CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF MASTER'S THESIS

Title:	Identification of Pre-Arrhythmogenic Features in ECG and Other Data Using Machine Learning
Student:	Bc. Dmitriy Bobir
Supervisor:	RNDr. Jiří Kroc, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2017/18

Instructions

The thesis is focused on localization of pre-arrhythmogenic features that are present in electrocardiograms (ECGs) a long time before the life-threatening arrhythmia occurs. ECG data from an animal model is used. Information hidden in ECGs that is not visible by a naked eye (even by cardiologists) can be revealed using special mathematical techniques that measure information content often used within complex systems. The outcome of this work can have a great impact on clinical research, and if successful, it can help to save many lives.

The input data are entropy curves (EC) (about 20-30) computed for the same ECG curve using different parameters (50 ECG curves in total). The goal of the work is to localize early markers on curves (or their combinations) enabling to discriminate ECs of rabbits that will have arrhythmia from those without it. The intention is to test a whole range of ML techniques against those data and find the most effective discriminating technique(s).

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague January 9, 2017

Czech Technical University in Prague

Faculty of Information Technology

Department of Theoretical Computer Science

Master's thesis

Identification of Pre-Arrhythmogenic Features in ECG and Other Data Using Machine Learning

Bc. Dmitriy Bobir

Supervisor: RNDr. Jiří Kroc, Ph.D.

9th May 2017

Acknowledgements

I would like to express my sincere gratitude for the guidance and support received from my supervisor RNDr. Jiří Kroc, Ph.D. and to Ing. Václav Chudáček, Ph.D. for consulting in data analysis sphere.

I am glad to say a sincere thank you to my dear ones, my family, who supported me throughout my studies and showed their love and care.

Declaration

In Prague on 9th May 2017 . . . .

Citation of this thesis

Bobir, Dmitriy. *Identification of Pre-Arrhythmogenic Features in ECG and Other Data Using Machine Learning.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Diplomová práce se zabývá lokalizací pre-arytmogenních markerů, které jsou přítomny v elektrokardiogramech (EKG) dlouho před tím, až jednu hodinu, než nastoupí život ohrožující arytmie. Jsou použita EKG data ze zvířecího modelu (králík). Informace skryté v EKG neviditelné pouhým okem (a to i kardiologem) mohou být odhaleny použitím speciálních matematických metod, které měří informační obsah a jsou často používány v komplexních systémech. Různé algoritmy a techniky strojového učení jako třeba Support Vector Machine, k-nejblizších sousedů, Random Forest, logistická regrese, vícevrstvý perceptron, Group Method of Data Handling a ensemble učení budou testovány a výsledky diskutovány. Výstup této práce může silně ovlivnit klinický výzkum, a pokud bude práce úspěšná, pak může pomoci zachránit mnoho životů.

Klíčová slova EKG, časové řady, arytmie, zvířecí model, predikce, klasifikace, strojové učení, Random Forest, Support Vector Machine, logistická regrese, k-nejblizších sousedů, vícevrstvý perceptron, group method of data handling, ensemble učení

Abstract

The thesis is focused on localization of pre-arrhythmogenic features that are present in electrocardiograms (ECGs) a long time, up to one hour, before the life-threatening arrhythmias occur. ECG data from an animal model (rabbits) are used. Information hidden in ECGs, which is not visible by a naked eye (even by cardiologists), can be revealed using special mathematical techniques that are measuring information content and that are often used within complex systems. Different machine learning algorithms and approaches such as Support Vector Machine, k-nearest neighbors, Random Forest, logistic regression, Multilayer Perceptron, Group Method of Data Handling and ensemble learning will be tested and the results will be discussed. The outcome of this work can have a great impact on clinical research, and if successful, it can help to save many lives.

Keywords ECG, time series, arrhythmia, animal model, prediction, classification, machine learning, Random Forest, Support Vector Machine, logistic regression, k-nearest neighbors, multilayer perceptron, group method of data handling, ensemble learning

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This section describes history of data analysis resulted in machine learning, presents available input data and formulates goals of this work. For the advanced readers, there is a subsection 1.4 with direct links to the experiments.

This work is an integral part of research project on arrhythmia prediction using Entropy Curves (ECs). Detailed description and reasoning for deriving ECs was beyond this scope of this work. This work provides more accurate predictions in data analysis using machine learning algorithms compared to using ordinary statistical tools. The forthcoming project paper would include theoretical base for ECs computation and would rely on this work for the first-of-the-kind of ECs using machine learning algorithms.

1.1 History of data analysis and machine learning

People started analyzing relationships between input and output data (results) long time ago. To represent such relationships, a linear function and later non-linear functions were used. Following the development of mathematical apparatus, researchers started using available tools to analyze data by approximating into some function, which reflected the data characteristics in the best way. This approach was called *regression analysis* and further facilitated the progress in data analytics, which ultimately resulted in the advanced statistic methods with the subsequent creation of machine learning algorithms. However, even with the establishment of the relevant theoretical base, application of such algorithms to real data was impossible due to limited capacity of human mind to process such complex data. The golden era of machine learning started with the progress in computer technology development, which allowed using machine learning algorithms for real data analysis.

Nowadays machine learning algorithms are widely implemented in many research spheres, including health care (e.g. analysis of ECGs).

1.2 Overview of input data

The input data are entropy curves (ECs) (about 16) computed for each ECG curve using different parameters. The most important of these parameters is L parameter with 16 different values. There are 37 different ECG curves in total: 13 ECGs of the non-arrhythmogenic rabbits and 24 ECGs of the arrhythmogenic rabbits. The rabbits had *anesthesia*. All these data are *real measurements* of the *real* rabbits. However, due to the abnormally high amount of data obtained from ECGs, the original values of ECs computed for the ECGs were averaged within five seconds. Figure 1.1 provides an example of several ECs of arrhythmogenic and non-arrhythmogenic rabbits. Black vertical lines indicate the starting moment when the rabbits receive drug infusions (further referred to as Drug Infusion Initiation or DII). The used drug induces emergence of arrhythmia in later times. Obviously, there are no more infusions after rabbit's death (in case of Figure with arrhythmogenic rabbit, the rabbit became dead after the third DII).

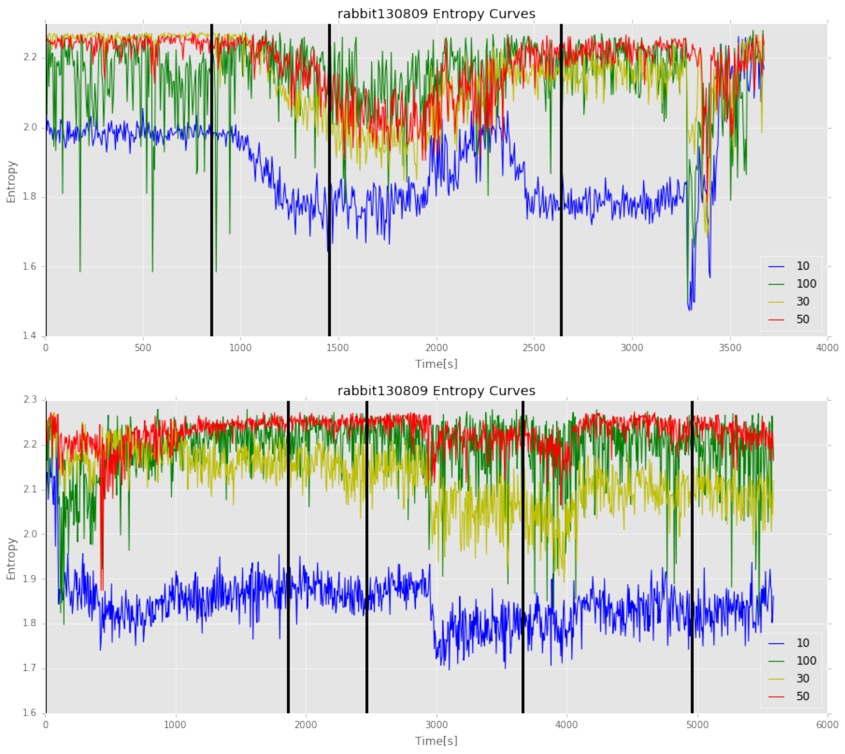


Figure 1.1: Example of ECs with parameters 10, 30, 50 and 100 of (Top) the arrhythmogenic rabbit (Bottom) the non-arrhythmogenic rabbit. Black vertical lines indicate DII.

Different number of DIIs was used to induce arrhythmia as some rabbits got arrhythmia and others didn't. It is important to mention that *all rabbits* had DIIs, even non-arrhythmogenic rabbits. However, those rabbits without arrhythmia could survive. Further in the text, dead and alive rabbits are equivalent to arrhythmogenic and non-arrhythmogenic rabbits, i.e. those rabbits which suffered from deadly arrhythmias and those which did not.

Each rabbit had so-called *control interval* before the first DII was applied. These control intervals have different lengths, but approximately equal to *ten minutes* (600 seconds). During the period of the control interval, all rabbits were at rest and calm.

1.3 Goals

The goal of the work is to localize early markers on curves (or their combinations) enabling to discriminate Entropy Curves (ECs) of rabbits that will have arrhythmia from those without it. The intention is to test a whole range of ML techniques against those data and find the most effective discriminating technique(s). All measurements were performed on the *real* rabbits.

1.4 Advanced readers

The readers not interested in the provided theoretical background can proceed to the description of the conducted experiments 5, achieved results 6, discussion of the achieved results 7 and the conclusion of this work 7.

CHAPTER 2

Theoretical background

This chapter introduces and describes common terms of time series analysis and machine learning which are important for this work. Then a review of the used machine learning algorithms and approaches are provided.

2.1 Basic concepts of time series analysis

2.1.1 Stationarity of time series

The times series is *stationary* if its statistical properties (variance, mean etc.) do not change with time. A good explanation of the time series stationarity may be found in [10][1]. Figure below provides the examples of stationary and non-stationary time series.

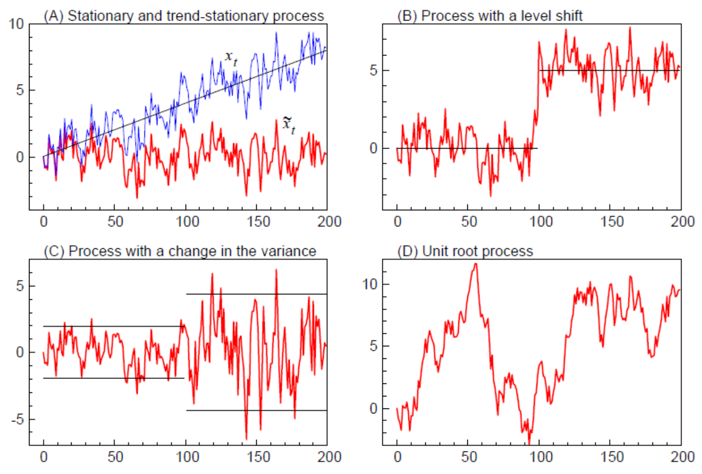


Figure 2.1: Stationary and non-stationary time series [1].

2.1.2 Differencing of time series

In general, *differencing* of time series is used to stabilize the mean and is often used with log transformation that stabilizes the variance of time series.

$$y_t' = y_t - y_{t-1}$$

The differenced time series is the change between consecutive values in the original time series, and is expressed as:

$$y_t'' = y_t' - y_{t-1}' = (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) = y_t - 2y_{t-1} - y_{t-2}$$

This expression defines *first order differencing.* Sometimes, the first order difference is not enough and thus, the *second order differencing* is introduced as follows:

In practice, it is almost never necessary to use higher level differences.

The main reason why transformations like differencing and log are commonly used is the fact that the majority of statistical forecasting methods are based on the assumption that the time series is *approximately stationary* (real data may almost *never* be transformed to completely stationary time series).

2.1.3 Energy of time series

Energy of time series is expressed as the sum of the squares of each element in this time series:

$$Energy(X) = ||x||^2 = \sum_{i=1}^{N} x_i^2$$

where x is time series $(x_1, x_2 \ldots, x_N)$

2.2 Basic concepts of machine learning

Machine learning and artificial intelligence solutions may be classified into two categories: “supervised” and “unsupervised” learning to analyze data. It is clear in our case we are dealing with supervised method as there are already classified ECGs either as arrhythmogenic or non-arrhythmogenic.

2.2.1 Supervised and unsupervised methods of data analysis

2.2.1.1 Supervised methods

In supervised learning, a random sub-sample of all records is taken and is manually classified. Relatively rare events may need to be over sampled to get a large enough sample size. The manually classified records are then used to train a supervised data analysis algorithm. After a model based on the training data is developed, the algorithm should be able to classify new records using the given trained labels.

2.2.1.2 Unsupervised methods

In contrast, unsupervised methods help to analyze data that has not been classified previously (there is no error or reward signal), namely, the data or actions are novel and are recognized as potentially satisfying some new category or another already identified category during the analysis. These methods simply seek those samples which are most dissimilar from the norm and then can be examined more closely. Outliers are a basic form of non-standard observation. Tools used for checking data quality can be used for the detection of accidental errors.

2.2.2 Features

Feature in machine learning is a measurable (nominal or categorical) characteristic of the observed phenomenon (e.g. subject, event etc.). Fox example, any human has the following features: age (nominal), salary (nominal), nationality (categorical) and many more others. Several features may be considered as a *feature vector.*

2.2.2.1 Standardization and normalization of features

In many cases, the input data contain nominal features at different scales, e.g. salary (10000-45000$), size of family (1-10) and age (0-140). Without preprocessing, a feature with wide distributions of values (salary) will dominate and an algorithm will be unable to learn properly from the features with compact distributions of values (family size, age) despite of the fact that these features may be the most discriminative. Some machine learning algorithms such as Decision Tree, Random Forest or Extra-Trees 2.3.7 do not depend on the feature scales and work correctly without feature rescaling, however, many others machine learning algorithms such as SVM 2.3.5, k-NN 2.3.4, logistic regression 2.3.1 and neural networks 2.3.2 are unable to handle features at different scales properly.

Standardization and *normalization* are common techniques for feature rescaling. Normalization is a process of scaling individual samples to have *unit norm* and is expressed as:

$$normalize(V) = \frac{V}{|V|}$$

where V is a vector $(v_1, v_2, \ldots, v_n)$, $|V|$ is the vector norm (also known as L2 norm) of V and is expressed as

$$|V| = \sqrt{v_1^2 + v_2^2 + ... + v_n^2}$$

Standardization refers to the following approaches:

- scaling features to a range

- scaling features to zero mean and unit variance (similar to Gaussian distribution)

The second case will be described in detail: considering feature m, std(m) as standard deviation of feature m, $mean(m)$ as mean value of feature m, n data samples $X = (x_1, x_2, \ldots, x_n)$ and $x_i(m)$ as value of feature m in sample x_i, scaling m to zero mean and unit variance may be performed as follows:

$$scale(x_i) = \frac{x_i(m) - mean(m)}{std(m)}$$

2.2.3 Classifier

In supervised machine learning, a *classifier* is an output of a learning algorithm that was trained using the training samples. In other words, a classifier is a model that describes relationships between the training samples and, given new samples, returns output based on the learnt relationships.

In mathematical terms, a classifier is a hypothesis about the true (often unknown) function f:

$$Y = f(X)$$

where X is a set of input variables and Y is a set of the appropriate true outputs for X.

There are two phases of classifier's life cycle:

- Learning phase
- Evaluation phase

During the learning phase, the selected learning algorithm is trained on the input data and classifier is constructed as a result. During the evaluation phase, the already constructed classifier is used to predict output of new data.

2.2.4 Objective function

During the learning phase, a classifier optimizes so-called *objective function*. This function may be minimized or maximized depending on the researcher's goals. In case of supervised learning and minimization, the objective function is called *cost function* (also known as *loss function* or *error function*) and may be defined as the difference (e.g. absolute value of difference or squared difference etc.) between the computed classifier's output and the true output of the input samples. Another example of the objective function is a fitness function in evolutionary algorithms.

2.2.5 Cross validation

Cross validation is a technique for estimating generalization accuracy on the independent data set. This technique divides data into the determined number of the unique subsets (so-called *folds*) with subsequent testing performance of a machine learning algorithm on each fold while the algorithm is training on the data in the remaining folds. The final performance of the algorithm is the average performance across all cross validation iterations. Figure 2.2 [1] illustrates the process. The abovementioned case refers to k-fold cross validation where k is a number of folds, however, there are other modifications of cross validation technique such as:

- *Leave-p-out cross validation* use p observations for testing and use remaining observations for training until all observations were used for testing
- *Leave-one-out cross validation* is an extreme case of leave-p-out cross validation where $p = 1$

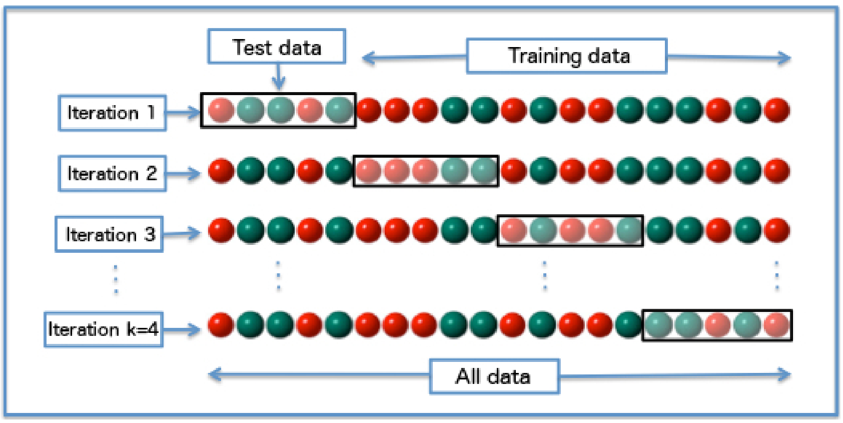


Figure 2.2: Example of 4-fold cross validation

2.2.6 Regularization

Regularization introduces additional information to the objective function (optimization of this objective function is the main goal of machine learning algorithm) in order to improve generalization accuracy and to prevent overfitting of the machine learning algorithms. Regularization in terms of this work was denoted as C: the smaller is C, the strongest is regularization. The value of C parameter is a trade-off between classifier performance on the training data

[1]`https://en.wikipedia.org`

and the simplicity of the objective function. Figure 2.3 illustrates the case where two functions model the data with zero loss, however, the regularized green function is simpler and provides better generalization accuracy on the given data.

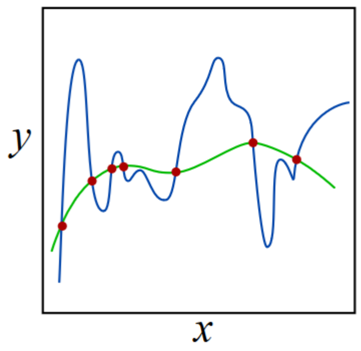


Figure 2.3: Regularization example: green function is regularized and blue one is not [2].

2.3 Overview of the used machine learning algorithms

This section contains descriptions of used machine learning algorithms. Description of each algorithm consists of basic and important features and properties of the described algorithm with regard to binary classification.

2.3.1 Logistic regression

Logistic regression, or *logit regression*, or *logit model* [11][12], is a regression model where the *dependent variable* is categorical and is determined by features. Similar to other types of regression analysis, logistic regression is a method of predictive analysis. Logistic regression is used to describe data and to explain the relationship between a dependent variable and one or more input (explanatory) variables (nominal, ordinal etc.). Logistic regression is mainly divided into the following two groups: *binary (binomial)* in case of two dependent variables (pass/fail, win/lose etc.) and *multinomial* in case of more than two dependent variables. The first type (binary) will be discussed below as it completely satisfies our goal of detecting arrhythmogenic ECG from non-arrhythmogenic one.

The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) features using a logistic function, which is the cumulative logistic distribution. Binary responses (cases) are assumed to be *independently distributed*, i.e. *independent*. Logistic regression assumes that the dependent variable (binary response) is a *stochastic event*.

Logistic regression is named so because it uses *standard logistic function* $\delta(t)$, which refers to *sigmoid* function type defined as follows for any real input t:

$$\delta(t) = \frac{1}{(1 + e^{-t})}$$

where $\delta(t)$ lies in interval $(0, 1)$. A graph of the logistic function within interval $[-6, 6]$ is shown on Figure 2.4.

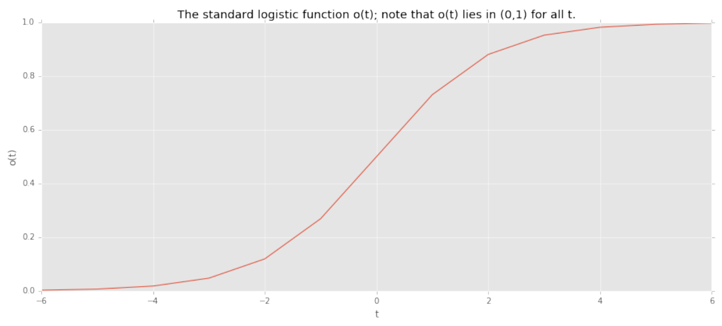


Figure 2.4: The standard logistic function

Let us assume that t is a linear function of two input variables (features) x_1 and x_2. Then t may be expressed as:

$$t = B_0 + B_1 x_1 + B_2 x_2$$

where B is a regression coefficient. Thus, the logistic function may now be written as:

$$X = (x_1, x_2)$$

$$F(X) = \frac{1}{1 + e^{-(B_0 + B_1 x_1 + B_2 x_2)}}$$

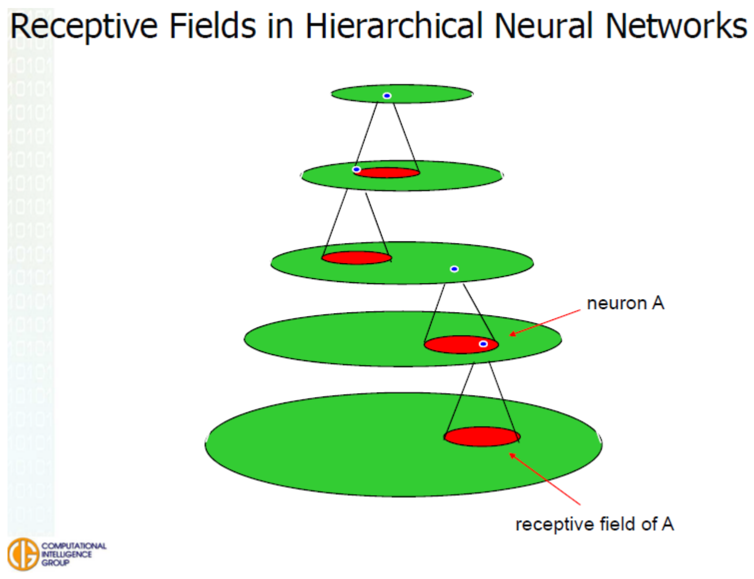
$F(X)$ is the probability that the dependent variable equals a case ("success"), given some linear combination of the predictors. The formula for $F(X)$ illustrates that the probability that the dependent variable equals a case ("success") is equal to the value of the logistic function of the linear regression expression. This is important as it shows that the value of the linear regression expression may vary from negative to positive infinity and yet, after transformation, the resulting expression for the probability $F(X)$ takes values between 0 and 1.

For example, if we analyze a pesticide's kill rate, the outcome event is either killed or alive. Since even the most resistant bug can only exist in one of these two states, logistic regression defines a likelihood of the bug getting killed. If the likelihood of killing the bug is > 0.5 it is assumed dead, if it is < 0.5 it is assumed alive.

The regression coefficients are usually estimated using *maximum likelihood estimation* (MLE) rather than *ordinary least squares* (OLS) method. In general, for a fixed set of data and underlying statistical model, the method of MLE selects the set of values of the model parameters that maximizes the likelihood function. Intuitively, this maximizes the “agreement” of the selected model with the observed data, and for discrete random variables it indeed maximizes the probability that the resulting distribution properly reflects the observed data.

2.3.2 Neural networks

Artificial Neural Networks (ANNs, or simply *Neural Networks* as NNs) are inspired by the structure and functional aspects of biological neural networks. NNs are a computational approach used in computer science and other research disciplines, which is based on a large collection of neural units (*artificial neurons*) loosely mimicking the way a biological brain solves problems. ANN is an adaptive system that is able to learn by adapting its connectivity patterns during the learning phase. In biological neural systems, neurons of similar functionality are usually organized in separate *areas* (or *layers*). Often, there is a *hierarchy* of interconnected layers with the lowest layer receiving sensory input and neurons in higher layers computing more complex functions [13][14][15]. Figure 2.3.2 illustrates the point clearly. A short summary of neural networks follows.


Receptive Fields in Hierarchical Neural Networks
neuron A
receptive field of A
COMPUTATIONAL INTELLIGENCE GROUP

Hierarchical ANN contains the following layers:

- Input
- Hidden (this layer may consist of many layers of hidden neurons rather than just one layer, or even does not exist at all)
- Output

Figure 2.5 provides an example of ANN.

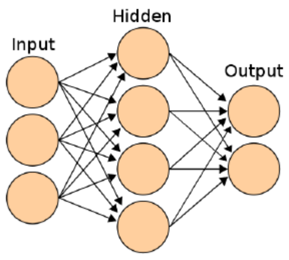


Figure 2.5: A hierarchical Artificial Neural Network [3]

ANN is typically defined by three types of parameters:

- The interconnection pattern between the different layers of neurons or neurons themselves.
- The weights of the interconnections, which are updated as part of the learning process (and thus, represent *memory* of ANN).
- The activation function that converts a neuron's weighted input into its output.

2.3.2.1 How ANNs works

Artificial Neural Networks basically have two operation phases:

- Adaptive (learning/training phase), where a network *learns* from data.
- Active (evaluation phase), where a network *evaluates* data.

ANNs are built from *artificial neurons* (also named as neurons, units, nodes). Basically, each neuron receives input from one (many) other neuron(s). Then this neuron changes its internal state (*activation*) using the appropriate *activation function* based on the current input and then sends the *same* output signal to one (many) other neuron(s).

In most cases, an activation function is a non-linear function such as Heaviside step function or a sigmoid function. It is important to keep in mind that different neurons may have different activation functions.

The strength of a connection, whether it is *excitatory* or *inhibitory*, depends on the state of synapses (i.e. on the weights) of the receiving neuron. NN achieves learning by adapting the states of its synapses (weights of the connections). Figure 2.6 shows a typical artificial neuron. A neuron input

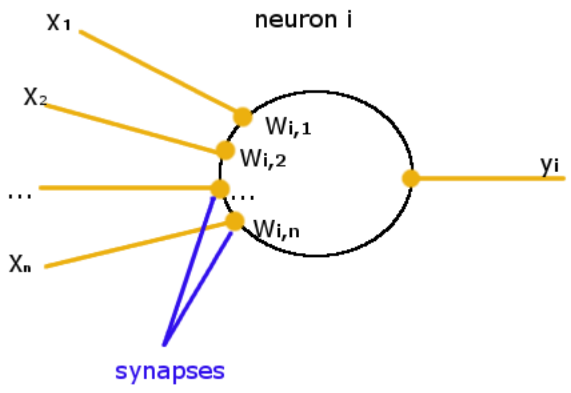


Figure 2.6: An artificial neuron

signal is expressed as follows:

$$neuron_i(t) = \sum_{j=0}^{n} w_{i,j}(t)x_j(t)$$

And output signal is expressed as:

$$y_i = f_i(neuron_i(t))$$

where f_i is an activation function of neuron i.

As soon as network output signal has been defined, it is necessary to estimate an error using a *loss function* 2.2.4.

2.3.2.2 Backpropagation

The backward propagation of errors or *backpropagation* [13][16] is a common method of training artificial neural networks and is used in conjunction with an optimization method such as gradient descent. The algorithm repeats a two phase cycle, propagation and weight update. When an input vector is presented to the network, it is propagated forward through the network, layer by layer (see Figure 2.7), until it reaches the output layer. The output of the network is then compared to the desired output, using a *loss function*, and an error value is calculated for each of the neurons in the output layer (see Figure 2.8). The error values are then propagated backwards, starting from the output (see Figure 2.9), until each neuron has an associated error value which roughly represents its contribution to the original output.

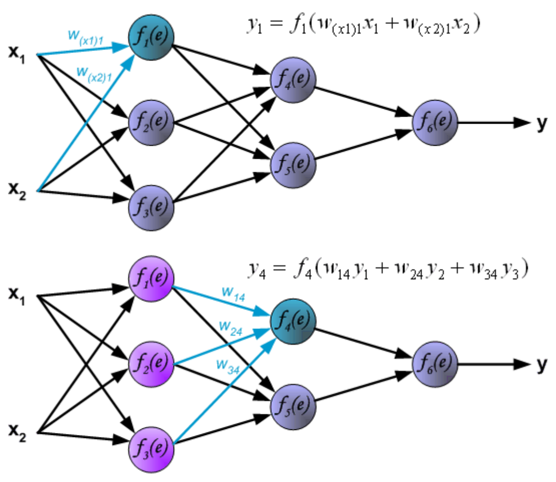


Figure 2.7: Backpropagation: input propagation [4]

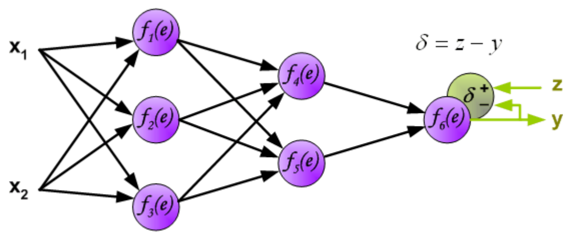


Figure 2.8: Backpropagation: final error computation, where y is a network output and z is a true output [4].

Backpropagation uses these error values to calculate the gradient of the loss function with respect to the weights in the network. During the second phase, this gradient is used as part of the optimization method, which in turn uses this gradient to update the weights (see Figure 2.10), in an attempt to minimize the loss function.

There are three methods of updating network weights:

- On-line training
- Batch training
- Mini-batch training

The abovementioned text and Figures in this section illustrate the case of *on-line training* where weights are updated after propagation of each training sample. During the batch training, weight changes are accumulated over all training samples (so-called *epoch*, i.e. all samples were propagated through the

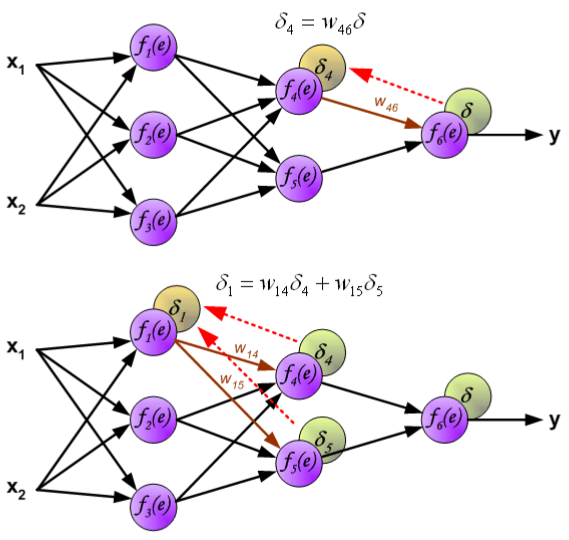


Figure 2.9: Backpropagation: backward error propagation [4]

network) and only then weight updating is performed. The mini-bath training defines some number n, which is the number of training samples for epoch, and then the same strategy as in the batch training is performed. The mini-batch training with $n = 1$ is actually the on-line training. It is important to note that Wilson and Martinez in their paper [17] in 2003 demystified widely spread delusion that batch training is faster than on-line training in terms of convergence speed, moreover, they proved the opposite.

Coefficient η affects network's teaching speed (step size or learning rate). In other words, the coefficient affects readiness of the network to accept changes: if it is low, then the neural network changes its internal state slowly and more epochs (forward and backward passes of all training samples, i.e. backpropagation cycles) are required to achieve the optimum, and if it is high, then the network changes its internal state so fast that it begins to straggle around the optimum without confidence in achieving it.

The importance of the backpropagation process is that, as the network is trained, the neurons in the intermediate layers organize themselves in such a way that the different neurons learn to recognize different characteristics of the total input space. After training, when an arbitrary input pattern is present and contains noise or is incomplete, neurons in the hidden layer of the network will respond with an active output if the new input contains a pattern that resembles a feature that the individual neurons have learned to recognize during their training.

Backpropagation requires a known, desired output for each input value in order to calculate the loss function gradient – therefore, the method is usually viewed as a supervised learning method; nonetheless, it is also used in

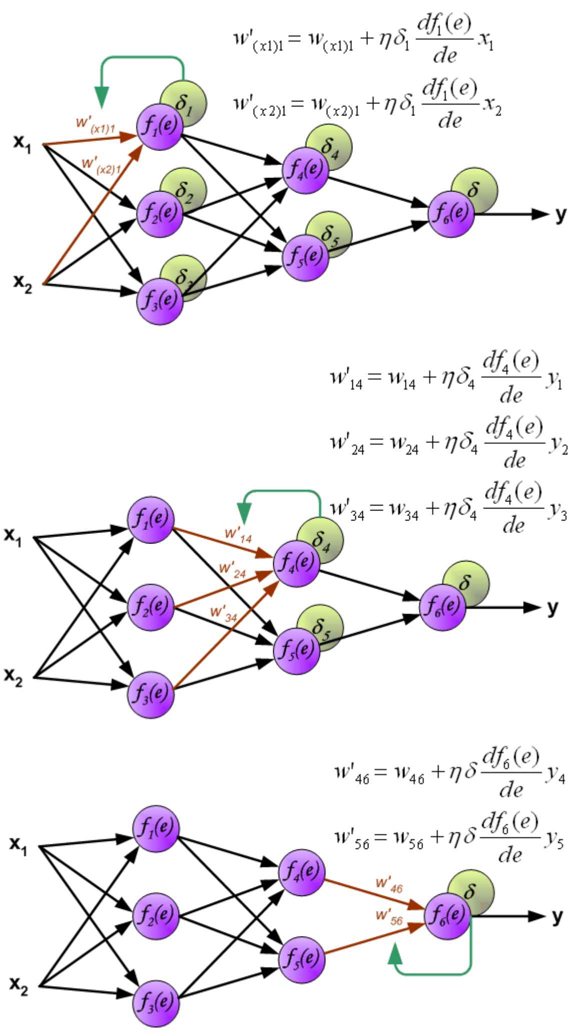

$w'_{(x1)1} = w_{(x1)1} + \eta\delta_1 \frac{df_1(e)}{de} x_1$
$w'_{(x2)1} = w_{(x2)1} + \eta\delta_1 \frac{df_1(e)}{de} x_2$
$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$
$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$
$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$
$w'_{46} = w_{46} + \eta\delta \frac{df_6(e)}{de} y_4$
$w'_{56} = w_{56} + \eta\delta \frac{df_6(e)}{de} y_5$
x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
δ_1
δ_2
δ_3
δ_4
δ_5
δ
y

Figure 2.10: Backpropagation: Updating weights [4]

some unsupervised networks such as *autoencoders* in deep learning. Backpropagation requires the activation function used by the artificial neurons to be differentiable. *Sigmoid functions* completely satisfy this condition.

2.3.2.3 Multi-Layer Perceptron (MLP)

MLP uses supervised learning that means it can be applied to our problem of distinguishing arrhythmogenic ECG from non-arrhythmogenic one without any additional actions as we have already labeled data. It also means that *backpropagation* may be used for minimizing the loss function.

MLP consists of multiple layers of nodes (neurons) in a directed graph with each layer fully connected to the next one (see Figure 2.11). Each neuron has an activation function according to differentiation condition of activation function in backpropagation. MLP may distinguish data that is *not linearly separable*.

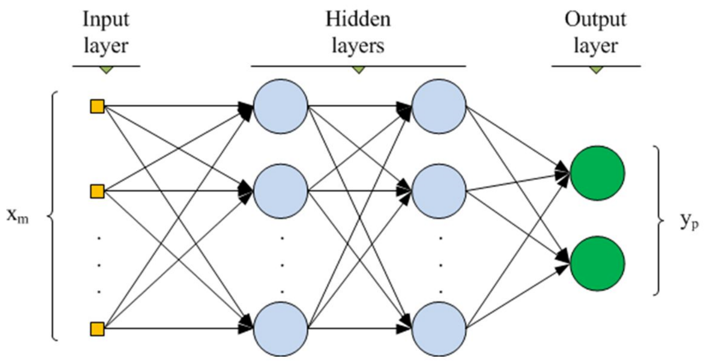


Figure 2.11: Multilayer perceptron [5]

2.3.2.4 Group Method of Data Handling (GMDH)

Group method of data handling (GMDH) was proposed in 1968 by Prof. Alexey G. Ivakhnenko in the Institute of Cybernetics in Kiev. GMDH represents a set of inductive algorithms for computer-based mathematical modeling of multi-parametric datasets that features fully automatic structural and parametric optimization of models. These algorithms are characterized by inductive procedure that performs sorting-out of gradually complicated polynomial models and selects the best solution by means of the so-called *external criterion*. A good overview of GMDH development process and algorithms may be found in [18]. Modifications of GMDH may be effectively used in the variety of problem domains such as the time series forecasting [19] and EEG signal classification [20]. A short summary of GMDH approach is provided further in the text.

The main difference between GMDH neural network and other NNs is that it is constructed by induction (sometimes called self-organization) and has *only* one output: optimal polynomial that approximates input data. That is the reason why GMDH NN is also called *polynomial neural network* and why it belongs to *parametric* neural networks. GMDH NN is a *feedforward* neural network and it uses *supervised learning*.

The main properties of GMDH neural network are:

- It is constructed from the minimal form to the final one step by step.
- Each neuron has exactly two input values.

- Each layer consists of all possible pairwise combinations of values of the previous layer, but we select only limited amount of these combinations (neurons) based on the external criterion (for example, k best neurons).
- Its parameters are set during learning phase.
- Perspective neurons survive.
- External criterion indicates when network construction must be stopped.

General relationship between input and output variables of neuron may be expressed by Volterra functional series, discrete analogue of which is Kolmogorov-Gabor polynomial:

$$Y(x_1,..,x_n) = a_0 + \sum_{i=1}^{n} a_i x_i + \sum_{i=1}^{n}\sum_{j=i}^{n} a_{ij} x_i x_j + \sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=j}^{n} a_{ijk} x_i x_j x_k + ...$$

where $X(x_1, x_2, \dots, x_n)$ is vector of input variables and $A(a_1, a_2, \dots, a_n)$ is vector of coefficients or weights. It may also be expressed by Ivakhnenko polynomial for an individual neuron (see Figure 2.12) in layer k:

$$y^k = a^k(i^{k-1})^2 + b^k(i^{k-1})(j^{k-1}) + c^k(j^{k-1})^2 + d^k i^{k-1} + e^k j^{k-1} + f^k)$$

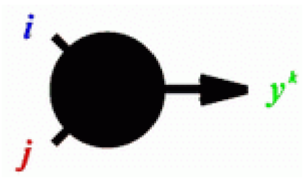


Figure 2.12: An individual neuron in GMDH

Figure 2.13 shows an example of GMDH NN with survived/dead neurons, where an approximation function g is expressed as:

$$g : R^n \rightarrow R$$

where n is a number of input variables (in case of Figure 2.13 it is 4).

External criterion is one of the key features of GMDH. The criterion describes requirements for the model (a subset of the connected neurons, so-called *partial model*), for example minimization of least squares. The criterion is always calculated using a subsample of data that has not been used for estimation of coefficients (training data set), i.e. so-called *validation data set. Criterion of Regularity* (CR) is the most popular criterion. CR is expressed as *least squares* of a model on the validation data set:

$$CR = argmin \frac{1}{N_B} \sum_{i=1}^{N_B} (y_i - \hat{y}_i(A))^2$$

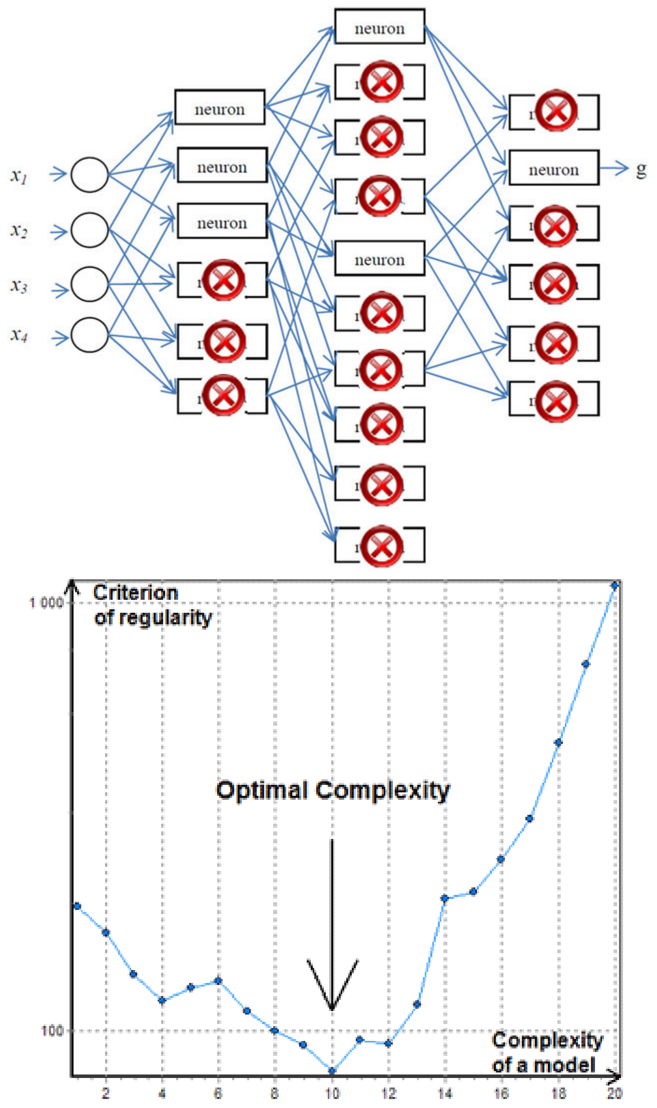


Figure 2.13: (Top) An example of GMDH NN with survived/dead neurons [6] (Bottom) GMDH Optimal complexity [3]

where where N_B is a number of samples in validation data set B, y_i is the true output, $\hat{y}_i$ is the model output for samples in B and the model is trained on the train data set A.

The best model during the training phase (i.e. partial model) is indicated by the minimum of the external criterion characteristic. This minimum CR is used as stop criterion in the following way: if CR of the best model in the next layer is greater than the current best one, it is time to stop network construction in the current layer (see Figure 2.13). Complexity of model is defined as a number of layers.

2.3.3 Dynamic Time Warping (DTW)

Dynamic Time Warping (DTW) is a well-known algorithm for measuring similarity (or dissimilarity from another point of view) of two time dependent sequences, which may also vary in speed. DTW is based on the *Levenshtein distance* (also known as *edit distance*) and was originally introduced for speech recognition [21] as *Euclidean distance* could not help in case of time dependent sequences. Euclidean distance does not take into account the *shape* of compared sequences, but only distance between their points at the same time. On the other side, DTW algorithm computes an optimal match between two given sequences by aligning them to each other and thus, the algorithm concentrates on the shape similarity of the sequences. Figure 2.14 illustrates such alignment of two sequences by DTW.

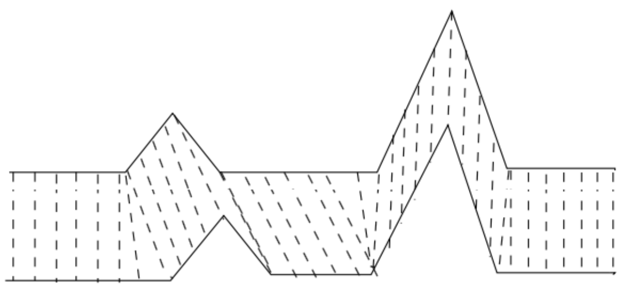


Figure 2.14: Alignment in Dynamic Time Warping [3]

The following text describes in more details how DTW works [21][22]. Suppose there are two sequences $C = (c_1, \dots c_n)$ and $Q = (q_1, \dots q_m)$, where c and q are so-called features, n and m are lengths of C and Q respectively.

In order to align sequences C and Q, it is necessary to compute a *cost matrix* of size $n * m$. This matrix is computed using Levenshtein distance approach as follows:

- First row

$$cost\_matrix(1, j) = \sum_{k=1}^{j} d(c_1, q_k)$$

- First column

$$cost\_matrix(i, 1) = \sum_{k=1}^{i} d(c_k, q_1)$$

- All other elements

$$cost\_matrix(i,j) = min\begin{cases} cost\_matrix(i-1,j) + d(c_i,q_j) \\ cost\_matrix(i,j-1) + d(c_i,q_j) \\ cost\_matrix(i-1,j-1) + d(c_i,q_j) \end{cases}$$

A distance $d(c_i, q_j)$ between two points c_i and q_j was originally defined as $d(c_i, q_j) = |c_i - q_j|$[21], but it may also be Euclidean distance $d(c_i, q_j) = (c_i - q_j)^2$ or another distance metric. A distance measure is also called *local cost measure* or *local distance measure* and may be different (e.g. Manhattan, Euclidean) as was mentioned previously. A selected distance measure must be a function

$$d : F \times F \rightarrow R_{(\geq 0)}$$

where F represents the space of features, R is the space of positive real numbers (including 0).

Typically, a distance between two points is small, when they are similar; otherwise, the distance is greater. Example of the computed cost matrix is shown on Figure 2.15.

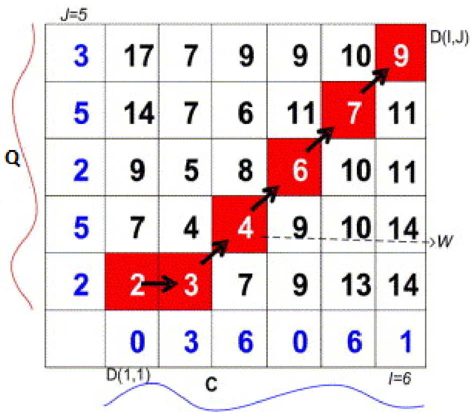


Figure 2.15: Computed cost matrix with colored optimal warping path [7].

Some match P, so-called *warping path*, between two sequences is expressed as a set of contiguous cost matrix elements (see Figure 2.15, red squares):

$$P = (p_1, \dots, p_k)$$

Warping path must satisfy several conditions:

- *Boundary condition*: $p_1 = (1,1)$ and $p_k = (n,m)$, which means warping path must be started in the beginning of both two sequences and must be finished in the end of these both sequences.

- *Monotonicity*: $n_1 \leq n_2 \leq ... \leq n_k$ and $m_1 \leq m_2 \leq ... \leq m_k$, which means no steps backward are made.
- *Continuity*: only adjacent cells are permitted (diagonal included).

Total cost of warping path is expressed as:

$$C(P) = \sum_{i=1}^{k} p_i$$

in case of Figure 2.15 with $d(c_i, q_j) = |c_i - q_j|$ or

$$C(P) = \sqrt{\sum_{i=1}^{k} p_i}$$

in case of Euclidean distance, or other total cost function which reflects the selected distance measure.

And *optimal warping path* is path with the minimum cost (see Figure 2.15).

Each element of the cost matrix corresponds to the alignment between two appropriate points. If the sequences are equal or are very similar, the appropriate warping path coincides with diagonal. Otherwise, the path deviates from diagonal. The following Figure 2.16 it clearly.

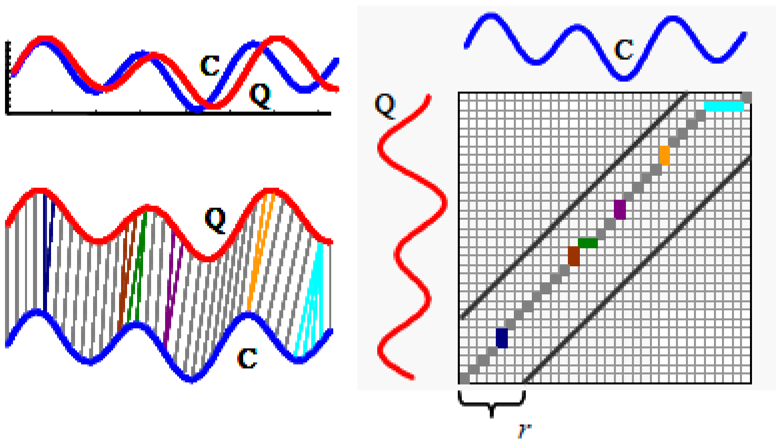


Figure 2.16: (Top-Left) Image contains two original sequences C and Q. (Bottom-Left) An alignment between sequences C and Q. (Right) Cost matrix for sequences C and Q [8].

DTW is a very powerful and accurate algorithm for comparing time dependent sequences. However, the main disadvantage of this algorithm is its poor $O(n^2)$ time complexity. Many modifications of the original DTW have been developed in order to speed up this algorithm.

2.3.3.1 Modifications of DTW

There are modifications of DTW [22] such as:

- *Step size condition (continuity)* determines which elements are considered as adjacent so that only such elements are permitted.
- *Local weights for multiplying* the distance measures in order to favor some direction (vertical, horizontal or diagonal). It is important to mention that diagonal direction consists of vertical and horizontal directions and thus, all their corresponding weights must be taken into account.
- *Global constraints* that determine a set of admissible warping paths and thus, speed up the algorithm.
- *Approximation* (e.g. Piecewise DTW [23]), following the idea to divide original time series into some parts (so-called frames) and afterwards compute their mean. The vector of computed means is now a reduced time series representation.
- *Multilevel approach* (FastDTW [24]), which works by:
  - converting the time series into the lower resolution by grouping the original elements into larger ones
  - finding the minimum distance at this level
  - subsequently restoring the original resolution of the grouped elements based on the identified warp path
  - identifying minimum warp path locally within each group of elements
- *Subsequence DTW* [22] finds a subsequence within a longer sequence that optimally fits the shorter sequence.

A modification of DTW with global constraints will be discussed below as the most common and widely used variation. This variation is used to impose constraints on the admissible warping paths. This approach speeds up DTW and, moreover, prevents pathological alignments by globally controlling the route of warping path. A *warping window* (or *global constraint region*) specifies region with admissible warping paths (denoted as r and represented as two parallel straight lines in Figure 2.16). Warping window may be specified as percentage of the length of the longer sequence. It is interesting to mention that the Euclidean distance between two sequences may be interpreted as a special case where $r = 0\%$. Two well-known types of constraint regions are *Sakoe-Chiba band* and *Itakura parallelogram* (See Figure 2.17). It is necessary to mention that an optimal warping path may not be a part of the used constraint region.

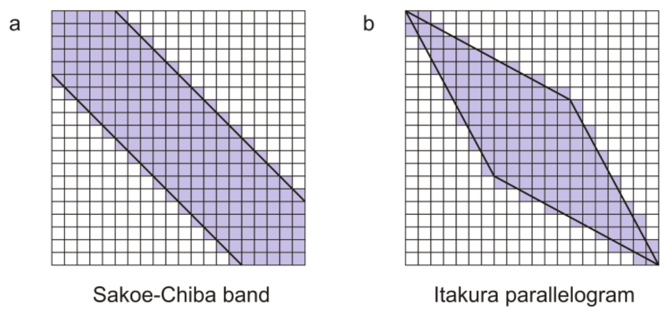


Figure 2.17: Constraint regions (warping windows) in DTW [9].

2.3.3.2 Three myths about DTW

In 2005, Ratanamahatana and Keogh in paper [25] have demystified three widely spread delusions about DTW. A short summary of their work is provided below:

- Comparison of sequences of the different lengths and reinterpolating them to equal length produces no statistically significant difference in accuracy or precision/recall.
- Larger constraints on the used warping window do not improve accuracy or precision/recall. Actually, 10% constraint on warping window is too large for real world data processing.
- No improvements in the DTW speed are required as it is $O(n)$ with a simple lower bounding technique (4S, see the abovementioned paper for more details).

2.3.4 k-Nearest Neighbors (k-NN)

k-Nearest Neighbors (k-NN) algorithm is a supervised machine learning algorithm. It is popular among researchers and is commonly used for classification tasks; thus, it may help us in classification of ECGs. This algorithm may be also used for regression tasks, but due to its specifics, it is more natural to use the algorithm in classification. Simplicity, accuracy and only one parameter k make this algorithm a great choice for many cases. The description of the algorithm's behavior in accordance with classification tasks is given further in the text.

The idea of k-NN algorithm is to find k nearest neighbors of a given new sample using some measure of the distance (e.g. Euclidean). Then the predicted class is determined by majority vote of these neighbors. The contribution of each neighbor to the prediction may also be weighted in such a way that the closest neighbors take bigger weights. In case of Euclidean distance, the weights may be computed as

$$w_i = \frac{1}{d(i,a)}$$

where a is a new given sample, and $d(i,a)$ is Euclidean distance between a and i-th element in the training data set.

Training phase of k-NN algorithm consists of storing given training data set with the appropriate true outputs for each sample in this data set. During the evaluation phase, the algorithm uses k nearest neighbors for prediction as was described above. The following Figure 2.18 [2] illustrates k-NN algorithm.

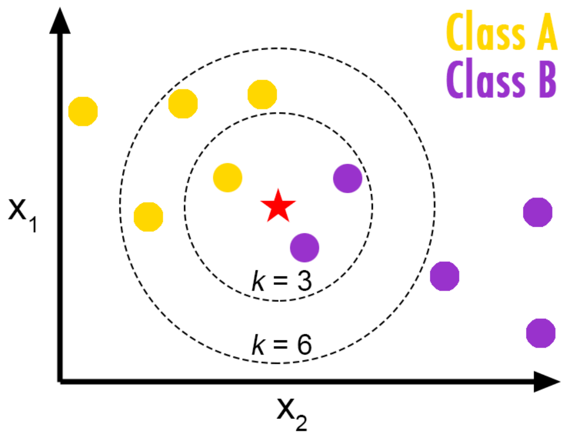


Figure 2.18: k-NN algorithm with data of the two features x_1 and x_2. $(k = 3)$ A new sample (star) is predicted as class B. $(k = 6)$ A new sample is predicted as class A.

However, there are two main disadvantages of this algorithm:

- Its sensitivity to noise in given training data: with growing noise, larger quantities of nearest neighbors are required for the accurate prediction. However, it still cannot guarantee accuracy, if the nearest neighbors are noisy.
- Its dependency on the size of the training data: time for finding the defined amount of nearest neighbors grows with the increasing size of the training set (which is unacceptable in case of millions and more training samples).

Many modifications of k-NN have been made in order to get rid of these weaknesses. Some of them are:

[2] `http://bdewilde.github.io/blog/blogger/2012/10/26/classification-of-hand-written-digits-3/`

- SMART-TV [26] for dealing with high-dimensional datasets (high amount of features).
- k-NN ensemble classifiers exploiting algorithm's instability on a set of input features [27]
- DB-kNN, V-kNN, W-kNN, CB-kNN [28]

Dynamic Time Warping (DTW) similarity measure, which was described in this work earlier, may be used as a distance measure for k-NN algorithm. Surprisingly, k-NN with DTW is a very robust classifier for time dependent sequences and, it is extremely hard to beat this combination as was shown in [8]

There is also an advantageous approach for measuring similarity of multivariate time series using k-NN algorithm [29]. It uses a similarity measure *Eros* (extended frobenius norm), an index structure *Muse* (multilevel distance-based index structure for *Eros*) and a feature subset selection technique *Ropes* (recursive feature elimination on common principal components for *Eros*). Research conducted in this paper shows that this approach outperforms k-NN with DTW in precision/recall by a minimum of 6.5% and also takes less time.

2.3.5 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised machine learning algorithm for both classification and regression tasks. The main idea of SVM is to map the input vectors into some high dimensional feature space through predefined non-linear mapping. In case of classification in this created high dimensional space, a linear decision surface (so-called *hyperplane*) is constructed with special properties that ensure a high generalization ability of SVM [30]. A short summary of the concept follows.

SVM algorithm is based on constructing an optimal hyperplane with the maximal margin between so-called *support vectors.* Support vectors are selected from the training samples in the boundary region of the two classes. Another important thing is that the number of the used support vectors is small in comparison to the size of training data set. Figure 2.19 [3] illustrates the optimal hyperplane and support vectors (they are colored).

If the input data are linear separable, then SVM algorithm (so-called *linear SVM*) searches for the optimal hyperplane in the unchanged feature space of the input data. However, in the case the input data are linear non-separable, SVM maps the input data using so-called *kernel function* to the higher dimensional feature space. The used kernel function must satisfy *Mercer's theorem* [31] and correspond to *some type of inner product* in the constructed high dimensional feature space. One of the advantages of SVM is its universality

[3]`http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html`

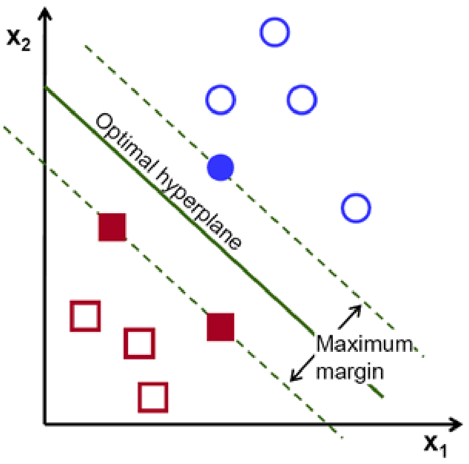


Figure 2.19: The optimal hyperplane of SVM

as the algorithm allows using different kernel functions, which depend on the researcher's assumptions and problem domain.

Another advantage of SVM algorithm is its effectiveness in the high dimensional feature spaces. Cortes and Vapnik in their paper [30] used a 7-degree polynomial as kernel function (which resulted in the new constructed high dimensional feature space with dimensionality approximately equal to 10^16) with SVM that was trained on the 7,300 samples. The number of the used support vectors was 190 and the error rate was equal to 4.3%. This experiment indicated no overfitting problems. However, it is important to note that leave-one-out cross validation, which provides the basis for computationally efficient model selection strategies, is prone to overfitting. Fortunately, Cawley and Talbot in their paper [32] proposed regularization approach for dealing with this problem.

SVM is widely used in many problem domains (e.g. ECG [33][34][35][36][37] and EEG [38] classifications) and often outperforms other machine learning algorithms.

Support Vector Machine is a powerful algorithm, but its computing and storage requirements increase rapidly with the number of training vectors. The core of SVM is a quadratic programming problem (QP), separating support vectors from the rest of the training data. The time complexity of SVM is $O(max(n,d) * min(n,d)^2)$ according to [39] where n is number of samples and d is number of features.

2.3.6 Ensemble Learning

Ensemble learning is an approach that allows constructing a set of classifiers of learning algorithms and then classifying new samples by some kind of voting of predictions of these classifiers.

Dietterich [40] discovered that ensembles of classifiers are generally much more accurate than the individual classifiers themselves. However, some assumptions must be taken into account to make this statement possible, i.e. individual classifiers of ensemble must satisfy the following two requirements (from [40]):

- be accurate and
- be divers

An *accurate* classifier is the one that has an error rate better than just random guessing on the new samples (i.e. error rate $< 50\%$). The two classifiers are diverse if they make different errors on the new samples.

The importance of diversity and accuracy will be described using the following example. Assume there are three classifiers with the same error on some sample x and thus, the outputs of these classifiers are the *same*, i.e. the classifiers are identical (not diverse). It is clear that the ensemble approach is useless in this case as all classifiers always vote for the same output. On the other hand, if the classifiers make *different* errors on some sample x, the outputs may vary and thus, the output with most votes wins (majority vote). Now let us consider account accuracy of the used classifiers in addition to their amount. If classifiers have accuracy less than random guessing ($< 50\%$), the voting result tends to converge to the *wrong outputs* with the *increasing* number of classifiers. Otherwise, the voting result converges to the *right outputs*.

The following Figure 2.20 [40] reflects the probability of wrong voting with the increasing number of classifiers (21 in totals) with their error rate equal to 0.3. Each classifier makes its errors independently of the others.

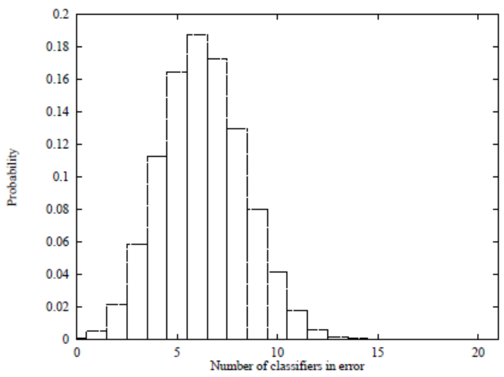


Figure 2.20: The number of used classifiers in ensemble vs the probability of error

Moreover, in practice, there are three fundamental reasons why construction of the good ensembles is possible [40]. These reasons are:

- Statistical
- Computational
- Representational

These three reasons will be shortly described in the text below. The following Figure 2.21 [40] illustrates the abovementioned reasons. The inner curve defines a set of classifiers with a good enough accuracy based on the training data. The point labeled f is the true classifier.

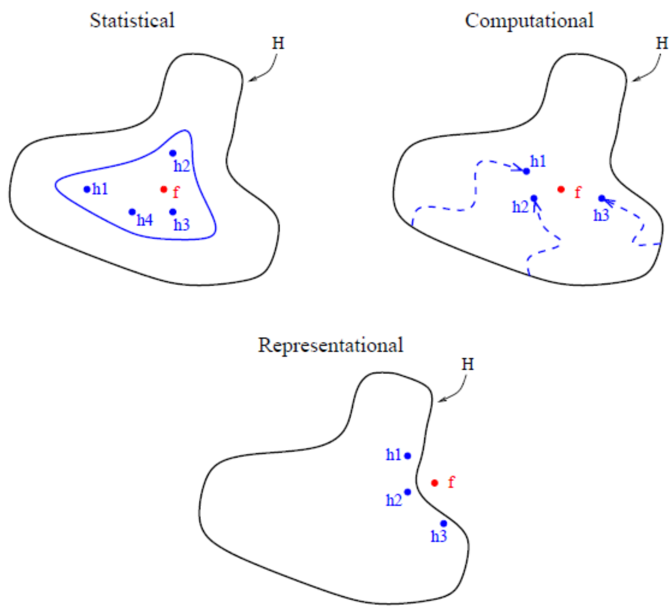


Figure 2.21: Three fundamental reasons why an ensemble may work better than a single classifier

2.3.6.0.1 Statistical reason Learning algorithms may be viewed as searching a space H of classifiers in order to identify the best classifier in this space. The statistical problem *arises in case of a too small training set* as the data is not enough to construct a strong classifier and thus, many different classifiers with the same accuracy may be found. The risk of selecting the wrong classifier will be reduced by constructing an ensemble of classifiers out of these identified classifiers of the same accuracy and "averaging" votes (or majority vote) of classifiers in this ensemble.

2.3.6.0.2 Computational reason Assume there is enough training data so that the statistical problem is absent. In this case, finding the best classifier may still be a problem as many learning algorithms work by performing some form of local search and thus, *they may get stuck in local optima* (e.g. gradient descent in neural networks, greedy splitting rule in decision trees). Optimal training is impossible in case of neural networks and decision trees as it is NP-Complete problem [41] [42]. Thus, an ensemble constructed by classifiers that run the local search from many different starting points may provide a better approximation to the true function than any individual classifier.

2.3.6.0.3 Representational reason The representational problem arises when a *learning algorithm is unable to represent the true function by any classifier in H*, what frequently occurs in case of real data. It is possible to expand a space of representable functions by forming weighted sums of classifiers from the space H. However, the main assumption in this case is that H is an effective space of all possible classifiers of a learning algorithm for a given training data. This assumption must be taken into account primarily for learning algorithms that have possibility to represent any true function, given enough training data (e.g. neural networks, decision trees).

2.3.6.1 Methods for constructing ensembles

In this subsection general methods [40] for constructing ensembles that may be applied to different learning algorithms for classification task will be discussed.

2.3.6.1.1 Manipulating the input features
This general technique for generating multiple classifiers assumes a manipulation of the input features of the available data. In 1996, Cherkauer [43] in his work describes identification of volcanoes by ensemble of neural networks with grouping features, which were based on different image processing operations. The resulting ensemble classifier was as good as the prediction made by human experts.

However, it is obvious that this technique works only in case of *highly redundant input features*, because some features typically will not be used when grouping occurs.

2.3.6.1.2 Manipulating the output targets
This technique is used for constructing good ensembles by manipulating the true outputs of the input data. The main assumption is that the *number of classes K is large.* Then new learning problems can be constructed by randomly partitioning the K classes into two subsets A and B. The input data can then be relabeled so that any of the original classes in set A are given the derived label 0 and the original classes in set B are given the derived label 1. This relabeled data is then given to the learning algorithm, which constructs

a classifier. By repeating this process L times (generating different subsets A and B) an ensemble of L classifiers is obtained [40].

Now given a new data point x, each classifier predicts its class. If a prediction is 0, then each class in A receives a vote. If a prediction is 1 then each class in B receives a vote. After each of the L classifiers has voted, the class with the highest number of votes is selected as the prediction of the ensemble.

2.3.6.1.3 Manipulating the training examples

There are three main approaches how an ensemble of classifiers may be constructed in this case:

- Bagging (Bootstrap Aggregating)
- Boosting
- Stacking

These methods work especially well for *unstable* learning algorithms. Unstable learning algorithms are those whose output is significantly changing due to small changes of their input. Such algorithms are neural networks, decision trees and other rule-based learning algorithms.

2.3.6.1.3.1 Bagging

Bagging method is based on the independent classifiers learnt on the random subset selection with repetition from the training set (so-called *bootstrap replicate*) with subsequent voting of their outputs in order to get the best probable output (majority vote). This method may be easily parallelized due to independent nature of learning algorithms. Experiments by Dietterich [40] have proved that Bagging outperforms AdaBoost in case of noise in data. The following Figure 2.22 [4] illustrates Bagging approach.

2.3.6.1.3.2 Boosting

Boosting method (e.g. AdaBoost) has a consecutive nature. The idea is that the training samples with bigger error on the previous classifiers have more chance to be selected as training data for the next classifier by modifying samples' weights. The ensemble output is a weighted (based on classifier performance) average of the individual classifiers in ensemble. Figure 2.23 [5] illustrates Boosting approach clearly.

[4] `https://www.packtpub.com/mapt/book/big_data_and_business_intelligence/9781783555130/7/ch07lvl1sec46/Bagging++building+an+ensemble+of+classifiers+from+bootstrap+samples`

[5] `http://vinsol.com/blog/2016/06/28/computer-vision-face-detection/`

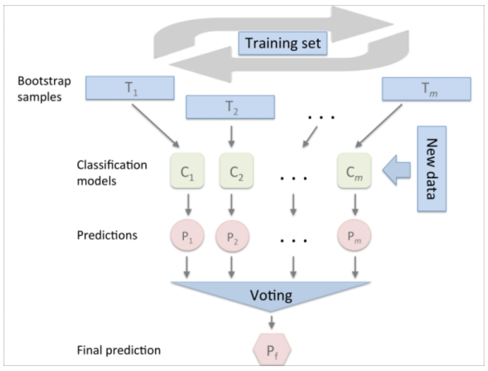


Figure 2.22: Ensemble methods: Bagging

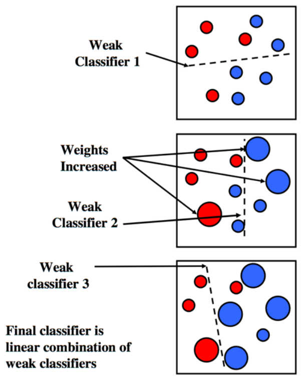


Figure 2.23: Ensemble methods: Boosting

2.3.6.1.3.3 Stacking

The main idea of *stacking method* [44][45] is to use predictions of different classifiers as the input to another *meta-classifier* which determines the final output.

After conducting research, the following two ways of interpreting this idea were revealed:

- The original way proposed by Wolpert[44]
- The original idea by Wolpert interpreted differently, which was tested in Kaggle [6] data science competition by Wille [7], and described by Faron [8].

The original approach [44] suggests, given some sample x, using a combination of classifiers' outputs for this sample (i.e. their predictions) as input

[6]https://www.kaggle.com/

[7]`https://dnc1994.com/`, `https://dnc1994.com/2016/05/rank-10-percent-in-first-kaggle-competition-en/`

[8]`https://www.kaggle.com/getting-started/18153#103381`

to meta-classifier. The output of this meta-classifier is viewed as the final one. First, each learning algorithm constructs a classifier learnt from the training data using k-cross-validation and, as a result, a vector of outputs of these classifiers for each sample is identified from the training data. Second, a meta-classifier is learnt based on these output vectors and the true outputs of the appropriate samples. As a result, the classifiers evaluate a new sample and form a vector of their predictions, which is processed as input to meta-classifier, and the output of meta-classifier is the desired final one.

The following is intriguing: Kai Ming Ting and Ian H. Witten reveal in their work [45], that for successful stacking it is necessary to use *output class probabilities* rather than class predictions and, moreover, only *multi-response linear regression algorithm* (MLR) is suitable for *the level-1 generalizer* (meta-classifier in our case).

A different interpretation of Wolpert's idea focuses on using classifiers' predictions (made for the appropriate validation samples (indicated in orange color on Figure 2.24 [9]) in the same way as k-fold cross-validation, 5-fold CV in the case of Figure 2.24) as an additional feature for the original training data. Then an additional new learning algorithm uses the training data extended with this new feature to construct a meta-classifier. Similar to the original training data, the original test data must also be extended taking into account a new feature in one of the following two ways:

- *Variant A*: A value to the new feature is assigned by the averaged voting of classifiers (Model 1-5 on Figure 2.24).

- *Variant B*: A new classifier based on the original training data is constructed by a learning algorithm and then is applied to the original test data in order to compute their predicted outputs. These outputs are mapped to the appropriate original test data as new features.

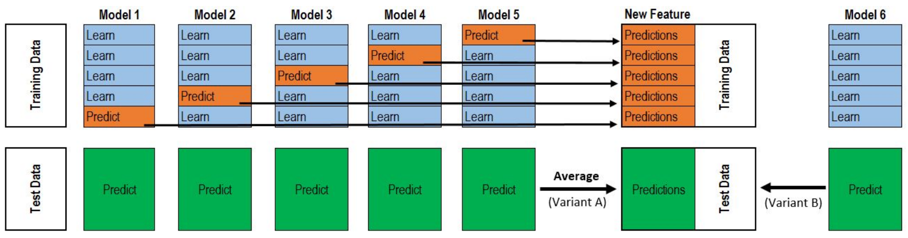


Figure 2.24: Ensemble methods: Stacking

Then the meta-classifier constructed from the extended training data is applied to the extended test data in order to estimate its accuracy.

[9] https://www.kaggle.com/getting-started/18153#103381

2.3.6.1.4 Injecting randomness

This method of ensemble construction is the most common one. The idea of this method is to inject randomness into a learning algorithm. As an example, training neural network with backpropagation and setting its initial weights to random values may result in quite different outputs.

However, a research conducted by Parmanto, Munro and Doyle [46] reveals that cross-validated committees (which is simplification of stacking method) and Bagging outperform injection of randomness in case of neural networks represented by random initial weights.

2.3.7 Decision trees

Decision trees are a common and widely used tool of data analysis. Simple decision trees have been used for a long time as they may be easily and rapidly created by a researcher and, moreover, have strong explanatory power and are easily interpreted by other people. Decision trees refer to *supervised learning*, however, further research for unsupervised learning appear nowadays such as in 2005 by Karakos, Khudanpur and Eisner[47].

As the advantages of decision trees are clear (well interpreted, requires little data preparations, robust etc), it is time to explicitly mention their disadvantages:

- Trees tend to be less accurate than other approaches.
- They are unstable (small changes in input may cause a big change in the tree).
- They tend to be overfitting.
- Training an optimal decision tree is NP-Complete problem[42].
- Trees may be very large.

Decision trees are classified into various types such as CART (Classification and Regression Trees)[48], C4.5[49], CHAID[50], CRUISE[51][52], GUIDE[53] and QUEST[54]. The abovementioned types of decision trees differ from each other in the methods of performing branching. All of these types may be used for classification and therefore, fulfill the goal of this work.

However, the search of the publicly available sources did not reveal source code for most of these algorithms.

The most widely implemented types of decision trees are CART and C4.5:

- CART algorithm is used for both classification and regression that uses binary split with Gini impurity criterion (also known as Gini index).
- C4.5 algorithm is used for classification that uses binary or multiway split with information gain.

Nevertheless, despite using different *splitting rules*, these algorithms are very similar as they repeatedly divide data set into parts with the *maximum homogeneity*. Moreover, in fact, multiway split is used much more rarely than binary split as tree complexity grows much more rapidly in case of multiway split. Due to this fact, the following text refers mainly to classification trees with binary split.

These decisions trees are constructed by repeatedly answering to simple questions (so-called attribute *value tests*), which are determined by *features* (also known as attributes, explanatory variables and so-called *splitters*) of input data, with the input data partition into the smaller parts in each question. Thus, nodes of a tree are represented by questions, its *leaves* represent a final output as a single case or only cases with the same output and its branches represent conjunctions of questions (and so features) that lead to those outputs. The following Figure 2.25[3] shows a simple example with input data of three features: sex, age and the number of spouses or siblings aboard.

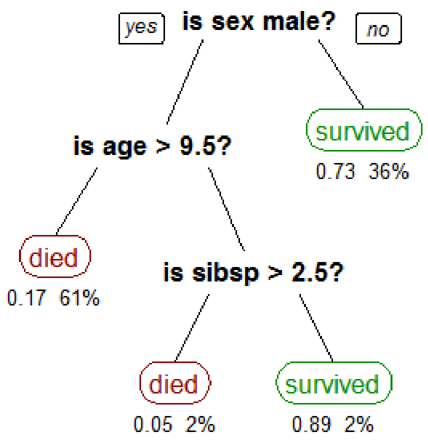


Figure 2.25: A tree showing survival of passengers on the Titanic ("sibsp" is the number of spouses or siblings aboard). The figures under the leaves show the probability of outcome and the percentage of observations in the leaf.

2.3.7.1 Classification and Regression Trees (CART)

Classification and Regression Trees (CART) is an algorithm that refers to both types of decision trees: classification and regression trees. It was introduced in 1984 by Breiman, Freidman, Olshen, Stone [48]. CART decision trees are constructed by repeating *binary split* (simple answering *yes/no*).

CART algorithm searches for all possible features and their values in order to find the best binary split of data. The best split is defined as partition of data into two groups based on *Gini impurity measure* also known as *Gini index*. In case of the classification tree with training samples, the Gini index impurity criterion of a node with some input samples is expressed as follows:

$$Gini\_Impurity = 1 - \sum_{i=1}^{J} f_i^2$$

where there is a set of samples with J classes, and f_i is a fraction of samples labeled with class I in the set.

This criterion defines the node splits, where each split maximizes the decrease in impurity. Reducing error, whether using classification or regression, is a goal behind CART[55].

This process is repeated in each node (question) on increasingly smaller sets until maximum defined depth of tree is achieved or the data set contains only one element and can no longer be divided. The following Figure 2.26 illustrates this process for data with two input features X_1 and X_2.

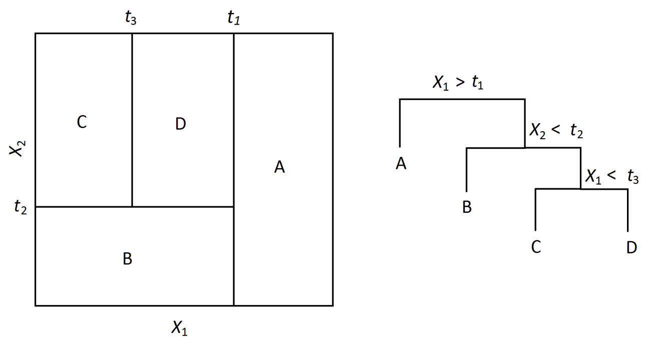


Figure 2.26: CART Recursive splitting. (Left) Data partitioning in 2D space specified by features X_1 and X_2 (Right) Top-down CART.

Due to the nature of CART approach (splitting a data set by variable resulting in two groups with maximum homogeneity), it is clear that CART *automatically* selects the *most important features* and thus, it may be used also for *important feature detection* (which is also one of the goals of this work).

2.3.7.2 C4.5

C4.5 algorithm for classification trees was developed by Ross Quinlan[49]. C4.5 is an extension of Quinlan's earlier ID3 algorithm. R. Quinlan in his work[49] shows that C4.5 produces small and accurate trees for many real-life problems.

C4.5 and its predecessor ID3 create trees using a concept of information gain to evaluate a “goodness” of a test used for split. In particular, they choose a test that extracts the maximum amount of information from a set of cases with the constraint that only one feature (attribute) will be tested at once. Entropy and information gain are expressed as follows:

$$Shannon\ entropy = H(X) = -\sum_{i=1}^{n} P(x_i) \log_b P(x_i)$$

where X is a discrete random variable (classes in case of classification) with possible values $(x_1, \dots, x_n)$ and $P(x)$ is the probability of class x.

$$Information\ gain = IG(X, a) = H(X) - H(X|a)$$

where $H(X)$ is Shannon entropy, a is a feature (attribute), and the split is performed by this feature a.

Quinlan[49] also adds that information gain has a significant weakness in case of feature with unique values, namely a strong bias, as partitioning a training set by this feature leads to a large number of subsets containing just one value. Thus, such subsets contain value corresponding to a single class and so $H(X|a) = 0$ and information gain using this feature for partitioning is maximal. It is clear that such partition is useless as it leads to extreme overfitting. That is why Quinlan [49] considers *information gain ratio* as a replacement of information gain. Consider the information content of a message pertaining to a case that indicates not the class to which the case belongs, but the outcome of the test (subset of samples). The formula represents the potential information generated by dividing X into n subsets and is similar to entropy formula mentioned above:

$$split\ info(X, a) = -\sum_{i=1}^{n} \frac{|X_i|}{|X|} \log_2 \frac{|X_i|}{|X|}$$

where a is a feature by which the partition into subsets was performed, $|X|$ is the number of node input samples, $|X_i|$ is a number of samples in subset i.

On the other side, the information gain measures the information relevant to classification that arises from the same division, and then the information gain ratio is expressed as follows

$$information\ gain\ ratio(X) = \frac{IG(X, a)}{split\ info(X, a)}$$

and expresses the proportion of information generated by the split that appears helpful for classification. The split that *maximizes* information gain ratio criterion is selected.

C4.5 handles some common issues that arise in decision tree construction. First, this algorithm deals with cases where some *feature values are unknown* (just ignore these samples). This straightforward approach was used as Quinlan's research [56] revealed the following issue: no single approach is uniformly superior. Thus, Quinlan uses the approach that is satisfactory for him, without making any stronger claims about it. Second, C4.5 uses a *pruning approach* in order to avoid overfitting. C4.5's pruning method is based on estimating the error rate of every subtree and then replacing the subtree with a leaf node, if the estimated error rate of the leaf (computed in the same way as during the tree construction phase) is lower.

In addition, recently conducted research [57] shows that, in case of C4.5, increasing a number of non-representative training samples is useless as C4.5 algorithm constructs the same decision tree in all such cases.

2.3.7.3 Ensemble of trees

2.3.7.3.1 Random forest

An idea of *Random Forests* (RFs) was presented in 2001 by Breiman [58]. Breiman defines RF as a classifier consisting of a collection of tree-structured classifiers unlike CART and C4.5 mentioned above:

$$h(x, \theta_k) \quad \text{for } k = 1, \ldots;$$

where x is a an input sample for its output prediction, k is the number of classifiers and θ_k are independent identically distributed random vectors which define properties of trees (e.g. number of examples in a training set in bagging). The nature and dimensionality of θ_k depend on its use within the process of tree construction. After the training phase is performed, each tree casts a unit vote for the most popular class at input x. Thus, the RF prediction is made by majority vote of classifiers in case of classification and by averaging classifiers' predictions in case of regression. All mathematical assumptions and definitions may be found in [58].

Each tree in RF is constructed in the following way[59]:

- If the number of cases (samples) in the training set is N, the researcher should select N cases at random - but with replacement, from the original data. About a third of cases are left out of the selected cases (so-called *oob* to refer to *out-of-bag* and used for the same goal as test data set, i.e. for error rate estimation) and are used to get a running unbiased estimate of the classification error as trees are added to the forest. It is also used to get estimates of importance of features. The selected cases will be the training set for growing the tree.

- If there are M input features, a number $m << M$ (m is much lesser than M) is specified such that at each node, m features are randomly selected out of the M, and the best split on these m is used to split the node. The value of m is constant during the forest growing.

- Each tree is grown to the largest extent possible. There is no pruning.

Breiman[58] shows in his paper that the forest error rate depends on:

- The *correlation* between any two trees in the forest. Increasing the correlation increases the forest error rate.

- The *strength* of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate

Reducing m (randomly selected features out of the M for subsequent using in splitting) reduces both the correlation and the strength, whereas increasing m increases both. Somewhere in between is an "optimal" range for m, which is usually quite wide. A value of m in the range can quickly be found using the *oob error rate*. This is the only adjustable parameter to which random forests are somewhat sensitive.

One of the very useful things that RF performs is *feature importance estimation*. In every tree in the forest, use the *oob cases* (used as test data set) and count the number of votes cast for the correct class. Now randomly permute the values of some feature f in the *oob cases* and again use these cases as input for the tree. Subtract the number of votes for the correct class in the *feature-f-permuted oob data* from the number of votes for the correct class in the *untouched oob data*. The average of this number over all trees in the forest is the raw importance score for the feature f.

Another very important character of RF is handling missing data, which is done by *proximity measure*. It is defined as frequency of unique pairs of training samples (in and out of bag) that end up in the same terminal node (leaf) and are normalized by dividing by the number of trees in the forest. Then this proximity measure is used as weights in missing value replacement[59].

Thus, the advantages of RF are:

- It is unexcelled in accuracy among current algorithms.

- It runs efficiently on large data bases.

- It can handle thousands of input variables without variable deletion.

- It has methods for balancing error in class population unbalanced data sets.

- Accuracy and feature importance are generated automatically.

- It handles overfitting.

- It is not very sensitive to outliers in the training data.

- No need for pruning.

- It has a few parameters, easy to set.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.

The main limitation of RF refers to regression problems as predictions cannot be done beyond the training data range, and extreme values often cannot be predicted accurately due to the nature of decision trees.

Recently conducted research[57] confirms that C4.5 and Random Forest outperform other algorithms of decision trees, and Random Forest outperforms C4.5.

2.3.7.3.2 Time Series Forest (TSF)

Time series are nowadays widely introduced in our lives within the spheres of medicine, geology, physics, finance etc. Classifying the time series was viewed as a necessary thing, and a new tree-ensemble method was proposed in 2013[60].

The method is named *Time Series Forest* (TSF) and uses the idea of Random Forests[58] with regard to constructing multiple decision trees with a majority vote of these trees as a final prediction, random sampling of training data set for each tree, random sampling of features (that makes the computational complexity linear in the time series length) and independently growing trees, which makes parallelization possible .

TSF also introduces a new splitting rule as a combination of entropy gain and a distance measure, referred to as *Entrance gain*, in order to evaluate high quality splits. TSF captures simple temporal features over time series intervals (so-called *interval features*) such as mean, standard deviation and slope and then uses them for distinguishing time series from one another. The following Figure 2.27 [60] provides an example of time series, which may be distinguished by this method.

Though TSF uses simple summary statistical features, it may also use more complex features such as wavelets.

Experimental results[60] show that implementing Time Series Forest is comparable to widely used alternatives such as 1-Nearest Neighbor with Dynamic Time Warping (DTW) and Random Forest, moreover, TSF often outperforms them despite the fact that 1-NN with DTW is one of the best classifiers for time series and it is extremely hard to beat[61].

2.4 Features

The main goal of any classification/regression task is to produce some kind of output based on the input data. Each sample in this input data is described by one or many so-called *features* (sometimes called attributes or dimensions). Features may be represented by nominal, numerical values or even by more

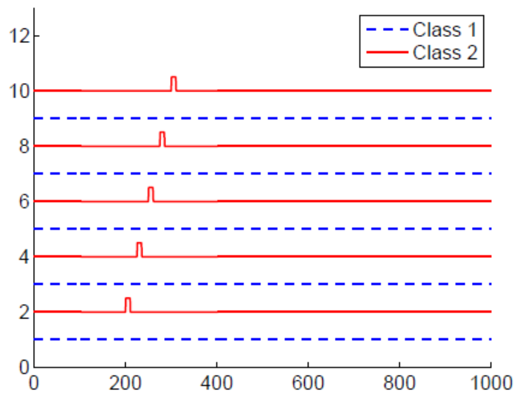


Figure 2.27: The time series from class 2 have sudden changes between time 201 and time 400. An interval feature such as the standard deviation between time 201 and time 400 can distinguish the time series from the two classes.

complex structures such as vectors, matrices etc. Sometimes features cannot be used in data mining directly and must be preprocessed in some way. However, even in case of nominal or numerical features, it may not be possible to produce accurate enough outputs without data preprocessing. Inaccurate outputs may be often a result of very high dimensional feature space (also known as *curse of dimensionality*), lack of good enough (i.e. representative) features or, less often, occur due to the lack of features themselves. Two techniques, *feature extraction* and *feature selection*, are widely used to deal with the abovementioned problems.

2.4.1 Feature Extraction

Feature extraction process is used in case of the lack of features: new features are extracted from the data in some way in order to construct feature space that will be used for building a quality classifier. Examples of simple statistic features, which may be extracted from time series, include mean, standard deviation, min and max values etc.

Some advanced feature extraction algorithms have been proposed for time series such as *Discrete Fourier Transform* (DFT), *Discrete Wavelet Transform* (DWT) and *Singular Value Decomposition* (SVD). The main disadvantage of SVD algorithm is its poor time complexity $O(m + n^3)$, where m is a number of time series in the data set, n is a length of these time series and $m >> n$.

On the other side, DFT and DWT both are computationally efficient: *Fast Fourier Transform* (FFT) algorithm for DFT has time complexity $O(n \log_2 n)$ and DWT has even better time complexity $O(n)$[62][63]. Thus, the only reasonable algorithms for time series feature extraction from those mentioned are FFT and DWT.

2.4.1.1 Discrete Fourier Transform (DFT)

In 1822, Joseph Fourier invented Fourier analysis and showed that periodic functions may be represented as a sum of harmonics (periodic functions), sines and cosines. The approach has been developing since that time, and many follow-up researches have been performed, including extension of the original idea for the non-periodic functions. This extension is known as *Fourier Transform.*

Nowadays, *Discrete Fourier Transform* (DFT) is the most common method of Fourier analysis. DFT converts a finite sequence of equally time spaced samples (time series) into a sequence of equally time spaced samples of the *Discrete-Time Fourier Transform* (DTFT) of the same length as the original sequence), which is a continuous complex function of frequency. In this case, *complex function* means function that operates with complex values consisting of real and imaginary parts. Both Fourier analysis and Fourier Transform works with functions (i.e. with continuous variables and thus, they are computationally expensive), whilst any signal can be measured only in a finite number of points, which are observed in discrete time. Thus, it is a reason why DFT was developed. The short summary is provided below (see the complete mathematical background in [62]).

One of the important features of DFT is the fact that DFT *preserves energy* (defined in section 2.1.3) of time series as *in Rayleigh Energy theorem* (*Parseval's theorem*). The useful consequence of this theorem is that DFT *preserves Euclidean distance* between two time series.

A new computed sequence of complex values, known as DFT coefficients, is sufficient for the reconstruction of the original sequence. Moreover, k-th DFT coefficient from the beginning is a conjugate of k-th DFT coefficient from the end, which means only half of all coefficients should necessarily be stored and, therefore, this property may be used for simple data reduction by half of the original sequence length. It is also necessary to mention that often only first few coefficients are significant and, thus, useful[64][65]. This also follows from the terms of energy of the time series as the first few DFT coefficients (and the last few due to their symmetry) contain, in general, most of energy and so the coefficients are expected to capture the raw shape of time series. The following Figure 2.28 shows DFT on the EC curve of an arrhythmogenic rabbit.

Based on what was said before, time series may be approximated by only first few coefficients while preserving the basic trends of the time series. And better approximations up to the identity may be constructed using larger quantities of DFT coefficients.

The disadvantage of DFT algorithm is its poor time complexity $O(n^2)$, where n is the length of time series. Fortunately, *Fast Fourier Transform*

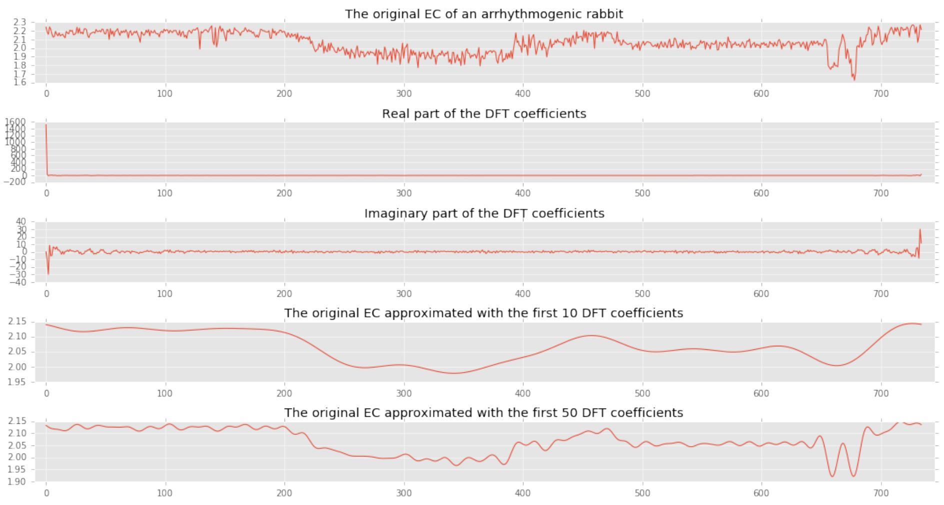


Figure 2.28: The example of DWT coefficients and of EC reconstruction using DFT coefficients. From top to bottom: original EC, real part and then imaginary part of DFT coefficients, EC reconstruction with the first 10 and the first 50 DFT coefficients.

(FFT) algorithm for DFT was developed in 1965 by Cooley and Tukey[66]. It is an efficient algorithm with good time complexity $O(n)$.

2.4.1.2 Wavelet Transform (WT)

The weakness of Fourier analysis is in extracting meaningful and sufficient information from time series, which are non-periodic (time-varying) and/or non-stationary. The classical Fourier Transform (and DFT as well) operates within harmonic (i.e. periodic) functions such as sines and cosines and, thus, it is clear that compression of a non-periodic time series, moreover, non-stationary time series, using this method is complicated and cannot be performed efficiently[67]. Figure 2.29[68] below illustrates the problem clearly: two different time-varying signals have the same power distribution into their frequency components, from which the original signals are composed. Obviously, the periodograms of the time-varying signals cannot capture differences in time domain where these signals have the different behavior.

In other words, it was necessary to develop some method that would analyze not only frequency domain, but would also take into account time domain of the inspected time series. The problem was resolved in 1946 in time-frequency analysis with *Short Time Fourier Transform* (STFT)[69]. The idea of STFT is based on *using sliding windows of the fixed size* on an inspected time series with subsequent computation of Fourier Transform for each of those windows. The disadvantages of this method are:

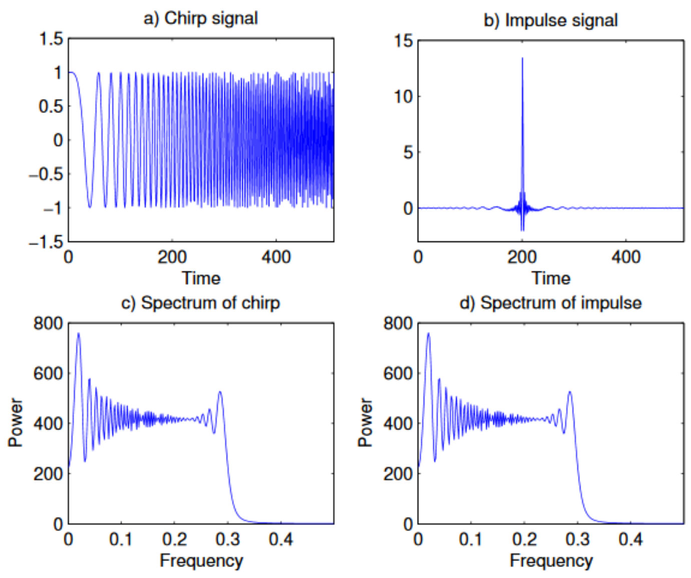


Figure 2.29: Two different signals with exactly the same periodograms

- precision of the obtained information depends on the size of the used window
- size of the used window is the same for all frequencies and, thus, the method may not provide enough information (i.e. some frequencies may require more detailed research)

The theory of Wavelet analysis was developed based on the Fourier analysis, and *Wavelet Transform* (WT) is the logical sequel of STFT. Wavelet Transform introduces sliding windows of *variable size* and replaces harmonic functions with family of functions called *wavelets* (“wavelet” is interpreted as “small wave”). Wavelet must be oscillatory and must have limited duration. The term “wavelet function” is used to refer to either orthogonal or non-orthogonal wavelets. The term “wavelet basis” refers only to an orthogonal set of functions. The use of wavelet basis implies the use of *Discrete Wavelet Transform* (DWT), while a non-orthogonal wavelet function may be used in either DWT or *Continuous Wavelet Transform* (CWT)[70].

Unlike the original idea of the time-frequency approach, WT uses *time-scale* domain analysis, where lower scales (i.e. shorter windows) correspond to the higher frequencies, which capture local trend, and higher scales (i.e. longer windows) correspond to the lower frequencies, which capture more general trend of the time series. Thus, Wavelet analysis is based on splitting a signal into shifted and scaled versions of the selected wavelet. Figure 2.30 illustrates the whole idea.

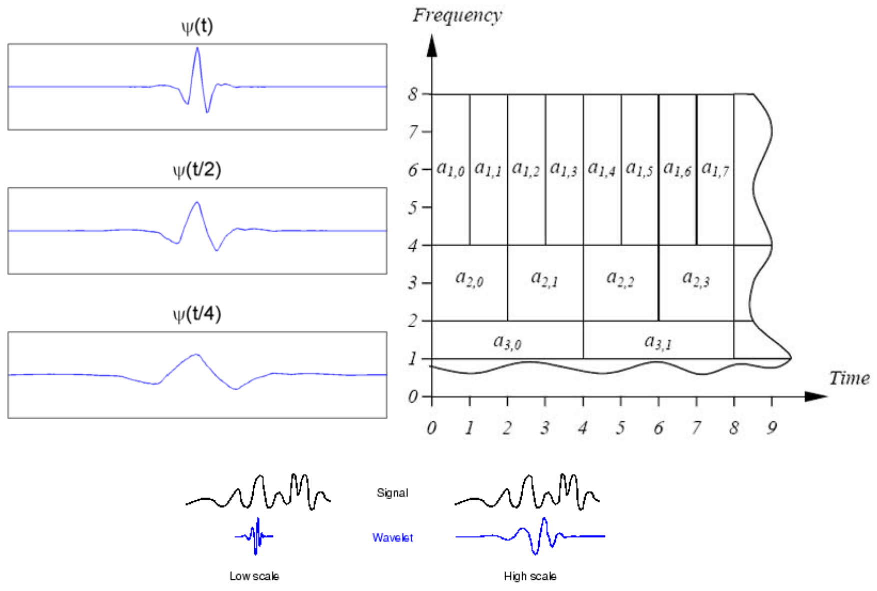

ψ(t)
ψ(t/2)
ψ(t/4)
Frequency
Time
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,4}$ $a_{1,5}$ $a_{1,6}$ $a_{1,7}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$
Signal
Wavelet
Low scale
High scale

Figure 2.30: (Top-left) Wavelet at scale 1,2 and 4 (Top-right) Illustration of relationships between time, frequency and scale (denoted as ”a”) (Bottom) Example of signal approximation with wavelets at low/high scale

Theory of wavelets is based on the *multi-resolution analysis* (MRA)[71] and the fact that a function may be approximated by *multilevel resolution approach*. In return, MRA is based on the relationship between the *scaling filter H* and the *wavelet filter G*. Mathematical details for theory of wavelets, including CWT and DWT, may be found in several papers[72][73][62][74].

2.4.1.2.1 Discrete Wavelet Transform (DWT)

CWT is highly redundant, because it analyzes a signal at all possible scales and translations, which also requires an infinite number of wavelets for the underlying analysis. Thus, especially at the large scales, the wavelet spectrum at adjacent times is highly correlated and so there is a redundancy. Moreover, CWT is computationally expensive and for that reason is practically unusable despite being very stable in terms of *shift invariance* and offering a very detailed view on the inspected signal. DWT was developed as a faster and more efficient algorithm. DWT is based on using the appropriate scale and translation sampling strategy (in powers of two: 2^1, 2^2 etc.), called *dyadic sampling*, while reducing the number of wavelets used for decomposition. In other words, DWT decomposes a signal into mutually orthogonal set of wavelets; it is the main difference from CWT, which decomposes a signal into non-orthogonal set of wavelets containing redundant information. However, due to the used sampling strategy for scales and translations, DWT *is not shift-invariant* in contrast to CWT, and a simple shift in a signal can cause a significant realignment of signal energy in the DWT coefficients. It is important to mention that DWT *captures all essential details* of the signal despite the applied sampling strategy.

DWT also *preserves* energy of time series just like DFT does, but DWT differs from DFT in spreading this energy among its coefficients. In general, the first few DFT coefficients accumulate the majority of energy and, therefore, selecting them is enough for reconstructing the raw shape of the original time series. Spread of energy in DWT coefficients is quite different as only the few *most significant (in absolute value)* coefficients allow reconstructing the original time series with enough precision. In other words, by removing the insignificant coefficients and then reconstructing the signal using these truncated coefficients, it is possible to smooth the signal without smoothing over all of the interesting peaks the way it happens with a moving average. Figure 2.31 below shows differences in approximation depending on whether most significant coefficients or the first few coefficients are used.

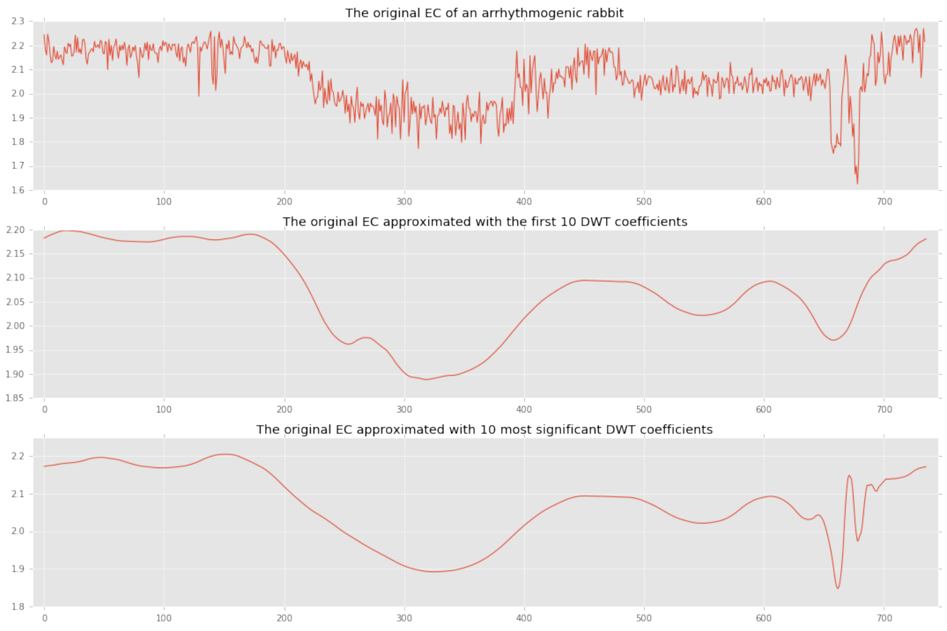


Figure 2.31: Example of EC reconstruction by DWT: (center) by the first 10 coefficients (bottom) by 10 most significant coefficients. The original EC is at the top.

DTW coefficients are calculated by passing a signal through a *filter bank*[74] consisting of *high-pass filter* (wavelet function, also called *mother wavelet*) and *low-pass filter* (scaling function, also called *father wavelet*). In order to clarify roles of the abovementioned elements and their relationships, it may be useful to read a paper by Guido[75], as there is often misunderstanding. On each pass through the filter bank, the scaling function captures the low frequency data (they concentrate most of the energy) from the previous approximation. Low frequency components are reduced in each subsequent approximation due to the effect of dyadic sampling (i.e. the length is reduced by 2 at each level). Figure 2.32 [10] below illustrates the process.

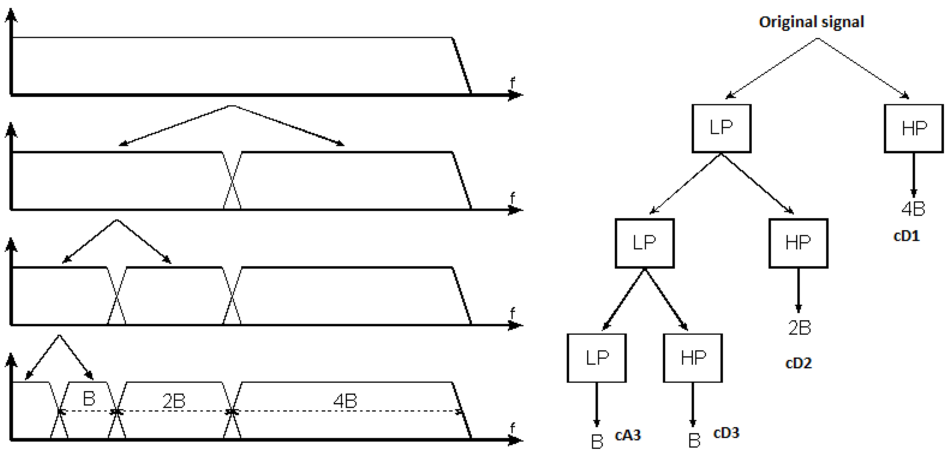


Figure 2.32: DWT decomposition. *LP* is low-pass and *HP* is high-pass filters. *cA*3, *cD*3, *cD*2, *cD*1 are the result DWT coefficients, where *cA* is approximation coefficients and *cD*∗ are details coefficients.

Techniques based on Wavelet Transform are used nowadays in many areas such as medicine, audio, image processing etc. Many problems in these areas have not been solved, but nowadays there is a significant progress. For example, in 2004, Martinez, Almeida and Olmos et al.[76] proposed and evaluated a robust ECG delineation system based on DWT outperforming other algorithms. Zhang and Ho[77] proposed unsupervised feature extraction algorithm for automatic selection of feature dimensionality.

Discussion on time series analysis and mining, based on DWT, may be found in [78].

DWT has good time complexity $O(n)$[62].

2.4.1.2.2 Discrete Wavelet Transform: wavelet selection

There are many different wavelet families like Haar, Daubechies, Symlets, Coiflets, Biorthogonal etc. Each wavelet family is represented by one or many wavelets: Haar for Haar; “db1”, “db2”, . . . , “db20” for Daubechies; “sym2”, . . . , “sym20” for Symlets etc., where the numbers indicate length of the used filter (note that Haar wavelet is identical to “db1”).

[10] `https://www.slideshare.net/dineshkumarc1/tl-50801785`

Recent research on the resting-state fMRI data for detecting schizophrenia[79] showed that the selected filter length had a great effect on the classification results, while wavelet family had an insignificant effect.

Chan and Fu[80] proposed and studied the use of Haar Wavelet Transform technique for dimensionality reduction with the subsequent similarity search within the inspected set of time series with no false dismissals competitive to DFT, and they also studied the effect of Haar wavelet-based approximation function for time warping distance, called Low Resolution Time Warping.

Popivanov and Miller[81] showed that any bi-orthonormal wavelet (Symlet, Daubechies, Coiflets) may be used in similarity search with no false dismissals. Moreover, they also detected a filter length impact on the result precision and considered the fact that filter length reflects the length of patterns in the inspected time series.

Dond, Sun and Xu[82] studied DWT based feature extraction in tissue classification problem. They showed that the used wavelet family has great impact on the classification results, and that the detail signals D (derived from details coefficients) as features outperform approximation signal (derived from approximation coefficients), moreover, the first level detail signal $D1$ as feature had the greatest classification accuracy.

Tumari, Sudirman and Ahmad[83] in selecting a suitable wavelet for cognitive memory using EEG signal detected an impact of the selected wavelet family on the results.

2.4.2 Feature Selection

The feature selection is another important process of data preprocessing. The number of the available features can greatly increase in case different feature extraction techniques are used. However, not all of them are actually useful and help in classification, moreover, high amount of the used features lead to overfitting as the feature vector of each sample becomes unique. Different techniques for selecting the important features have been proposed such as recursive feature elimination for recursive considering smaller subsets of features and feature importance estimation using embedded capabilities of machine learning algorithms (e.g. Random Forest 2.3.7.3.1).

Chapter 3

Technologies

The following software was used in this work:

- Python 3.5
- Jupyter Notebook [11]
- Scikit-learn (also known as sklearn) for machine learning [12]
- Anaconda3 for many useful scientific packages [13]
- Trial version of GMDH Shell [14] for GMDH Neural Network
  - GMDH Shell was used as no good tested and verified implementations of GMDH approach were found.

[11] http://jupyter.org/
[12] http://scikit-learn.org/
[13] https://www.continuum.io/
[14] https://www.gmdhshell.com/time-series-forecasting

Chapter 4

Data preparation

In data analysis and data mining, the original data almost always require some preprocessing before making further assumptions and applying machine learning algorithms. In our case, it also matters.

Due to the fact that the rabbits are subject to different number of DIIs (as was mentioned in section 1.2), it is necessary to select the minimum number of DIIs for all rabbits in order to perform further research properly (it is obvious that comparison of ECs must be performed for all rabbits after the first DII, the second one, etc.). Moreover, a time interval between DIIs can also vary. This fact implies the following restriction: it is necessary to find the minimum of maximum time ranges (so-called *Min-max theorem*) after each DII in order to compare ECs not only after the same DII, but also within the common minimum time range. The following Figure 4.1 illustrates the abovementioned statements.

After performing the abovementioned data transformations, it was identified that the minimum number of DIIs for all rabbits is equal to *one* (i.e. some rabbits died between second and third DIIs). As each rabbit has control interval, and these control intervals differ in length, it is necessary to follow the same logic as in case of intervals between DIIs: to find the minimum length of control interval and truncate each control interval to that identified minimum length.

Thus, the two intervals will be used in experiments: control interval and the first interval (after the first DII).

The following notation will be introduced in order to clarify two types of ECs:

- *Original EC*: no preprocessing
- *Truncated EC*: the original ECs were truncated to the minimum number of DIIs with the appropriate minimum time range after each DII. The truncated ECs also contain the truncated control interval.

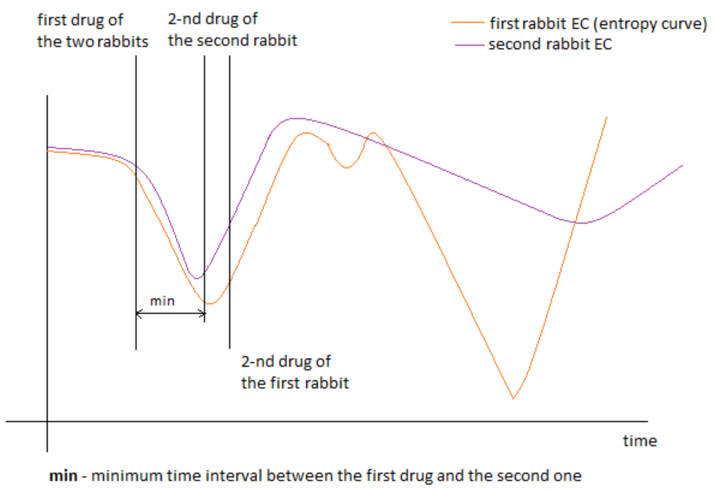


Figure 4.1: Min-max theorem: selecting minimum time range from the maximum time ranges.

CHAPTER 5

Experiments

This section contains complete information about the conducted experiments and their settings. Important metrics for performance evaluation are also described in detail.

5.1 Performance evaluation

Once classifier has been constructed and exists, it is necessary to estimate its performance. There are different approaches on how to do that in case of classification and in case of regression.

5.1.1 Performance metrics for classification

The following metrics exist in case of classification:

- Accuracy
- Precision (Positive Predictive Value)
- Recall (sensitivity)
- Specificity
- F_1 score
- ROC AUC

The most common metric for performance estimation is *accuracy.* It is the proportion of correctly classified samples to all samples.

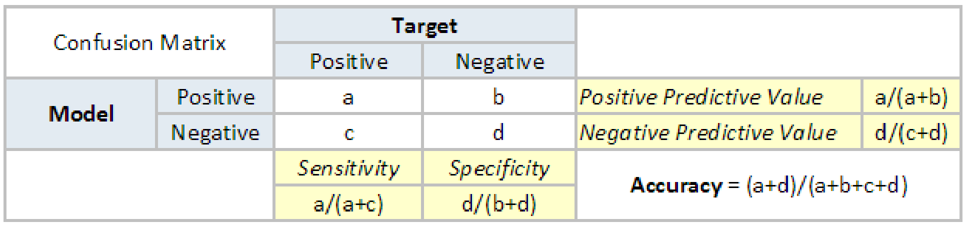
However, accuracy is not enough in case of binary classification such as detection of diseased/healthy people as accuracy tells only the probability that a patient is classified as diseased or healthy. However, the classification accuracy in real world is rarely equal to 100% and, thus, misclassified cases are present.

In such cases, accuracy is insufficient metric itself and the more valuable metric is required as the primary thing to know is *"what is the probability that a patient classified as diseased is indeed diseased?"*. *Precision, recall, specificity, F_1 score and ROC AUC* were introduced in order to answer this and similar questions.

It is necessary to provide some definitions in order to specify five abovementioned metrics. In context of disease detection, positive class ("1") means the classifier predicts that the disease is present and negative class ("0") means the classifier predicts that the disease is not present. The following notations are made based on the abovementioned assumptions:

- True positives (TP): diseased patients classified as diseased
- False positives (FP): healthy patients classified as diseased
- True negatives (TN): healthy patients classified as healthy
- False negatives (FN): diseased patients classified as healthy

The most common instrument to capture the abovementioned notations is *confusion matrix* (see Figure 5.1 [15]. The notation is the following: "Target" as actual value and "Model" as the predicted value.



Confusion Matrix		**Target**			
		Positive	Negative		
Model	Positive	a	b	*Positive Predictive Value*	a/(a+b)
	Negative	c	d	*Negative Predictive Value*	d/(c+d)
		Sensitivity	*Specificity*	**Accuracy** = (a+d)/(a+b+c+d)	
		a/(a+c)	d/(b+d)		

Figure 5.1: Confusion matrix

The following Figure 5.2 [16] illustrates the notations and their relationships in the different way.

Now, accuracy, precision, recall and specificity can be expressed as:

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

[15]`https://www.analyticsvidhya.com/blog/2016/02/7-important-model-evaluation-error-metrics/`)

[16]`https://en.wikipedia.org/wiki/Precision_and_recall`

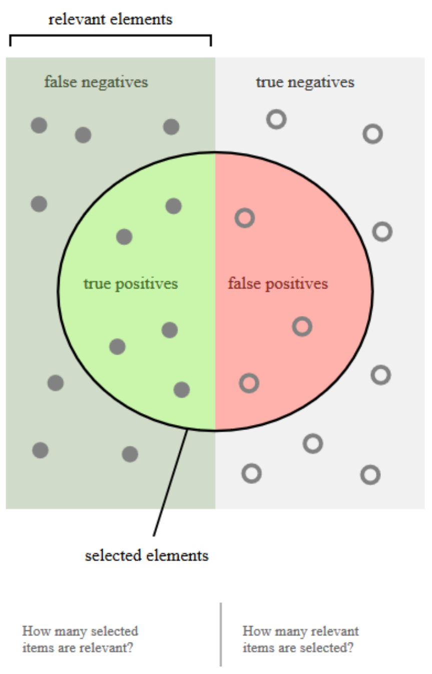


Figure 5.2: Another view on the TP, FP, TN, FN

$$specificity = \frac{TN}{TN + FP}$$

ROC (Receiver Operating Characteristic) curve is a graphical representation of relationship between *recall* and *(1 – specificity)*, indicated as y-axes and x-axes, and illustrates the performance of a binary classifier (see Figure 5.3 [17]). The maximum performance is achieved at point $[0, 1]$ and diagonal represents the case of random guessing. ROC AUC (Area Under the Curve) is a numerical estimation of classifier's quality in case of using ROC curve. ROC AUC is equal to 0.5 in case of random guessing.

The last metric of interest is F_1 score. F_1 score is the harmonic mean of precision and recall and is expressed as:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Multiplication by 2 is used for scaling purposes in case precision and recall are both equal to 1.

5.1.2 Performance metrics for regression

The following performance metrics are used in case of regression:

[17] `https://stats.stackexchange.com/questions/132777/what-does-auc-stand-for-and-what-is-it`

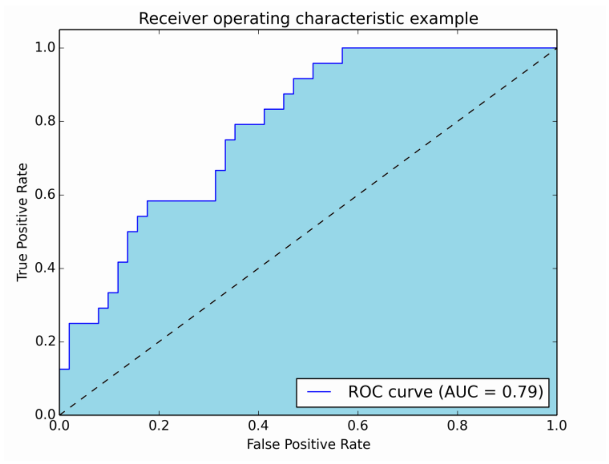


Figure 5.3: ROC AUC

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- R2 score

MAE and MSE are expressed as follows:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|f_i - y_i|$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

where N is a number of samples, f_i is the classifier output for sample i and y_i is true output of sample i.

R2 score requires extra definitions:

$$\bar{y} = \sum_{i=1}^{N} y_i$$

where $\bar{y}$ is a mean of the observed data, y_i is true output of sample i.

$$total\ sum\ of\ squares = SS_{tot} = \sum_{i=1}^{N}(\bar{y} - y_i)^2$$

$$residual\ sum\ of\ squares = SS_{res} = \sum_{i=1}^{N}(f_i - y_i)^2$$

where N is a number of samples, f_i is the classifier output for sample i and y_i is true output of sample i.

Finally R2 score is expressed as:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

5.2 Settings of experiments

In the conducted experiments, arrhythmogenic rabbits were marked as positive ("1") class and non-arrhythmogenic rabbits were marked as negative ("0") class.

The following approaches were used in the conducted experiments:

- ECs were grouped by value of L parameter (1, 5, 10 etc.) and classification results were computed for each group
- F_1 score (mean of the computed metrics, precision and recall, for each class, positive and negative) for parameter optimization and for general evaluation of classification results
  - accuracy, ROC AUC, recall (sensitivity) and specificity for more detail view of results
  - average of accuracy, ROC AUC, recall and specificity (further referred to as ARARS score) was used for indication of classification quality
- 10-fold cross validation x 10 times
  - Each cross validation divided data into folds and each fold preserved the initial ratio of samples between classes (so-called *stratified k-fold cross validation*)
  - All cross validations were performed with different shuffling of the input data (i.e. they had different data partitions into folds)
  - Cross validation results were averaged for each fold in order to compute final results
  - Average of all folds for the appropriate group of ECs was the final result for such group

The input data set is slightly unbalanced with proportion 1:2 (13 negative and 24 positive samples) and it may result in overfitting of the used algorithms on the positive samples. However, use of average results of ten different stratified 10-fold cross validations and parameter optimization of algorithms based on F_1 score mentioned above may fix this issue of unbalanced class distributions. Moreover, other metrics such as recall, specificity and ROC AUC were used for verification of quality of the achieved results. Despite of the abovementioned steps, it may still be not enough for handling imbalance in the input data and thus, the used algorithms were tested in normal unbalanced mode and in so-called balanced mode based on the weights for balancing contributions of class samples to classification.

Optimization of algorithm parameters was performed using exhaustive search [18] over the specified parameter values, i.e. all possible combinations of the specified values were tested.

5.3 Used machine learning algorithms

The experiments were conducted using the following machine learning algorithms:

- Random Forest (RF)
- Support Vector Machine (SVM)
- Logistic regression (LR)
- k-nearest neighbors (k-NN)

Ensemble learning, namely AdaBoost, was performed with slightly different set of algorithms as SVM, k-NN and Logistic regression algorithms are very sensitive on their parameters and the input data. This sensitivity results in classification performance less than 50% and, therefore, these algorithms could not be used in AdaBoost algorithm. The used algorithms were:

- Extremely Randomized Trees (ExtraTrees)
- Decision tree (CART)
- Random Forest

5.4 Feature extraction

This section describes features, which were extracted from the input data and used for further analysis. In case of univariate features, box-and-whisker plots of *arrhythmogenic* (dead) and *non-arrhythmogenic* (alive) rabbits were used in order to make an assumption that the inspected feature can discriminate between two abovementioned classification groups. Evaluating usefulness of multivariate features (e.g. subintervals in this context due to the fact the whole time series is represented as several values, each of the values is computed for a single subinterval) is impossible and, thus, they always will be used for further analysis by machine learning algorithms.

[18]GridSearchCV `http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`)

5.4.1 Statistic features

There are simple and advanced statistic features. Simple statistic features are commonly used for every numerical data and were the first choice, however, these statistic features didn't provide sufficient classification results. Advanced statistic features are less common and were introduced in order to reach better classification results.

The following statistic features were used:

- Simple
  - Mean
  - Standard deviation
  - Variance
  - Min and Max
  - 25th, 50th and 75th percentiles
- Advanced
  - Integral
  - Skewness
  - Kurtosis
  - Slope
  - $(Max - Min)/length$ of time series
  - Energy of time series
  - Sum of values of time series
  - Trend (uptrending, downtrending or without trend)

Some of the abovementioned statistic features will be described below.

5.4.1.1 Integral

In order to integrate ECs given by discrete values, the *composite Simpson's rule*[19] was used.

5.4.1.2 Skewness

Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. The skewness value can be positive or negative. *Negative skew* indicates that the *tail* on the left side of the probability density function is *longer* or *fatter* than on the right side. *Positive skew* indicates that the tail on the right side is *longer* or *fatter* than on the left side. Skewness is 0 in case of the symmetric distribution. Skewness was computed for all values of EC.

[19] `https://en.wikipedia.org/wiki/Simpson%27s_rule#Composite_Simpson.27s_rule`

5.4.1.3 Kurtosis

Kurtosis is similar conceptually to skewness, because kurtosis describes shape of the probability distribution. Kurtosis is the fourth central moment divided by the square of the variance. Normal distribution has kurtosis equal to 0.

5.4.1.4 Slope

A *slope* (also called *gradient*) was computed for each EC curve and is expressed as

$$slope = \frac{\triangle\ y}{\triangle\ x} = \frac{y(x_{last}) - y(x_{first})}{x_{last} - x_{first}}$$

where $y(x)$ is value of time series (EC) in time x. Actually x denotes i-th element of time series rather than time from the original measures. Thus, x_{last} is equal to the length of EC and x_{first} is equal to 1.

If a slope

- > 0: time series is increasing, i.e. it goes up.
- < 0: time series is decreasing, i.e. it goes down.
- $= 0$: time series is constant

5.4.1.5 Sum and Cumulative sum

Sum was used on the following data:

- On the whole EC data
- On each drug interval in truncated EC data
- On truncated EC data with concatenated intervals

Cumulative sum was used on the following data:

- On the computed gradients of the truncated EC for indication of downtrending (-1), uptrending (1) and no trending (0)

5.4.1.6 Variance

Inspection of variance of EC values revealed some dependencies in the particular intervals and in the concatenated intervals. They are especially good for EC #60 in control interval, EC #500 in the first interval, EC #5 in the concatenated intervals (see Figures 5.4 below). Thus, this feature was used in further classification.

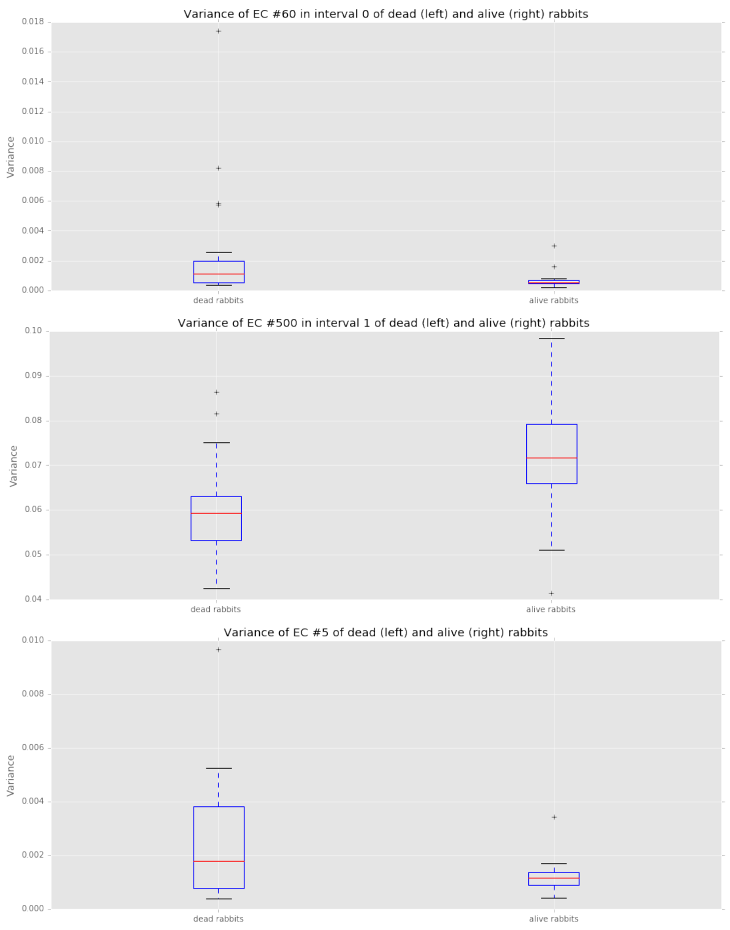


Figure 5.4: Variance: distribution of values for dead (left) and alive (right) rabbits. From the top: EC #60 in control interval, EC #500 in the first interval and EC #5 in the concatenated intervals.

5.4.1.7 Standard deviation

Standard deviation is a square root of variance, and the same behavior is expected. However, standard deviation sometimes has outliers that differ from those for variance (see Figure 5.5 below for example of such difference) and for that reason may be useful.

5.4.1.8 Maximum and minimum values

Difference of maximum and minimum values of time series (divided by the length of time series) demonstrates interesting dependencies in some cases (Figures 5.6 below show examples) and so it was used in further data analysis. Also due to the same reason simple maximum and minimum values were used.

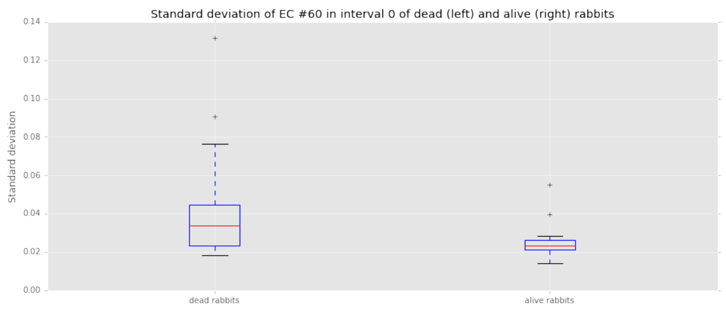


Figure 5.5: Standard deviation: distribution of values for dead (left) and alive (right) rabbits for EC #60 in control interval.

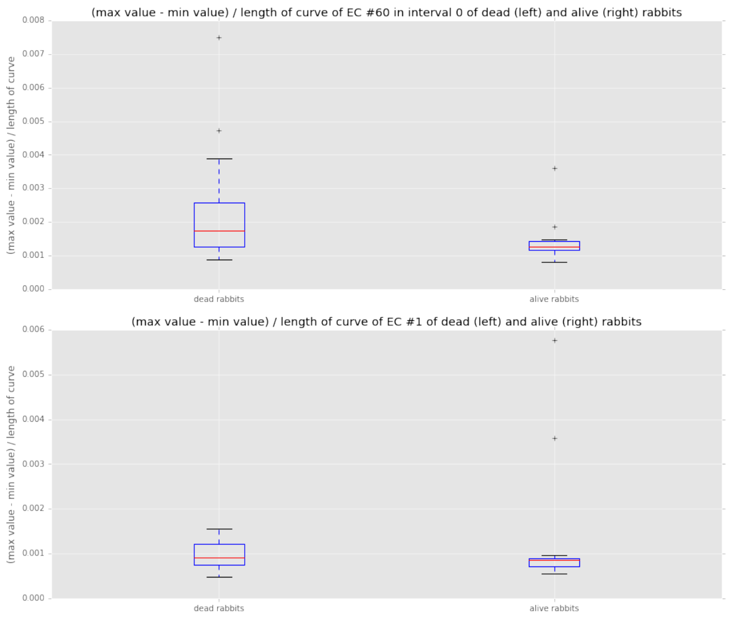


Figure 5.6: $max - min/length$: distribution of values for dead (left) and alive (right) rabbits. From the top: EC #60 in control interval and EC #1 in the concatenated intervals.

5.4.1.9 Energy of time series

The cumulative sum of energy 2.1.3 in each interval and in the concatenated intervals was inspected. The perspective results were revealed in the transitional (non-final) cumulative sums and in the energy of the whole ECs (Figures 5.7 below show examples) and, thus, the energy of ECs was used as feature in machine learning algorithms. In order to take transitional cumulative sums into account, values of EC time series were divided into one minute subintervals consisting of 12 EC values (as EC values were measured in a five seconds interval).

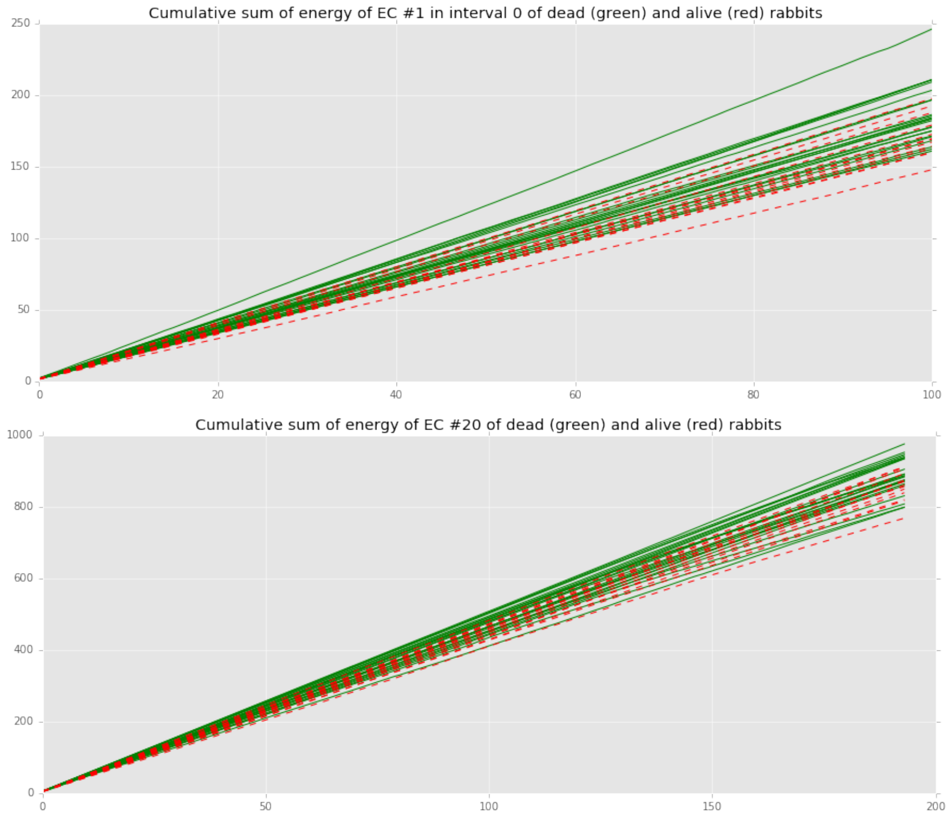


Figure 5.7: Cumulative sum progress for dead (green) and alive (red) rabbits. From the top: EC #1 in control interval and EC #20 in the concatenated intervals.

5.4.1.10 Features in subintervals

In the case of one minute subintervals, it is impossible to evaluate usefulness manually as there are too many dimensions (subintervals) with their own values of the same feature, which results in very complicated behavior. Thus, all subinterval features were used in further data analysis by machine learning algorithms. These features are listed below:

- Difference between max and min EC values in subinterval
  - (max - min)/(length of subinterval) as the final subinterval may have different length than previous ones.

- Maximum value in subinterval
- Minimum value in subinterval
- Mean of values in subinterval
- Energy of time series in the subinterval 2.1.3
- Continuous cumulative sum of energy in subinterval
  - (sum of energy in current subinterval) + (sum of energy in the previous subinterval)
- Sum of values of time series in subinterval
- Continuous sum of values in subinterval
  - (sum of values in current subinterval) + (sum of values in the previous subinterval)
- Slope of subinterval

  $$slope = \frac{\triangle\ y}{\triangle\ x} = \frac{last\ y - first\ y}{last\ x - first\ x}$$

  where y are time series values and x are indices of these values
- Variance of subinterval
- Standard deviation of subinterval
- Integral of a curve defined by values in subinterval
- Skewness of a curve defined by values in subinterval
- Kurtosis of a curve defined by values in subinterval
- Trend of the subinterval

5.4.2 Discrete Fourier Transform

The truncated ECs were used as input for DFT algorithm and the first 15 complex (with real and imaginary parts) DFT coefficients were computed for each EC. Then the important real and imaginary coefficients were identified using Random Forest for each value of L parameter. In the final step, five most important coefficients (from real and imaginary parts considered together) were selected and used as features.

The second used approach consisted in classification of ECs reconstructed by the first ten DFT coefficients. The five most important time moments were identified for the reconstructed ECs using Random Forest and used as features.

The achieved classification performance for these two approaches may be found for SVM in 6.0.2.1, for Random Forest in 6.0.1.1, for logistic regression in 6.0.3.1 and for k-NN in 6.0.4.1.

5.4.3 Discrete Wavelet Transform

The first approach consisted in using DWT algorithm for computation of DWT coefficients for each EC at the maximum level of signal decomposition. However, unlike DFT algorithm, in order to perform DWT algorithm properly, it is necessary to find the most suitable wavelet 2.4.1.2.2. The most suitable wavelet for each value of L parameter was found based on ECs reconstructed by ten most significant in absolute value coefficients (the same number of coefficients were used later on as features due to the fact that the identified suitable wavelet for different number of the DWT coefficients may be different). The quality of DWT approximations of ECs was estimated by Mean Squared Error (MSE) 5.1.2. The tested wavelet families were:

- Daubechies (db1-db20)
- Symlets (sym2-sym20)
- Coiflets (coif1-coif17)

Haar wavelet (or "db1" as they are identical) was identified as the most suitable wavelet for ECs of each value of L parameter. After that, ten most significant DWT coefficients computed for each EC using Haar wavelet and sorted in descending order by their values were used as features for subsequent classification.

The second approach consisted in using important time moments of ECs reconstructed by ten most significant in absolute value DWT coefficients. The five most important coefficients identified by Random Forest were used as features.

The results of the conducted experiments for each of these two approaches may be found for SVM in 6.0.2.1, for Random Forest in 6.0.1.1, for logistic regression in 6.0.3.1 and for k-NN in 6.0.4.1.

5.4.4 Standardization of features

The extracted features such as statistic features, DFT and DWT coefficients or combinations of these features with other features are measured by different scales and thus, these extracted features must be rescaled. Scikit-learn provides instruments [20] for both standardization and normalization 2.2.2.1 and

[20] `http://scikit-learn.org/stable/modules/preprocessing.html`

was used for standardization purposes. The extracted features were standardized to *zero mean and unit variance.*

5.5 Feature selection

The ability of Random Forest to estimate feature importance was used on different feature sets in order to reveal the most useful features. The settings of Random Forest algorithm [21] were the following:

- Number of trees: 300
- All features were considered when the algorithm was searching for the best split
- 10-fold cross validation for estimating average feature importance across different data partitions into folds
- Two split criterions were used: Gini impurity measure 2.3.7.1 and information gain 2.3.7.2. The union of the top five important features for each criterion was used as a final estimation in order to get a more complete set of important features (note that it results sometimes in the number of the selected important features greater than five).

Other not mentioned parameters had their default values.

Results of this approach in application to different feature sets will be described below.

5.5.1 Selection of statistic features

The number of all extracted statistic features is 16 and some of these features are definitely not useful. Feature importance estimated by Random Forest was used in order to identify the most important statistic features. Figure 5.8 below illustrates the estimated importance of statistic features computed for the original EC #10. Blue lines represent standard deviation of the appropriate feature.

5.5.2 Subsampling of the important time moments

The truncated ECs are quite long (approximately 100 values) and thus, several strategies of subsampling had been performed and tested by SVM (with parameter optimization) in order to identify important time moments of ECs. The following subsampling strategies were used:

- Every 10-th element of EC (including the first element)

[21] `http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

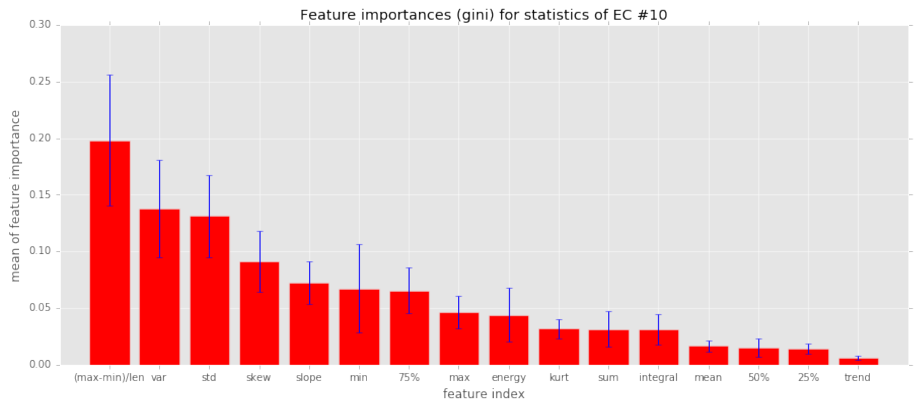


Figure 5.8: Computed feature importance of statistic features for EC #10. The best five in descending order: $(max - min)/length$, variance, standard deviation, skewness and slope.

- Every 15-th element of EC (including the first element)
- Random subsampling

No one of these subsampling strategies succeeded. Random subsampling strategy was performed several times with no success. However, subsampling each 15-th element of EC results in ARARS score of 80% in one case.

Then Random Forest was used for the whole time series (Entropy Curve of a rabbit) in order to find good time moments for classification (see Figure 5.9 below). This approach was successful with achieved ARARS score approximately equal to 90%.

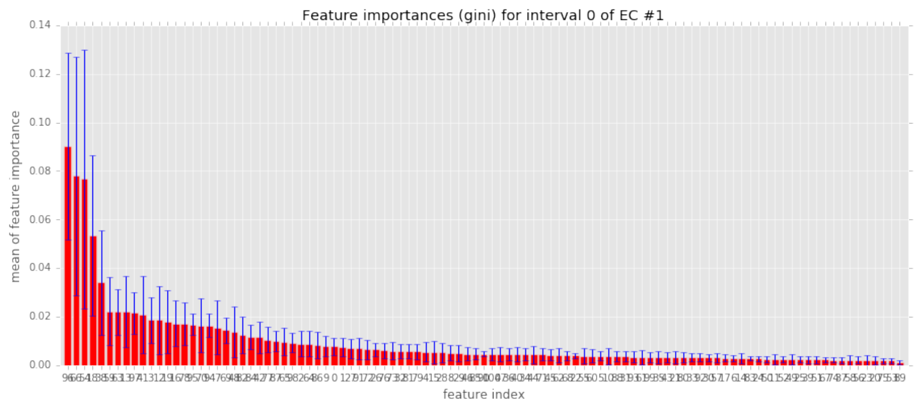


Figure 5.9: Computed feature importance of important time moments for EC #1 and control interval. The best six are 96, 97, 66, 38, 18, 54.

The importance of the identified time moments (indices 96, 97, 66, 38, 18, 54 in the case of Figure 5.9) may be easily seen and verified using Box-and-Whisker plot. Figure 5.10 shows an example of distribution of EC values in the important time moment identified by Random Forest for verification of importance of this time moment.

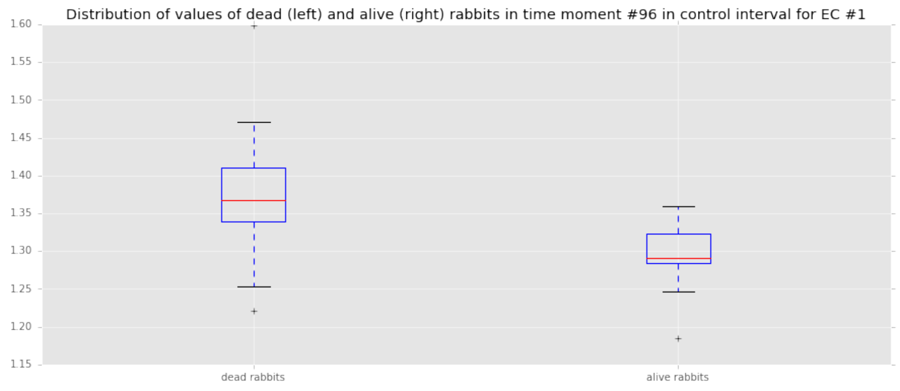


Figure 5.10: Important time moment #96: distribution of values for dead (left) and alive (right) rabbits.

5.5.3 Selection of the most discriminative values of L parameter

The final approach applies Random Forest algorithm to the set of multidimensional points (constructed from all ECs of the appropriate rabbit and repeating this step for each rabbit) for determining ECs, which are the most discriminative. Figure 5.11 shows an example of such multidimensional point construction.

5.6 Used features and their combinations

This subsection describes features and their combinations used as input for machine learning algorithms.

All created features may be divided into four groups:

- Statistic features
- Time moments of ECs
- Combinations of features
- DFT and DWT approaches

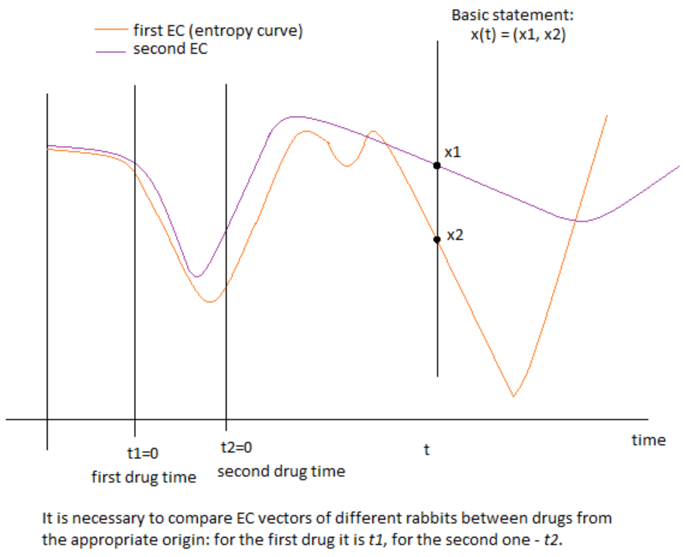

first EC (entropy curve)
second EC
Basic statement:
x(t) = (x1, x2)
x1
x2
t1=0
first drug time
t2=0
second drug time
t
time
It is necessary to compare EC vectors of different rabbits between drugs from the appropriate origin: for the first drug it is t1, for the second one - t2.

Figure 5.11: Example of a multidimensional point construction

Statistic features are common choice of features for almost any data. Mean, minimum, maximum, percentiles – these statistics are simple and describe any type of numerical data and thus, they were initially used as features. Unfortunately, the performance of machine learning algorithms on these simple features was poor with ARARS score less than 75% in almost all cases. Then more specific statistic features reflecting shape of ECs, distributions of EC values and time series specific information were introduced and added to the simple statistic features in order to improve results. This approach resulted in additional winners (values of L parameter of ECs) and in the slightly improved overall performance.

The next used approach consisted in reducing the length of the original ECs using different subsampling strategies 5.5.2 in order to reveal hidden patterns. Subsampling using Random Forest succeeded in location of the important time moments of ECs and machine learning algorithms, trained on these time moments, achieved high results even with ARARS score greater than 90% in rare cases.

Though the already achieved results were quite good, their combination could achieve better results. Thus, the different combinations of the important time moments and statistic features including concatenation of the drug intervals together were tested and the classification performance had achieved overall improvements (more cases with ARARS score in the range of 85-90

The last approach, namely usage of DFT and DWT algorithms, was realized due to the fact that the input data (i.e. ECs) are time series and may be interpreted as signals. DFT and DWT algorithms are widely used in signal processing due to their robustness and compression capabilities and thus, they may also be used in prediction of arrhythmia. DFT and DWT coefficients computed for each EC were used as features in the first step. Then the important time moments of ECs reconstructed by several DFT (and DWT) coefficients were identified by Random Forest and used further as features.

The list below contains used features and their combinations as well as short abbreviations:

- **imp_time**: top five important time moments of the truncated ECs revealed by Random Forest
- **imp_time_rollmean**: top five important time moments of the truncated ECs with rolling mean revealed by Random Forest
- **original**: all values of truncated ECs
- **original_rollmean**: all values of the truncated ECs with rolling mean
- **osimple_stats**: simple statistic features of the original ECs
- **oadvanced_stats**: simple + advanced statistic features of the original ECs
- **tsimple_stats**: simple statistic features of the truncated ECs
- **tadvanced_stats**: simple + advanced statistic features of the truncated ECs
- **oimp_stats**: top five important statistic features (simple + advanced) of the original ECs revealed by Random Forest
- **timp_stats**: top five important statistic features (simple + advanced) of the truncated ECs revealed by Random Forest
- **best_truncated**: top five important truncated ECs revealed by Random Forest
- **best_imp_time**: top five important time moments of top five important truncated ECs revealed by Random Forest
- **best_imp_ostats**: top five important statistic features (simple + advanced) of top five important truncated ECs revealed by Random Forest
- **concat**: concatenation of values of control and first intervals of the truncated ECs
- **concat_imp_time**: concatenation of the top five important time moments of control and first intervals of the truncated ECs

- **concat_imp_time_ostats**: concatenation of top five important time moments of control and first intervals of the truncated ECs with the subsequent concatenation with all statistic features (simple + advanced) computed for the original ECs (i.e. control_imp_time + first_imp_time + ostats)

- **concat_imp_time_inter_ostats**: concatenation of top five important time moments of each interval of the truncated ECs with all statistic features (simple + advanced) computed for the original ECs (i.e. control_imp_time + ostats, first_imp_time + ostats)

- **concat_imp_time_cstats**: concatenation of top five important time moments of control and first intervals of the truncated ECs with the subsequent concatenation with all statistic features (simple + advanced) computed for this concatenated important time moments (i.e. control_imp_time + first_imp_time + cstats)

- **concat_imp_time_inter_tstats**: concatenation of top five important time moments of each interval (control and first) of the truncated ECs with all statistic features computed for the appropriate interval (i.e. control_imp_time + tstats, first_imp_time + tstats)

- **concat_imp_time_inter_imp_tstats**: concatenation of top five important time moments of each interval (control and first) of the truncated ECs with important statistic features computed for the appropriate interval (i.e. control_imp_time + imp_tstats, first_imp_time + imp_tstats)

- **imp_dft_coeffs**: top five important DFT coefficients (concatenation of 15 real and 15 imaginary parts together and top five from them all) computed for the truncated ECs. Important time moments were identified by Random Forest.

- **imp_dft**: top five important time moments (identified by Random Forest) of ECs reconstructed by the first ten DFT coefficients

- **dwt_coeffs**: ten most significant in absolute value DWT coefficients for each EC. Coefficients were sorted in descending order.

- **imp_dwt**: top five important time moments (identified by Random Forest) of ECs reconstructed by ten most significant (in absolute value) DWT coefficients

- **imp_time_cstats**: all statistic features computed for the concatenated important time moments of ECs of each drug interval

Chapter 6

Experiment results

This section contains selected results of the conducted experiments for SVM, Random Forest, Logistic Regression, k-NN, MLP, GMDH and ensemble learning. All results may be found in Appendix **??**. At the end of the section, the best achieved results for each performance metric and the most effective values of L parameter of ECs are presented.

Due to the high amount of the obtained results and the fact that many of these results were very poor, unsatisfying results were not part of this work.

6.0.1 Random Forest

Two of the significant advantages of Random Forest algorithm, especially in case of small data sets, are its resistance to overfitting and handling the outliers 2.3.7.3.1. These advantages are very important in case of the input data of this work as there are only 37 different samples for each value of L parameter of ECs, which indicates very small data set.

Algorithm of Random Forest may be optimized in several ways such as choosing size of the selected subset of subsamples, maximum admissible depth of tree classifiers, amount of features for splitting etc. Scikit-learn implementation of Random Forest [22] supports many parameters, however, only some of them are important for algorithm optimization and must be set explicitly. These parameters are:

- **n_estimators**: the number of tree classifiers in Random Forest
- **max_features**: the number of features to consider when looking for the best split
- **criterion**: criterion for measure quality of a split
- **max_depth**: the maximum depth of the trees in forest

[22]`http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

- **min_samples_leaf**: the minimum number of samples in a leaf node
- **oob_score**: whether to use out-of-bag samples to estimate the generalization accuracy or not

Inclusion of **oob_score** may seem a weird decision at first, but it is actually not. Using **oob_score** in Random Forest 2.3.7.3.1 has the same goal as using cross validation, i.e. for estimating generalization accuracy on data set independent from the training data set. However, in this work, the generalization accuracy was estimated by ten different cross validations, which results in more reliable final results than **oob_score** may provide and thus, use of oob test samples is not required in this case.

The set of values for each other parameter must be tested in order to find and then construct effective Random Forest classifier.

The criterions for measuring quality of split, namely Gini index and information gain, are tree specific parameters and thus, they must be tested.

Optimization of **max_features** parameter is very important as the performance of Random Forest algorithm is very sensitive to this parameter which was mentioned in section (Random Forest). In case of the available input data, where amount of features is approximately 5 (in case of the important time moments) up to 16 (in case of all statistic features), some values of **max_features** parameter become very similar and may be omitted. For example, two of the available options of this parameter are $\sqrt{all\_features}$ and $\log_2(all\_features)$. However, these two values of parameter are very close to each other or even identical in case of 5-16 features as the computed maximum number of features is either $\sqrt{5} = 2$ and $\log_2 5 = 2$ or $\sqrt{16} = 4$ and $\log_2 16 = 4$. Thus, one of the abovementioned values may be omitted without losing any important information. Even in case they are slightly different, these two computed values used together do not result in effective search of the parameter space and thus, one of the values may be omitted.

Random Forest trains each of its decision tree classifiers on the various subsamples (with replacement) of the original input data set with size 37. The tested values of **max_depth** parameter must reflect the size of the input data set and the fact that the maximum possible depth of tree equals to $\log_2 37 = 6$ (note that scikit-learn implements Random Forest with binary split). The computed maximum tree depth parameter is too small and the only viable option of this parameter is tree depth equal to 3 or 4. Thus, the following two values of maximum depth were tested: 3 and *None* as the case where nodes are expanded until all leaves are pure in terms of classes or until all leaves contain less than two samples.

The last parameter to consider is **min_samples_leaf**. It is obvious that the tested values of this parameter must be less than 13 due to the fact that there are 13 non-arrhythmogenic rabbits and 24 arrhythmogenic rabbits. It is also important to mention that this parameter is connected to the maximum depth of trees in Random Forest. Even more, the tree depth may be controlled in some way by this parameter as the construction of each tree is stopped when there are no more nodes with number of samples greater than value of **min_samples_leaf**. Thus, due to the small size of the input data and the fact that testing of maximum tree depth parameter will be performed, it is not necessary to test **min_samples_leaf** parameter.

The table 6.1 below contains summary of the tested values of Random Forest parameters.

Parameter	**Tested values**
n_estimators	10, 25, 40, 80, 130
max_features	all features, $\log_2$, 50%, 70%
criterion	Gini index, information gain
max_depth	3, None

Table 6.1: Optimized parameters of Random Forest algorithm

6.0.1.1 Achieved results

Table 6.2 contains achieved results.

Feature combination	**EC**	**recall**	**specificity**	**ROC AUC**	**accuracy**	**Interval**
imp_time	400	0.89	0.76	0.82	0.84	control
	100	0.88	0.77	0.83	0.84	first
concat_imp_time	100	0.97	0.67	0.82	0.86	-
	500	0.9	0.73	0.81	0.84	-
concat_imp_time_cstats	500	0.96	0.74	0.84	0.88	-
concat_imp_time_inter_tstats	100	0.9	0.71	0.8	0.83	first

Table 6.2: Achieved results for Random Forest

6.0.2 SVM

Scikit-learn implementation of SVM [23] was used for the experiments and the following parameters were optimized:

- C regularization 2.2.6 parameter

[23] `http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`

- *gamma*: kernel coefficient
- kernel: type of the kernel function for using in the algorithm

Different kernel functions must be tested in order to find the most adequate one for the given data set and problem domain. The used implementation supports several types of kernel functions such as Radial Basis Function (RBF), sigmoid, polynomial and linear kernels. The *gamma* parameter relates to RBF, polynomial and sigmoid kernels and defines an influence area of a single training sample selected as support vector. Let consider RBF kernel and *gamma*: If *gamma* is high, the influence area is small up to the selected support vectors themselves and it leads to overfitting. On the other side, with low values of *gamma*, the selected support vectors have large influence areas containing all training samples, which again leads to wrong classification. In other words, SVM cannot capture the shape of data distribution in both abovementioned cases, which leads to wrong results.

The table 6.3 below contains summary of the tested values of SVM parameters.

Parameter	**Tested values**
C	$2^{-10}, 2^{-9}, \ldots, 1, \ldots, 2^{9}, 2^{10}$
gamma	$2^{-10}, 2^{-9}, \ldots, 1, \ldots, 2^{9}, 2^{10}$ and $1/n\_features$ [24]
kernel	rbf, sigmoid, linear

Table 6.3: Optimized parameters of SVM algorithm

6.0.2.1 Achieved results

Table 6.4 contains achieved results.

6.0.3 Logistic Regression

Scikit-learn implements regularized logistic regression [25]. The regularization 2.2.6 may be performed with one of the two methods: L1 [84] (least absolute shrinkage and selection operator or LASSO) and L2 (also known as Tikhonov regularization [26]. The comparison of L1 and L2 regularization methods with focusing on logistic regression in case of many irrelevant features was performed in 2004 by Andrew Y. Ng [85] and showed that logistic regression with L1 regularization is effective even in case of the number of irrelevant features exponentially higher than the amount of the training data. It is important

[25]`http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

[26]`https://en.wikipedia.org/wiki/Tikhonov_regularization`

Feature combination	**EC**	**recall**	**specificity**	**ROC AUC**	**accuracy**	**Interval**
imp_time	30	0.82	0.85	0.84	0.83	control
	300	0.91	0.93	0.92	0.92	
	300	0.79	0.90	0.85	0.83	first
	500	0.86	0.99	0.92	0.90	
	10	0.88	0.79	0.84	0.86	
imp_time_rollmean	60	0.86	0.82	0.84	0.84	first
oadvanced_stats	10	0.78	0.92	0.85	0.83	-
oimp_stats	20	0.71	0.90	0.80	0.77	-
concat_imp_time	10	0.96	0.75	0.85	0.89	-
	300	0.85	0.95	0.90	0.89	
	500	0.92	0.88	0.90	0.90	
concat_imp_time_ostats	10	0.91	0.85	0.88	0.89	-
	5	0.76	0.94	0.85	0.82	
	500	0.84	0.84	0.84	0.84	
	90	0.89	0.84	0.86	0.87	
concat_imp_time_inter_ostats	10	0.795000	0.995	0.895000	0.864500	control
	10	0.86	0.85	0.85	0.86	first
	20	0.83	0.87	0.85	0.85	
	400	0.89	0.86	0.87	0.87	
	500	0.83	0.91	0.87	0.85	

Table 6.4: Achieved results for SVM

as the input data set for this work is small (37 different ECGs in total) and though the number of the used features does not exceed the size of this data set, the number of features can achieve approximately half of the data set size in case of feature combinations.

Let w is a vector of regression coefficients, n is a number of input samples, X is a matrix of input samples, y is a vector of true outputs, c is an error term or noise and C is the inverse of regularization strength (the smaller is C the strongest is regularization), then L1 regularized logistic regression minimizes the following cost function [27]

$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Figure 6.1: Logistic regression with L1 (top) and L2 (bottom)

Scikit-learn implementation of logistic regression also supports different

[27] `http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression`

so-called *solvers*: approaches which are used to optimize objective function during the training. The available solvers are:

- “liblinear” (coordinate descent algorithm)
- “newton-cg” (newton conjugate descent)
- “lbfgs” (limited memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm)
- “sag” (Stochastic Average Gradient descent).

Thus the parameters for optimization are solver type, C (the inverse of regularization strength) and regularization method (L1 or L2) (link to section Regularization). The table 6.5 below summarizes the tested values of these parameters.

Parameter	**Tested values**
C	ten evenly spaced samples in the interval $[0.1, 1]$ and 50 evenly spaced samples in the interval $[2, 5000]$
solver	liblinear, newton-cg, lbfgs, sag
regularization method	L1, L2

Table 6.5: Optimized parameters of Logistic regression algorithm

6.0.3.1 Achieved results

Table 6.6 contains the achieved results.

Feature combination	**EC**	**recall**	**specificity**	**ROC AUC**	**accuracy**	**Interval**
imp_time	500	0.89	0.82	0.85	0.86	first
oadvanced_stats	10	0.74	0.97	0.85	0.82	-
concat_imp_time	300	0.89	0.81	0.85	0.86	-
concat_imp_time_ostats	10	0.91	0.87	0.89	0.90	-
	400	0.86	0.84	0.85	0.86	
	50	0.96	0.74	0.85	0.88	
	90	0.95	0.78	0.86	0.89	
concat_imp_time_inter_ostats	50	0.99	0.84	0.91	0.94	first
concat_imp_time_cstats	10	0.90	0.94	0.92	0.92	-
	300	0.91	0.81	0.86	0.88	
	500	0.96	0.75	0.85	0.89	
concat_imp_time_inter_tstats	500	0.91	0.80	0.85	0.87	first

Table 6.6: Achieved results for Random Forest

6.0.4 k-NN

k-nearest neighbors algorithm is quite simple and must be optimized primarily in number of the nearest neighbors. Scikit-learn implementation of k-NN [28] supports some additional parameters. The following parameters were optimized:

- **n_neighbors**: amount of neighbors to use
- **weights**: how the contribution of the neighbors is weighted (e.g. by inverse of their distance)
- **p**: power parameter for the Minkowski distance

The common used types of **weights** parameter are uniform (all neighbors are weighted equally) and distance (neighbors are weighted by the inverse of their distance, i.e. the closest neighbors contribute to the classification the most).

In order to clarify the meaning of **p** parameter, it is necessary to define Minkowski distance. Minkowski distance of order p between two points $X = (x_1, x_2, .., x_n)$ and $Y = (y_1, y_2, \dots, y_n) \in R^n$ is expressed as [29]:

$$dist(X, Y, p) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}$$

where $|\dots|$ means absolute value.

Thus, Minkowski distance of order 1 is Manhattan distance and of order 2 is Euclidean distance. Amount of the considered nearest neighbors cannot be obviously greater than the amount of available samples of negative class (i.e. greater than 13) and thus, the values of 1-10 were tested.

The table 6.7 below contains the tested values of parameters.

The optimized parameters of k-NN with DTW are slightly different from the standard k-NN. First, Minkowski distance was replaced with DTW distance. Second, computation of DTW distances between time series is computationally expensive and thus, the amount of the nearest neighbors for testing was reduced to only three values. The tested values of weights parameter are the same as in case of standard k-NN algorithm. k-NN with DTW approach was applied to the reduced number of the feature sets as it was very computationally demanding (approximately 30 hours for each feature set) and due the fact that k-NN with DTW indicated worse results then standard k-NN.

The table 6.8 summarizes information about tested parameters in case of k-NN with DTW.

[28] `http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`

[29] `https://en.wikipedia.org/wiki/Minkowski_distance`

Parameter	**Tested values**
n_neighbors	1-10
weights	uniform, distance
p	1,2,3

Table 6.7: Optimized parameters of k-NN algorithm

Parameter	**Tested values**
n_neighbors	1, 3, 5
weights	uniform, distance

Table 6.8: Optimized parameters of k-NN with DTW

6.0.4.1 Achieved results

6.0.4.1.1 k-NN

Table 6.9 contains achieved results.

6.0.4.1.2 k-NN with DTW

Table 6.10 contains achieved results.

6.0.5 Neural Networks

Neural networks require a large enough training data set in order to reveal hidden patterns and relationships. 37 different ECGs were available and this likely indicates a very small data set for neural networks. This assumption was confirmed by training MLP and GMDH on the input data (it resulted in prediction of one class either positive or negative in all cases). In order to deal with this problem, ECs of different values of L parameter were grouped together for each drug interval which constructed new data sets of size 592. Then the neural networks were trained on these new data sets and provided meaningful results. However, the amount of tested feature sets was reduced

Feature combination	EC	recall	specificity	ROC AUC	accuracy	Interval
imp_time	500	1.00	0.69	0.84	0.89	control
	20	0.71	0.99	0.85	0.81	first
	500	0.90	0.92	0.91	0.91	
	90	0.93	0.70	0.81	0.84	
imp_time_rollmean	60	0.79	0.91	0.85	0.83	control
oimp_stats	50	0.72	1.00	0.86	0.81	-
	10	0.62	0.96	0.79	0.74	
	5	0.69	0.90	0.79	0.76	
concat_imp_time	10	0.79	0.91	0.85	0.83	-
	100	0.78	0.99	0.88	0.85	
	500	0.94	0.77	0.85	0.88	
	300	0.96	0.70	0.83	0.87	
concat_imp_time_cstats	10	0.90	0.84	0.87	0.88	-
	500	0.81	0.94	0.88	0.85	
	400	0.76	0.94	0.85	0.82	
concat_imp_time_inter_imp_tstats	500	0.87	0.86	0.86	0.86	first
imp_dft_coeffs	40	0.84	0.83	0.84	0.83	control
imp_dwt	90	0.93	0.91	0.92	0.92	first

Table 6.9: Achieved results for k-NN

Feature combination	EC	recall	specificity	ROC AUC	accuracy	Interval
imp_time	500	0.95	0.75	0.85	0.88	first

Table 6.10: Achieved results for k-NN

due to the fact that concatenation of ECs of different L parameters in case of important features identified by Random Forest produces data set of samples with different and inconsistent features (e.g. ECs with $L = 30$ may have important statistic features different from important statistic features of ECs with $L = 5$ and thus, these groups of ECs cannot be considered together). Though the important features could be estimated for the new data sets with grouped ECs of different value of L parameter, the goal of this work is to produce meaningful and well-interpreted results, including identifying the most perspective values of L parameter, which is impossible in the abovementioned case. Thus, the neural networks were used rather for illustration purposes than for the serious research.

6.0.5.1 GMDH

The important advantage of GMDH neural network is its self-organization. The used application GMDH Shell [30] performs effective construction of GMDH NN and the only parameters to set by user are:

[30] `https://www.gmdhshell.com/time-series-forecasting`

- Complexity of neuron function in terms of polynomial order and other related settings
- Maximum number of layers
- The initial layer width, which defines how many neurons are added to the set of inputs at each new layer
- Loss function

The custom polynomial was used as the neuron function with the following properties:

- Maximum power of variable: 2
- Minimum power of variable: 0
- Maximum total power (as sum of powers) in a polynomial term: 4
- Maximum number of variables in a polynomial term: 3

Due to the fact that GMDH Shell cannot run several cross validations, leave-one-out cross validation was used. The following table 6.11 summarizes information about the tested values of parameters.

Parameter	**Tested values**
maximum number of layers	33
initial layer width	50, 200, 400
loss function	Root Mean Squared Error (RMSE)

Table 6.11: Optimized parameters of GMDH

6.0.5.1.1 Achieved results The table 6.12 contains selected achieved results.

Feature combination	**recall**	**specificity**	**ROC AUC**	**accuracy**
oadvanced_stats	0.8	0.74	0.88	0.78

Table 6.12: Achieved results for GMDH

6.0.6 Ensemble learning

6.0.6.1 Combining different algorithms together

ECs of particular L parameters with the best results were selected for each used algorithm. Best results in this section mean results with ARARS score greater than or equal to 75%. Then the combination of SVM, Random Forest, k-NN and Logistic regression algorithms in the following two ways was used:

- Algorithms were combined together by L parameter of ECs
- Combination of algorithms across all drug intervals with ECs of any L parameter, i.e. combination of all available good enough values of L parameter in any drug interval together

In both cases, majority voting was used as final prediction.

6.0.6.1.1 Combination of different algorithms for each value of L parameter This approach combined together predictions of machine learning algorithms for each value of L parameter and for each drug interval (if it existed) with subsequent majority voting. Table 6.13 contains the selected results.

Feature combination	EC	recall	specificity	ROC AUC	accuracy	Interval
imp_time	500	0.870000	0.985	0.927500	0.909667	first
oadvanced_stats	10	0.730000	0.985	0.857500	0.818333	-
oimp_stats	50	0.788333	0.950	0.869167	0.843000	-
concat_imp_time	300	0.898333	0.895	0.896667	0.896000	-
	500	0.943333	0.820	0.881667	0.900000	
concat_imp_time_ostats	10	0.876667	0.905	0.890833	0.887833	-
	90	0.921667	0.860	0.890833	0.900667	
concat_imp_time_cstats	10	0.908333	0.995	0.951667	0.938500	-
	300	0.860000	0.895	0.877500	0.871833	
	500	0.946667	0.850	0.898333	0.913167	

Table 6.13: Achieved results for combination of different algorithms by value of L parameter

6.0.6.1.2 Combination of classifiers for all values of L parameter at once This ensemble approach used combination of machine learning algorithms and all available information (i.e. ECs from different drug intervals and of different values of L parameter) in order to make a final prediction by majority voting. Moreover, each algorithm used only those values of L parameter, which achieved ARARS score of at least 75% for the appropriate algorithm.

The table 6.14 below illustrates achieved results. The values of L parameter, which were used for prediction, may be found in another table 6.15.

Feature combination	**Used algorithms**	**recall**	**specificity**	**ROC AUC**	**accuracy**
imp_time	SVM	1.0	1.0	1.0	1.0
	RF	0.99	0.88	0.93	0.95
	k-NN	0.99	1.0	0.99	0.99
	LR	0.96	0.87	0.91	0.93
oadvanced_stats	SVM	0.83	0.86	0.84	0.84
oimp_stats	SVM	0.83	0.97	0.9	0.88
	SVM, RF, k-NN, LR	0.8	0.99	0.9	0.87
timp_stats	SVM	0.93	0.93	0.93	0.93
concat_imp_time_inter_imp_tstats	SVM, RF, k-NN, LR	1.0	1.0	1.0	1.0
imp_dft_coeffs	SVM	0.99	0.99	0.99	0.99
dwt_coeffs	SVM	0.95	0.85	0.9	0.9

Table 6.14: Achieved results with combination of different algorithms and using different values of L parameter at once

Feature combination	**Used algorithms**	**Used ECs**
imp_time	SVM	Control: 1, 100, 30, 300, 40, 500, 60 First: 10, 100, 20, 200, 30, 300, 40, 5, 50, 500, 60, 70, 90
	RF	Control: 300, 400, 500, 90 First: 100, 5, 500
	k-NN	Control: 100, 30, 500, 60 First: 10, 100, 20, 300, 40, 400, 5, 50, 500, 90
	LR	Control: 100 First: 10, 200, 300, 5, 500, 90
oadvanced_stats	SVM	10, 40, 50
oimp_stats	SVM	10, 20, 300, 40, 5, 50, 500, 90
	SVM, RF, k-NN, LR	RF: 40 LR: 10, 40 k-NN: 10, 300, 5, 50, 90 SVM: 10, 20, 300, 40, 5, 50, 500, 90
timp_stats	SVM	Control: 20, 300 First: 40, 60
imp_dft_coeffs	SVM	Control: 40, 400, 5, 90 First: 1, 20, 200, 30, 80
dwt_coeffs	SVM	First: 10, 40, 60

Table 6.15: The used values of L parameter for combination of different algorithms with using different values of L parameter at once

6.0.7 Best results

This section contains ten best achieved scores 6.16 for each of the used performance metrics (recall, specificity, ROC AUC and accuracy) in the cases with ARARS score greater than 80% and regardless of the used algorithm or approach.

6.0.8 Best values of L parameter

This section contains summary (see table 6.17) of the most useful values of L parameter with ARARS score greater than 80%. In case of values of L parameter with ARARS score greater than 90%, their highest ARARS scores are explicitly stated. As it can be seen from the table, the most useful values of L parameter are 10, 90 and 500.

Position	**Recall**	**Specificity**	**ROC AUC**	**Accuracy**
1	1.0	1.0	1.0	1.0
2	1.0	1.0	0.99	1.0
3	1.0	1.0	0.99	0.99
4	1.0	1.0	0.95	0.99
5	1.0	0.99	0.93	0.95
6	0.99	0.99	0.93	0.94
7	0.99	0.99	0.93	0.94
8	0.99	0.99	0.93	0.93
9	0.99	0.99	0.92	0.93
10	0.98	0.99	0.92	0.93

Table 6.16: Top 10 achieved results regardless of the used approach

As it was mentioned in section 5.5.3, the most discriminative values of L parameter were revealed by constructing a multidimensional set of points with particular values of L parameter as dimensions. The following values of L parameter were identified as important for each drug interval:

- Control interval: 1, 10, 20, 40, 70
- The first interval (after the first DII): 1, 5, 20, 40, 80

Taking into account the table 6.17 below with the real observations, it is obvious that the used approach showed neither outstanding nor poor performance. Most of the identified values of L (5, 10, 20, 40 and 80) were confirmed by real observations (though only value of 10 belongs to the best observed values). However, the remaining two identified values (1 and 70) were completely wrong.

	L															
	1	**5**	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**200**	**300**	**400**	**500**
RF												x			x	x
SVM		x	x	x	x	x		x		x	0.92		x	0.92	x	0.93
k-NN		x	x	x		x	x	x			0.92	x		x	x	0.91
LR			0.92				0.92				x			x	x	x
Ensemble			0.95				x			x	x	0.92		x	x	0.93

Table 6.17: The most useful values of L parameter with ARARS score of at least 80%

6.0.9 Outlier detection

There are two different approaches in distinguishing abnormal observations from the regular ones depending on the training data set. Novelty detection is applied where the training data set does not contain any outliers. Thus, the goal of the novelty detection is detection of abnormal observations in the new observations. Outlier detection is applied in case there are abnormal observations in the training data set and it is necessary to take into account the regular observations ignoring anomalies [86].

Due to the small input data set (37 different ECGs in total), outlier detection and novelty detection are useless and, moreover, may lead to serious misunderstanding of the problem domain as the identified outliers (in case of outlier detection) and assumptions about the true distribution of regular observations (in case of novelty detection) may not be correct. Thus, these approaches must be used with caution.

An example of using outlier detection in order to improve classification performance will be provided further. An algorithm of Isolation Forest or iForest proposed by Fei Tony Liu and Kai Ming Ting [87] was used for outlier detection. The basic approach of anomaly detection consists in identifying profile of regular observations and marking observations, which do not conform to this profile as anomalies. The disadvantage of this approach is concentration on the regular observations rather than abnormal observations. iForest works differently: it constructs ensemble of trees (iTrees in context of iForest) for the data and then the observations with short average path lengths in the iTrees are viewed as anomalies. In other words, the anomalies are those observations, which can be isolated as soon as possible. The Figure 6.2 below shows an example of outlier detection by iForest for non-arrhythmogenic rabbit and EC #30 in the control interval.

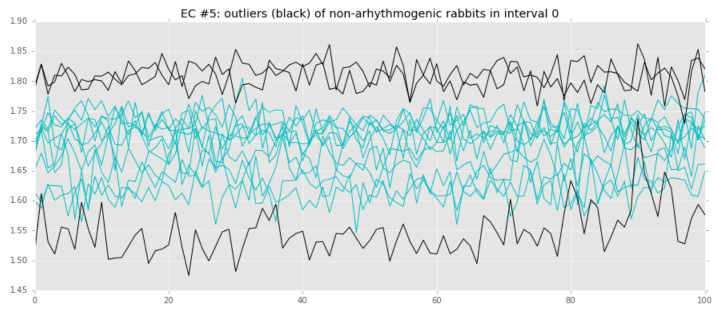


Figure 6.2: Outlier detection on the non-arrhythmogenic rabbits.

6.0.9.1 SVM

6.0.9.2 Random Forest

6.0.9.3 Logistic regression

6.0.9.4 k-NN

Chapter **7**

Real time prediction on the one minute subintervals

This section provides experiment results in case of real time prediction on the increasing number of continuous one minute subintervals.

First, the control and the first drug intervals were divided into one minute subintervals. Then, the prediction was initiated for each drug interval on the first one minute subinterval, later on the concatenated first and second subintervals etc. In other words, prediction was performed on the increasing number of continuous subintervals, i.e. the prediction was real time. SVM with subinterval statistic features (described in section 5.4.1) was used for the prediction purposes.

The final score of the real time prediction was computed for recall, specificity, ROC AUC and accuracy by averaging ten different 10-fold cross validation results. The following two Figures 7.1 illustrate the progress of maximum value for each abovementioned metric for the control and the first drug intervals. The highest scores were achieved on the first five minutes in the control interval and on the first three minutes after the first DII.

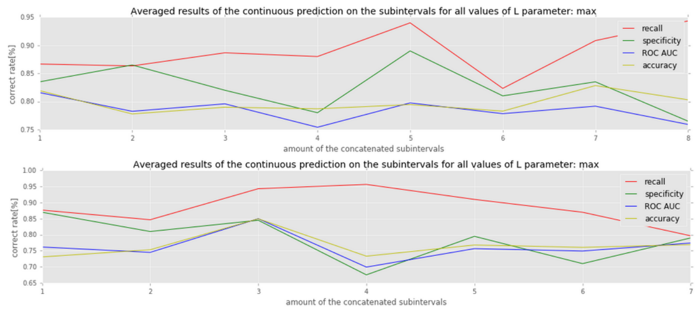


Figure 7.1: Progress of results in case of real time prediction on the one minute subintervals.

Discussion

This work has achieved good results (even with approximately 99% ARARS score in case of ensemble learning with SVM) in the prediction of arrhythmia on the animal model with rabbits. However, it is necessary to take into account the small size of the analyzed input data: there were only 37 different ECGs. The main problem in this case could be overfitting of machine learning algorithms. However, this weakness was rectified by:

- penalizing the contribution of more frequent class samples
- using reliable performance metrics 5.1.1 (F1, recall, specificity, ROC AUC)
- using ten different cross validation runs with different data partitioning into the cross validation folds with subsequent averaging the computed results.

Despite of the conducted improvements, the computed results must be considered with caution as such small data set may not provide enough generalization of the considered problem domain. For example, important time moments 5.5.2 revealed by Random Forest and verified by Box-and-Whisker plot may not exist in case of increased size of the input data under consideration. Figure 7.2 below provides a clear illustration of this case.

However, the high classification performance was also achieved with more reliable features such as statistic features 5.4.1 and coefficients of DFT 5.4.2 and DWT 5.4.3 transformations.

Outlier detection has the same problem as was mentioned in the text above, i.e. the small input data set. Though the classification performance for the data set without outliers is generally higher than the one with outliers, the achieved results primarily illustrate the possible case of outlier detection and potential improvements. The results must be considered with caution as due to the small input data set, the currently detected outliers may not be outliers identified in the larger data set. Moreover, it is unsafe to delete

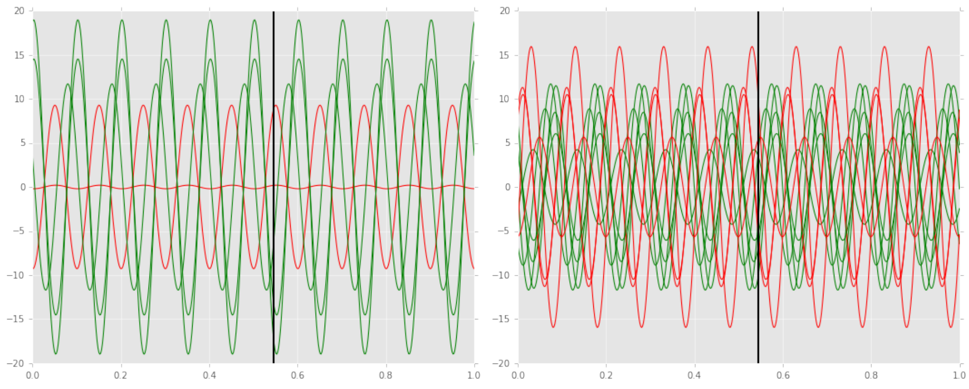


Figure 7.2: Example of the wrong time moment. (Left) The time moment exists (Right) There are more samples to disposition and the time moment is not now valid.

outliers without understanding their properties and causes of why they behave like outliers.

Conclusion

In this work, different machine learning algorithms and approaches such as Random Forest, SVM, logistic regression, k-NN, GMDH and ensemble learning were tested in order to predict arrhythmias. Different features were extracted from the provided input data such as classical statistic features and time series specific features. Discrete Fourier Transform and Discrete Wavelet Transform were used for compressing input time series with subsequent usage of the computed coefficients in prediction. Subsampling of time series data was performed in order to identify the most discriminative time moments. Finally, different features were combined together in different ways and their classification performance was evaluated.

This work presents good results in prediction of arrhythmias on the animal model up to one hour before life-threatening situation occurs. The maximum achieved scores in recall and specificity were as high as 99% in the case of ensemble learning (link to selected results). SVM 6.0.2.1, logistic regression 6.0.3.1 and k-NN 6.0.4.1 algorithms competed against one another in the highest achieved results with ARARS score around 90%. Taking into account the achieved results, the most useful values of L parameter were identified 6.0.8.

Outlier detection approach 6.0.9 was also tested and classification performance achieved overall improvements. However, deletion of outliers in case of the small input data set is unsafe and the achieved results must be viewed with caution (link to Discussion).

Finally, the real time prediction on the one minute subintervals was performed 7. It was revealed that the highest scores were achieved on the first five minutes in the control interval and the first three minutes in the interval after the first DII.

The conducted experiments completely fulfil the goals of this work in prediction of arrhythmias, identification of the useful values of L parameter and the early markers before occurrence of the life-threatening situation. The results demonstrated in this work illustrate the potential of arrhythmia prediction with the given type of input data.

Bibliography

[1] Nielsen, H. B. Non-stationary time series and unit root variance. january 2017, econometrics 2, Lecture Note 5.

[2] Regularization. 2017. Available from: `https://en.wikipedia.org/wiki/Regularization_(mathematics)`

[3] Wikipedia The Free Encyclopedia. 2017. Available from: `https://www.wikipedia.org/`

[4] Principles of training multi-layer neural network using backpropagation. 2017. Available from: `http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html`

[5] Artificial Neural Networks in NetLogo. 2017. Available from: `http://www.cs.us.es/~fsancho/?e=135`

[6] Khaled Assaleh, Y. A. K., Tamer Shanableh. Group Method of Data Handling for Modeling Magnetorheological Dampers.

[7] Haderlein, T. Dynamic Time Warping (DTW). 2013, lecture, Západočeská univerzita v Plzni.

[8] Xiaopeng Xi, C. S. L. W., Eamonn Keogh. Fast Time Series Classification Using Numerosity Reduction.

[9] Carmelo Cassisi, M. A. A. C., Placido Montalto; Pulvirenti, A. Similarity Measures and Dimensionality Reduction Techniques for Time Series Data Mining.

[10] Grandell, J. Time series analysis. 2017, lecture notes.

[11] WALKER, S. H.; DUNCAN, D. B. Estimation of the probability of an event as a function of several independent variables. *Biometrika*, volume 54, no. 1-2, 1967: p. 167, doi:10.1093/

biomet/54.1-2.167, /oup/backfile/content_public/journal/biomet/54/1-2/10.1093/biomet/54.1-2.167/2/54-1-2-167.pdf.

[12] Freedman, D. A. *Statistical Models: Theory and Practice.* Cambridge University Press, 2009.

[13] Rojas, R. *Neural Networks - A Systematic Introduction.* Springer, 1996.

[14] Šlapák, I. M. Metody výpočetní inteligence. 2011, lecture, FIT CTU in Prague.

[15] Kordík, P. Gradient Learning of Feedforward Networks. 2009, lecture, FIT CTU in Prague.

[16] Shashi Sathyanarayana, P. A Gentle Introduction to Backpropagation.

[17] Wilson, D.; Martinez, T. R. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, volume 16, no. 10, 2003: pp. 1429 – 1451, ISSN 0893-6080.

[18] Mort, L. A. . N. The development of self-organization techniques in modeling: a review of the Group Method of Data Handling (GMDH). Technical report, The University of Sheffield, 2001.

[19] Josef Taušer, P. B. Exchange Rate Predictions in International Financial Management by Enhanced GMDH Algorithm.

[20] Schetinin, V. Polynomial Neural Networks Learnt to Classify EEG Signals. Technical report cs.NE/0504058, Apr 2005. Available from: http://cds.cern.ch/record/832743

[21] H. Sakoe, S. C. Dynamic programming algorithm optimization for spoken word recognition.

[22] Müller, M. *Information Retrieval for Music and Motion.* Springer-Verlag Berlin Heidelberg, first edition, 2007.

[23] Keogh, E. J.; Pazzani, M. J. Scaling Up Dynamic Time Warping for Datamining Applications. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, New York, NY, USA: ACM, 2000, ISBN 1-58113-233-6, pp. 285–289, doi:10.1145/347090.347153.

[24] Salvador, S.; Chan, P. Toward Accurate Dynamic Time Warping in Linear Time and Space. *Intell. Data Anal.*, volume 11, no. 5, Oct. 2007: pp. 561–580, ISSN 1088-467X.

[25] Chotirat Ann Ratanamahatana, E. K. Three Myths about Dynamic Time Warping Data Mining. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, ACM, 2005, ISBN 978-0-89871-593-4.

[26] Abidin, T.; Perrizo, W. SMART-TV: A Fast and Scalable Nearest Neighbor Based Classifier for Data Mining. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, New York, NY, USA: ACM, 2006, ISBN 1-59593-108-2, pp. 536–540, doi:10.1145/1141277.1141403.

[27] Carlotta Domeniconi, B. Y. On Error Correlation and Accuracy of Nearest Neighbor Ensemble Classifiers. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, ACM, 2005, ISBN 978-0-89871-593-4.

[28] Voulgaris, Z.; Magoulas, G. D. Extensions of the K Nearest Neighbour Methods for Classification Problems. In *Proceedings of the 26th IASTED International Conference on Artificial Intelligence and Applications*, AIA '08, Anaheim, CA, USA: ACTA Press, 2008, ISBN 978-0-88986-710-9, pp. 23–28.

[29] Yang, K.; Shahabi, C. An Efficient K Nearest Neighbor Search for Multivariate Time Series. *Inf. Comput.*, volume 205, no. 1, Jan. 2007: pp. 65–98, ISSN 0890-5401, doi:10.1016/j.ic.2006.08.004.

[30] Cortes, C.; Vapnik, V. Support-Vector Networks. *Mach. Learn.*, volume 20, no. 3, Sept. 1995: pp. 273–297, ISSN 0885-6125, doi: 10.1023/A:1022627411411.

[31] Mercer, J. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society*, 1909.

[32] Cawley, G. C.; Talbot, N. L. C. Preventing Over-Fitting During Model Selection via Bayesian Regularisation of the Hyper-Parameters. *J. Mach. Learn. Res.*, volume 8, Dec. 2007: pp. 841–861, ISSN 1532-4435.

[33] DEVASHREE JOSHI, R. G. PERFORMANCE ANALYSIS OF FEATURE EXTRACTION SCHEMES FOR ECG SIGNAL CLASSIFICATION. *International Journal of Electrical, Electronics and Data Communication*, 2013.

[34] Asl, B. M.; Setarehdan, S. K.; et al. Support Vector Machine-based Arrhythmia Classification Using Reduced Features of Heart Rate Variability Signal. *Artif. Intell. Med.*, volume 44, no. 1, Sept. 2008: pp. 51–64, ISSN 0933-3657, doi:10.1016/j.artmed.2008.04.007.

[35] Kampouraki, A.; Manis, G.; et al. Heartbeat Time Series Classification with Support Vector Machines. *Trans. Info. Tech. Biomed.*, volume 13, no. 4, July 2009: pp. 512–518, ISSN 1089-7771, doi:10.1109/TITB.2008.2003323.

[36] K.S. Park, D. L. S. S. J. L. Y. C. I. K. S. K., B.H. Cho. Hierarchical support vector machine based heartbeat classification using higher order statistics and hermite basis function. *Computers in Cardiology*, 2008.

[37]

[38] Guler, I.; Ubeyli, E. D. Multiclass Support Vector Machines for EEG-Signals Classification. *Trans. Info. Tech. Biomed.*, volume 11, no. 2, Mar. 2007: pp. 117–126, ISSN 1089-7771, doi:10.1109/TITB.2006.879600.

[39] Chapelle, O. Training a Support Vector Machine in the Primal. *Neural Comput.*, volume 19, no. 5, May 2007: pp. 1155–1178, ISSN 0899-7667, doi:10.1162/neco.2007.19.5.1155.

[40] Dietterich, T. G. Ensemble Methods in Machine Learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, London, UK, UK: Springer-Verlag, 2000, ISBN 3-540-67704-6, pp. 1–15.

[41] Blum, A.; Rivest, R. L. Training a 3-node Neural Network is NP-complete. In *Proceedings of the First Annual Workshop on Computational Learning Theory*, COLT '88, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988, pp. 9–18.

[42] Hyafil, L.; Rivest, R. L. Constructing Optimal Binary Decision Trees is NP-Complete. *Information Processing Letters*, volume 5, 1976: pp. 15–17.

[43] Cherkauer, K. Human Expert-Level Performance on a Scientific Image Analysis Task by a System Using Combined Artificial Neural Networks. In *Proceedings of AAAI-96 Workshop on Integrating Multiple Learned Model for Improving and Scaling Machine Learning Algorithms*, 1996, pp. 15–21.

[44] Wolpert, D. H. Stacked Generalization. *Neural Netw.*, volume 5, no. 2, Feb. 1992: pp. 241–259, ISSN 0893-6080, doi:10.1016/S0893-6080(05)80023-1.

[45] Ting, K. M.; Witten, I. H. Stacked Generalization: When Does It Work? In *Proceedings of the Fifteenth International Joint Conference on Artifical Intelligence - Volume 2*, IJCAI'97, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, ISBN 1-555860-480-4, pp. 866–871.

[46] Parmanto, B.; Munro, P. W.; et al. Improving Committee Diagnosis with Resampling Techniques. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, Cambridge, MA, USA: MIT Press, 1995, pp. 882–888.

[47] Damianos Karakos, J. E., Sanjeev Khudanpur. UNSUPERVISED CLASSIFICATION VIA DECISION TREES: AN INFORMATION-THEORETIC PERSPECTIVE. 2005, center for Language and Speech Processing, Johns Hopkins University.

[48] Leo Breiman, C. J. S. R. O., Jerome Friedman. *Classification and regression trees*. 1984.

[49] Quinlan, J. R. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, ISBN 1-55860-238-0.

[50] An Exploratory Technique for Investigating Large Quantities of Categorical Data. *Applied Statistics*, 1980: pp. 119–127.

[51] Classification trees with unbiased multiway splits. *Journal of the American Statistical Association*, 2001: pp. 598—-604.

[52] Classification Trees with Bivariate Linear Discriminant Node Models. *Journal of Computational and Graphical Statistics*, 2003: pp. 512—530.

[53] IMPROVING THE PRECISION OF CLASSIFICATION TREES. *The Annals of Applied Statistics*, 2009. Available from: `https://arxiv.org/pdf/1011.0608.pdf`

[54] Loh, W.-Y.; Shih, Y.-S. SPLIT SELECTION METHODS FOR CLASSIFICATION TREES. *Statistica Sinica*, 1997. Available from: `http://www3.stat.sinica.edu.tw/statistica/oldpdf/A7n41.pdf`

[55] Morgan, J. Classification and Regression Tree Analysis. Technical report, Boston University School of Public Health, 2014.

[56] Quinlan, J. R. Unknown Attribute Values in Induction. In *Proceedings of the Sixth International Workshop on Machine Learning*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, ISBN 1-55860-036-1, pp. 164–168.

[57] Salih ÖZSOY, S. K., Gökhan GÜMÜŞ. C4.5 Versus Other Decision Trees: A Review. *Computer Engineering and Applications*, 2015.

[58] Breiman, L. Random Forests. *Mach. Learn.*, volume 45, no. 1, Oct. 2001: pp. 5–32, ISSN 0885-6125, doi:10.1023/A:1010933404324.

[59] Breiman, L.; Cutler, A. Random Forests. 2017. Available from: `https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#intro`

[60] Deng, H.; Runger, G. C.; et al. A Time Series Forest for Classification and Feature Extraction. *CoRR*, volume abs/1302.2277, 2013. Available from: `http://arxiv.org/abs/1302.2277`

[61] Xi, X.; Keogh, E.; et al. Fast Time Series Classification Using Numerosity Reduction. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, New York, NY, USA: ACM, 2006, ISBN 1-59593-383-2, pp. 1033–1040, doi:10.1145/1143844.1143974. Available from: `http://doi.acm.org/10.1145/1143844.1143974`

[62] Shasha, D.; Zhu, Y. *High Performance Discovery In Time Series: Techniques And Case Studies*. SpringerVerlag, 2004, ISBN 0387008578.

[63] Brigham, E. O. *The Discrete Fourier Transform*. 1974.

[64] Agrawal, R.; Faloutsos, C.; et al. Efficient Similarity Search In Sequence Databases. In *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, FODO '93, London, UK, UK: Springer-Verlag, 1993, ISBN 3-540-57301-1, pp. 69–84. Available from: `http://dl.acm.org/citation.cfm?id=645415.652239`

[65] Faloutsos, C.; Ranganathan, M.; et al. Fast Subsequence Matching in Time-series Databases. *SIGMOD Rec.*, volume 23, no. 2, May 1994: pp. 419–429, ISSN 0163-5808, doi:10.1145/191843.191925. Available from: `http://doi.acm.org/10.1145/191843.191925`

[66] Cooley, J. W.; Tukey, J. W. An algorithm for the machine calculation of complex Fourier series. 1965.

[67] Ruqaiya Khanam, S. N. A. ECG Signal Compression for Diverse Transforms. *Information and Knowledge Management*, 2012.

[68] Sandsten, M. Time-Frequency Analysis of Time-Varying Signals and Non-Stationary Processes An Introduction, 2016.

[69] Gabor, D. Theory of communication. 1946.

[70] Farge, M. WAVELET TRANSFORMS AND THEIR APPLICATIONS TO TURBULENCE. *Annual Reviews Fluid Mechanics*, 1992.

[71] Mallat, S. G. A Theory for Multiresolution Signal Decomposition: The Wavelet Representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, volume 11, no. 7, July 1989: pp. 674–693, ISSN 0162-8828, doi: 10.1109/34.192463.

[72] C M Leavey, J. S., M N James; Sutton, R. An introduction to wavelet transforms: a tutorial approach. *Insight*, 2002.

[73] Daubechies, I. *Ten Lectures on Wavelets*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992, ISBN 0-89871-274-2.

[74] Vetterli, M.; Herley, C. Wavelets and Filter Banks: Theory and Design. *Trans. Sig. Proc.*, volume 40, no. 9, Sept. 1992: pp. 2207–2232, ISSN 1053-587X, doi:10.1109/78.157221. Available from: `http://dx.doi.org/10.1109/78.157221`

[75] Guido, R. C. A note on a practical relationship between filter coefficients and scaling and wavelet functions of Discrete Wavelet Transforms. *Applied Mathematics Letters*, 2011.

[76] Juan Pablo Martínez, S. O. A. P. R. P. L., Rute Almeida. A Wavelet-Based ECG Delineator: Evaluation on Standard Databases. *IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING*, 2004.

[77] Hui Zhang, Y. Z. M.-S. L., Tu Bao Ho. Unsupervised Feature Extraction for Time Series Clustering Using Orthogonal Wavelet Transform. *Informatica*, 2006.

[78] Chaovalit, P.; Gangopadhyay, A.; et al. Discrete Wavelet Transform-based Time Series Analysis and Mining. *ACM Comput. Surv.*, volume 43, no. 2, Feb. 2011: pp. 6:1–6:37, ISSN 0360-0300, doi:10.1145/1883612.1883613. Available from: `http://doi.acm.org/10.1145/1883612.1883613`

[79] Zhang Z, G. C. L. K.-B. D., Telesford QK. Choosing Wavelet Methods, Filters, and Lengths for Functional Brain Network Construction. *PLoS ONE*, 2016. Available from: `https://doi.org/10.1371/journal.pone.0157243`

[80] Kin-Pong Chan, F.; Wai-chee Fu, A.; et al. Haar Wavelets for Efficient Similarity Search of Time-Series: With and Without Time Warping. *IEEE Trans. on Knowl. and Data Eng.*, volume 15, no. 3, Mar. 2003: pp. 686–705, ISSN 1041-4347, doi:10.1109/TKDE.2003.1198399. Available from: `http://dx.doi.org/10.1109/TKDE.2003.1198399`

[81] Similarity Search Over Time-Series Data Using Wavelets. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, Washington, DC, USA: IEEE Computer Society, 2002, pp. 212–. Available from: `http://dl.acm.org/citation.cfm?id=876875.878959`

[82] Sun, G.; Dong, X.; et al. Tumor Tissue Identification Based on Gene Expression Data Using DWT Feature Extraction and PNN Classifier. *Neurocomput.*, volume 69, no. 4-6, Jan. 2006: pp. 387–402, ISSN 0925-2312, doi:10.1016/j.neucom.2005.04.005. Available from: `http://dx.doi.org/10.1016/j.neucom.2005.04.005`

[83] S. Z. Mohd Tumari, A. H. A., R. Sudirman. Selection of a Suitable Wavelet for Cognitive Memory Using Electroencephalograph Signal. *Engineering*, 2013.

[84] Tibshirani, R. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society*, 1994.

[85] Ng, A. Y. Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, New York, NY, USA: ACM, 2004, ISBN 1-58113-838-5, pp. 78–, doi:10.1145/1015330.1015435. Available from: `http://doi.acm.org/10.1145/1015330.1015435`

[86] Chandola, V.; Banerjee, A.; et al. Anomaly Detection: A Survey. *ACM Comput. Surv.*, volume 41, no. 3, July 2009: pp. 15:1–15:58, ISSN 0360-0300, doi:10.1145/1541880.1541882. Available from: `http://doi.acm.org/10.1145/1541880.1541882`

[87] Liu, F. T.; Ting, K. M.; et al. Isolation-Based Anomaly Detection. *ACM Trans. Knowl. Discov. Data*, volume 6, no. 1, Mar. 2012: pp. 3:1–3:39, ISSN 1556-4681, doi:10.1145/2133360.2133363. Available from: `http://doi.acm.org/10.1145/2133360.2133363`

APPENDIX A

Acronyms

EC Entropy Curve

ECG Electrocardiogram

MLE Maximum Likelihood Estimation

OLS Ordinary Least Squares

ANN Artificial Neural Network

NN Neural Network

DTW Dynamic Time Warping

MLP Multilayer perceptron

CR Criterion of Regularity

MLR Multi-response Linear Regression algorithm

SVM Support Vector Machine

RF Random Forest

LR Logistic regression

k-NN k-nearest neighbors

GMDH Group Method of Data Handling

CART Classification and Regression Trees

DFT Discrete Fourier Transform

DWT Discrete Wavelet Transform

WT Wavelet Transform

ARARS	Accuracy, ROC AUC, recall, specificity
TSF	Time Series Forest
FFT	Fast Fourier Transform
CWT	Continuous Wavelet Transform
MRA	Multi-resolution analysis

Appendix **B**

Contents of enclosed CD

```
│  readme.txt ........................ the file with CD contents description
├── src ........................................ the directory of source codes
│   └── thesis ............... the directory of LaTeX source codes of the thesis
└── text ............................................ the thesis text directory
    └── thesis.pdf ........................... the thesis text in PDF format
```