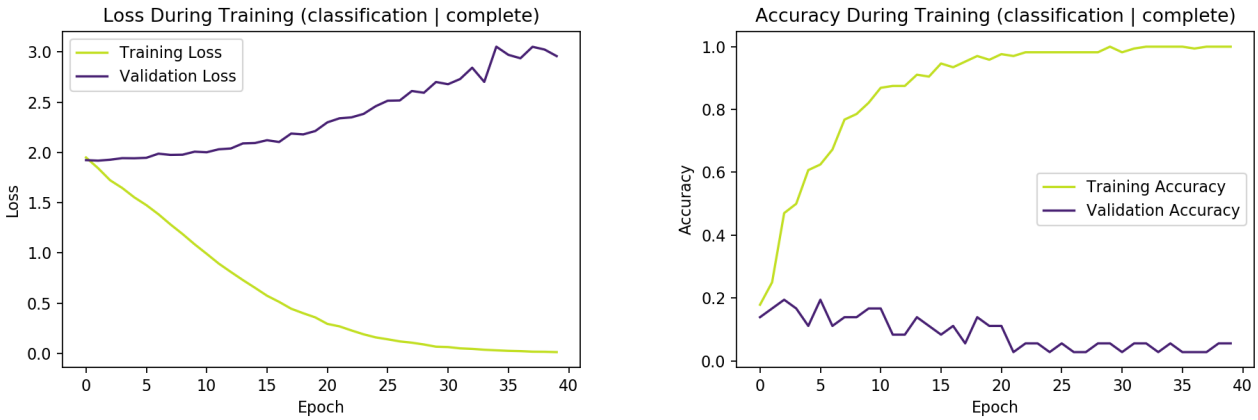


Figure 47: **Error distribution** for DeepSleep performing weighted complete classification.

```
Training Accuracy: 0.1388888888888889
Validation Accuracy: 0.1111111111111111
Test Accuracy: 0.05405405405405406

Estimated Bias: 1.4305555555555556


Epochs: 40
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: none
```

9.6.3 Complete Classification (oversampled)



(a) Loss (b) Accuracy

Figure 48: **Training** and **validation** loss and accuracy for DeepSleep performing oversampled complete classification.

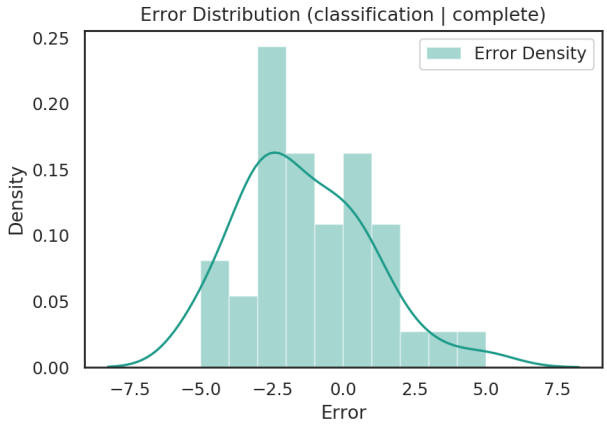
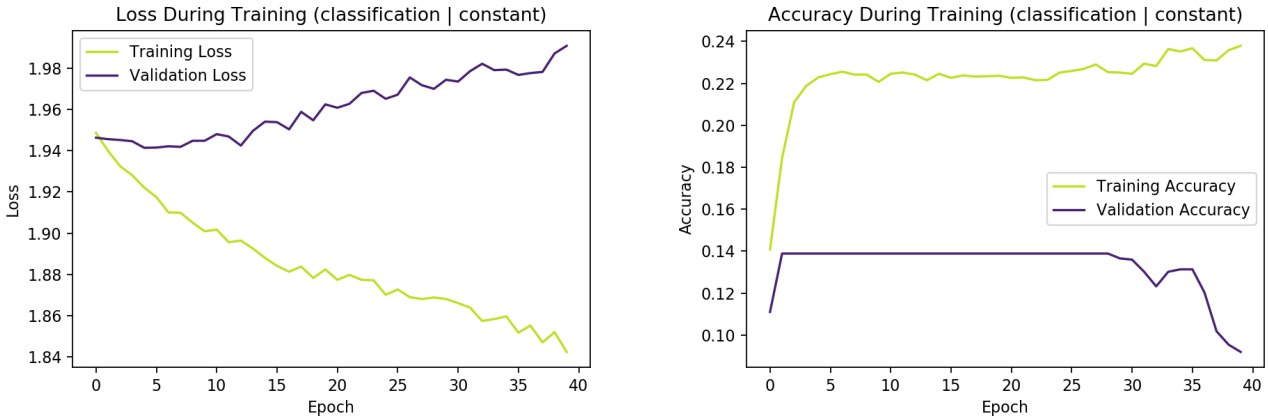


Figure 49: **Error distribution** for DeepSleep performing oversample complete classification.

```
Training Accuracy: 0.35185185185185186
Validation Accuracy: 0.16666666666666666
Test Accuracy: 0.16216216216216217

Estimated Bias: -1.3333333333333333


Epochs: 40
Learning Rate: 1e-06
Batch Size: 1
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: oversample
```

9.6.4 Constant Classification (standard)



(a) Loss

(b) Accuracy

Figure 50: **Training** and **validation** loss and accuracy for DeepSleep performing standard constant classification.

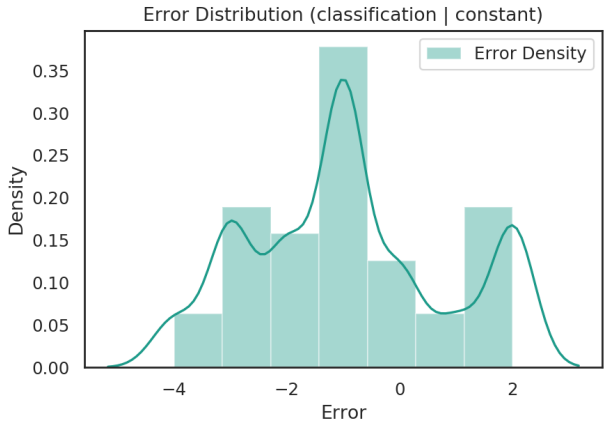
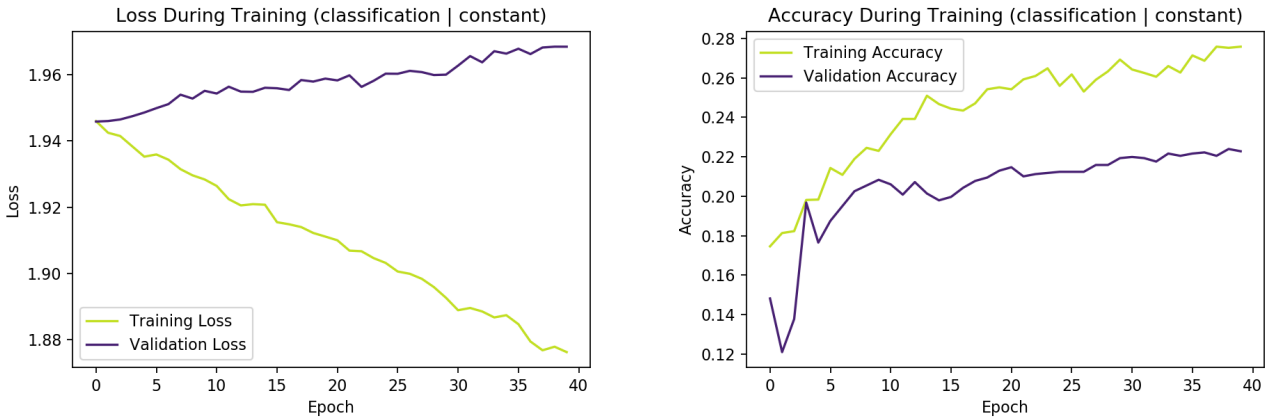


Figure 51: **Error distribution** for DeepSleep performing standard constant classification.

```
Training Accuracy: 0.2222222222222222
Validation Accuracy: 0.1388888888888889
Test Accuracy: 0.10810810810810811

Estimated Bias: -0.7777777777777778


Epochs: 40
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: none
```

9.6.5 Constant Classification (weighted)



(a) Loss

(b) Accuracy

Figure 52: **Training** and **validation** loss and accuracy for DeepSleep performing weighted constant classification.

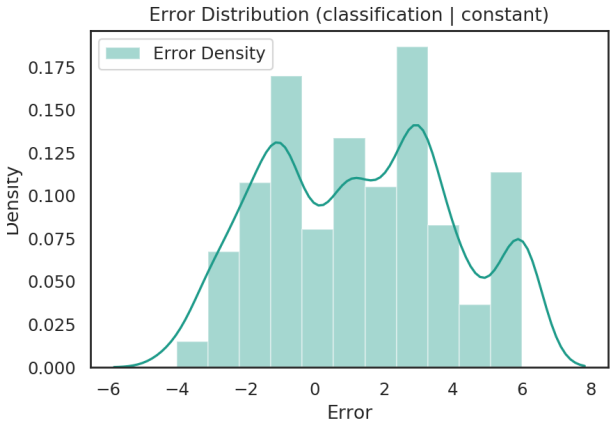
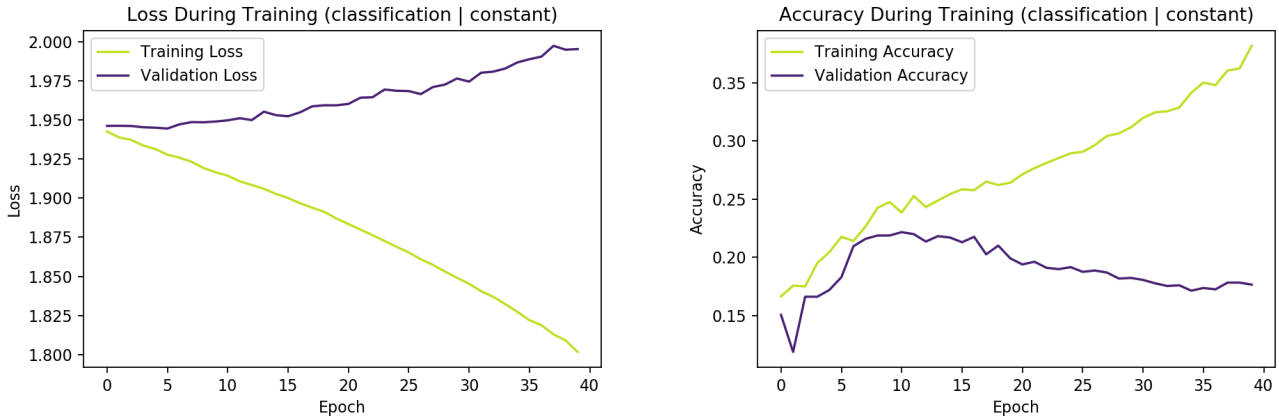


Figure 53: **Error distributions** for DeepSleep performing weighted constant classification.

```
Training Accuracy: 0.1388888888888889
Validation Accuracy: 0.1111111111111111
Test Accuracy: 0.05405405405405406

Estimated Bias: 1.4305555555555556


Epochs: 40
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: none
```

9.6.6 Constant Classification (oversampled)



(a) Loss

(b) Accuracy

Figure 54: Training and validation loss and accuracy for DeepSleep performing oversampled constant classification.

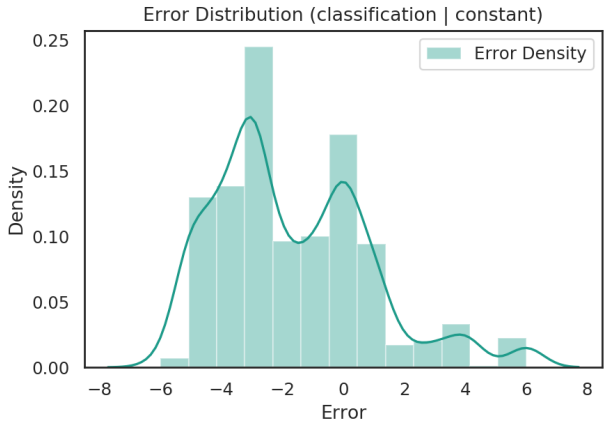
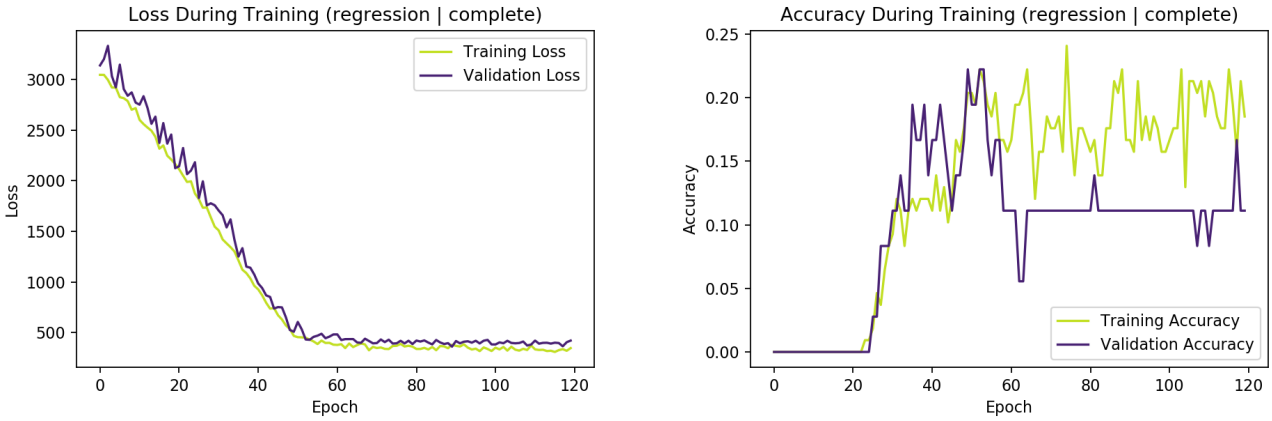


Figure 55: **Error distribution** for DeepSleep performing oversample constant classification.

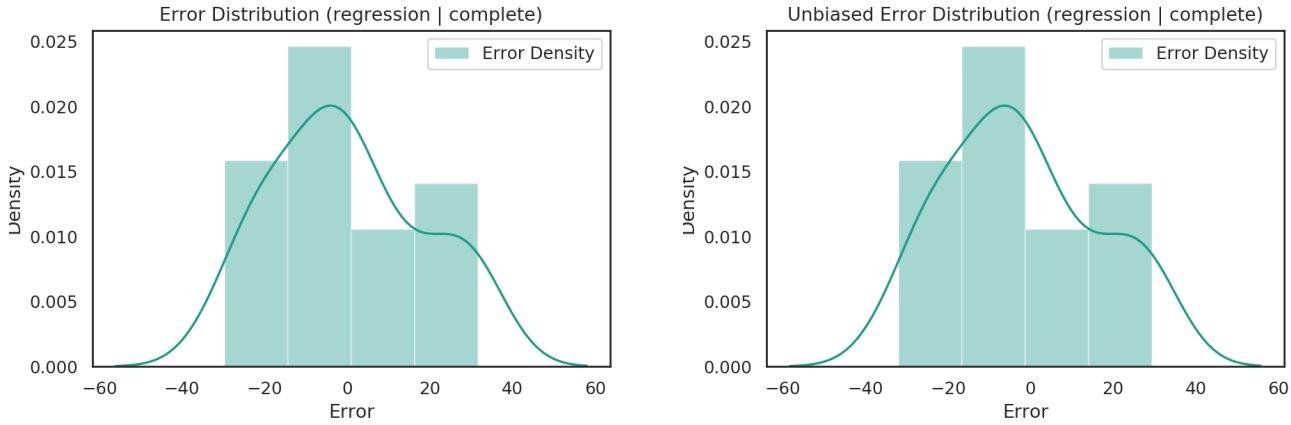
```
Training Accuracy: 0.23148148148148148
Validation Accuracy: 0.19444444444444445
Test Accuracy: 0.13513513513513514

Estimated Bias: -1.1603009259259258


Epochs: 40
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: oversample
```

9.6.7 Complete Regression (standard)



(a) Loss (b) Accuracy

Figure 56: **Training** and **validation** loss and accuracy for DeepSleep performing standard complete regression.



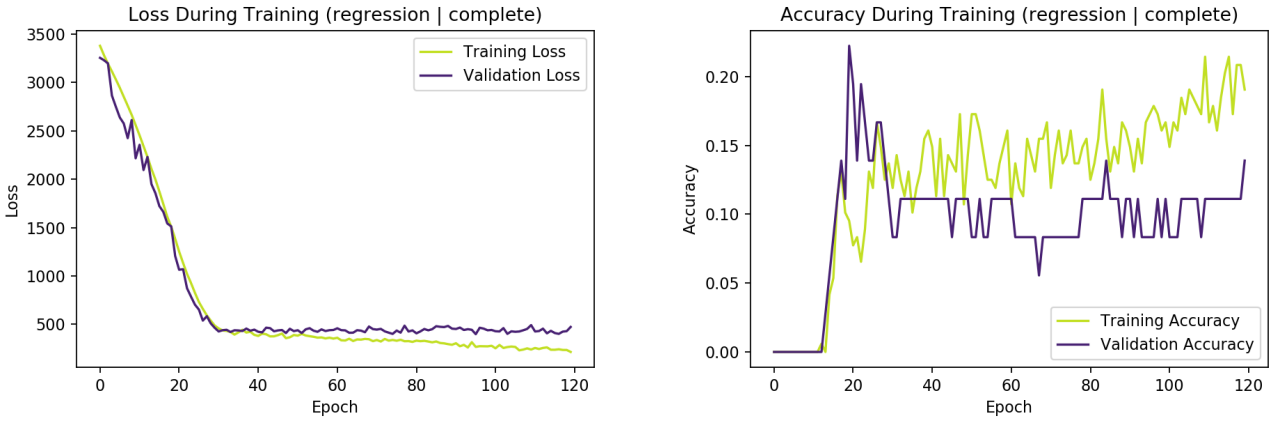
(a) Biased (b) Unbiased

Figure 57: **Error distributions** for DeepSleep performing standard complete regression.

```
Training Accuracy: 0.1388888888888889
Validation Accuracy: 0.1111111111111111
Test Accuracy: 0.2972972972972973

Estimated Bias: 2.2446789741516113

Unbiased Training Accuracy: 0.1574074074074074
Unbiased Validation Accuracy: 0.19444444444444445
```

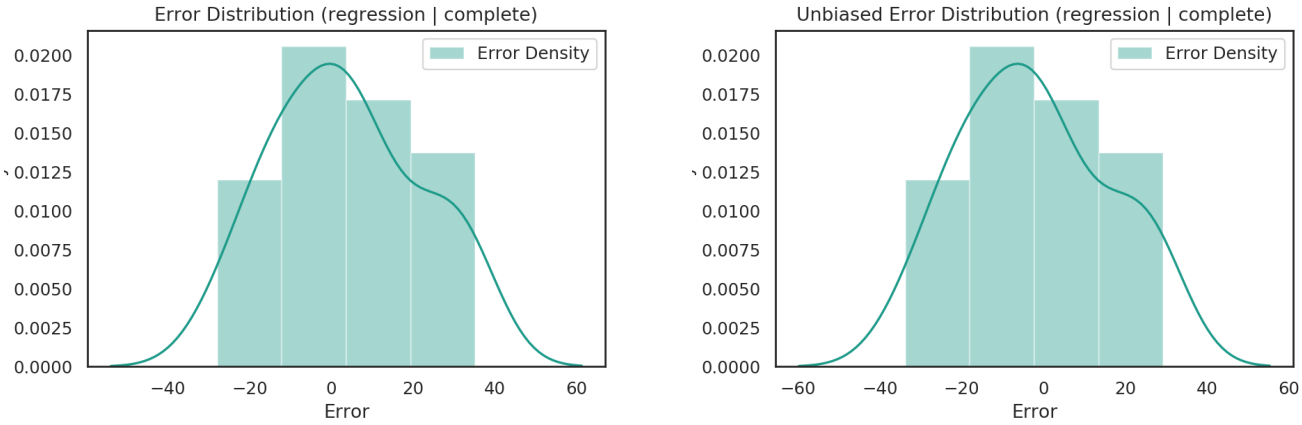
```
Unbiased Test Accuracy: 0.21621621621621623



# Estimates
Unbiased Mean:   -2.2642951011657715
Standard Deviation:  17.99371910095215
95% Prediction Interval: -37.53198453903198 to
   ↪ 33.00339433670044

# Estimate from percentiles
95% Prediction Interval: -32.59143934249878 to
   ↪ 26.39871873855591

# Pure PI
95% Prediction Interval: -35.26768943786621 to
   ↪ 35.26768943786621

Epochs: 120
Learning Rate: 1e-06
Batch Size: 8
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: none
```

9.6.8 Complete Regression (oversampled)



(a) Loss (b) Accuracy

Figure 58: **Training** and **validation** loss and accuracy for DeepSleep performing oversampled complete regression.



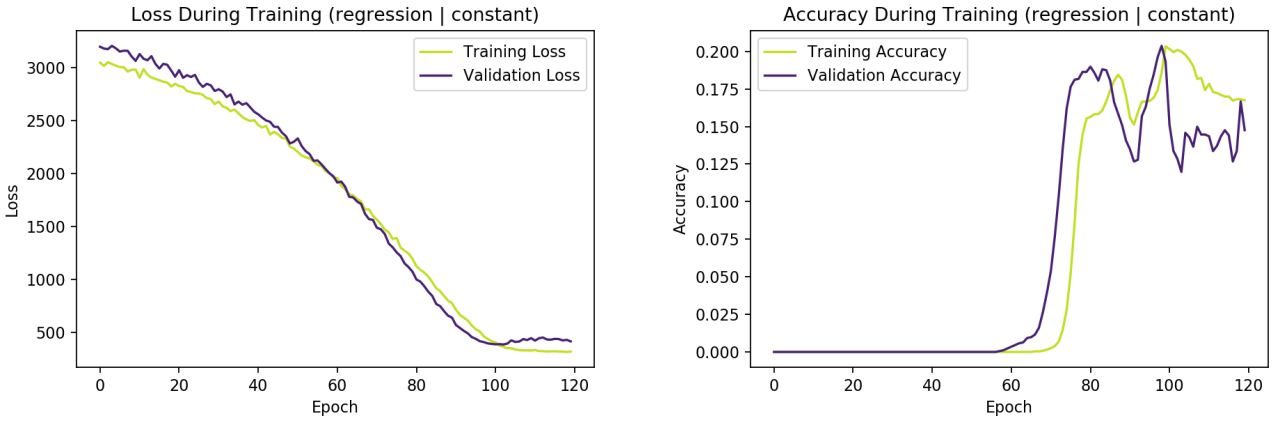
(a) Biased (b) Unbiased

Figure 59: **Error distributions** for DeepSleep performing oversampled complete regression.

```
Training Accuracy: 0.1388888888888889
Validation Accuracy: 0.1111111111111111
Test Accuracy: 0.2972972972972973

Estimated Bias: 2.2446789741516113

Unbiased Training Accuracy: 0.1574074074074074
```

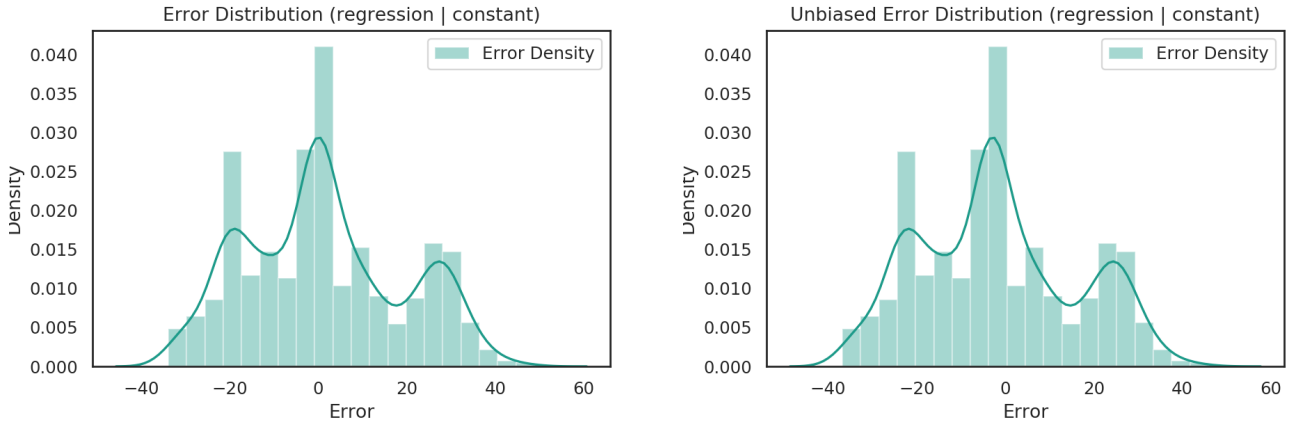
```
Unbiased Validation Accuracy: 0.19444444444444445
Unbiased Test Accuracy: 0.21621621621623



# Estimates
Unbiased Mean:   -2.2642951011657715
Standard Deviation:  17.99371910095215
95% Prediction Interval: -37.53198453903198 to
   ↪ 33.00339433670044

# Estimate from percentiles
95% Prediction Interval: -32.59143934249878 to
   ↪ 26.39871873855591

# Pure PI
95% Prediction Interval: -35.26768943786621 to
   ↪ 35.26768943786621

Epochs: 120
Learning Rate: 1e-06
Batch Size: 8
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: none
```

9.6.9 Constant Regression (standard)



(a) Loss (b) Accuracy

Figure 60: **Training** and **validation** loss and accuracy for DeepSleep performing standard constant regression.



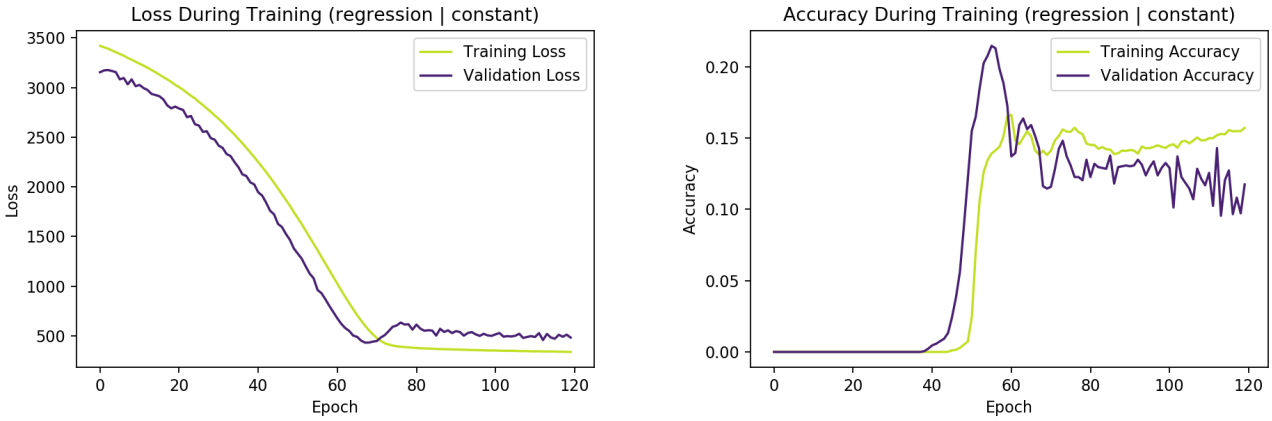
(a) Biased (b) Unbiased

Figure 61: **Error distributions** for DeepSleep performing standard constant regression.

```
Training Accuracy: 0.12962962962962962
Validation Accuracy: 0.1111111111111111
Test Accuracy: 0.2972972972972973

Estimated Bias: 2.9516193866729736

Unbiased Training Accuracy: 0.16666666666666666
Unbiased Validation Accuracy: 0.16666666666666666
```

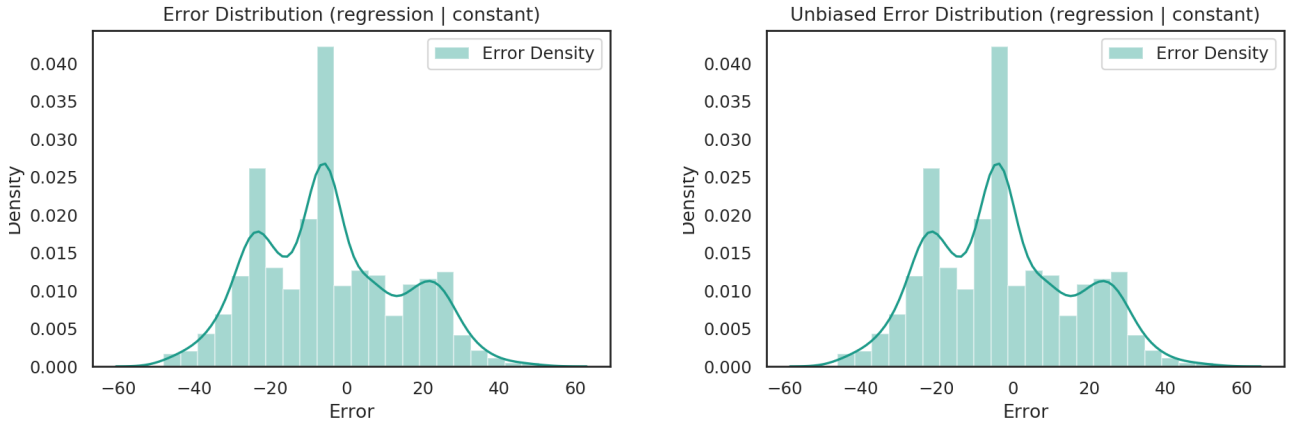
```
Unbiased Test Accuracy: 0.32432432432432434



# Estimates
Unbiased Mean:   -1.8727755546569824
Standard Deviation:  17.619007110595703
95% Prediction Interval: -36.40602949142456 to 32.6604783821106

# Estimate from percentiles
95% Prediction Interval: -33.808252811431885 to
   ↪ 28.86894941329956

# Pure PI
95% Prediction Interval: -34.53325393676758 to
   ↪ 34.53325393676758

Epochs: 120
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: none
```

9.6.10 Constant Regression (oversampled)



(a) Loss (b) Accuracy

Figure 62: **Training** and **validation** loss and accuracy for DeepSleep performing oversampled constant regression.



(a) Biased (b) Unbiased

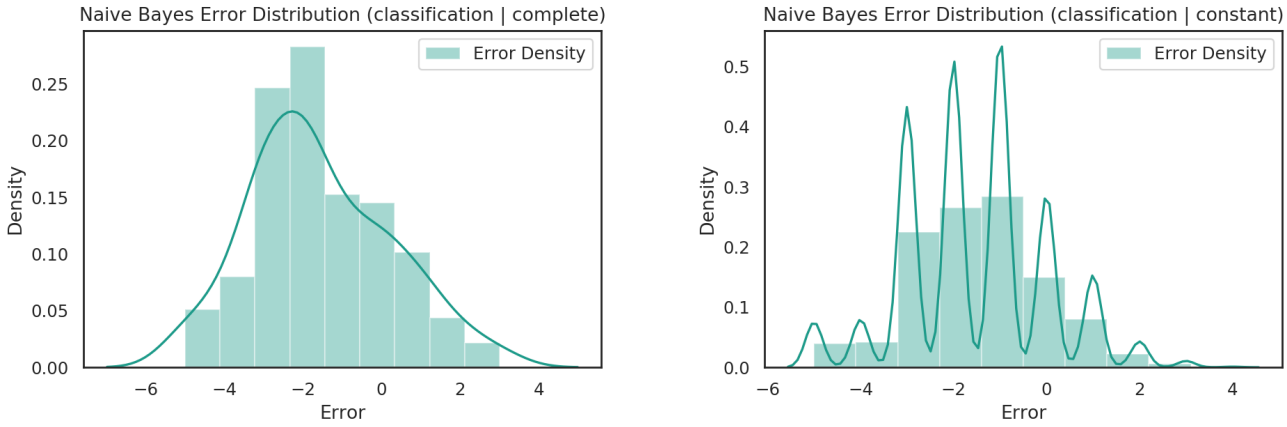
Figure 63: **Error distributions** for DeepSleep performing oversampled constant regression.

```
Training Accuracy: 0.12962962962962962
Validation Accuracy: 0.1111111111111111
Test Accuracy: 0.2972972972972973

Estimated Bias: 2.9516193866729736

Unbiased Training Accuracy: 0.16666666666666666
```

```
Unbiased Validation Accuracy: 0.16666666666666666
Unbiased Test Accuracy: 0.32432432432432434



# Estimates
Unbiased Mean:   -1.8727755546569824
Standard Deviation:  17.619007110595703
95% Prediction Interval: -36.40602949142456 to 32.6604783821106

# Estimate from percentiles
95% Prediction Interval: -33.808252811431885 to
   ↪ 28.86894941329956

# Pure PI
95% Prediction Interval: -34.53325393676758 to
   ↪ 34.53325393676758

Epochs: 120
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: none
```

9.7 Physionet: Naive Bayes

9.7.1 Error Distributions Classification



(a) Complete

(b) Constant

Figure 64: **Error distributions** for Naive Bayes performing classification.

9.7.2 Classification Metrics (Complete)

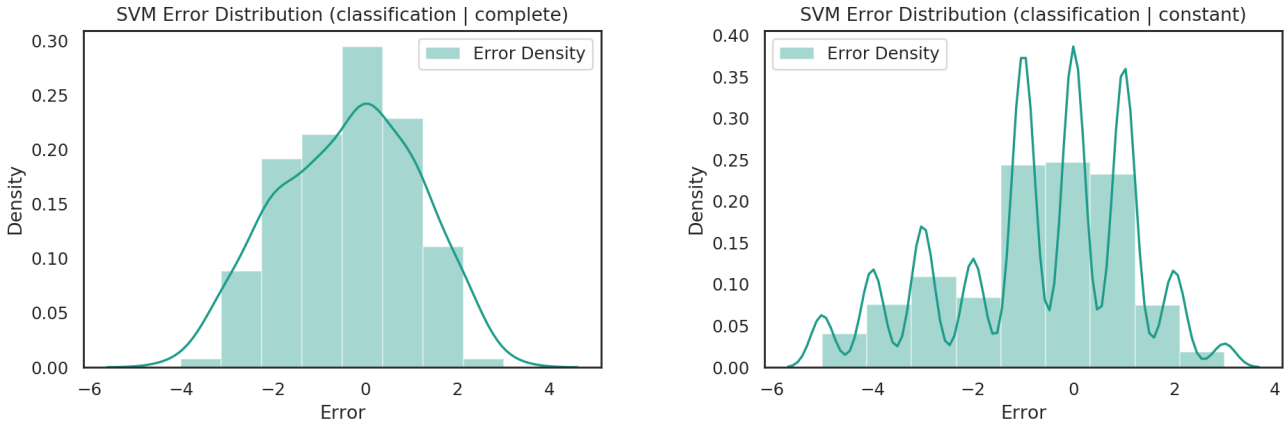
```
Accuracy Train: 0.15345821325648415
Accuracy Test: 0.12903225806451613

Max Size CV: Taken -> 1388, cap -> 80000, available -> 1388.
Max Size Fit: Taken -> 1388, cap -> 160000, available -> 1388.
Best Parameters: {'classifier__alpha': 0.1}
```

9.7.3 Classification Metrics (Constant)

```
Accuracy Train: 0.1345821325648415
Accuracy Test: 0.12387096774193548

Max Size CV: Taken -> 80000, cap -> 80000, available -> 333120.
Max Size Fit: Taken -> 160000, cap -> 160000, available ->
    ↪ 333120.
Best Parameters: {'classifier__alpha': 0.1}
```

9.8 Physionet: Support Vector Machine

9.8.1 Error Distribution Classification



(a) Complete

(b) Constant

Figure 65: **Error distributions** for SVM performing classification.

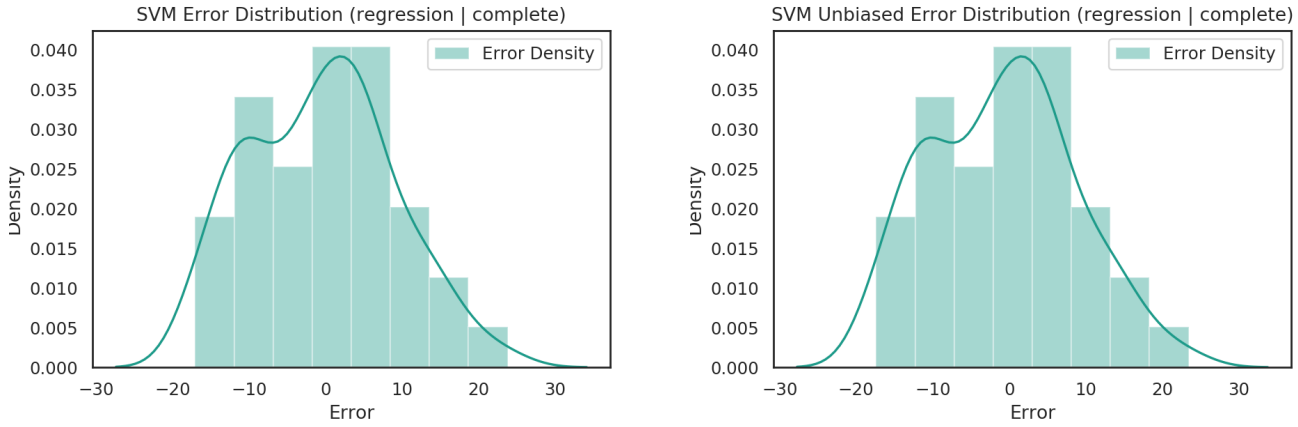
9.8.2 Classification Metrics (Complete)

```
Accuracy Train: 0.28530259365994237
Accuracy Test: 0.25806451612903225

Max Size CV: Taken -> 1388, cap -> 20000, available -> 1388.
Max Size Fit: Taken -> 1388, cap -> 40000, available -> 1388.
Best Parameter: {'classifier__C': 0.3}
```

9.8.3 Classification Metrics (Constant)

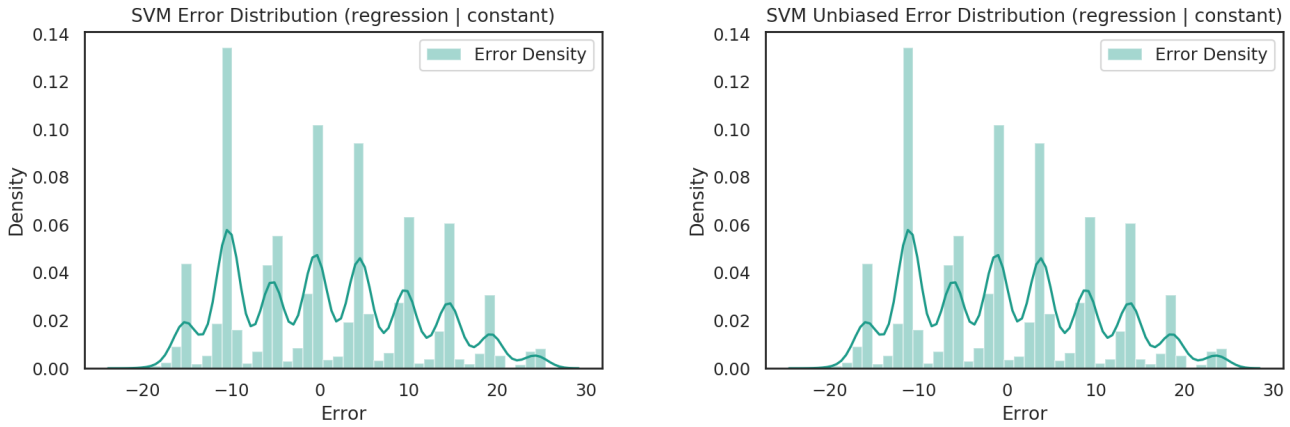
```
Accuracy Train: 0.32046109510086457
Accuracy Test: 0.24645161290322581

Max Size CV: Taken -> 20000, cap -> 20000, available -> 333120.
Max Size Fit: Taken -> 40000, cap -> 40000, available ->
    ↪ 333120.
Best Parameter: {'classifier__C': 0.1}
```

9.8.4 Error Distribution Regression



(a) Biased (b) Unbiased

Figure 66: **Error distributions** for SVM performing complete regression.



(a) Biased (b) Unbiased

Figure 67: **Error distributions** for SVM performing constant regression.

9.8.5 Metrics (Complete)

```
Error Train: 9.64557229111488
Error Test: 9.469148082282882
MSE Train: 93.03706482312316
MSE Test: 89.66476540420159
Accuracy Train: 0.40129682997118155
Accuracy Test: 0.2967741935483871
```

```
Estimated Bias: 0.3058456157408026

Unbiased Error Test: 9.494179999898268
Unbiased MSE Test: 90.13945387046829
Unbiased Accuracy Test: 0.3032258064516129

Max Size CV: Taken -> 1388, cap -> 20000, available -> 1388.
Max Size Fit: Taken -> 1388, cap -> 40000, available -> 1388.
Best Parameter: {'classifier__C': 0.5, 'classifier__epsilon':
   ↪ 9.0}



# Estimates
Unbiased Mean:   -0.9289490803362098
Standard Deviation:  9.448624634125899
95% Prediction Interval: -19.44825336322297 to
   ↪ 17.59035520255055

# Estimate from percentiles
95% Prediction Interval: -17.672241635623124 to
   ↪ 17.105945794179235

# Pure PI
95% Prediction Interval: -18.51930428288676 to
   ↪ 18.51930428288676
```

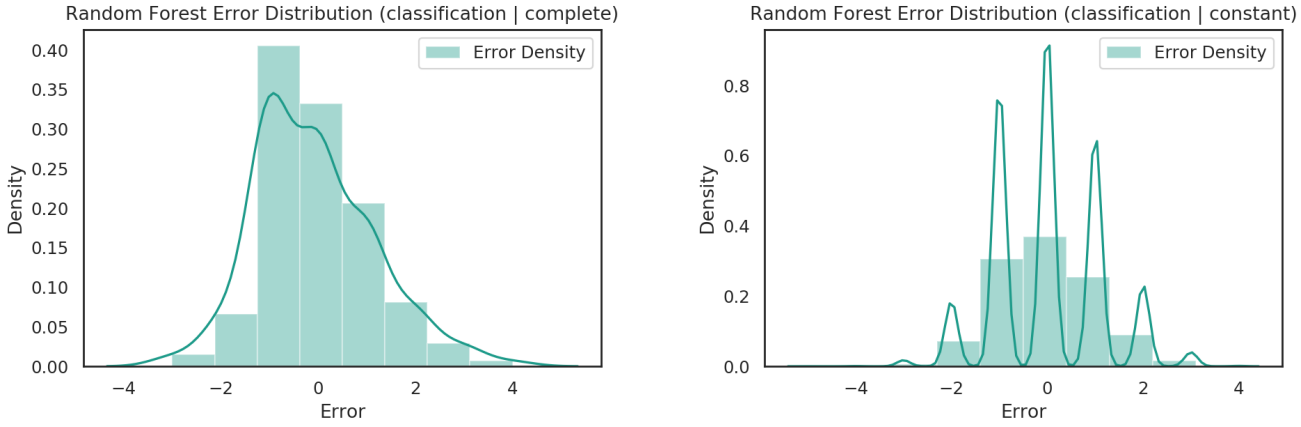
9.8.6 Metrics (Constant)

```
Error Train: 9.586708545199993
Error Test: 10.224316885063686
MSE Train: 91.90498073061056
MSE Test: 104.53665576619841
Accuracy Train: 0.4025936599423631
Accuracy Test: 0.2903225806451613

Estimated Bias: 0.7095176653591754

Unbiased Error Test: 10.19811073411625
Unbiased MSE Test: 104.00146254529707
Unbiased Accuracy Test: 0.2929032258064516

Max Size CV: Taken -> 20000, cap -> 20000, available -> 333120.
Max Size Fit: Taken -> 40000, cap -> 40000, available ->
```

```
    ↪ 333120.
Best Parameter: {'classifier__C': 0.1, 'classifier__epsilon':
    ↪ 10.0}


# Estimates
Unbiased Mean:   0.02239401849744366
Standard Deviation:  10.198086146588123
95% Prediction Interval: -19.965854828815278 to
    ↪ 20.010642865810166

# Estimate from percentiles
95% Prediction Interval: -15.982368872426829 to
    ↪ 19.081800168272558

# Pure PI
95% Prediction Interval: -19.988248847312722 to
    ↪ 19.988248847312722
```

9.9 Physionet: Random Forest

9.9.1 Error Distribution Classification



(a) Complete (b) Constant

Figure 68: **Error distributions** for Random Forest performing classification.

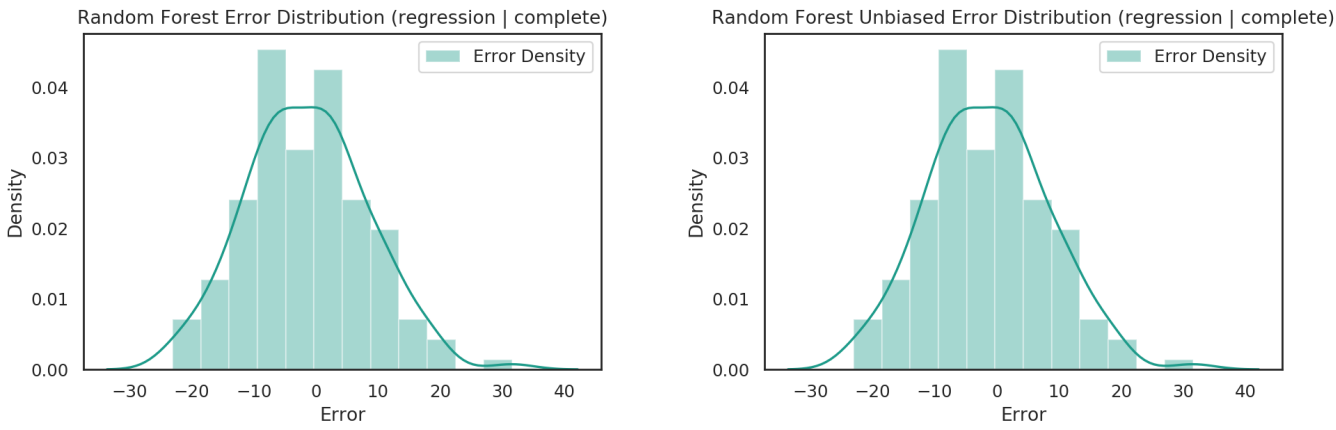
9.9.2 Classification Metrics (Complete)

```
Accuracy Train: 0.9978386167146974
Accuracy Test: 0.2903225806451613

Max Size CV: Taken -> 1388, cap -> 40000, available -> 1388.
Max Size Fit: Taken -> 1388, cap -> 80000, available -> 1388.
Best Parameters: {'classifier__max_depth': 20, '
    ↪ classifier__min_samples_leaf': 1, '
    ↪ classifier__min_samples_split': 5, '
    ↪ classifier__n_estimators': 40}
```

9.9.3 Classification Metrics (Constant)

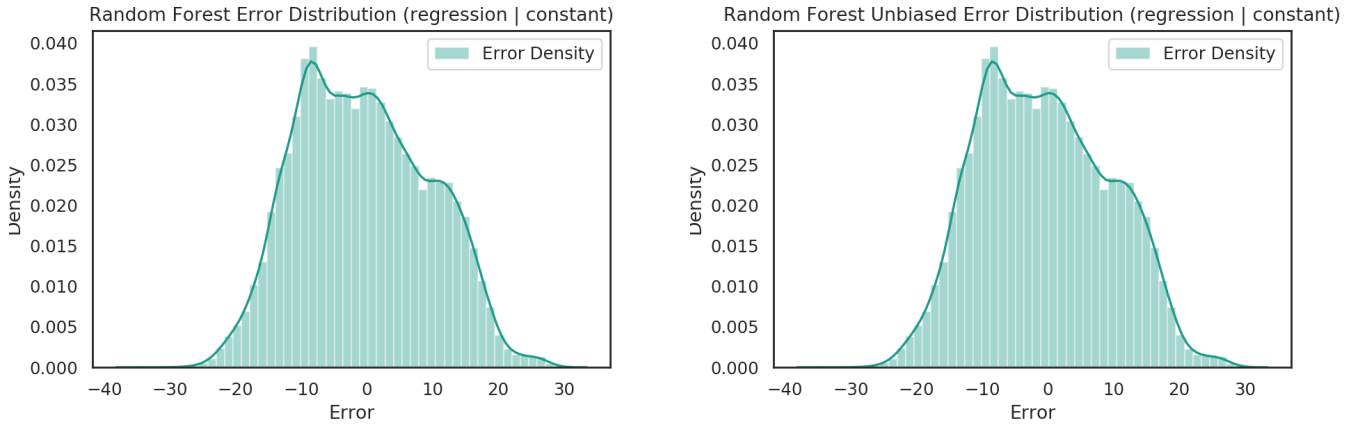
```
Accuracy Train: 0.7180115273775216
Accuracy Test: 0.3393548387096774

Max Size CV: Taken -> 40000, cap -> 40000, available -> 333120.
Max Size Fit: Taken -> 80000, cap -> 80000, available ->
    ↪ 333120.
Best Parameters: {'classifier__max_depth': 25, '
    ↪ classifier__min_samples_leaf': 1, '
    ↪ classifier__min_samples_split': 2, '
    ↪ classifier__n_estimators': 40}
```

9.9.4 Error Distribution Regression



(a) Biased (b) Unbiased

Figure 69: **Error distributions** for Random Forest performing complete regression.



(a) Biased (b) Unbiased

Figure 70: **Error distributions** for Random Forest performing constant regression.

9.9.5 Regression Metrics (Complete)

```
Error Train: 5.385236875473574
Error Test: 9.767040608053689
MSE Train: 29.000776204960385
MSE Test: 95.39508223936977
Accuracy Train: 0.6044668587896254
Accuracy Test: 0.36129032258064514

Estimated Bias: -0.01933269765439031

Unbiased Error Test: 9.764250690267152
Unbiased MSE Test: 95.34059154238257
Unbiased Accuracy Test: 0.36129032258064514

Max Size CV: Taken -> 1388, cap -> 40000, available -> 1388.
Max Size Fit: Taken -> 1388, cap -> 80000, available -> 1388.
Best Parameters: {'classifier__max_depth': 10, '
    ↪ classifier__min_samples_leaf': 2, '
    ↪ classifier__min_samples_split': 3, '
    ↪ classifier__n_estimators': 25}



# Estimates
Unbiased Mean:    -1.3996221519637018
```

```
Standard Deviation:  9.663418099933121
95% Prediction Interval: -20.33992162783262 to
   ↪ 17.540677323905218

# Estimate from percentiles
95% Prediction Interval: -21.578668504019948 to
   ↪ 16.40176526387753

# Pure PI
95% Prediction Interval: -18.94029947586892 to
   ↪ 18.94029947586892
```

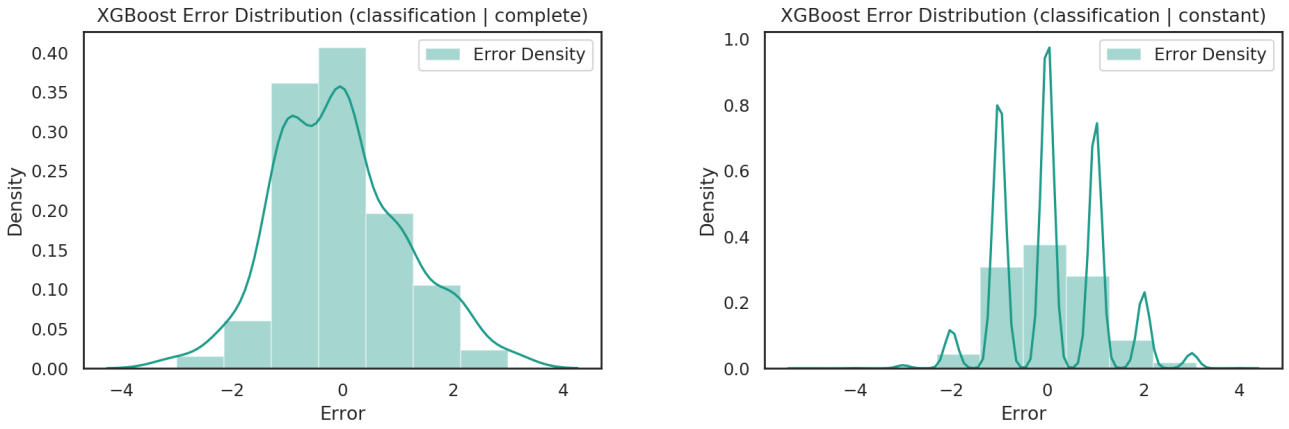
9.9.6 Regression Metrics (Constant)

```
Error Train: 8.145002571675944
Error Test: 10.019389012261868
MSE Train: 66.34106689260773
MSE Test: 100.38815617903384
Accuracy Train: 0.4642651296829971
Accuracy Test: 0.3058064516129032

Estimated Bias: -0.02348690366275692

Unbiased Error Test: 10.018083376328569
Unbiased MSE Test: 100.36199453507082
Unbiased Accuracy Test: 0.30451612903225805

Max Size CV: Taken -> 40000, cap -> 40000, available -> 333120.
Max Size Fit: Taken -> 80000, cap -> 80000, available ->
   ↪ 333120.
Best Parameters: {'classifier__max_depth': 15, '
   ↪ classifier__min_samples_leaf': 5, '
   ↪ classifier__min_samples_split': 3, '
   ↪ classifier__n_estimators': 30}



# Estimates
Unbiased Mean:   -0.5451976490194831
Standard Deviation:  10.00323717896234
95% Prediction Interval: -20.151542519785668 to
   ↪ 19.061147221746705

# Estimate from percentiles
```

```
95% Prediction Interval: -18.4198134151032 to
    ↪ 17.737244421146624

# Pure PI
95% Prediction Interval: -19.606344870766186 to
    ↪ 19.606344870766186
```

9.10 Physionet: XGBoost

9.10.1 Error Distribution Classification



(a) Complete

(b) Constant

Figure 71: **Error distributions** for XGBoost performing classification.

9.10.2 Classification Metrics (Complete)

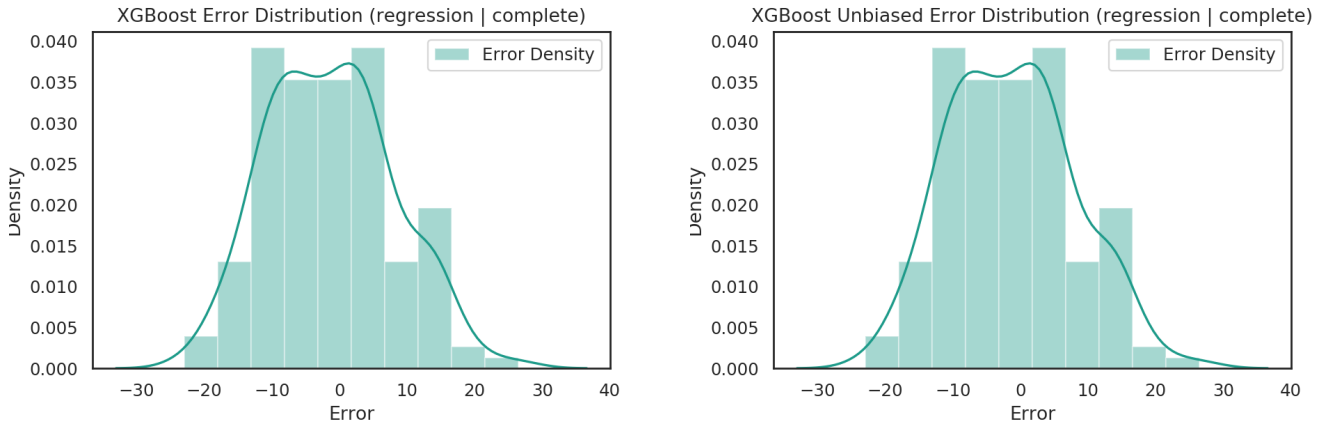
```
Accuracy Train: 0.46325648414985593
Accuracy Test: 0.34838709677419355

Max Size CV: Taken -> 1388, cap -> 80000, available -> 1388.
Max Size Fit: Taken -> 1388, cap -> 160000, available -> 1388.
Best Parameter: {'classifier__booster': 'gblinear', '
    ↪ classifier__learning_rate': 0.9, 'classifier__max_depth':
    ↪  1, 'classifier__n_estimators': 10, '
    ↪ classifier__reg_lambda': 0.9}
```

9.10.3 Classification Metrics (Constant)

```
Accuracy Train: 0.5389048991354467
Accuracy Test: 0.33806451612903227
```

```
Max Size CV: Taken -> 80000, cap -> 80000, available -> 333120.
Max Size Fit: Taken -> 160000, cap -> 160000, available ->
    ↪ 333120.
Best Parameter: {'classifier__booster': 'gbtree', '
    ↪ classifier__learning_rate': 0.3, 'classifier__max_depth':
    ↪  10, 'classifier__n_estimators': 10, '
    ↪ classifier__reg_lambda': 0.9}
```

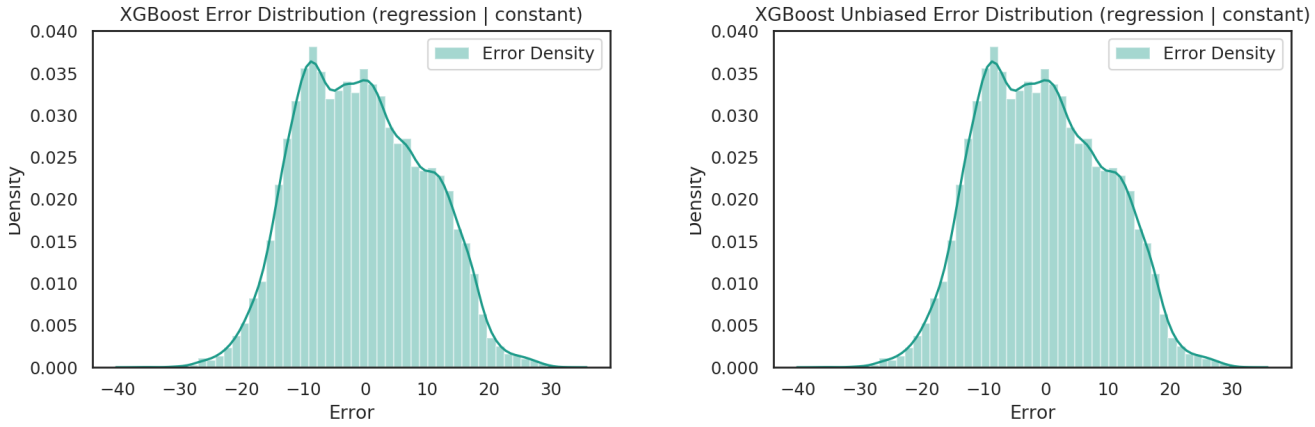
9.10.4 Error Distribution Regression



(a) Biased

(b) Unbiased

Figure 72: **Error distributions** for XGBoost performing complete regression.



(a) Biased (b) Unbiased

Figure 73: **Error distributions** for XGBoost performing constant regression.

9.10.5 Regression Metrics (Complete)

```
Error Train: 8.997183218806496
Error Test: 9.366664592044053
MSE Train: 80.94930587277321
MSE Test: 87.73440557985178
Accuracy Train: 0.4243515850144092
Accuracy Test: 0.3225806451612903

Estimated Bias: -0.006891253358692532

Unbiased Error Test: 9.365704524577186
Unbiased MSE Test: 87.71642124168557
Unbiased Accuracy Test: 0.3225806451612903

Max Size CV: Taken -> 1388, cap -> 20000, available -> 1388.
Max Size Fit: Taken -> 1388, cap -> 40000, available -> 1388.
Best Parameter: {'classifier__booster': 'gblinear', '
    ↪ classifier__learning_rate': 0.6, 'classifier__max_depth':
    ↪  1, 'classifier__n_estimators': 10, '
    ↪ classifier__reg_lambda': 0.3}



# Estimates
Unbiased Mean:   -1.301083767798639
```

```
Standard Deviation:  9.274890957356666
95% Prediction Interval: -19.479870044217705 to
   ↪ 16.877702508620423

# Estimate from percentiles
95% Prediction Interval: -18.680066693213675 to
   ↪ 13.971456133934762

# Pure PI
95% Prediction Interval: -18.178786276419064 to
   ↪ 18.178786276419064
```

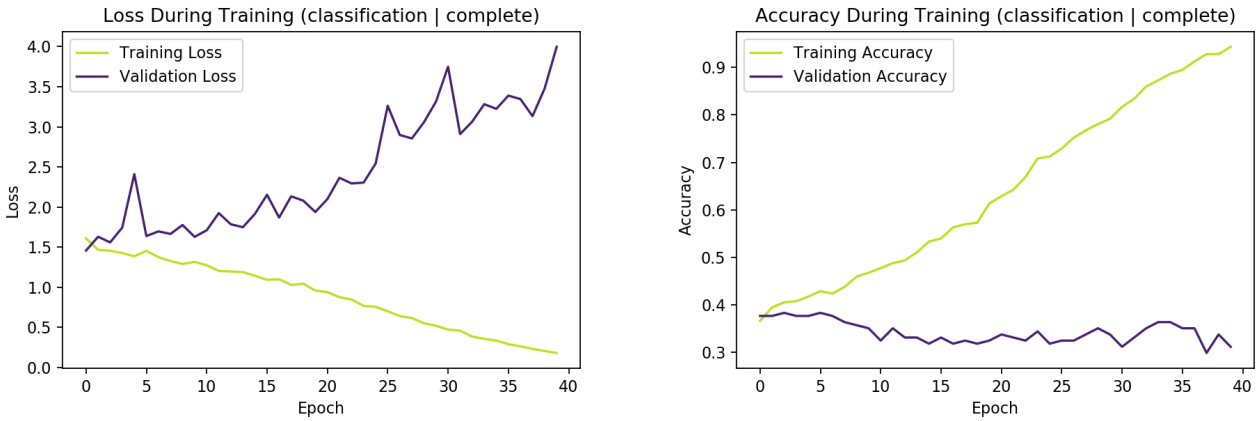
9.10.6 Regression Metrics (Constant)

```
Error Train: 8.441196445016525
Error Test: 10.090245091363965
MSE Train: 71.25379742335961
MSE Test: 101.81304600379458
Accuracy Train: 0.44927953890489913
Accuracy Test: 0.30451612903225805

Estimated Bias: -0.04478388917377428

Unbiased Error Test: 10.08781048559894
Unbiased MSE Test: 101.76392039335991
Unbiased Accuracy Test: 0.30709677419354836

Max Size CV: Taken -> 80000, cap -> 80000, available -> 333120.
Max Size Fit: Taken -> 160000, cap -> 160000, available ->
   ↪ 333120.
Best Parameter: {'classifier__booster': 'gbtree', '
   ↪ classifier__learning_rate': 0.6, 'classifier__max_depth':
   ↪  8, 'classifier__n_estimators': 8, '
   ↪ classifier__reg_lambda': 0.6}



# Estimates
Unbiased Mean:   -0.5260734799087688
Standard Deviation:  10.07408393289914
95% Prediction Interval: -20.27127798839108 to
   ↪ 19.219131028573546

# Estimate from percentiles
```

```
95% Prediction Interval: -18.653887295979324 to
    ↪ 17.799911093455496

# Pure PI
95% Prediction Interval: -19.745204508482313 to
    ↪ 19.745204508482313
```

9.11 Physionet: DeepSleep

9.11.1 Complete Classification (standard)



(a) Loss

(b) Accuracy

Figure 74: **Training** and **validation** loss and accuracy for DeepSleep performing standard complete classification.

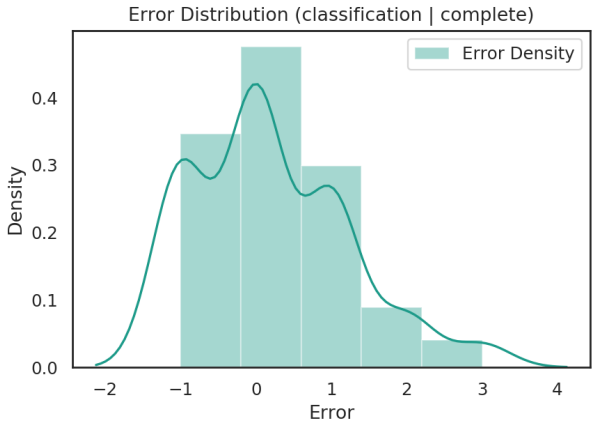
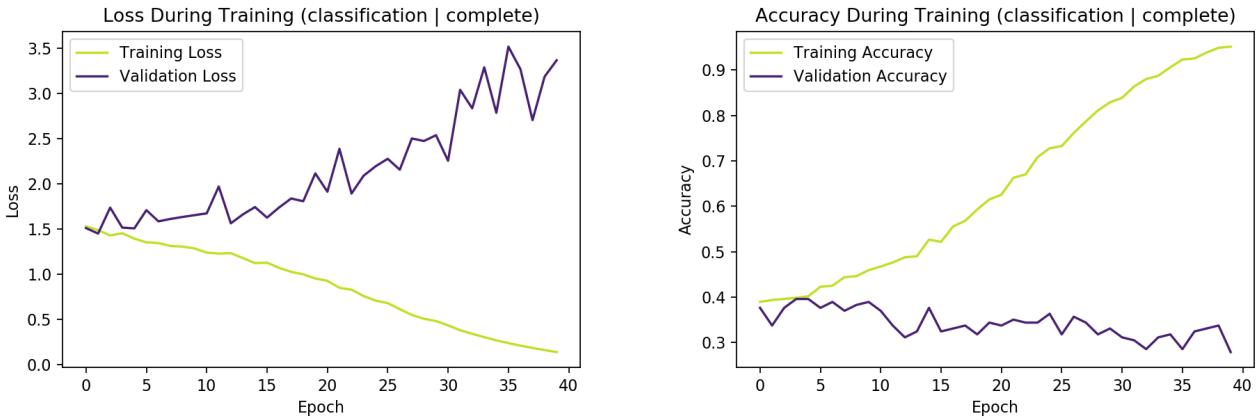


Figure 75: **Error distribution** for DeepSleep performing standard complete classification.

```
Training Accuracy: 0.3946515397082658
Validation Accuracy: 0.37662337662337664
Test Accuracy: 0.38064516129032255

Estimated Bias: 0.2922077922077922


Epochs: 40
Learning Rate: 1e-06
Batch Size: 1
Side-Arm MLP: 32
LSTM Hidden: 128
Channels: 64
Sampling: none
```

9.11.2 Complete Classification (weighted)



(a) Loss

(b) Accuracy

Figure 76: **Training** and **validation** loss and accuracy for DeepSleep performing weighted complete classification.

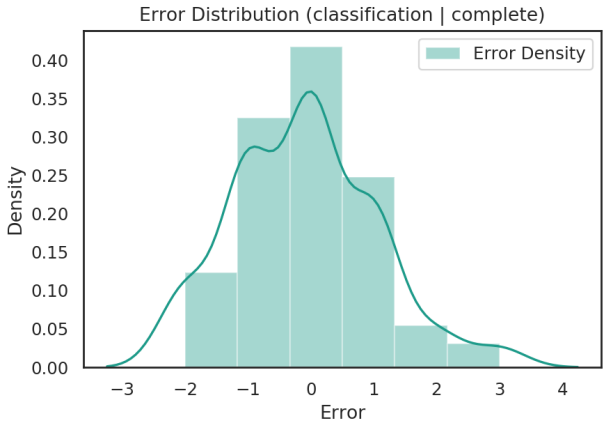
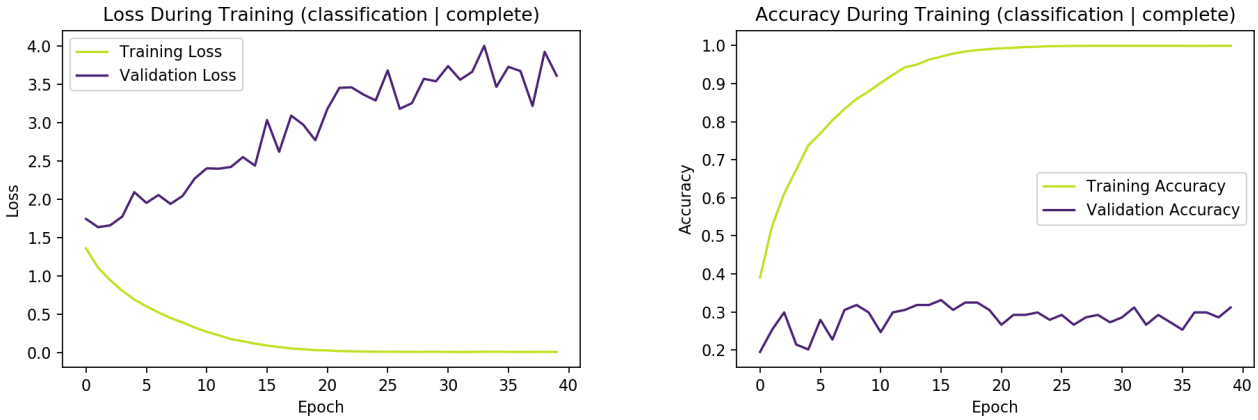


Figure 77: **Error distribution** for DeepSleep performing weighted complete classification.

```
Training Accuracy: 0.21231766612641814
Validation Accuracy: 0.17532467532467533
Test Accuracy: 0.16129032258064516

Estimated Bias: -1.1312770562770562


Epochs: 40
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 64
LSTM Hidden: 128
Channels: 64
Sampling: none
```

9.11.3 Complete Classification (oversampled)



(a) Loss

(b) Accuracy

Figure 78: **Training** and **validation** loss and accuracy for DeepSleep performing oversampled complete classification.

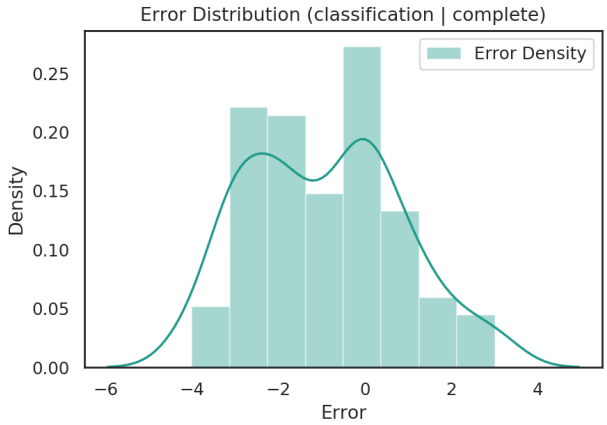
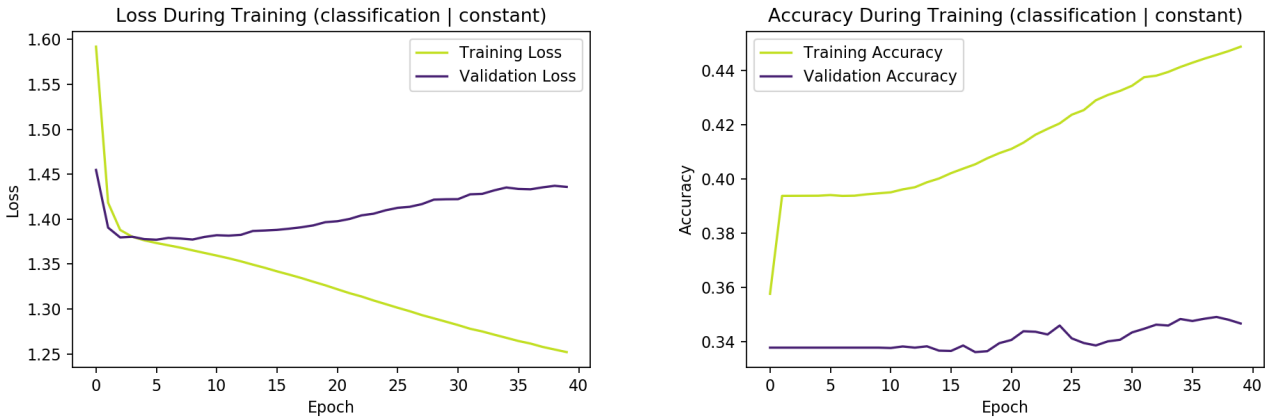


Figure 79: **Error distribution** for DeepSleep performing oversample complete classification.

```
Training Accuracy: 0.2901134521880065
Validation Accuracy: 0.2532467532467532
Test Accuracy: 0.23870967741935484

Estimated Bias: -1.0389610389610389


Epochs: 40
Learning Rate: 1e-06
Batch Size: 1
Side-Arm MLP: 32
LSTM Hidden: 128
Channels: 64
Sampling: oversample
```

9.11.4 Constant Classification (standard)



(a) Loss

(b) Accuracy

Figure 80: **Training** and **validation** loss and accuracy for DeepSleep performing standard constant classification.

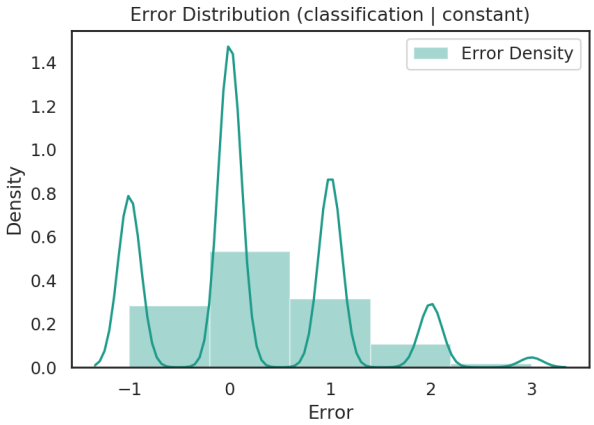
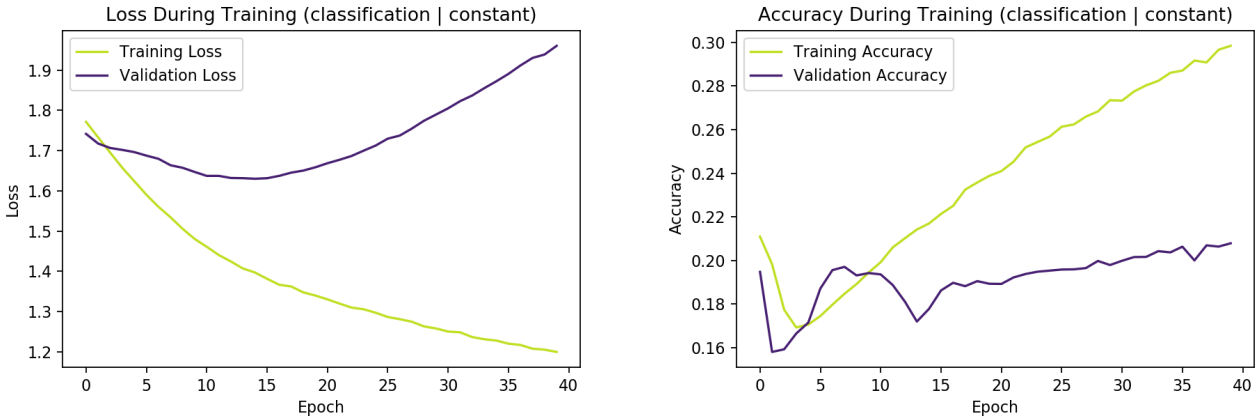


Figure 81: **Error distribution** for DeepSleep performing standard constant classification.

```
Training Accuracy: 0.39384116693679094
Validation Accuracy: 0.33766233766233766
Test Accuracy: 0.4258064516129032

Estimated Bias: 0.4090909090909091


Epochs: 40
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 64
LSTM Hidden: 128
Channels: 64
Sampling: none
```

9.11.5 Constant Classification (weighted)



(a) Loss

(b) Accuracy

Figure 82: **Training** and **validation** loss and accuracy for DeepSleep performing weighted constant classification.

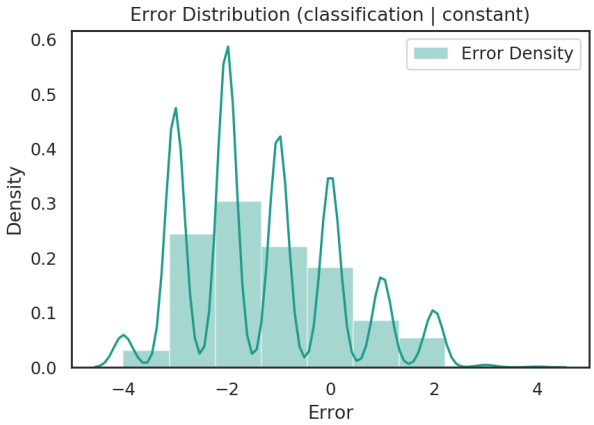
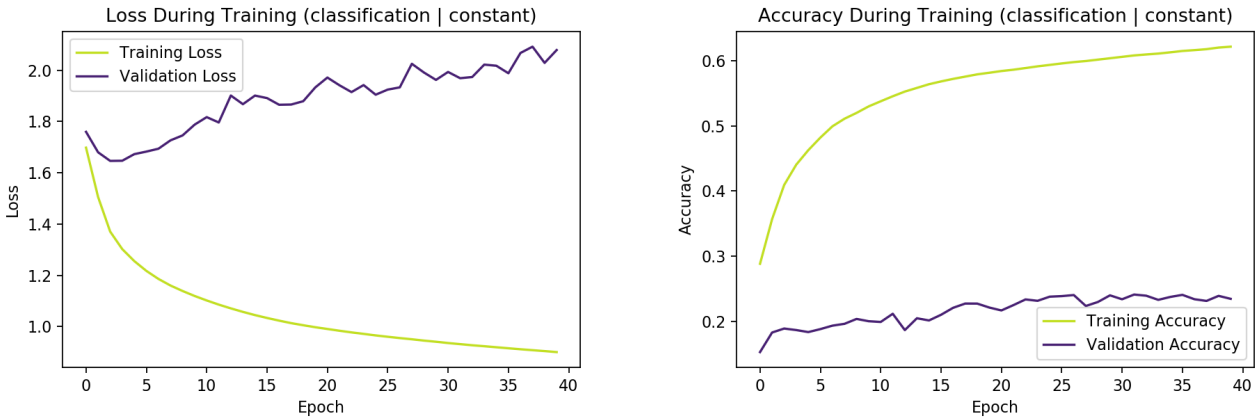


Figure 83: **Error distributions** for DeepSleep performing weighted constant classification.

```
Training Accuracy: 0.21231766612641814
Validation Accuracy: 0.17532467532467533
Test Accuracy: 0.16129032258064516

Estimated Bias: -1.1312770562770562


Epochs: 40
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 64
LSTM Hidden: 128
Channels: 64
Sampling: none
```

9.11.6 Constant Classification (oversampled)



(a) Loss

(b) Accuracy

Figure 84: **Training** and **validation** loss and accuracy for DeepSleep performing oversampled constant classification.

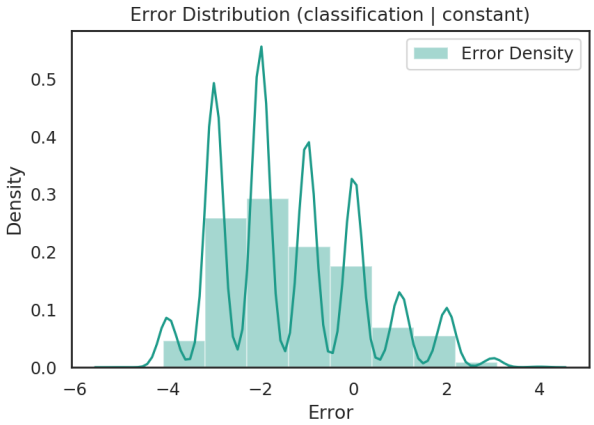
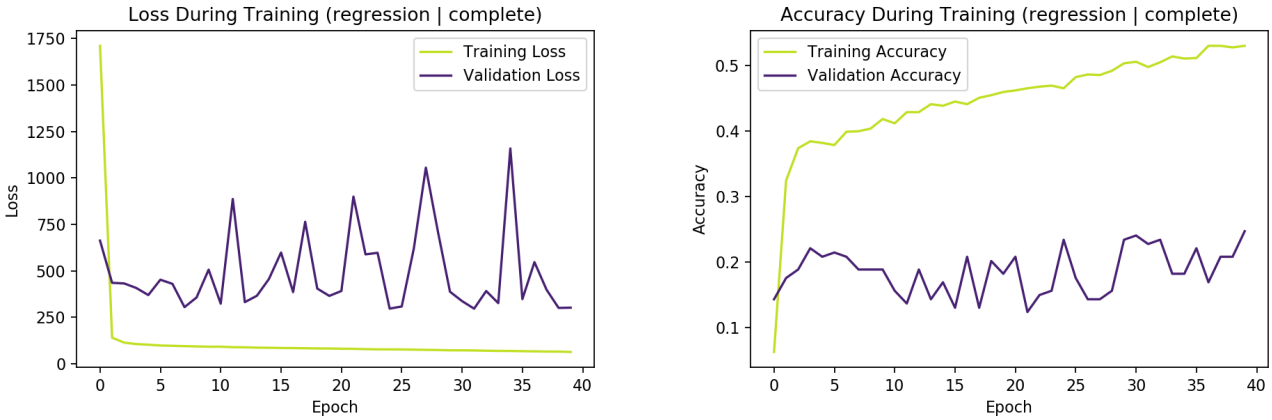


Figure 85: **Error distribution** for DeepSleep performing oversample constant classification.

```
Training Accuracy: 0.18638573743922204
Validation Accuracy: 0.19480519480519481
Test Accuracy: 0.16129032258064516

Estimated Bias: -1.2135281385281385


Epochs: 40
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 64
LSTM Hidden: 128
Channels: 64
Sampling: oversample
```

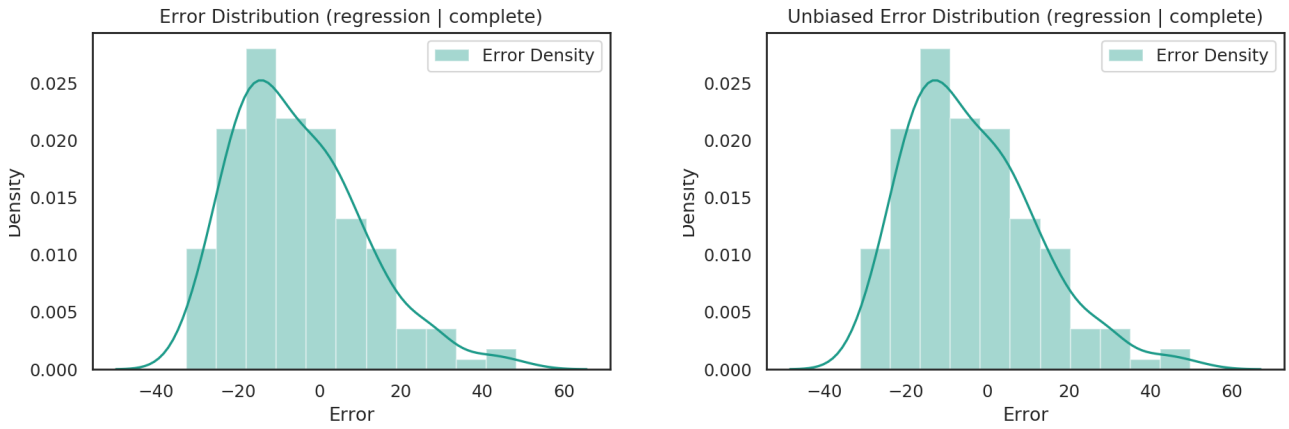
9.11.7 Complete Regression (standard)



(a) Loss

(b) Accuracy

Figure 86: **Training** and **validation** loss and accuracy for DeepSleep performing standard complete regression.



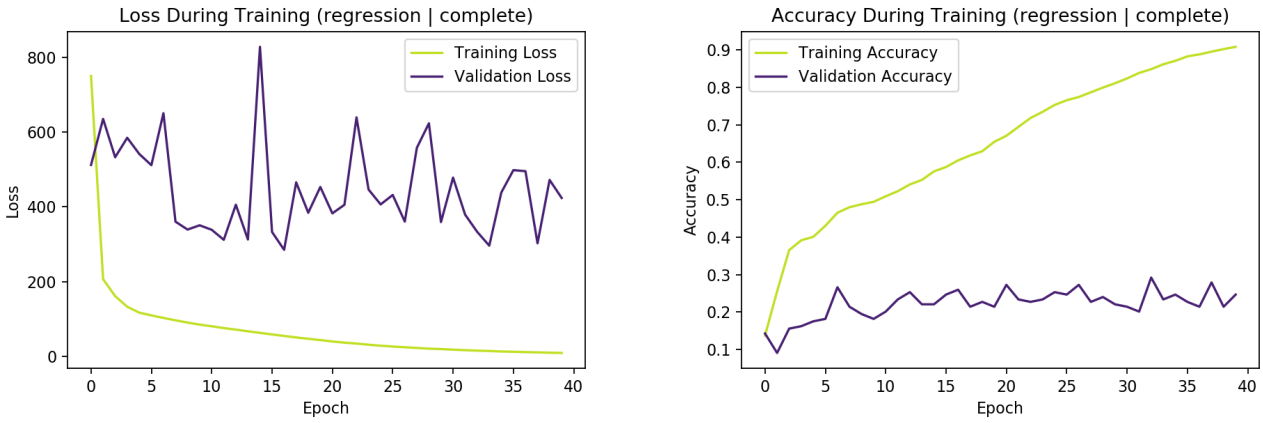
(a) Biased (b) Unbiased

Figure 87: **Error distributions** for DeepSleep performing standard complete regression.

```
Training Accuracy: 0.23500810372771475
Validation Accuracy: 0.2792207792207792
Test Accuracy: 0.2709677419354839

Estimated Bias: -1.444936990737915

Unbiased Training Accuracy: 0.2406807131280389
Unbiased Validation Accuracy: 0.2857142857142857
Unbiased Test Accuracy: 0.25161290322580643



# Estimates
Unbiased Mean:   -3.8098931312561035
Standard Deviation:  15.746270179748535
95% Prediction Interval: -34.672582683563235 to
   ↪ 27.052796421051024

# Estimate from percentiles
95% Prediction Interval: -30.28651723861694 to
   ↪ 26.75727529525758

# Pure PI
95% Prediction Interval: -30.862689552307128 to
   ↪ 30.862689552307128
```

```
Epochs: 40
Learning Rate: 1e-06
Batch Size: 1
Side-Arm MLP: 32
LSTM Hidden: 128
Channels: 64
Sampling: none
```

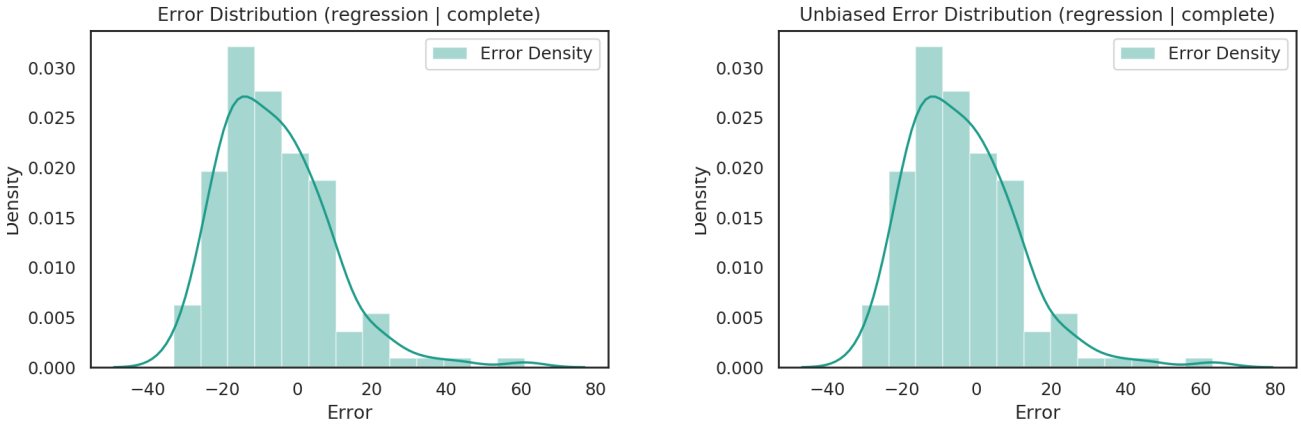
9.11.8 Complete Regression (oversampled)



(a) Loss

(b) Accuracy

Figure 88: **Training** and **validation** loss and accuracy for DeepSleep performing oversampled complete regression.



(a) Biased

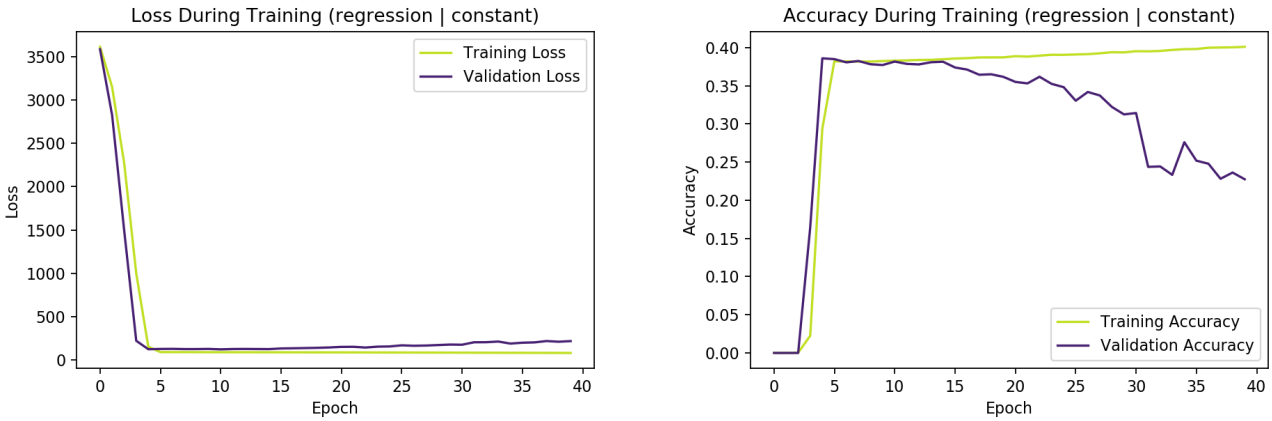
(b) Unbiased

Figure 89: **Error distributions** for DeepSleep performing oversampled complete regression.

```
Training Accuracy: 0.23500810372771475
Validation Accuracy: 0.2792207792207792
Test Accuracy: 0.2709677419354839

Estimated Bias: -1.444936990737915

Unbiased Training Accuracy: 0.2406807131280389
Unbiased Validation Accuracy: 0.2857142857142857
Unbiased Test Accuracy: 0.25161290322580643



# Estimates
Unbiased Mean:   -3.8098931312561035
Standard Deviation:  15.746270179748535
95% Prediction Interval: -34.672582683563235 to
   ↪ 27.052796421051024

# Estimate from percentiles
95% Prediction Interval: -30.28651723861694 to
   ↪ 26.75727529525758

# Pure PI
95% Prediction Interval: -30.862689552307128 to
   ↪ 30.862689552307128
```

```
Epochs: 40
Learning Rate: 1e-06
Batch Size: 1
Side-Arm MLP: 32
LSTM Hidden: 128
Channels: 64
Sampling: none
```

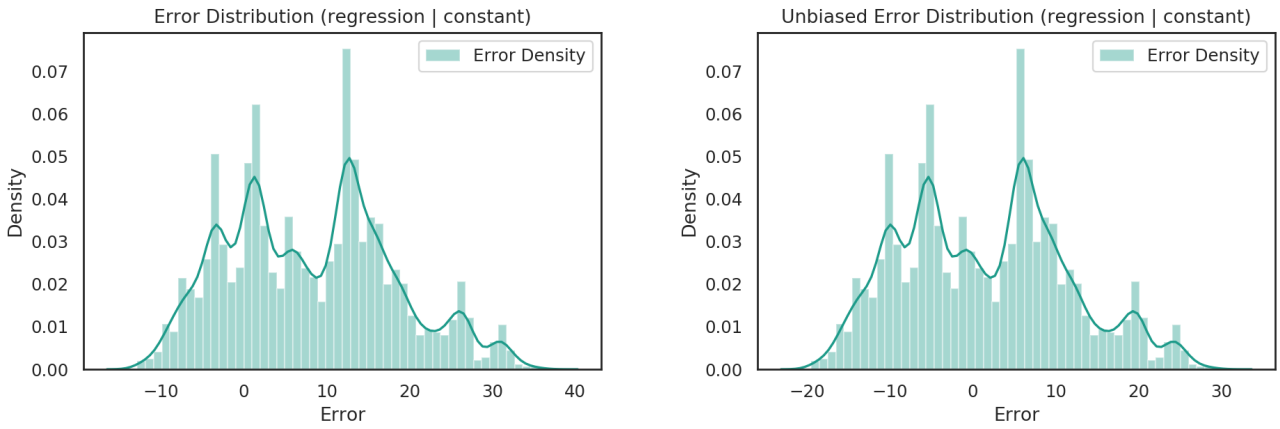
9.11.9 Constant Regression (standard)



(a) Loss

(b) Accuracy

Figure 90: **Training** and **validation** loss and accuracy for DeepSleep performing standard constant regression.



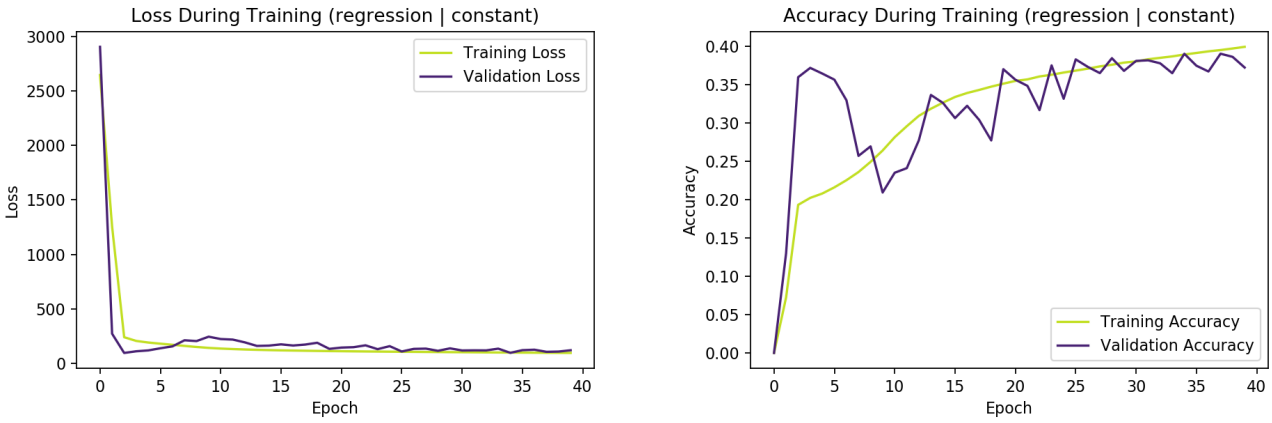
(a) Biased (b) Unbiased

Figure 91: **Error distributions** for DeepSleep performing standard constant regression.

```
Training Accuracy: 0.34440842787682335
Validation Accuracy: 0.4025974025974026
Test Accuracy: 0.32903225806451614

Estimated Bias: 6.632369518280029

Unbiased Training Accuracy: 0.3881685575364668
Unbiased Validation Accuracy: 0.43506493506493504
Unbiased Test Accuracy: 0.33548387096774196



# Estimates
Unbiased Mean:   1.4748682975769043
Standard Deviation:  9.991869926452637
95% Prediction Interval: -18.109196758270265 to
   ↪ 21.058933353424074

# Estimate from percentiles
95% Prediction Interval: -13.790889072418212 to
   ↪ 24.047116947174075

# Pure PI
95% Prediction Interval: -19.58406505584717 to
   ↪ 19.58406505584717
```

```
Epochs: 40
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 64
LSTM Hidden: 128
Channels: 64
Sampling: none
```

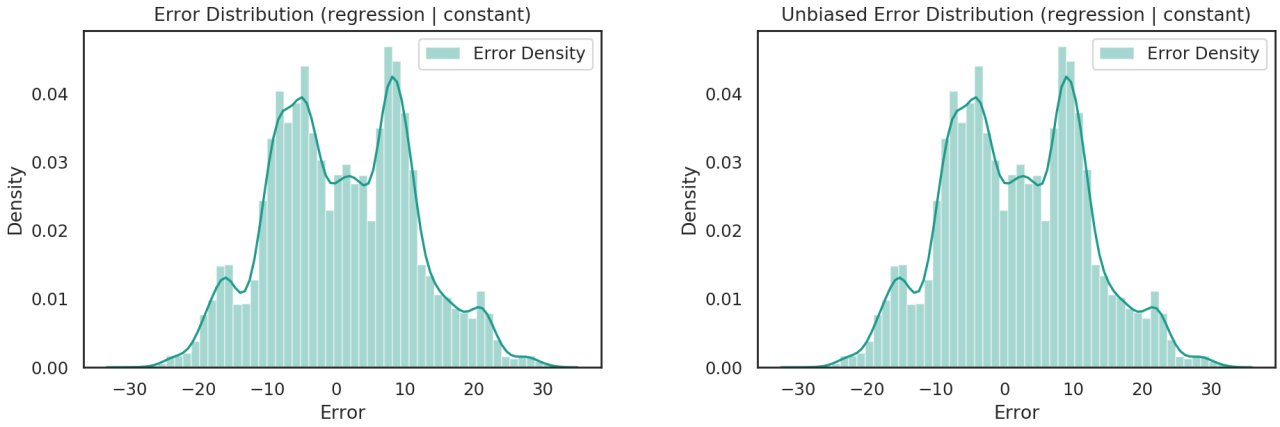
9.11.10 Constant Regression (oversampled)



(a) Loss

(b) Accuracy

Figure 92: **Training** and **validation** loss and accuracy for DeepSleep performing oversampled constant regression.



(a) Biased

(b) Unbiased

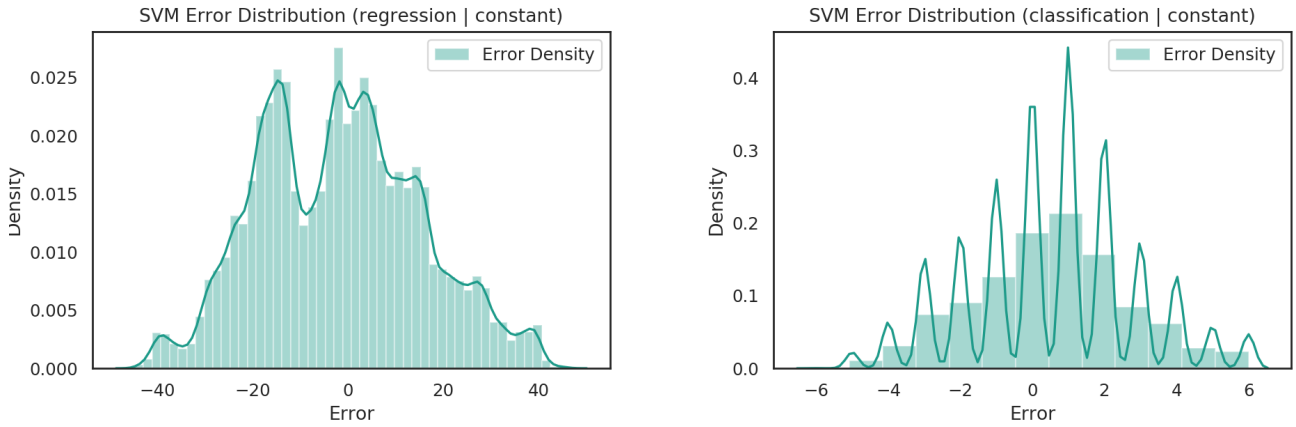
Figure 93: **Error distributions** for DeepSleep performing oversampled constant regression.

```
Training Accuracy: 0.34440842787682335
Validation Accuracy: 0.4025974025974026
Test Accuracy: 0.32903225806451614

Estimated Bias: 6.632369518280029

Unbiased Training Accuracy: 0.3881685575364668
Unbiased Validation Accuracy: 0.43506493506493504
Unbiased Test Accuracy: 0.33548387096774196



# Estimates
Unbiased Mean:   1.4748682975769043
Standard Deviation:  9.991869926452637
95% Prediction Interval: -18.109196758270265 to
   ↪ 21.058933353424074

# Estimate from percentiles
95% Prediction Interval: -13.790889072418212 to
   ↪ 24.047116947174075

# Pure PI
95% Prediction Interval: -19.58406505584717 to
   ↪ 19.58406505584717
```

```
Epochs: 40
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 64
LSTM Hidden: 128
Channels: 64
Sampling: none
```

9.12 Simulated Dataset: Support Vector Machine

9.12.1 Error Distribution Constant



(a) Regression (b) Classification

Figure 94: **Error distributions** for SVM performing constant predictions.

9.12.2 Classification Metrics (Constant)

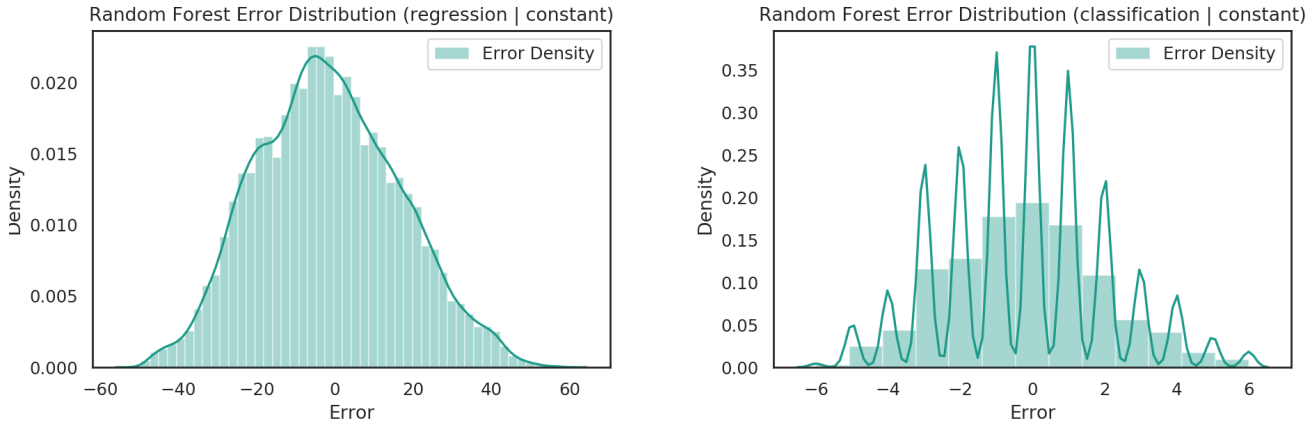
```
Accuracy Train: 0.33284722222222224
Accuracy Test: 0.1545945945945946

Max Size CV: Taken -> 20000, cap -> 20000, available ->
   ↪ 1382400.
Max Size Fit: Taken -> 40000, cap -> 40000, available ->
   ↪ 1382400.
Best Parameter: {'classifier__C': 0.9}
```

9.12.3 Regression Metrics (Constant)

```
Error Train: 16.949706747187438
Error Test: 17.121786249106304
MSE Train: 287.2925588156514
MSE Test: 293.1555643600857
Accuracy Train: 0.23902777777777778
Accuracy Test: 0.3116216216216216

Estimated Bias: -0.19375707975318024

Unbiased Error Test: 17.104019330925027
Unbiased MSE Test: 292.54747727265703
Unbiased Accuracy Test: 0.3075675675675676

Max Size CV: Taken -> 20000, cap -> 20000, available ->
   ↪ 1382400.
Max Size Fit: Taken -> 40000, cap -> 40000, available ->
   ↪ 1382400.
Best Parameter: {'classifier__C': 0.5, 'classifier__epsilon':
   ↪ 10.0}



# Estimates
Unbiased Mean:   -1.4723211203457387
Standard Deviation:  17.040532497291295
95% Prediction Interval: -34.87176481503668 to
   ↪ 31.927122574345198

# Estimate from percentiles
95% Prediction Interval: -32.7702906262144 to 31.92275665507487

# Pure PI
95% Prediction Interval: -33.39944369469094 to
   ↪ 33.39944369469094
```

9.13 Simulated Dataset: Random Forest

9.13.1 Error Distribution Constant



(a) Regression

(b) Classification

Figure 95: **Error distributions** for Random Forest performing constant predictions.

9.13.2 Classification Metrics (Constant)

```
Accuracy Train: 0.9998611111111111
Accuracy Test: 0.21243243243243243

Max Size CV: Taken -> 40000, cap -> 40000, available ->
    ↪ 1382400.
Max Size Fit: Taken -> 80000, cap -> 80000, available ->
    ↪ 1382400.
Best Parameters: {'classifier__max_depth': 25, '
    ↪ classifier__min_samples_leaf': 1, '
    ↪ classifier__min_samples_split': 3, '
    ↪ classifier__n_estimators': 40}
```

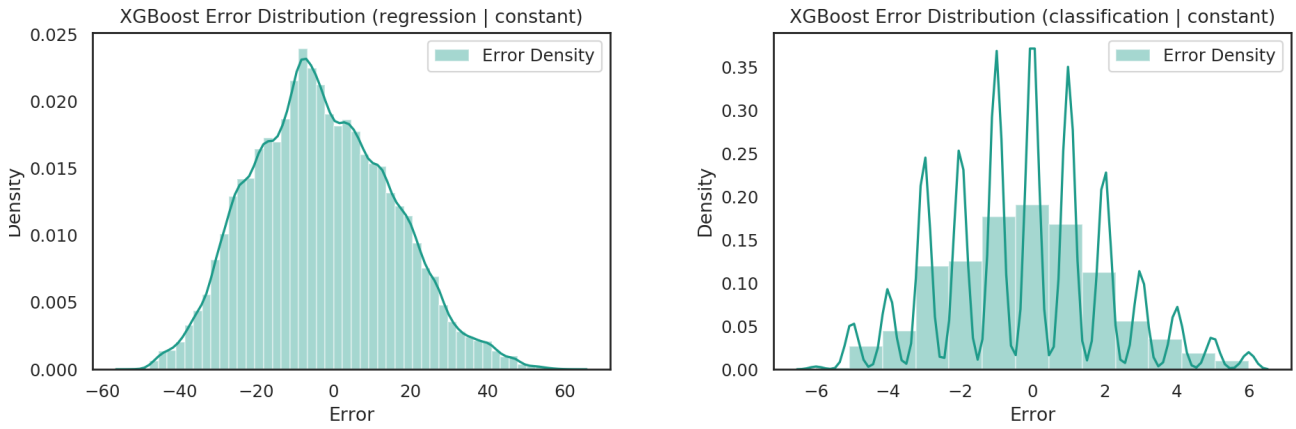
9.13.3 Regression Metrics (Constant)

```
Error Train: 11.141972649655129
Error Test: 18.355457171724442
MSE Train: 124.14355452566295
MSE Test: 336.9228079830102
Accuracy Train: 0.5988888888888889
Accuracy Test: 0.2181081081081081

Estimated Bias: -0.08633061238376737

Unbiased Error Test: 18.347990692266745
Unbiased MSE Test: 336.64876244350705
```

```
Unbiased Accuracy Test: 0.21756756756756757

Max Size CV: Taken -> 40000, cap -> 40000, available ->
   ↪ 1382400.
Max Size Fit: Taken -> 80000, cap -> 80000, available ->
   ↪ 1382400.
Best Parameters: {'classifier__max_depth': 30, '
   ↪ classifier__min_samples_leaf': 1, '
   ↪ classifier__min_samples_split': 2, '
   ↪ classifier__n_estimators': 30}



# Estimates
Unbiased Mean:   -1.544021045996928
Standard Deviation:  18.282908998652964
95% Prediction Interval: -37.378522683356735 to
   ↪ 34.29048059136288

# Estimate from percentiles
95% Prediction Interval: -36.12435710027982 to
   ↪ 34.208976233053505

# Pure PI
95% Prediction Interval: -35.83450163735981 to
   ↪ 35.83450163735981
```

9.14 Simulated Dataset: XGBoost

9.14.1 Error Distribution Constant



(a) Regression (b) Classification

Figure 96: **Error distributions** for XGBoost performing constant predictions.

9.14.2 Classification Metrics (Constant)

```
Accuracy Train: 0.9904166666666666
Accuracy Test: 0.23945945945945946

Max Size CV: Taken -> 80000, cap -> 80000, available ->
    ↪ 1382400.
Max Size Fit: Taken -> 160000, cap -> 160000, available ->
    ↪ 1382400.
Best Parameter: {'classifier__booster': 'gbtree', '
    ↪ classifier__learning_rate': 0.6, 'classifier__max_depth':
    ↪  10, 'classifier__n_estimators': 10, '
    ↪ classifier__reg_lambda': 0.6}
```

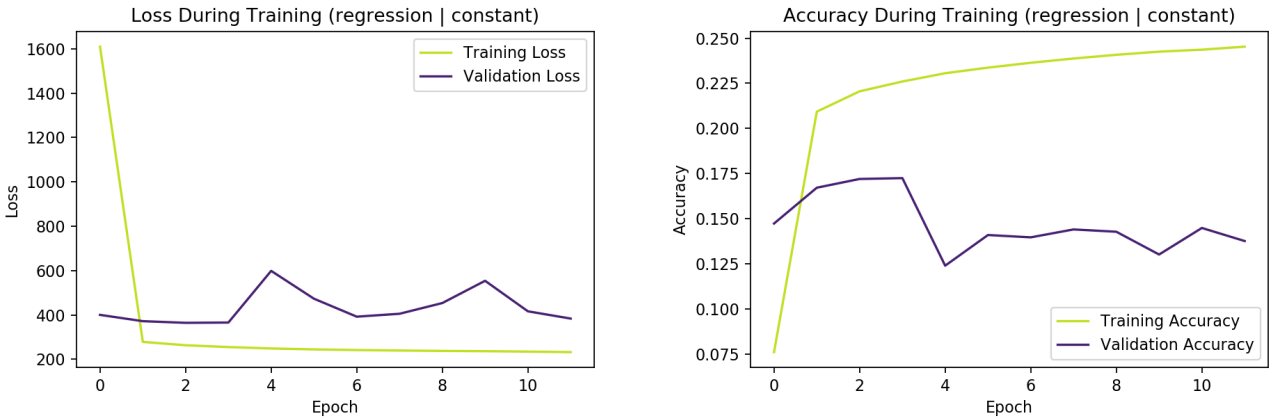
9.14.3 Regression Metrics (Constant)

```
Error Train: 12.83116718157446
Error Test: 18.229100872373785
MSE Train: 164.63885124151346
MSE Test: 332.30011861517863
Accuracy Train: 0.3779166666666667
Accuracy Test: 0.21351351351351353

Estimated Bias: -1.4541447871461235

Unbiased Error Test: 18.05227936693211
Unbiased MSE Test: 325.8847903417628
```

```
Unbiased Accuracy Test: 0.22054054054054054

Max Size CV: Taken -> 80000, cap -> 80000, available ->
   ↪ 1382400.
Max Size Fit: Taken -> 160000, cap -> 160000, available ->
   ↪ 1382400.
Best Parameter: {'classifier__booster': 'dart', '
   ↪ classifier__learning_rate': 0.3, 'classifier__max_depth':
   ↪  10, 'classifier__n_estimators': 10, '
   ↪ classifier__reg_lambda': 0.6}



# Estimates
Unbiased Mean:   -1.4788094633012205
Standard Deviation:  17.991606735169974
95% Prediction Interval: -36.74235866423437 to
   ↪ 33.78473973763193

# Estimate from percentiles
95% Prediction Interval: -35.12104297808698 to
   ↪ 34.01366986103958

# Pure PI
95% Prediction Interval: -35.26354920093315 to
   ↪ 35.26354920093315
```

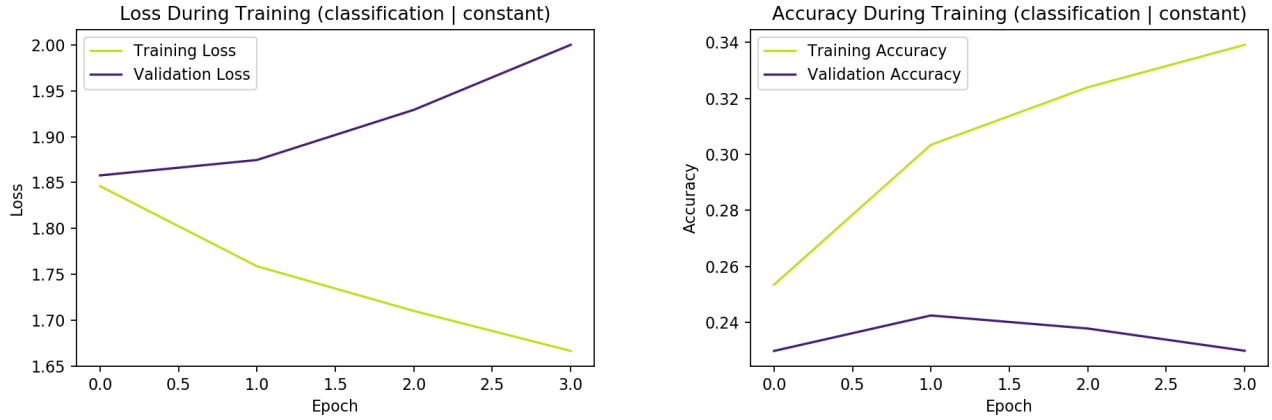
9.15 Simulated Dataset: DeepSleep

9.15.1 Error Distribution Constant



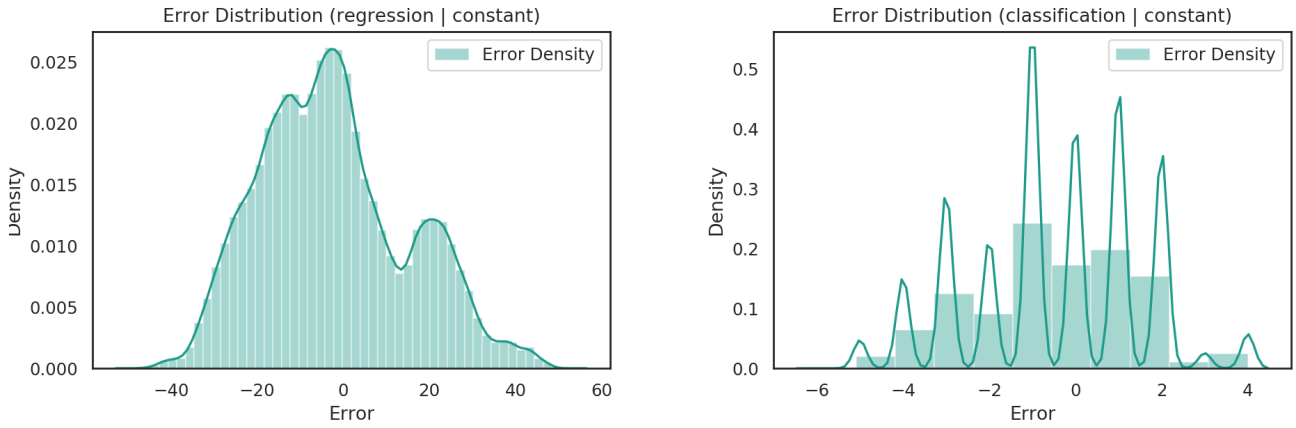
(a) Loss (b) Accuracy

Figure 97: **Training** and **validation** loss and accuracy for DeepSleep performing constant regression.



(a) Loss (b) Accuracy

Figure 98: **Training** and **validation** loss and accuracy for DeepSleep performing constant classification.



(a) Regression

(b) Classification

Figure 99: **Error distributions** for DeepSleep performing constant predictions.

9.15.2 Classification Metrics (Constant)

```
Training Accuracy: 0.2975462962962963
Validation Accuracy: 0.25430555555555556
Test Accuracy: 0.1308108108108108

Estimated Bias: -0.7665538194444445


Epochs: 4
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: none
```

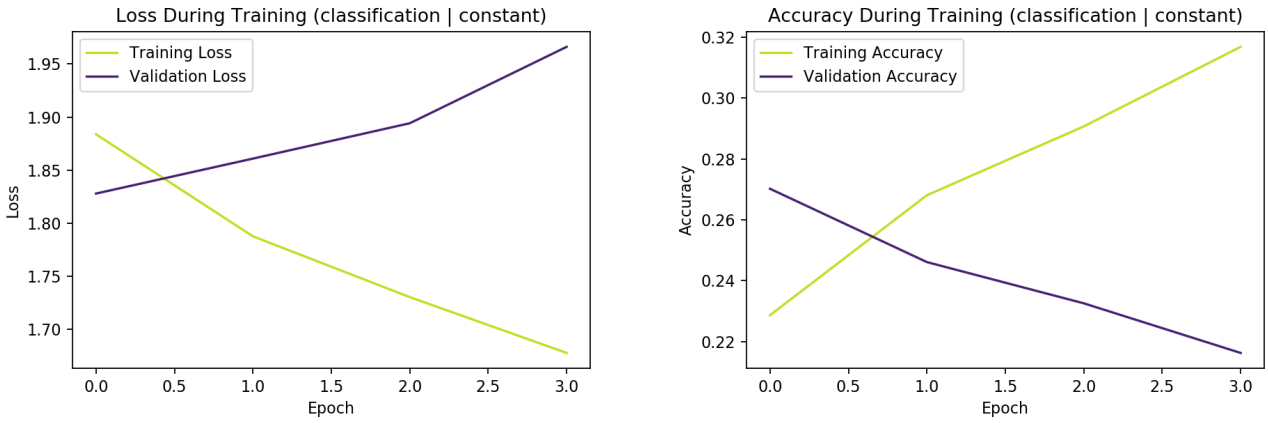
9.15.3 Regression Metrics (Constant)

```
Training Accuracy: 0.20935185185185184
Validation Accuracy: 0.18013888888888888
Test Accuracy: 0.2766216216216216

Estimated Bias: -1.2730299234390259

Unbiased Training Accuracy: 0.20083333333333334
Unbiased Validation Accuracy: 0.15513888888888888
Unbiased Test Accuracy: 0.2468918918918919



# Estimates
Unbiased Mean:   -0.9957787990570068
Standard Deviation:  17.21916961669922
95% Prediction Interval: -34.745351247787475 to
   ↪ 32.75379364967346

# Estimate from percentiles
95% Prediction Interval: -30.639387369155884 to
   ↪ 33.387352132797226

# Pure PI
95% Prediction Interval: -33.74957244873047 to
   ↪ 33.74957244873047
```

```
Epochs: 12
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: none
```

9.16 Simulated Dataset: DeepSleep

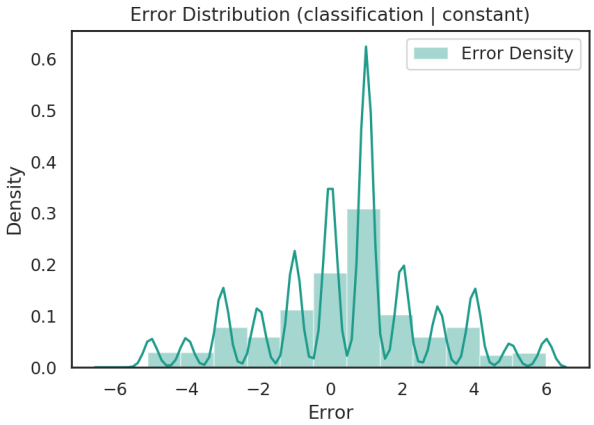
9.16.1 Error Distribution Weighted Constant



(a) Loss

(b) Accuracy

Figure 100: **Training** and **validation** loss and accuracy for DeepSleep performing constant weighted classification.



(a) Classification

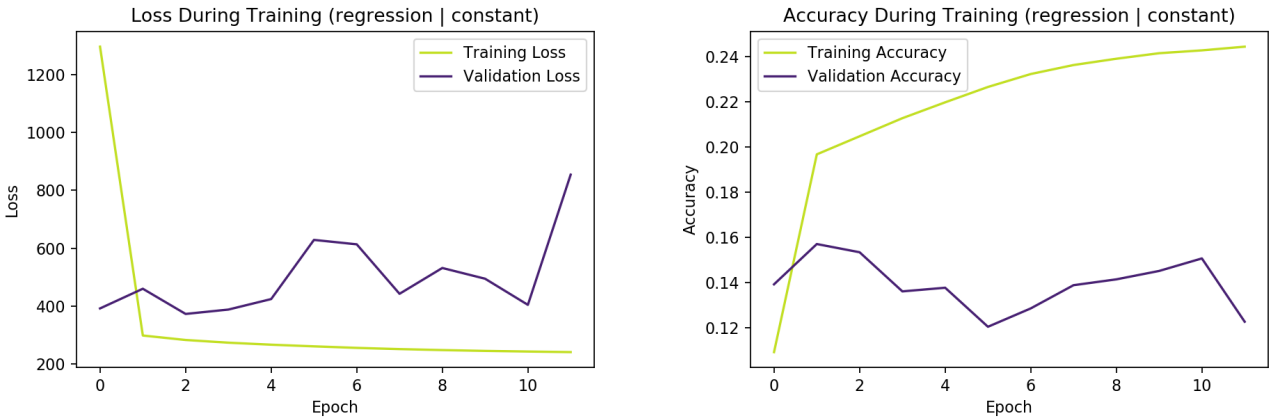
Figure 101: **Error distributions** for DeepSleep performing constant weighted predictions.

9.16.2 Classification Metrics (Constant)

```
Training Accuracy: 0.2926388888888889
Validation Accuracy: 0.31333333333333335
Test Accuracy: 0.19405405405405404

Estimated Bias: 0.34203125


Epochs: 4
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: none
```

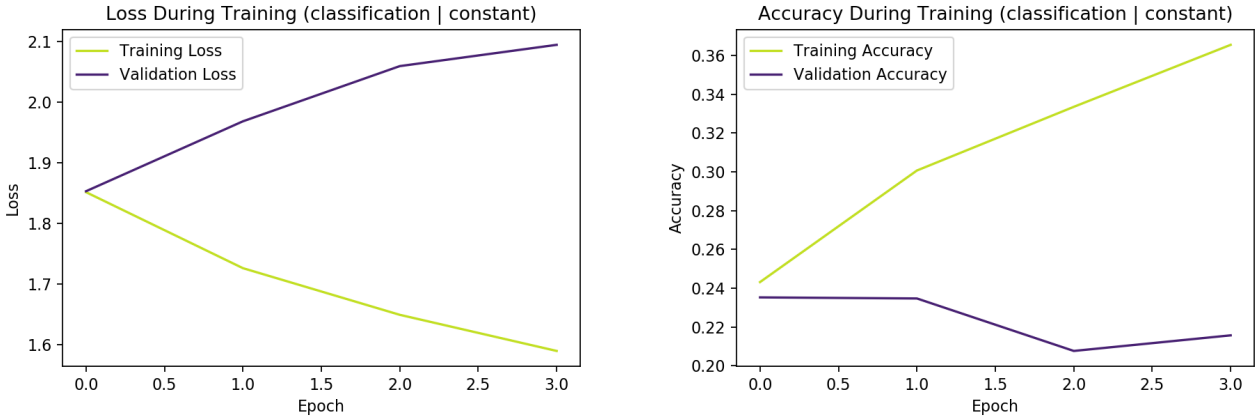
9.17 Simulated Dataset: DeepSleep (oversample)

9.17.1 Error Distribution Oversampled Constant



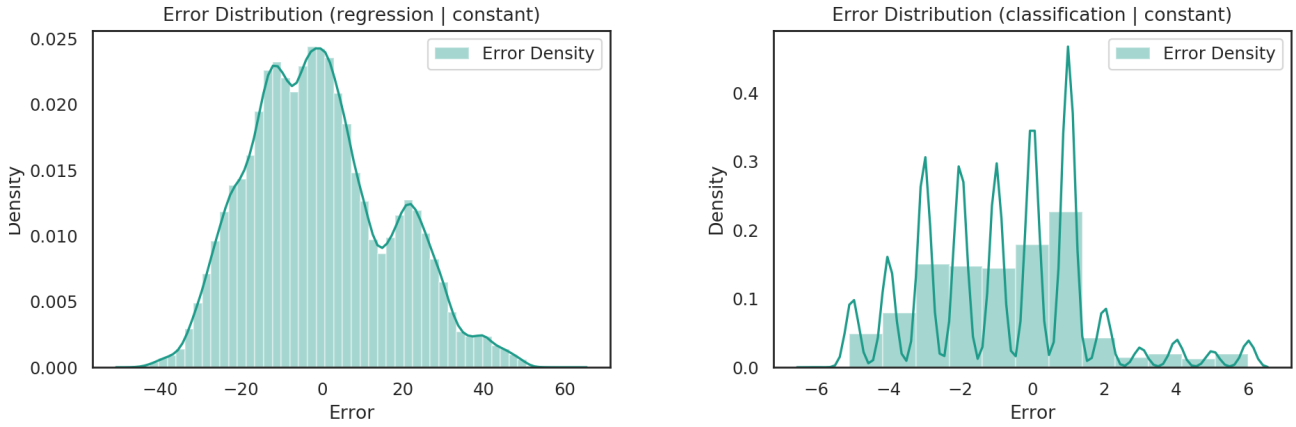
(a) Loss (b) Accuracy

Figure 102: **Training** and **validation** loss and accuracy for DeepSleep performing constant oversampled regression.



(a) Loss (b) Accuracy

Figure 103: **Training** and **validation** loss and accuracy for DeepSleep performing constant oversampled classification.



(a) Regression

(b) Classification

Figure 104: **Error distributions** for DeepSleep performing constant oversampled predictions.

9.17.2 Classification Metrics (Constant)

```
Training Accuracy: 0.3075462962962963
Validation Accuracy: 0.26416666666666666
Test Accuracy: 0.1764864864864865

Estimated Bias: -0.6629542824074074


Epochs: 4
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: oversample
```

9.17.3 Regression Metrics (Constant)

```
Training Accuracy: 0.22657407407407407
Validation Accuracy: 0.14541666666666667
Test Accuracy: 0.28256756756756757

Estimated Bias: 0.6429332494735718

Unbiased Training Accuracy: 0.23425925925925925
Unbiased Validation Accuracy: 0.1523611111111111
Unbiased Test Accuracy: 0.2951351351351351



# Estimates
Unbiased Mean:   -1.2536522150039673
Standard Deviation:  17.210420608520508
95% Prediction Interval: -34.98607660770416 to
   ↪ 32.47877217769623

# Estimate from percentiles
95% Prediction Interval: -31.34851315021515 to
   ↪ 33.622129225730795

# Pure PI
95% Prediction Interval: -33.732424392700196 to
   ↪ 33.732424392700196
```

```
Epochs: 12
Learning Rate: 1e-06
Batch Size: 512
Side-Arm MLP: 128
LSTM Hidden: 256
Channels: 128
Sampling: oversample
```