Masaryk University
Faculty of Informatics

ECG Arrhythmia Detection and Classification

Bachelor's Thesis

Adam Ivora

Brno, Spring 2020

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Adam Ivora

Advisor: doc. RNDr. Tomáš Brázdil, Ph.D.
Consultant: Ing. Filip Plešinger, Ph.D.

Acknowledgements

Thanks to
Natálka, for her love, never-ending support and encouragement,
Jana, for being there for me,
Filip, for his valuable advice, corrections and answers to every single one of my questions,
all the lively and lovely people from the Laboratory of Electronic and Multimedia Applications, for the thought-provoking debates,
Adam, for always cheering me up,
Damon Albarn, for his musical masterpieces.

Abstract

In the thesis we design a machine learning (ML) model for classification of heart arrhythmia from a 1-lead ECG (electrocardiogram) signal recorded by telemonitoring devices. A part of the thesis is devoted to the comparison of multiple ML methods on three datasets, two of which are publicly available.

Both the classical ML methods based on domain knowledge features and neural networks trained on the raw signal are tested in the comparison.

The F_1 macro averaged scores on the unseen data partitions are in the range of 0.43 to 0.79. Thanks to low inference time, the model can be used as a part of a server ECG processing application. The output of the thesis is a public repository with the comparison of the methods mentioned above on the publicly available ECG datasets.

Keywords

machine learning, electrocardiography, telemedicine, signal processing, neural networks

Contents

List of Tables

List of Figures

Introduction

Abnormalities of the heart rhythm (arrhythmia) are among the most common health problems in the population. They range in severity from benign palpitations to total loss of cardiac function and death [1]. One of the methods to diagnose heart arrhythmia is an analysis of electrocardiogram. However, a trained human expert is needed for precise diagnosis of a heart condition. Some arrhythmia, such as atrial fibrillation, require long-term monitoring for days or even weeks. However, long-term cardiac monitoring signals are too long to be thoroughly analysed by a human effectively, so the analysis has to be done automatically by a model. With automated arrhythmia classification, the expert only has to look at parts of the signal suggested by the model.

The model has to be able to classify heart rate abnormalities, the normal rhythm and noisy ECGs.

In Chapter 1, there is a brief explanation of the physiological background of ECG and description of heart arrhythmia we will classify. Chapter 2 contains an introduction to supervised learning lightly focused on learning from one-dimensional signals. Chapter 3 is a summary of software tools and libraries used (scikit-learn, PyTorch, ONNX Runtime, and more). Chapter 4 includes the description of datasets we used to train the models. The practical part of the thesis starts from chapter 5, where we discuss various machine learning models we tried to solve the problem. Chapter 6 is concerned with the training of the models. Chapter 7 is about the evaluation of the models and model export. In the last Chapter 8, we will conclude the work and talk about further improvements to the solution.

We show the final model performance in the comparison table in Section 7.2.

The full source code for the experiment is available under an MIT licence. For more information, see Appendix A.

1 Down the Electrocardiography Hole

1.1 Electrocardiogram

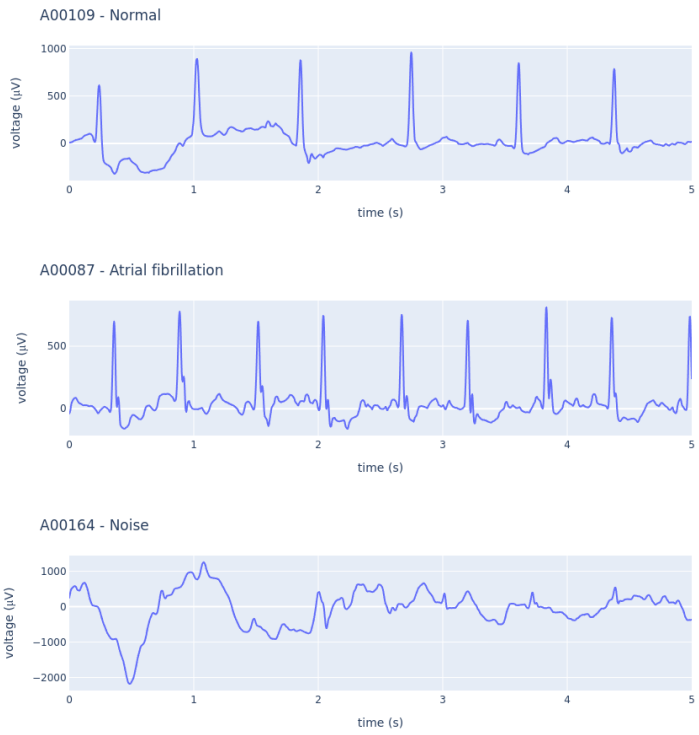


Figure 1.1: Electrocardiograms of normal rhythm, atrial fibrillation and a noisy signal (CINC2017 dataset [2]).

Electrocardiogram signal is a recording of the electrical activity of the heart. Electrodes on the skin detect small changes of voltage which are caused by heart muscle depolarization and the following repolarization. Two electrodes can measure the electric potential between them. This pair of electrodes is called an ECG lead. Every lead provides a different viewpoint of a heart's electrical activity. The most clinically used lead formation is 12-lead from ten electrodes. This work focuses on 1-lead (two electrodes) ECG often used in telemonitoring devices.

1.1.1 Telemonitoring

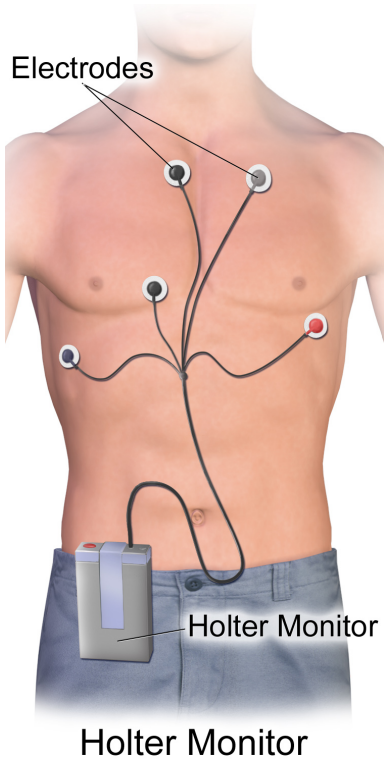


Figure 1.2: Holter monitor, a type of ECG telemonitoring device [3].

Telemonitoring is defined as "the use of information technology to monitor patients at a distance" [4]. In the context of ECG monitoring, a telemonitoring device is usually a small, wearable device such as a Holter monitor (Figure 1.2). Telemonitoring system can help decrease hospital admissions in cardiac patients [5].

1.1.2 Specific ECG elements

In order to identify heart arrhythmias, we have to recognize the characteristic patterns in ECG. The essential elements of an ECG include P wave, QRS complex and T wave [1]. All these waves are shown together in Figure 1.3.

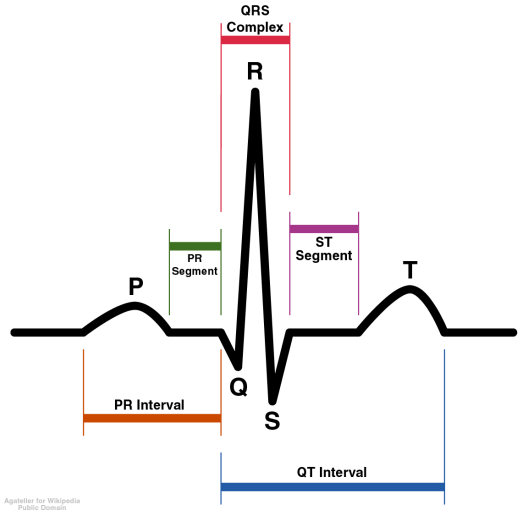


Figure 1.3: ECG of a single beat in normal sinus rhythm [6].

The P wave represents the electrical depolarization and the resulting muscle contraction of both atria, the upper chambers of the heart. The QRS complex represents the depolarization of the ventricles, lower chambers of the heart. The horizontal ST segment follows, and the cardiac cycle ends with a broad T wave, which represents the ventricular repolarization [7].

RR interval

RR interval, also called the inter-beat interval, is the time elapsed between two consecutive R waves which can be directly used to calculate the average heart rate HR:

$$\mathrm{HR} = \frac{60}{\text{RR interval in seconds}}, \tag{1.1}$$

where the RR interval in seconds is averaged over several consecutive R peaks.

Several relevant machine learning features can be extracted from the RR intervals. Heart rate variability methods are concerned precisely with the problem of analysing inter-beat intervals. We discuss these methods more thoroughly in section 2.3.5.

1.2 Selected arrhythmia descriptions

To get a basic idea of the labels that are being classified in this work, a brief description of a few heart arrhythmia follows.

1.2.1 Atrial fibrillation

Atrial fibrillation (AF) is an arrhythmia usually detected from ECG by irregular heartbeats (RR interval length is unstable) and an absence of a P wave preceding the QRS complex. AF is the most common cardiac rhythm disorder and is dangerous for the patient [8]. Atrial fibrillation increases the risk of heart failure, stroke, coronary heart disease and death [9].

An example of atrial fibrillation electrocardiogram is shown in Figure 1.1. In the figure, we can see the characteristic irregular RR intervals.

1.2.2 Premature ventricular contraction

A premature ventricular contraction (PVC) is a premature beat that most often happens by low oxygenation of the ventricles. It is easily recognizable by the usually enlarged QRS on the ECG (Figure 1.4) and the compensatory pause after the PVC which is longer than usual.

Single premature beats normally occur even in healthy people, but six or more PVCs per minute are considered pathological [10].

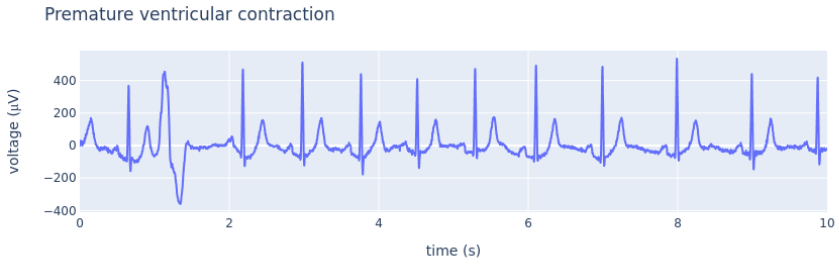


Figure 1.4: Premature ventricular contraction (`CPSC2018/A0080`) occuring at 1.1 seconds from the start of the recording.

1.2.3 Ventricular tachycardia

Ventricular tachycardia is a run of four or more premature ventricular contractions. An example of this arrhythmia is shown in Figure 1.5.

Ventricular tachycardia is a dangerous arrhythmia as it can progress into a ventricular flutter and ventricular fibrillation, which requires immediate defibrillation as there is no effective cardiac output [11].

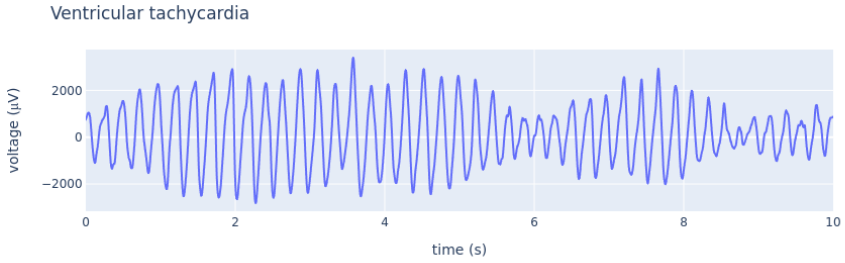


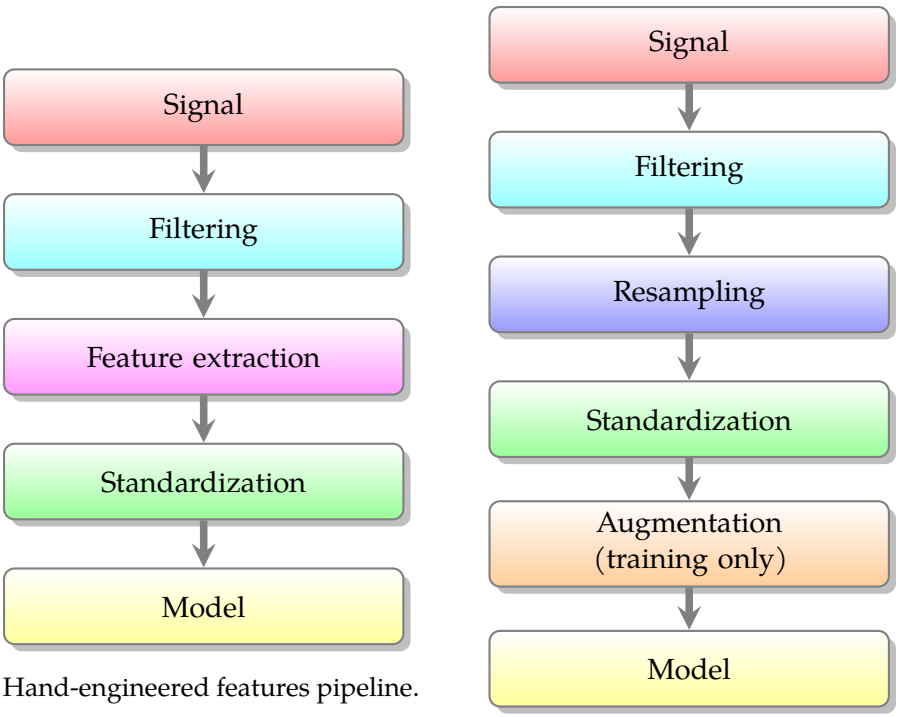
Figure 1.5: Ventricular tachycardia / fibrillation (`CUDB/cu05`) [12].

1.3 Aims of the thesis

The thesis aims to design and compare various machine learning models to classify heart rhythm. The resulting experiment is designated with respect to reproducibility and extensibility.

2 Feature Processing

Two different approaches to feature extraction are being compared in this work. One is the approach of crafting hand-engineered features from the signal by using domain knowledge. The second one is the use of neural networks to be used as automated feature extractors trained on the complete signal. Below, there are descriptions of the two approaches, and in Figure 2.1 we can see the data pipeline of both types of models.



Hand-engineered features pipeline.

Direct signal input pipeline.

Figure 2.1: Data flow in the two feature processing approaches.

Models with hand-engineered features

This approach is about using the domain knowledge to extract features from a signal that characterize the heart condition. We need to have a level of expertise in the field to build meaningful feature extractors that work even for noisy or abnormal signals.

Models with direct signal input

These methods use the signal input without much domain knowledge. The models act as automated feature extractors and try to learn the unknown mapping function between samples and the labels. The main requirement is a great amount of labelled, clean data. We resample (Section 2.2.1) the signal to reduce the number of samples, standardize it (Section 2.2.3) and then train the models. We can also use data augmentation (Section 6.3.1) to increase the number of training examples.

2.1 Signal processing

An electrocardiogram is an example of a discrete one-dimensional signal. Thus, the natural way to work with ECGs is to use methods of digital signal processing. A brief explanation of some signal processing key concepts follows. A more detailed introduction into signal processing is out of this thesis' scope, and there is already a great deal of quality material on the subject available online or in signal processing textbooks, e.g., *Discrete-time signal processing* [13].

Sampling

Heart electrical activity is a continuous analogue signal. To get a digital representation of the underlying analogue signal, we need to sample and quantize it. The sampling rate of a signal is the number of samples per time interval. The standard unit of sampling rate is Hz (1 Hz = 1 sample per second). ECG sampling rate usually falls within the range of 100–1000 Hz, but there does not exist a single sampling rate used ubiquitously.

Filtering

A discrete-time signal can be mapped from the real space into the Fourier (frequency) space by the Fourier transform. That means we can think about the signal not as a function of time, but a function of frequency [14].

Filtering is a process which removes the unwanted components of a signal. In this study, we use band-pass filtering. A band-pass

filter has two cut-off frequencies. The signals under the lower cut-off frequency and over the higher cut-off frequency are attenuated. Filter design is the study of designing signal processing filters that satisfy the requirements. More information about filter design can be found in *Discrete-time signal processing* [15].

2.2 Preprocessing

We used resampling to reduce the dimensionality of the signal and band-pass filtering to filter out frequency spectra which contain little information about heart rhythm.

2.2.1 Resampling

We can reduce the dimensionality of the signal without losing too much of its quality by resampling – changing its sampling rate. Most of the heartbeat information is in the lower frequencies under 50 Hz, and thus we can still see the electrocardiogram clearly when using relatively small sampling rate.

As we can see in Figure 2.2, the second recording resampled to 100 Hz still retains the quality of the signal, and the different segments of ECG are easily detectable. A sampling rate of 10 Hz is too low; parts of the recording are inevitably lost.

For feature extraction, we did not do any resampling. Some features, especially the frequency-domain ones, benefit from larger sampling rate. A sampling rate of 250 Hz or greater is ideal. For time-domain analysis, a frequency of 100 Hz is acceptable [16]. Thus, we resampled all recordings for neural network-based methods to 100 Hz using a polyphase filter. We used the `resample_poly` function with default parameters from SciPy, a Python library [17].

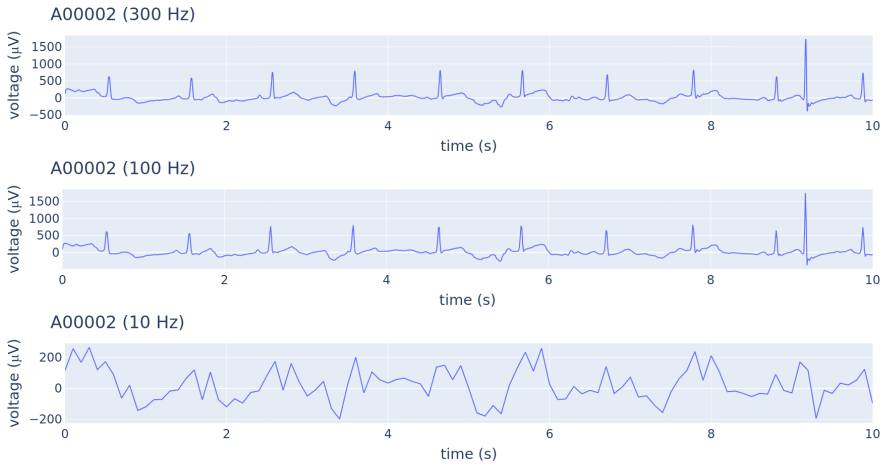


Figure 2.2: Recording `CINC2017/A00002` sampled at original frequency, 100 Hz and 10 Hz.

2.2.2 Band-pass filtering

As most of the relevant information of the signal is only in a narrow frequency spectrum, we can suppress the frequencies out of the spectrum in the signal. For this, we can use a digital band-pass filter which removes both the very low (0–0.5 Hz) and high (>40 Hz) frequency noise.

In the cases of datasets where there is no information about filtering, we preprocess the signals using a band-pass filter in the frequency range of 0–40 Hz. We do zero-phase filtering using a fifth-order Butterworth finite impulse response filter. We use the SciPy Python library for filter design.

2.2.3 Standardization

Models with hand-engineered features

For the methods where the features are extracted from the signal using domain knowledge, we standardize each feature individually by removing the mean of the training set and dividing by the standard deviation of the feature.

Models with direct signal input

We use a sample-wise standardization when working with direct signal input (distance-based methods, neural networks). For a signal $X = X_1X_2X_3...X_n$, the standardized version Z is defined as:

$$Z = \frac{X - \bar{X}}{\sigma(X)}, \tag{2.1}$$

where $\bar{X}$ is the mean and $\sigma(X)$ is the standard deviation of the signal.

The sample-wise standardization helps us to get all samples to the same range and baseline of sample values, as every patient can have its characteristic voltage range. When there is a large amount of high-amplitude noise in the recording, the sample standard deviation can be extreme, and the resulting heart waves can become hard to read after standardization. We do not deal with this problem, and it could theoretically be a problem which affects performance; thus, it is promising to explore the phenomenon more in the future.

2.3 Feature extraction

Electrocardiogram signal is high-dimensional (thousands of samples), and it can be helpful to extract features of the signal. We use generic signal statistics (mean, range, standard deviation, skew, kurtosis, extremes, percentiles). We also add domain-level features based on the length of the signal segments. For most of these methods, we need to extract and label the R peaks. Robust R peak detection in noisy electrocardiograms is not trivial, and it is still an open research problem [18].

2.3.1 R peak detection

For R peak detection, the Python physiological signal library NeuroKit2 (Section 3.2) is used. It supports a variety of R peak detection algorithms. We chose the NeuroKit custom method as it can extract R peaks even from some noisy signals (see Figure 2.3). In comparison, the standard Pan–Tompkins algorithm [19] does not detect any real R peaks in the noisy signal in the figure.

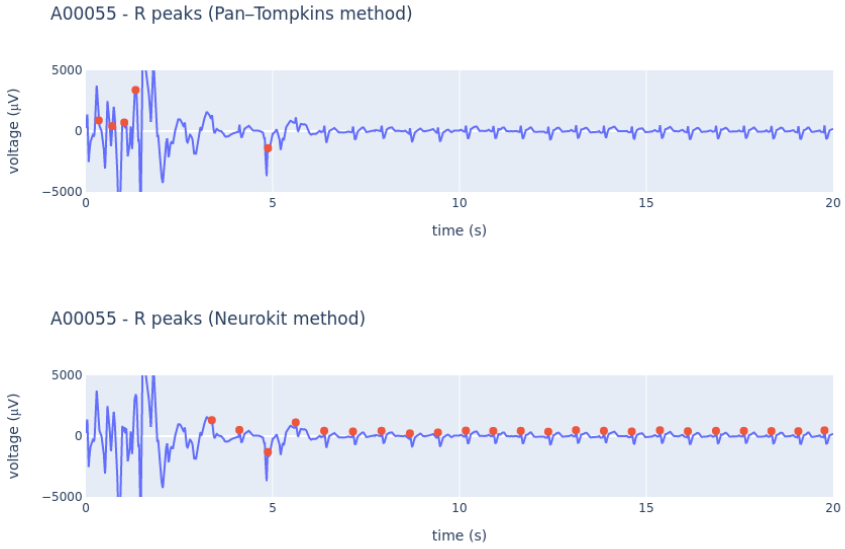


Figure 2.3: R peak detection in a noisy signal by Pan-Tompkins method (top) and Neurokit method (bottom). Recording from the CINC2017 dataset [2].

2.3.2 R peak features

With R peak annotations, we can extract potentially useful features for arrhythmia detection. These include RR interval statistics, the corresponding heart rate, and heart rate variability features (Section 2.3.5).

2.3.3 QRS correlation features

We also create features by correlating the area near the QRS peaks. Missing P waves, a symptom of atrial fibrillation, can be numerically detected by low correlation of the area before the QRS complex. These features have already been successfully tried for the detection of atrial fibrillation [20].

2.3.4 PRT segments

Neurokit2 also has an algorithm for segmenting the electrocardiogram into all the ECG segments referenced in Figure 1.3. We can extract intervals between the segments. Some arrhythmia are even defined by prolonged intervals of these segments. For example, the first degree atrioventricular block (*I-AVB*) is, by definition, a heart rhythm with a PR interval longer than 0.2 seconds [21]. The segmentation is a problem of at least the same complexity as R peak detection, so we cannot expect the detected PRT segments will be accurate all the time.

2.3.5 Heart rate variability

Heart rate variability (HRV) is the variation of the times between heartbeats (RR intervals). Heart rate variability features can be divided into three categories – time domain, frequency domain and non-linear features [22]. We use the Neurokit2 library, which is self-proclaimed to be "the most comprehensive software for computing HRV indices" [23]. As of NeuroKit2 version 0.0.40, the library can compute 19 time-domain, 11 frequency-domain and nine non-linear features. The inputs for all HRV methods are the positions of R peaks.

List of hand-engineered features

Here we present the summarized list of hand-engineered features used for training and inference. For a complete list of all features, see Appendix B.

Table 2.1: Features extracted from the signal.

Type	Count
Signal statistics	12
RR interval features	11
QRS correlation features	11
PRT interval features	55
HRV features	24
Total	113

3 Software Tools

The whole implementation was written in Python with the support of several libraries, mainly the machine learning and signal processing ones. A brief description of the tools used follows.

3.1 SciPy

The SciPy library is a scientific computation library with a heavy emphasis on time and memory-efficient numerical routines. It consists of about 600,000 lines of code in 16 subpackages and is being actively developed since 2001 [24].

In this work, the `signal` subpackage is used for signal processing and the `stats` subpackage for various statistic features computation.

3.2 NeuroKit2

NeuroKit2 is a work-in-progress successor to the library NeuroKit.py. The library provides access to advanced biological signal processing functions. The supported biosignals comprise ECG (electrocardiography signals), EDA (electrodermal activity), PPG (photoplethysmogram), EMG (electromyography signals) and EEG (electroencephalography signals) [25].

We use the ECG processing subpart of the package, namely the QRS detector (Section 2.3.1) and the PQRST segmentation routine (Section 2.3.4). We also compute the HRV features (Section 2.3.5) using this package.

3.3 Scikit-learn

Scikit-learn is a library which encompasses a range of machine learning algorithms for supervised and unsupervised classification problems. It includes functions for data preprocessing, model selection, dimensionality reduction, classification and more [26].

We use scikit-learn for the classical ML pipeline: preprocessing, training and evaluating models. The random forest and logistic regression models are also implemented by this library.

3.4 PyTorch

PyTorch is a high-performance object-oriented deep learning library. A dynamical computational graph is the core structure for neural network layers. It integrates acceleration libraries for both CPU (central processing unit) and GPU (graphics processing unit). An automatic differentiation framework is also included [27].

We use PyTorch for training neural network models. PyTorch also supports export of trained models to the ONNX format, which is useful for inference.

3.5 ONNX

Open Neural Network Exchange (ONNX) is an open-source format for trained ML models. It supports inference on both PC and mobile devices using the ONNX Runtime. The ONNX Runtime also supports multiple inference accelerators, one of which is also a GPU acceleration [28].

ONNX bindings exist for a great variety of programming languages: Python, C#, C++, C, Java, Ruby and JavaScript. We use the ONNX models and the ONNX Runtime C# package for easy model replacement/updates.

For exporting scikit-learn estimators, we use the ONNXMLTools library [29]. PyTorch already has built-in ONNX exporting functionality.

3.6 Other libraries

We use Pandas [30] for data analysis, progress bars from tqdm [31], NumPy [32] for its efficient vectorized computation model, tslearn [33] for the k-NN DTW classifier (Section 5.1.1), XGBoost [34] for the gradient boosted tree model (Section 5.2.2) and Plotly [35] for generating the plots in this thesis.

4 Datasets

We use three datasets in total for evaluating the models. Several arrhythmia public datasets exist, but their structure is variable. In particular, they are varied by the lengths of recordings, the number of classes and number of ECG leads. From these, we have chosen The PhysioNet Computing in Cardiology Challenge 2017 and The China Physiological Signal Challenge 2018 dataset. For an interested reader, comprehensive lists of arrhythmia datasets is a part of the PTB-XL publication [36]. We also used a private dataset which is larger than any open one we have found.

4.1 The PhysioNet Computing in Cardiology Challenge 2017

AF Classification from a short single lead ECG recording was the theme of The PhysioNet Computing in Cardiology Challenge (CINC) 2017 [2]. It is of our interest since the samples provided resemble our use case – single lead data from a telemonitoring device. The 8528 recordings ranging from 9 to 61 seconds are sampled at 300 Hz. The files are already preprocessed by a band-pass filter with a bandwidth of 0.5-40 Hz. There are only three classes included in the challenge original scoring metric – the noise class *X* was added subsequently. Five teams shared the first place in the competition. The winners mainly used ensemble methods: XGBoost + RNN [37], random forests [38, 39] and custom ensemble algorithms [40, 41].

Table 4.1: CINC2017 labels

Label	Condition	Count
N	Sinus Rhythm	5154
A	Atrial Fibrillation	771
O	Other Rhythm	2557
X	Noisy	46
Total		8528

4.2 The China Physiological Signal Challenge 2018

A set of 12-lead ECGs was made public for The China Physiological Signal Challenge (CPSC) 2018 [42]. There is a total of 9831 recordings from 9458 different patients. Out of those, 6877 is available in the form of a training set. The recordings are sampled at 500 Hz. The length of the recording ranges from six to 60 seconds. There is a total of nine distinct classes. There is a total of 477 multi-label recordings (the patient had more than one heart condition); we do not use them as some of our models are not fit for multi-label classification. We used only the first lead I as it is similar to a single-lead ECG. Moreover, the winners of the challenge achieved only slightly worse performance when using a single lead for classification [43].

Table 4.2: CPSC2018 labels

Label	Condition	Count
Normal	Sinus Rhythm	918
AF	Atrial Fibrillation	1098
I-AVB	Atrioventricular Block	704
LBBB	Left Bundle Branch Block	207
RBBB	Right Bundle Branch Block	1695
PAC	Premature Atrial Contraction	556
PVC	Premature Ventricular Contraction	672
STD	ST Segment Depression	825
STE	ST Segment Elevation	202
Total		6877

4.3 Private dataset

This dataset was acquired by telemonitoring thousands of patients throughout several years (MDT, s.r.o., Brno, Czechia). The sampling frequency is 200 Hz; the length of every recording is between 30 and 70 seconds. In the table below, we can see the distribution of class labels in the data.

Table 4.3: Private dataset labels

Label	Condition	Count
AF	Atrial Fibrillation	20000
AVB	Atrioventricular Block	20000
NK	Noisy	20000
PAC	Premature Atrial Contraction	20000
PVC	Premature Ventricular Contraction	20000
SR	Sinus Rhythm	20000
SVT	Supraventricular Tachycardia	14823
VT	Ventricular Tachycardia	4757
Total		139580

4.4 Other datasets

There exist more publicly available electrocardiogram datasets. Although we did not use them in the practical part, we wanted to give an overview of some of them and the reason why we do not compare the machine learning models on them in this work.

4.4.1 MIT-BIH

The MIT-BIH (Massachusetts Institute of Technology - Beth Israel Hospital) Arrhythmia Database [44] has been the standard for analysis of arrhythmia classification methods. The dataset is small – it contains 48 recordings from 47 patients. There is a high number of papers investigating this dataset. We get about 12,100 results on Google Scholar if we search for "MIT-BIH Arrhythmia Database" related articles [45]. We decided not to use the dataset because of its small number of patients and over saturation of articles analysing it.

4.4.2 PTB

The PTB (Physikalisch-Technische Bundesanstalt) Diagnostic ECG Database [46] is another ECG database that is widely cited (3,150 results on Google Scholar). It contains 549 records from 290 patients.

The only class with a sufficient number of recordings is Myocardial infarction, with 148 recordings. As we do not try to detect myocardial infarction in this thesis, it does not make much sense to include it in the benchmarks.

4.4.3 PTB-XL

This dataset [36] contains 21,837 clinical 12-lead ECG short recordings from 18885 patients and is the largest public ECG dataset as of July 2020. As it was published in April 2020, there was not enough time to do a detailed analysis of it. One comparison study focused on deep learning networks on the PTB-XL dataset is already available [47]. Further analysis of machine learning methods for classification of arrhythmia in the dataset is a promising future direction to explore.

5 Models

This chapter lists all the models that were tested on the selected arrhythmia datasets. The selection of models was influenced by the winning entries of CINC2017 and CPSC2018. Therefore, we chose models from the family of ensemble methods (random forest – Section 5.2.1, XGBoost – Section 5.2.2) based on hand-engineered features and neural network models trained on direct signal (CNN – Section 5.3.1, RNN – Section 5.3.2). As baseline classifiers, we used k-nearest neighbour classifier with DTW distance metric (Section 5.1.1) and a logistic regression (Section 5.1.2) model based on the hand-engineered features.

The descriptions of the machine learning models are not complete or detailed. For a more thorough review, we recommend the textbooks *Hands-On Machine Learning with Scikit-Learn & TensorFlow* [48] and *Deep Learning* [49].

5.1 Baselines

These baselines are simple models which can give us an idea of how are the other models better than a trivial classifier.

5.1.1 k-nearest neighbors with dynamic time warping

k-nearest neighbors (k-NN) is one of the simplest supervised learning algorithms. The main idea is that new examples are of the class of their nearest neighbours. The important parameter is k, which means how many neighbours influence the classification. The metric used when comparing the examples is also very important. The commonly used metrics include, but are not limited to, Minkowski distance, correlation, cosine similarity or dynamic time warping [50].

As for the implementation, we used the `KNeighborsTimeSeriesClassifier` with its default `dtw` metric from the tslearn Python library.

Dynamic time warping

The problem of classifying arrhythmia from ECG is a time series classification problem. Time series classification is a mature research field,

and there is plenty of algorithms trying to solve the problem [51]. However, most of the algorithms are evaluated on small datasets (hundreds of records) from The UCR (University of California, Riverside) Time Series Archive [52]. Dynamic time warping (DTW) is considered to be the standard for measuring time-series similarity. It was shown that a 1-NN classifier with DTW distance is a powerful time-series classification method [51].

5.1.2 Logistic regression

Logistic regression is one of the simpler supervised learning algorithms. For each sample $\mathbf{x}$ and class k, a score $s_k(\mathbf{x})$ is computed:

$$s_k(\mathbf{x}) = \mathbf{x}^\intercal \theta^{(k)}, \tag{5.1}$$

where $\theta^{(k)}$ is the parameter vector of the model for class k and the k^{th} row of the full parameter matrix Θ. Then the predicted class is:

$$\hat{y} = \text{argmax}_k s_k(\mathbf{x}). \tag{5.2}$$

The model is usually trained by minimizing the cross entropy cost function J:

$$J(\Theta) = -\frac{1}{m}\Sigma_{i=1}^{m}\Sigma_{k=1}^{K} y_k^{(i)} \log(\hat{p}_k^{(i)}), \tag{5.3}$$

where $y_k^{(i)}$ is an indicator variable in the case of mutually exclusive classes, m is the total number of samples, K is the total number of classes and $\hat{p}_k^{(i)}$ is the model estimated probability of sample i belonging to class k calculated using the softmax function from the score $s_k^{(i)}$ [53].

We use the multinomial logistic regression class `LogisticRegression` from the scikit-learn package.

5.2 Models with hand-engineered features

These models take the features from Section 2.3 as the input. Random forest and XGBoost classifiers are examples of ensemble machine learning models using the bagging and boosting ensemble techniques, respectively. Ensemble tree models were chosen because of ease of

feature selection – they are not as sensitive to the quality of features. Multilayer perceptron, a type of artificial neural network, was also trained with the hand-engineered features.

Bagging

Bagging (bootstrap aggregating) is the technique of training the same type of model using random subsets of the training set. The sampling of random subsets is done with replacement. By collective voting of the classifiers trained on all subsets, the resulting ensemble model can gain robustness and be better than any single base classifier [54].

5.2.1 Random forest

Random forest is an ensemble model of decision tree classifiers using the bagging technique. A set of decision trees is trained on different random subsets on the data. Moreover, the nodes in the decision trees are not split by the best feature globally, but the best feature selected from a random set of features [54].

Boosting

Boosting is the technique of training multiple weak classifiers sequentially. Every classifier is trying to correct the mistakes that the previous models made. By sequentially improving the results of previous models, the final ensemble classifier is robust machine learning model. One of the boosting algorithms is Gradient Boosting [55]. The underlying weak classifier is usually a decision tree.

5.2.2 XGBoost

XGBoost is an optimized library using the gradient boosting algorithm for training ensemble classifiers. XGBoost stands for Extreme Gradient Boosting [34].

We use the `XGBoostClassifier` which the Python package provides as a scikit-learn compatible classifier.

5.2.3 Multilayer perceptron

Multilayer perceptron is a type of an artificial neural network. It consists of an input layer, one or more hidden layers and an output layer. The neurons are densely connected between two neighbouring layers. Figure 5.1 shows a multilayer perceptron with one hidden layer, three neurons in the input layer, three in the hidden layer and two neurons in the output layer.

A neuron's output h is calculated as a linear combination of its inputs with a bias term followed by a (usually non-linear) activation function σ:

$$h_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{x}^{\intercal}\mathbf{w} + b), \tag{5.4}$$

where $\mathbf{x}$ is the input vector, $\mathbf{w}$ is the weight vector of the input and b is the bias term.

The multilayer perceptron we use has one hidden layer and uses the rectified linear unit (ReLU) activation function:

$$\sigma(x) = \max(0, x) \tag{5.5}$$

To train the network, we need to find the weights which minimize the loss function between the predicted and true outputs. This problem is discussed in Section 6.3.

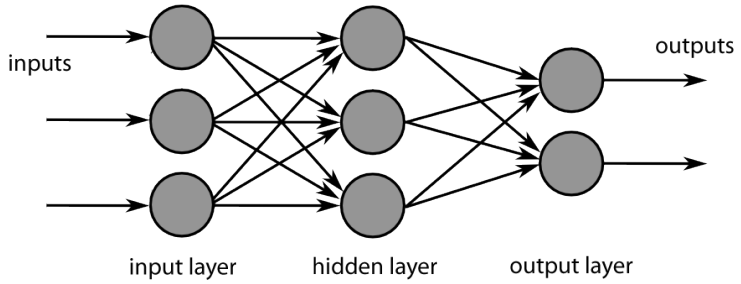


Figure 5.1: A schema of a multilayer perceptron architecture [56].

5.3 Models with direct signal input

These methods take as the input the band-filtered, standardized electrocardiogram signal.

5.3.1 Convolutional networks

Convolutional neural network (CNN) is also a type of artificial neural network. They consist of convolutional layers. Neurons in the convolutional layers are not densely connected to all neurons in the previous layer as in the case of a multilayer perceptron. Instead, the inputs to a single neuron are only from a small region from the previous layer. All neurons in a layer share the same weights, so the learned filters are spatially invariant [57].

CNNs are popular in the field of computer vision, where the filters are two-dimensional. In the case of electrocardiograms, the signal is one-dimensional, and so can be the CNN filters. Two-dimensional filters can also be used on the spectrograms of the signal, which are created by a short-time Fourier transform [58].

ResNet

ResNet (residual network) is a convolutional neural network architecture emerged in 2015 in the field of computer vision. It allowed training convolutional neural networks with significantly increased depth (more than 100 layers) without degradation of performance using deep residual learning.

A model from the ResNet family won the Image Large Scale Visual Recognition Challenge 2015 with a 3.57 % top-five classification error on Imagenet 1k, an image database of more than 14 million labelled images with 1000 different classes. The original ResNet paper [59] contains more information.

We use the ResNet architecture with 18 one-dimensional convolutional layers in the comparison.

5.3.2 Recurrent networks

A recurrent neural network (RNN) is a network which consists of neurons the same as in multilayer perceptrons. The difference is that the output connection of a neuron can be its own input connection. An RNN can take inputs in the form of a sequence and produce a vector – this is called a sequence-to-vector network [60].

Classical RNNs do not work very well on long sequences, as the network becomes very deep and the network can suffer from the

vanishing gradient problem as well as the short-term memory problem [61].

Gated recurrent unit (GRU) is often the solution to both of the stated problems. GRU was first introduced in the context of Natural Language Processing [62], and it works similarly as the LSTM (long short-term memory).

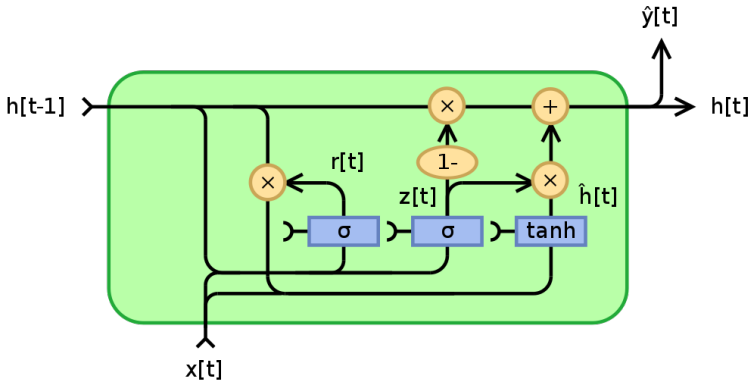


Figure 5.2: Gated recurrent unit [63].

The GRU cell (Figure 5.2) contains gates which control the parts of the state that should be erased and which parts of the input should become a part of the cell's hidden state [61].

We use an architecture of convolutional ResNet backbone followed by a gated recurrent unit (GRU). The backbone extracts 128-dimensional feature vectors which are then fed into the GRU. The GRU layer can then capture the longer-term dependencies between features extracted by the convolutional backbone.

Training the recurrent network on the raw input is an also option, but current recurrent network architectures are not well-suited to train on very long sequences for both performance and computational time reasons.

6 Training

This chapter is concerned about the training of the models. The models are trained on each dataset separately as there is little overlap between the labels in the datasets.

The precise model hyperparameters and architectures are also available in the source code (Appendix A).

6.1 Train-test split

We split all the datasets into train and test partitions which are same for all the models. The models are trained on the train partition, and the final evaluation is done on the test partition, which the model has not seen so that we can evaluate the generalization power of the model. For neural network-based models, we further separate a hold-out validation set out of the training set to evaluate learning performance during training.

The split is 90–10 % of data for the train and test partitions, respectively. When using the validation partition, the final split is 80–10–10 % of train, validation and test partitions.

All the models are trained and evaluated on the same train-test split.

Because the private datasets contain a large number of recordings from the same patients, the train-validation-test partitions are split using `GroupShuffleSplit` from scikit-learn by the telemonitoring device ID. Therefore, it should not happen that the same patient's record is both in the training and test partition.

6.1.1 Cross-validation

Cross-validation is an alternative way of evaluating the generalization performance of a model. The K-fold cross-validation splits a dataset into K partitions (folds), then trains the model K times training on $K - 1$ of the folds and evaluating on the last one. The K scores can be averaged to get the validation score.

For all models except CNN and RNN models, we use 5-fold cross-validation as the validation score. For CNN and RNN, we use the hold-out validation set score.

6.2 Models with hand-engineered features

This family of models is trained using the 5-fold cross-validation on 90 % of the data. All the models use balanced class weights – the fewer samples the class has, the larger the class weight is. The parameters used were found by a cross-validated grid search on the training partition of the CINC2017 dataset (Section 4.1). If a parameter of the classifier is not stated, we use the default value provided by the library.

Logistic regression

The missing features are imputed by the mean of the column and then standardized (Section 2.2.3).

The regularization parameter $C = 100.0$ is used.

Random forest

The missing features were imputed but not scaled, as random forests can deal with unscaled features easily.

The `max_depth` of the trees in the forest is set to 20, the minimum number of samples at a leaf node (`min_samples_leaf`) is 5, and the number of estimators in the forest is 1000.

XGBoost

We use a learning rate `eta` of 0.1, a minimum loss reduction (`gamma`) of 0.1. The maximum allowed depth (`max_depth`) of a tree is 7. The `min_child_weight` parameter is 4, number of estimators is 1000, the same as in the random forest classifier. The `subsample` parameter is set to 0.8.

MLP

The default learning rate of 0.001 with the Adam optimizer [64] is used for minimizing the cross-entropy loss function. A batch size of 128 is used.

Early stopping is used – when the validation score is not improving for ten consecutive epochs, the training stops. Early stopping can help the generalization power of the neural network [65].

6.3 Models with direct signal input

6.3.1 Data augmentation

Data augmentation techniques help us to increase the model's training dataset size by introducing transforms that do not change the label of the recording. We used random flipping of the sign of the signal, as it commonly happens that the patient switches the electrodes, and the resulting signal is inverted.

Random cropping of the signal would be possible if we had the information about the location of the arrhythmia. As the information is not available, random cropping could crop out the arrhythmic part, and the label would be wrong. Thus, we decided not to use random cropping.

ResNet

We use a ResNet implementation from the torchvision library used for computer vision tasks. The convolutional layers are rewritten to be one-dimensional. We use an 18-layer and 50-layer convolutional residual networks. We train the ResNets for 20 epochs and after each epoch measure the validation score. We take the model with the best validation score as the final trained model.

We use a batch size of 32 for the `ResNet18` model. The batch size is dependent mainly on the available CUDA memory. We use a cross-entropy loss function, the Adam optimizer with a learning rate of 0.0003. For regularization of the networks, we use a weight decay of 0.0001. We do not use weight decay for the batch normalization parameters, as it is shown to degrade the performance of learning [66].

We use gradient norm clipping [67], which helps us with unstable gradients problem.

CnnGru

For training the `CnnGru` classifier, we use learning hyperparameters same as the ResNet ones – Adam, a learning rate of 0.0003, weight decay of 0.0001, cross-entropy loss. We train with a batch size of 32 for ten epochs and again save the model trained on the epoch with the highest validation score.

k-NN with DTW

The `n_neighbors` parameter is set to 1, as we want to use the 1-nearest neighbour classifier. However, the evaluation takes too much time to calculate scores on even the smallest of our datasets. Therefore, we do not have the scores for this baseline. For our use case, the inference time is crucial, so this classifier is apparently not applicable.

6.4 Model persistence

To save the models for future evaluation, we used the PyTorch and joblib libraries, which both use the pickle module under the hood. This solution is not perfect because the saved models have no guarantee of being transferable to the future versions of the machine learning libraries. Our mitigation is to use the reproducible Python virtual environment which has exactly the versions of libraries we used. Also, the exported ONNX models (Section 3.5) are independent of any library and should be usable on any platform supporting the ONNX Runtime.

7 Evaluation

After training of the models, we compare them against each other. In this chapter, we will describe the F_1 score metric, which we use in the final comparison table and then discuss the final results.

7.1 Metrics

We need to evaluate the performance different models on the same data. For a binary classification problem with two classes, say *Pos* and *Neg*, a confusion matrix has two rows and two columns. True positives are the examples labelled as *Pos* and classified as *Pos*, true negatives are *Neg* classified as *Neg*, false positives are *Neg* classified as *Pos*, and false negatives are *Pos* classified as *Neg*.

Table 7.1: A binary classification task confusion matrix.

True Positives (TP)	True Negatives (TN)
False Positives (FP)	False Negatives (FN)

A confusion matrix for a multi-class problem has shape $N \times N$, where N is the number of classes. A single score instead of a full matrix is preferable for comparing the classifiers' performance. We calculate the F_1 score for all classifiers and use it for a rough comparison.

7.1.1 F1 score

The F_1 score is the harmonic mean of precision (positive predictive value) and recall (sensitivity) [68]:

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}, \tag{7.1}$$

where

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \text{ and recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{7.2}$$

The above definition is valid for binary classification. With multi-class classification, we do macro averaging. That means, we take a

separate F_1 score for each class versus all the others as a binary problem; then, we get the resulting macro F_1 score as the mean of all the class scores.

F_1 score is a simple way to compare classifiers. It favours classifiers with similar precision and recall; this is not always what we want when our task is classifying heart pathologies.

We use F_1 score for simplicity, but some types of arrhythmia are more dangerous than the others. For example, ventricular tachycardia is life-threatening as it can lead to deadly ventricular fibrillation. Therefore, a metric that takes into account the severity of different classes could prove useful in future work.

7.2 Results

In Table 7.3, we can see the F_1 scores, macro averaged over all classes of the datasets. Our best trained model on the CINC2017 dataset is the `MLPClassifier`, the best model on the CPSC2018 dataset is the `ResNet18Classifier` and on the private dataset it is the `CnnGruClassifier`.

On the CINC2017 dataset with four classes and thousands of recordings, the methods based on hand-engineered features and on the direct signal are comparable. However, when the dataset has more different classes or is bigger in size, the automated feature extractors can provide better performance and good inference time.

As the private dataset is the most similar to the telemonitoring data, we will likely use the methods trained on the direct signal. The hand-engineered feature methods could probably be tuned and have better results with more domain knowledge and expert-designed features.

Table 7.3: The F1 macro averaged scores on the test partitions.

Model	CINC2017	CPSC2018	Private
`LogisticRegression`	0.6419	0.4532	0.5773
`RandomForestClassifier`	0.6513	0.4323	0.6153
`XGBClassifier`	0.6480	0.4824	0.6345
`MLPClassifier`	**0.6824**	0.4840	0.6290
`ResNet18Classifier`	0.6539	**0.6296**	0.7871
`CnnGruClassifier`	0.6470	0.5432	**0.7923**

7.3 Inference and deployment

The models are automatically converted by the program to the Open Neural Network Exchange format (Section 3.5). Exported models can be loaded in any application that supports the ONNX Runtime.

8 Conclusion, Future Works

In this work, we train seven different models on three different ECG databases and compare their performance between each other. We find that at least in our experiment, both hand-crafted features and the neural network feature extractors each have both its positive and negative aspects.

The next step is the integration of the model to the ECG processing solution running on the telemonitoring server. As the solution is ready to work with ONNX models, the integration should not be problematic.

An another direction of study could be the ensemble model of both the feature processing approaches we tried.

The solution can also be expanded with more datasets to run the comparison on, for example, the PTB-XL database (Section 4.4.3) or the PhysioNet Computing in Cardiology Challenge 2020 dataset [69].

Bibliography

1. LILLY, Leonard S. Chapter 11. Mechanisms of Cardiac Arrhythmias. In: *Pathophysiology of heart disease: a collaborative project of medical students and faculty*. Wolters Kluwer, 2016, p. 268. ISBN 1605477230.
2. CLIFFORD, Gari; LIU, Chengyu; MOODY, Benjamin; LEHMAN, Li-wei; SILVA, Ikaro; LI, Qiao; JOHNSON, Alistair; MARK, Roger. AF Classification from a Short Single Lead ECG Recording: the Physionet Computing in Cardiology Challenge 2017. In: *2017 Computing in Cardiology Conference (CinC)*. Computing in Cardiology, 2017. Available from DOI: 10.22489/cinc.2017.065-469.
3. BLAUS, Bruce. *An illustration depicting a Holter monitor.* 2017. Available also from: https://commons.wikimedia.org/wiki/File:Holter_Monitor.png.
4. MEYSTRE, Stephane. The Current State of Telemonitoring: A Comment on the Literature. *Telemedicine and e-Health*. 2005, vol. 11, no. 1, pp. 63–69. Available from DOI: 10.1089/tmj.2005.11.63.
5. CORDISCO, Marie Elena; BENIAMINOVITZ, Ainat; HAMMOND, Kim; MANCINI, Donna. Use of telemonitoring to decrease the rate of hospitalization in patients with severe congestive heart failure. *The American Journal of Cardiology*. 1999, vol. 84, no. 7, pp. 860–862. Available from DOI: 10.1016/s0002-9149(99)00452-x.
6. ATKIELSKI, Anthony. *ECG of a heart in normal sinus rhythm.* 2007. Available also from: https://commons.wikimedia.org/wiki/File:SinusRhythmLabels.svg.
7. DUBIN, Dale. Chapter 1: Basic Principles. In: *Rapid Interpretation of EKG's, Sixth Edition*. Tampa, Fla: Cover Pub. Co, 2000, pp. 14–27. ISBN 0912912065.
8. LIP, Gregory Y. H. et al. Atrial fibrillation. *Nature Reviews Disease Primers*. 2016, vol. 2, no. 1, pp. 16016. ISSN 2056-676X. Available from DOI: 10.1038/nrdp.2016.16.

9. ODUTAYO, Ayodele; WONG, Christopher X; HSIAO, Allan J; HOPEWELL, Sally; ALTMAN, Douglas G; EMDIN, Connor A. Atrial fibrillation and risks of cardiovascular disease, renal disease, and death: systematic review and meta-analysis. *BMJ*. 2016, pp. i4482. Available from DOI: 10.1136/bmj.i4482.

10. DUBIN, Dale. Chapter 5: Rhythm, Part I. Premature Ventricular Contraction (PVC). In: *Rapid Interpretation of EKG's, Sixth Edition*. Tampa, Fla: Cover Pub. Co, 2000, p. 135. ISBN 0912912065.

11. DUBIN, Dale. Chapter 5: Rhythm, Part I. Runs of Ventricular Tachycardia. In: *Rapid Interpretation of EKG's, Sixth Edition*. Tampa, Fla: Cover Pub. Co, 2000, p. 156. ISBN 0912912065.

12. NOLLE, Floyd M; BOWSER, Richard W. *Creighton University Ventricular Tachyarrhythmia Database*. physionet.org, 1992. Available from DOI: 10.13026/C2X59M.

13. OPPENHEIM, Alan. *Discrete-time signal processing*. Upper Saddle River: Pearson, 2010. ISBN 0131988425.

14. OPPENHEIM, Alan. Chapter 8: The Discrete Fourier Transform. In: *Discrete-time signal processing*. Upper Saddle River: Pearson, 2010, pp. 541–600. ISBN 0131988425.

15. OPPENHEIM, Alan. Chapter 7: Filter Design Techniques. In: *Discrete-time signal processing*. Upper Saddle River: Pearson, 2010, pp. 439–511. ISBN 0131988425.

16. KWON, Ohhwan; JEONG, Jinwoo; KIM, Hyung Bin; KWON, In Ho; PARK, Song Yi; KIM, Ji Eun; CHOI, Yuri. Electrocardiogram Sampling Frequency Range Acceptable for Heart Rate Variability Analysis. *Healthcare Informatics Research*. 2018, vol. 24, no. 3, pp. 198. Available from DOI: 10.4258/hir.2018.24.3.198.

17. *scipy.signal.resample_poly — SciPy v1.5.0 Reference Guide*. 2020. Available also from: https://docs.scipy.org/doc/scipy-1.5.0/reference/generated/scipy.signal.resample_poly.html.

18. GAO, Hongxiang; LIU, Chengyu; WANG, Xingyao; ZHAO, Lina; SHEN, Qin; NG, E. Y. K.; LI, Jianqing. An Open-Access ECG Database for Algorithm Evaluation of QRS Detection and Heart Rate Estimation. *Journal of Medical Imaging and Health Informatics*.

2019, vol. 9, no. 9, pp. 1853–1858. Available from DOI: 10.1166/jmihi.2019.2800.

19. PAN, Jiapu; TOMPKINS, Willis J. A Real-Time QRS Detection Algorithm. *IEEE Transactions on Biomedical Engineering*. 1985, vol. BME-32, no. 3, pp. 230–236. Available from DOI: 10.1109/tbme.1985.325532.

20. PLESINGER, Filip; ANDRLA, Petr; VISCOR, Ivo; HALAMEK, Josef; BULKOVA, Veronika; JURAK, Pavel. Shape Analysis of Consecutive Beats May Help in the Automated Detection of Atrial Fibrillation. In: *2018 Computing in Cardiology Conference (CinC)*. Computing in Cardiology, 2018. Available from DOI: 10.22489/cinc.2018.036.

21. DUBIN, Dale. Chapter 6: Rhythm, Part II. 1°AV Block. In: *Rapid Interpretation of EKG's, Sixth Edition*. Tampa, Fla: Cover Pub. Co, 2000, p. 178. ISBN 0912912065.

22. ACHARYA, U. Rajendra; JOSEPH, K. Paul; KANNATHAL, N.; LIM, Choo Min; SURI, Jasjit S. Heart rate variability: a review. *Medical & Biological Engineering & Computing*. 2006, vol. 44, no. 12, pp. 1031–1051. Available from DOI: 10.1007/s11517-006-0119-0.

23. *Heart Rate Variability (HRV) — NeuroKit 0.0.39 documentation*. 2020. Available also from: https://neurokit2.readthedocs.io/en/latest/examples/hrv.html.

24. VIRTANEN, Pauli et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*. 2020, vol. 17, pp. 261–272. Available from DOI: https://doi.org/10.1038/s41592-019-0686-2.

25. MAKOWSKI, Dominique; PHAM, Tam; LAU, Zen J.; BRAMMER, Jan C.; LESPINASSE, François; PHAM, Hung; SCHÖLZEL, Christopher; S H CHEN, Annabel. *NeuroKit2: A Python Toolbox for Neurophysiological Signal Processing*. Zenodo, 2020. Available from DOI: 10.5281/ZENODO.3597887.

26. PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, vol. 12, pp. 2825–2830.

27. PASZKE, Adam et al. PyTorch: An Imperative Style, High Performance Deep Learning Library. In: WALLACH, H.; LAROCHELLE, H.; BEYGELZIMER, A.; D'ALCHÉ-BUC, F.; FOX, E.; GARNETT, R. (eds.). *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

28. BAI, Junjie; LU, Fang; ZHANG, Ke, et al. *ONNX: Open Neural Network Exchange* [`https://github.com/onnx/onnx`]. GitHub, 2019.

29. CORPORATION, Microsoft. *ONNXMLTools*. 2020. Available also from: `https://github.com/onnx/onnxmltools`.

30. REBACK, Jeff et al. *pandas-dev/pandas: Pandas 1.0.5*. Zenodo, 2020. Available from DOI: `10.5281/ZENODO.3898987`.

31. COSTA-LUIS, Casper O. da. tqdm: A Fast, Extensible Progress Meter for Python and CLI. *Journal of Open Source Software*. 2019, vol. 4, no. 37, pp. 1277. Available from DOI: `10.21105/joss.01277`.

32. WALT, Stéfan van der; COLBERT, S Chris; VAROQUAUX, Gaël. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering*. 2011, vol. 13, no. 2, pp. 22–30. Available from DOI: `10.1109/mcse.2011.37`.

33. TAVENARD, Romain et al. Tslearn, A Machine Learning Toolkit for Time Series Data. *Journal of Machine Learning Research*. 2020, vol. 21, no. 118, pp. 1–6. Available also from: `http://jmlr.org/papers/v21/20-091.html`.

34. CHEN, Tianqi; GUESTRIN, Carlos. XGBoost: A Scalable Tree Boosting System. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco, California, USA: ACM, 2016, pp. 785–794. KDD '16. ISBN 978-1-4503-4232-2. Available from DOI: `10.1145/2939672.2939785`.

35. INC., Plotly Technologies. *Collaborative data science*. Montreal, QC: Plotly Technologies Inc., 2015. Available also from: `https://plot.ly`.

36. WAGNER, Patrick; STRODTHOFF, Nils; BOUSSELJOT, Ralf-Dieter; KREISELER, Dieter; LUNZE, Fatima I.; SAMEK, Wojciech; SCHAEFFTER, Tobias. PTB-XL, a large publicly available electrocardiography dataset. *Scientific Data*. 2020, vol. 7, no. 1. Available from DOI: 10.1038/s41597-020-0495-6.

37. TEIJEIRO, Tomas; GARCIA, Constantino A.; CASTRO, Daniel; FÉLIX, Paulo. Arrhythmia Classification from the Abductive Interpretation of Short Single-Lead ECG Records. In: *2017 Computing in Cardiology Conference (CinC)*. Computing in Cardiology, 2017. Available from DOI: 10.22489/cinc.2017.166-054.

38. ZABIHI, Morteza; RAD, Ali Bahrami; KATSAGGELOS, Aggelos K.; KIRANYAZ, Serkan; NARKILAHTI, Susanna; GABBOUJ, Moncef. Detection of Atrial Fibrillation in ECG Hand-held Devices Using a Random Forest Classifier. In: *2017 Computing in Cardiology Conference (CinC)*. Computing in Cardiology, 2017. Available from DOI: 10.22489/cinc.2017.069-336.

39. MAHAJAN, Ruhi; KAMALESWARAN, Rishikesan; HOWE, John Andrew; AKBILGIC, Oguz. Cardiac Rhythm Classification from a Short Single Lead ECG Recording via Random Forest. In: *2017 Computing in Cardiology Conference (CinC)*. Computing in Cardiology, 2017. Available from DOI: 10.22489/cinc.2017.179-403.

40. DATTA, Shreyasi et al. Identifying Normal, AF and other Abnormal ECG Rhythms using a Cascaded Binary Classifier. In: *2017 Computing in Cardiology Conference (CinC)*. Computing in Cardiology, 2017. Available from DOI: 10.22489/cinc.2017.173-154.

41. HONG, Shenda; WU, Meng; ZHOU, Yuxi; WANG, Qingyun; SHANG, Junyuan; LI, Hongyan; XIE, Junqing. ENCASE: an ENsemble ClASsifiEr for ECG Classification Using Expert Features and Deep Neural Networks. In: *2017 Computing in Cardiology Conference (CinC)*. Computing in Cardiology, 2017. Available from DOI: 10.22489/cinc.2017.178-245.

42. LIU, Feifei et al. An Open Access Database for Evaluating the Algorithms of Electrocardiogram Rhythm and Morphology Abnormality Detection. *Journal of Medical Imaging and Health Informatics*. 2018, vol. 8, no. 7, pp. 1368–1373. Available from DOI: 10.1166/jmihi.2018.2442.

43. CHEN, Tsai-Min; HUANG, Chih-Han; SHIH, Edward S.C.; HU, Yu-Feng; HWANG, Ming-Jing. Detection and Classification of Cardiac Arrhythmias by a Challenge-Best Deep Learning Neural Network Model. *iScience*. 2020, vol. 23, no. 3, pp. 100886. Available from DOI: 10.1016/j.isci.2020.100886.

44. MOODY, George B; MARK, Roger G. *MIT-BIH Arrhythmia Database*. physionet.org, 1992. Available from DOI: 10.13026/C2F305.

45. *MIT-BIH Arrhythmia Database - Google Scholar*. 2020. Available also from: https://scholar.google.com/scholar?q=MIT-BIH+Arrhythmia+Database.

46. BOUSSELJOT, Ralf-Dieter; KREISELER, D; SCHNABEL, A. *The PTB Diagnostic ECG Database*. physionet.org, 2004. Available from DOI: 10.13026/C28C71.

47. STRODTHOFF, Nils; WAGNER, Patrick; SCHAEFFTER, Tobias; SAMEK, Wojciech. *Deep Learning for ECG Analysis: Benchmarks and Insights from PTB-XL*. 2020. Available from arXiv: 2004.13701 [cs.LG].

48. GÉRON, Aurélien. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019. ISBN 1492032646. Available also from: https://www.xarg.org/ref/a/1492032646/.

49. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

50. CUNNINGHAM, Padraig; DELANY, Sarah Jane. *k-Nearest Neighbour Classifiers: 2nd Edition (with Python examples)*. 2020. Available from arXiv: 2004.04523 [cs.LG].

51. BAGNALL, Anthony; LINES, Jason; BOSTROM, Aaron; LARGE, James; KEOGH, Eamonn. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*. 2016, vol. 31, no. 3, pp. 606–660. Available from DOI: 10.1007/s10618-016-0483-9.

52. DAU, Hoang Anh; BAGNALL, Anthony; KAMGAR, Kaveh; YEH, Chin-Chia Michael; ZHU, Yan; GHARGHABI, Shaghayegh; RATANAMAHATANA, Chotirat Ann; KEOGH, Eamonn. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*. 2019, vol. 6, no. 6, pp. 1293–1305. Available from DOI: 10.1109/jas.2019.1911747.

53. GÉRON, Aurélien. Chapter 4: Training Models. Logistic Regression. In: *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019, pp. 142–150. ISBN 1492032646. Available also from: https://www.xarg.org/ref/a/1492032646/.

54. GÉRON, Aurélien. Chapter 7: Ensemble Learning and Random Forests. Bagging and Pasting, Random Forests. In: *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019, pp. 192–199. ISBN 1492032646. Available also from: https://www.xarg.org/ref/a/1492032646/.

55. GÉRON, Aurélien. Chapter 7: Ensemble Learning and Random Forests. Boosting. In: *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019, pp. 203–208. ISBN 1492032646. Available also from: https://www.xarg.org/ref/a/1492032646/.

56. CHRISLB. *Diagram of a multi-layer feedforward artificial neural network.* 2012. Available also from: https://commons.wikimedia.org/wiki/File:MultiLayerNeuralNetworkBigger_english.png.

57. *Course materials and notes for Stanford class CS231n: Convolutional Neural Networks for Visual Recognition.* 2020. Available also from: https://cs231n.github.io/convolutional-networks/.

58. HUANG, Jingshan; CHEN, Binqiang; YAO, Bin; HE, Wangpeng. ECG Arrhythmia Classification Using STFT-Based Spectrogram and Convolutional Neural Network. *IEEE Access*. 2019, vol. 7, pp. 92871–92880. Available from DOI: 10.1109/access.2019.2928017.

59. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016. Available from DOI: 10.1109/cvpr.2016.90.

60. GÉRON, Aurélien. Chapter 15: Processing Sequences Using RNNs and CNNs. Recurrent Neurons and Layers. In: *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019, pp. 497–501. ISBN 1492032646. Available also from: https://www.xarg.org/ref/a/1492032646/.

61. GÉRON, Aurélien. Chapter 15: Processing Sequences Using RNNs and CNNs. Handling Long Sequences. In: *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019, pp. 511–520. ISBN 1492032646. Available also from: https://www.xarg.org/ref/a/1492032646/.

62. CHO, Kyunghyun; MERRIËNBOER, Bart van; GULCEHRE, Caglar; BAHDANAU, Dzmitry; BOUGARES, Fethi; SCHWENK, Holger; BENGIO, Yoshua. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1724–1734. Available from DOI: 10.3115/v1/D14-1179.

63. BLAD, John Erling. *Gated Recurrent Unit, fully gated version.* 2018. Available also from: https://commons.wikimedia.org/wiki/File:Gated_Recurrent_Unit,_base_type.svg.

64. KINGMA, Diederik P.; BA, Jimmy. *Adam: A Method for Stochastic Optimization*. 2014. Available from arXiv: 1412.6980 [cs.LG].

65. CARUANA, Rich; LAWRENCE, Steve; GILES, Lee. Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping. In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. Denver, CO: MIT Press, 2000, pp. 381–387. NIPS'00.

66. LAARHOVEN, Twan van. *L2 Regularization versus Batch and Weight Normalization*. 2017. Available from arXiv: 1706.05350 [cs.LG].

67. PASCANU, Razvan; MIKOLOV, Tomas; BENGIO, Yoshua. On the Difficulty of Training Recurrent Neural Networks. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. Atlanta, GA, USA: JMLR.org, 2013, III–1310–III–1318. ICML'13.

68. GÉRON, Aurélien. Chapter 3: Classification. Performance Measures. In: *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019, pp. 88–93. ISBN 1492032646. Available also from: `https://www.xarg.org/ref/a/1492032646/`.

69. PEREZ ALDAY, Erick Andres; GU, Annie; SHAH, Amit; LIU, Chengyu; SHARMA, Ashish; SEYEDI, Salman; BAHRAMI RAD, Ali; REYNA, Matthew; CLIFFORD, Gari. *Classification of 12-lead ECGs: the PhysioNet - Computing in Cardiology Challenge 2020*. PhysioNet, 2020. Available from DOI: `10.13026/F4AB-0814`.

A Source code

The source code for all experiments is available as a `.zip` file in the archive of the thesis at `https://is.muni.cz/th/ybs7e/`. It is also publicly available on GitHub at `https://github.com/adamivora/ecg_arrhythmia_classification`. A `README.md` file is included in the repository with the instructions to run the experiments.

The archive version contains the extracted feature Pandas data frames and trained models, so only dataset download is needed to run the comparison. The GitHub version does not contain any preprocessed data, and the full run through the pipeline can take more than three hours.

B Complete list of hand-engineered features

Table B.1 presents the full list of the features used to train models with hand-engineered features, along with descriptions of the features. The total number of numeric columns used for training is 113 – 24 HRV features and 89 segment and signal features.

The features of type `[float]` are actually a list of 11 statistical features listed in Table B.4. The HRV feature descriptions are based on the Neurokit2 documentation [23].

Table B.1: Complete list of extracted features.

Name	Description	Type
Segment and signal features:		
`N_QRS`	Number of QRS complexes detected.	`integer`
`P_Stats`	Statistics of the P segment length.	`[float]`
`R_Stats`	Statistics of the R segment length.	`[float]`
`T_Stats`	Statistics of the T segment length.	`[float]`
`PR_Stats`	Statistics of the PR intervals of the same beat.	`[float]`
`RR_Stats`	Statistics of the consecutive RR intervals.	`[float]`
`RT_Stats`	Statistics of the RT intervals of the same beat.	`[float]`
`QRS_Stats`	Consecutive beat correlation statistics.	`[float]`
`Signal_Stats`	Statistics of the band-pass preprocessed signal.	`[float]`
Time-domain HRV features:		
`HRV_RMSSD`	Root mean square of successive differences between RR intervals.	`float`
`HRV_MeanNN`	Mean of the RR intervals.	`float`
`HRV_SDNN`	Standard deviation of RR intervals.	`float`

HRV_SDSD	Standard deviation of successive differences between RR intervals.	float
HRV_CVNN	Calculated from the features HRV_SDNN / HRV_MeanNN.	float
HRV_CVSD	HRV_RMSSD / HRV_MeanNN.	float
HRV_MedianNN	Median of the absolute values of successive differences between RR intervals.	float
HRV_MadNN	Median absolute deviation of the RR intervals.	float
HRV_MCVNN	HRV_MadNN / HRV_MedianNN.	float
HRV_TINN	An approximation of the RR interval distribution.	float
HRV_HTI	The number of RR intervals divided by the height of histogram of RR intervals.	float
HRV_pNN20	Number of pairs of successive RR intervals that differ by more than 20 ms divided by total number of RR intervals.	float
HRV_pNN50	Number of pairs of successive RR intervals that differ by more than 50 ms divided by total number of RR intervals.	float
Frequency-domain HRV features:		
HRV_HF	SPD of the high frequency band.	float
HRV_VHF	SPD of the very high frequency band.	float
HRV_HFn	Normalized high frequency (HRV_HF / total power).	float
HRV_LnHF	Log transformation of HRV_HF.	float
Non-linear HRV features:		
HRV_SD1	Beat-to-beat variability.	float

HRV_SD2	Index of long-term RR interval fluctuations.	float
HRV_SD2SD1	The ratio of HRV_SD2 / HRV_SD1.	float
HRV_CSI	Cardiac Sympathetic Index.	float
HRV_CSI_Modified	Modified Cardiac Sympathetic Index.	float
HRV_CVI	Cardial Vagal Index.	float
HRV_SampEn	Sample entropy.	float

Table B.4: List of statistics used.

Name	Description
min	The minimum.
max	The maximum.
mean	The arithmetic mean.
median	The median (50^{th} percentile).
perc25	The 25^{th} percentile (first quartile).
perc75	The 75^{th} percentile (third quartile).
perc99	The 99^{th} percentile.
range	The range (max − min).
std	The standard deviation.
skew	The sample skewness (symmetry of the distribution).
kurtosis	The fourth standardized moment (tailedness of the distribution).