Politechnika Łódzka
Wydział Elektrotechniki, Elektroniki, Informatyki i Automatyki
Katedra Informatyki Stosowanej

PRACA DYPLOMOWA MAGISTERSKA

Analysis and Application of Data Mining Methods to Real Data
Analiza i zastosowanie metod Data Mining na realnych danych

Marcin Franc
131689

Dr inż. Andrzej Romanowski

Łódź, 10 września 2010

Contents

Abstract

The aim of this thesis is to select and integrate data mining methods which can be applied to analyze the measurement data from the astrophysical experiment LOPES. The integration part was achieved by design and implementation of the software which is capable to perform the mentioned analysis and which is provided as the attachment to the electronic version of this dissertation. The thesis starts with the presentation of the "practically oriented" theory which is directly related to the applied data mining methods. Afterwards the most important functional and technical requirements of the proposed solution are depicted. Next chapters are focused on the description of the design and development of the implemented software. In the last part one can find examples of the software's usage and results obtained with data from the LOPES experiment.

Streszczenie

Celem niniejszej dysertacji jest wybór i integracja metod eksploracji danych, które mogą zostać użyte podczas analizy wartości pomiarowych pochodzących z astrofizycznego eksperymentu LOPES. Implementacja wybranych metod została dokonana w przygotowanym pakiecie oprogramowania, który został dołączony do elektronicznej wersji tej pracy dyplomowej. W tezie przedstawiono zagadnienia teoretyczne związane z metodami eksploracji danych zorientowane na ich praktyczne zastosowanie. Następnie opisane zostały najważniejsze wymagania techniczne i funkcjonalne, a także szczegółowo zaprezentowano opracowanie i implementację projektu informatycznego stanowiącego część praktyczną niniejszej dysertacji. W osobnym rozdziale zaprezentowano praktyczne przykłady zastosowania oprogramowania, jak również wyniki uzyskane przy wykorzystaniu pomiarowych danych eksperymentalnych.

List of Figures

Chapter 1

Introduction

The thesis concerns the issue of the integration of data mining methods which can be successfully applied during the analysis of the data from the astrophysical experiment LOPES. Its main goal is the implementation of the software which is capable to perform such analysis and which description will be hereby provided. The first part consists of the analysis of the software requirements together with the necessary theory concerning applied data mining methods. The second part is the software engineering oriented discussion about the implemented solutions. The interdisciplinary character of the thesis entails that its audience consists both of physicist and computer scientists. Therefore the dissertation starts with the brief general description of the LOPES experiment as well as basic concepts of data mining.

1.1 Introduction to the LOPES experiment

The experiment is located in the Karlsruhe Institute of Technology (KIT) and its goal is to investigate the nature of so called cosmic rays. These are particles with energies above 10^{10} eV which occasionally collide with the Earth's atmosphere giving rise to an air shower of secondary particles. When these secondary particles are deflected by the Lorentz force in the geomagnetic field they emit a radio pulse which can be measured with experiments like LOPES. Radio emission from such cosmic ray air showers was first discovered by Jelley in 1965 and described in details in [20]. LOPES experiment operates in coincidence with another astrophysical project – KASCADE-Grande, which is used for calibration, tests and reference. LOPES analyzes the frequency range from 40 to 80 MHz and the employed hardware is depicted in the figure 1.1. The signal is picked up by the antenna, sent via a coax-

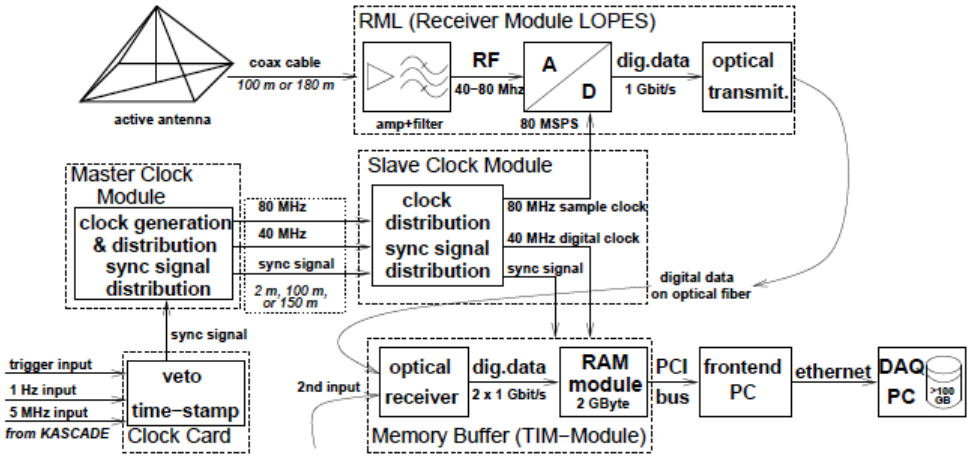

Figure 1.1: Outline of the LOPES hardware. Source: [16].

ial cable to the receiver module, digitized and sent to the memory module. From there it can be read out by the frontend PC and transmitted to the central data acquisition PC. The clock signals are generated by the master clock module and, together with the sync-signal, sent to the slave modules for further distribution.[1] For more reference see [8] and [16].

1.2 Basic data mining concepts

Almost each book which is related to the topic of data mining provides its very own definition of this term. In the author's opinion the most accurate one was provided by Berry et al. In the book [4, page 7] data mining is described as "the exploration and analysis of large quantities of data in order to discover meaningful patterns and rules". This definition is also not exactly a perfect one, since it does not clearly define what should be understood under the term of "large quantities of data". Such information is, however, not relevant from the point of view of this thesis and therefore the formal definition of the term ends here.

Data mining uses tools from three distinct domains:

- database technology;
- statistics;

[1] Compare to [16, page 32].

- machine learning and artificial intelligence.

An important feature of data mining methods is the fact, that their performance is evaluated on the basis of test examples and not abstract statistic identifiers. This is the reason why data mining models correspond relationships between input data and are not developed on the basis of abstract parameters. In other words the model is supposed to adapt itself to input data and not the other way round when input data is adjusted to the used model.[2]

Four stages of the data mining analysis

The data mining analysis can be divided into four stages:[3]

- data preparation;
- data exploration;
- data main analysis;
- application of the developed model.

During the data preparation phase it is determined which information is going to be used in the next stage. The appropriate data will be taken and its correctness verified. If the data originates from various sources, it has to be converted to the common format. During the next phase – data exploration – the general features of the data are extracted and evaluated. The third step starts with the initial selection of methods which could be used for solving the particular problem. The results will be afterwards assessed from the point of view of their usefulness (i. e. if they provide acceptable accuracy). The application of the developed model is considered as the last stage of the data mining analysis. The whole procedure is depicted in the figure 1.2. It is important to notice that those phases do not have to occur in a sequence. It can happen that the assumptions are changed during the course of the process and therefore one has to go back instead of moving on.

[2] This sentence is the paraphrase of the famous Albert Einsten's quotation — "If the facts don't fit the theory, change the facts."

[3] Compare to [10].

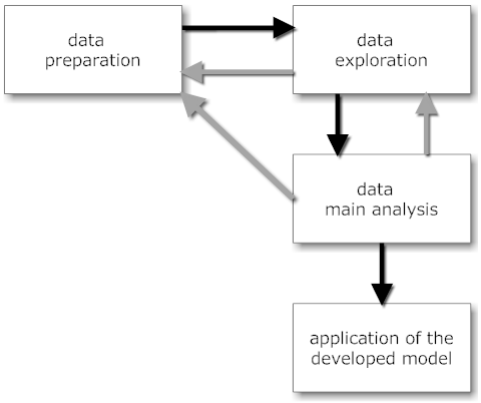

Figure 1.2: Four stages of the data mining analysis.

1.3 Data mining's applications in LOPES

A naive approach to the data mining analysis assumes that its methods could be used each time when some kind of data exists. From the logical point of view this statement is not utterly false[4] because the methods of data mining could be almost always applied after the necessary conversion of the input data. The question which has to be answered, is if such analysis brings any useful results. One can develop a data mining model which reflects relationships between arbitrary attributes from the input data set. Although the prediction's capabilities of such a model can even accidentally provide good results, it is not definitely any kind of a proof that discovered dependency provides any contribution to the solution of the given problem. Using the medical analogy, during the diagnostics of a patient doctors can scan the whole body and treat all anomalies as potential symptoms of an illness. The disadvantage of this method is the fact that not each anomaly is the real symptom and sometimes is just irrelevant concerning the particular problem. The important criterion of data mining analysis is therefore to know what exactly is supposed to be found.

The other important problem is the noise which occurs in input data and sometimes makes the analysis very complicated (if not impossible). Such a problem often occurs while analyzing the experiment's data due to the inaccuracy of measurements.

[4] Obviously the rules of the classical (boolean) logic exclude the possibility of the fact that the sentence is "not utterly false". The adjective "logical" was used only to present the existence of the purely theoretical option.

As far as the LOPES analysis is concerned, the first criterion is satisfied since the analysis always starts with an assumption which could be somehow proved from the physical point of view. The author has also tested plenty of data mining models using the LOPES data and despite the noise it was possible to apply them in order to perform classification, regression or prediction tasks. Therefore there exist no theoretical reasons why the integration of data mining methods in the analysis of the data from LOPES could not be performed.

1.4 Status quo

The current activities performed by physicist could be dived into the two groups. The first one concerns the detection of possible relationships between measured values[5] as well as development and assessment of the mathematical formula which describes those dependencies. As previously written, the fact that the target is always defined makes this analysis easier. The activities from the second group concern the exact explanation of the found relationship using the available physical theory. The scheme[6] is depicted in the figure 1.3. The activities of the first group will be labeled as the initial analysis.[7] Currently scientists conduct it using pure statistical methods which are implemented using programming language C++ and the ROOT library. The disadvantage of this approach is that instead on physics scientists have to first of all concentrate on the development of an application, which is used to find and describe a desired dependency. Moreover the development task takes usually more time as the explanation of the observed phenomenon. The reason is among others the fact that ROOT is not very well written library (see section 3.3.4) and C++ is not the easiest programming language to be applied (see section 4.2.1).

[5] The LOPES raw data has to be initially preprocessed in order to be used for any analysis purposes. Such preprocessing consists of calibration and partial noise suppression.

[6] The presented scheme is one of the possible applications of LOPES data. The other possibility is for instance forecasting the properties of a radio pulse using the theoretical model and afterwards comparing its results with the measured values.

[7] This name is used locally and has no connotations with the initial analysis – the stage of the data mining analysis.

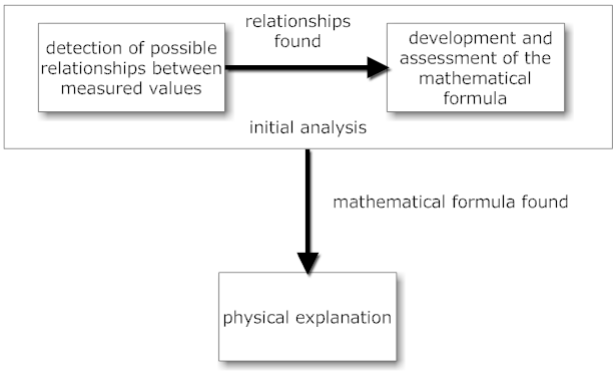

Figure 1.3: The flowchart of the current analysis process.

1.5 Motivation

As previously mentioned, currently the first part of the data analysis consists of the development of small applications which are supposed to perform mathematical and statistical operations and return a mathematical formula which can be afterwards analyzed from the point of view of the current physical theory. Such solution suffers, however, from some serious problems:

- Plenty of source code is written in a strictly "task-oriented" manner and it cannot be reused to perform even very similar tasks. Moreover it is often developed without consideration of paradigms of object oriented programming (which can be explained with the fact that structural languages like Fortran and C were previously applied).
- The currently used set of development tools makes the design and implementation of the quality software relatively difficult.

The mentioned problems result with the situation in which a lot of time and resources must be utilized to find a solution for a given problem. The generally formulated improvement possibility is to use data mining methods both to partially automate the relationships search and develop the high quality software which could be later reused and easily extended.

1.6 Goals of the thesis

The thesis is aimed to be written in the way allowing its approachable analysis by the target audience. The advanced topics – from the domain of data

mining, software engineering and programming – were explained and references to the technical literature[8] were always provided (mostly in the form of footnotes).[9] The main targets of this thesis are to assess data mining methods which can be applied to analyze the data from the LOPES, select the most promising ones and integrate those in the implemented software – the software requirements are presented in the section 3.1. The whole software engineering part bases on the opinions of the world greatest software architects and developers like Stroustrup, Alexandrescu or van Rossum[10] and therefore can be also partially used as the development reference containing the set of good programming practices.

[8] All online sources were last retrieved on 1st September 2010, 10:00 AM CEST.

[9] In other words the level of this thesis is located between "Ha" and "Ri" corresponding to "Shu-Ha-Ri" principle (jap."守破離") which describes the stages of learning to mastery the traditional japanesse arts and is congruent to apprentice-journeyman-master model.

[10] Stroustrup is the inventor and head developer of C++, Alexandrescu is considered as one of the greatest experts concerning design and programming using this language and van Rossum is the author of Python.

Chapter 2

Selection of the data mining methods

This chapter presents details concerning selection of the applied data mining methods. Their description as well as basic principles of their algorithms can be found in technical literature provided in the bibliography and are therefore not explained in this thesis.

The literature sources (especially [39]) suggest that the easiest solutions provide often acceptable results. This is the reason why the choice of the methods applied inside of the developed software has been done with the consideration of so called KISS principle[1]. The discussion starts with description of the preparation methods which were used to preprocess the input data.

2.1 Data preparation

The data preparation seems to be a topic which does not need any further explanation, however, it is just as much important as any other stage of the data mining analysis. As far as the LOPES data is concerned, its nature and quality are two most important reasons why this issue should be seriously taken into account – values are not stored in the form of a spreadsheet and contain often a considerable amount of noise. The third important goal of this step is to select the attributes which can provide the best results, which is also one of the software requirements (see section 3.1.1).

[1] KISS is the principle which states that simplicity should be a key goal in design. It is the abbreviation for "Keep it simple. Stupid!"

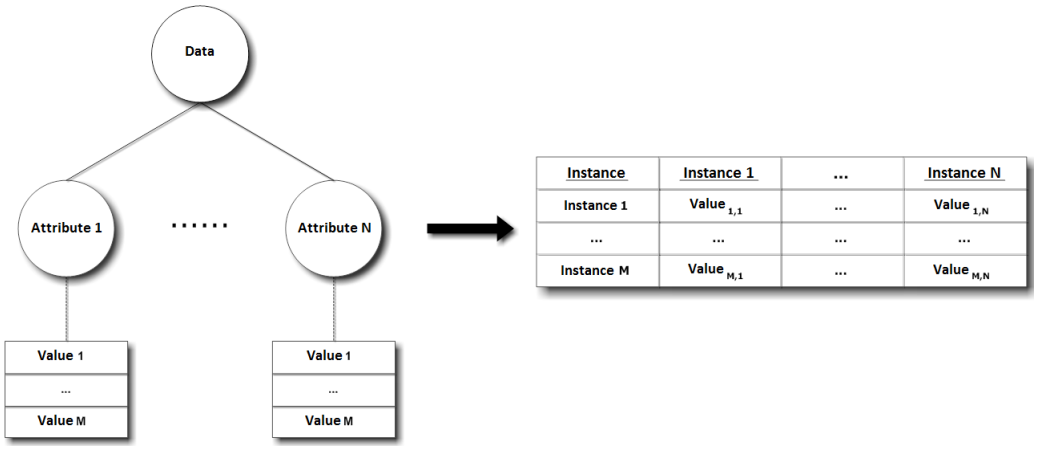

Figure 2.1: The conversion's scheme of the currently used data structure into a standard form.

2.1.1 Standard form

One of the best definition of a standard form is provided in the book [39]: "The concept of a standard form is more than formatting of data. A standard form helps to understand the advantages and limitations of different prediction techniques and how they reason with the data." The different methods used for data mining tasks have one common property – their world view which can be presented in the form of a spreadsheet. The measured values from LOPES experiment are stored in the form of serialized *TTree* structure from the ROOT library. The general scheme of both the structure and the conversion procedure is depicted in the figure 2.1. Its detailed description will be provided in the section 3.3.4. In the standard form table from the figure 2.1 each row represents an instance of input data which is characterized by its attributes (columns). $Value_{i,j}$ is also the value of the j-th attribute from the i-th instance.

2.1.2 Data smoothing

The idea behind the concept of the data smoothing is depicted with the formula 2.1 stating that in case of data stored as a standard form all instances contain a certain amount of noise which could be relatively easily removed – where i refers to the i-th instance.

$$value(i) = mean(i) + noise \tag{2.1}$$

The target of those methods is to substitute the original set of values with the reduced one since it is highly probable that such values can provide better

results. This is also the way to reduce noise from the input data. The book [21] mentions four techniques which can be used to perform data smoothing:

- binning;
- clustering;
- combined computer and human inspection;
- regression.

Both first and third method provide the best results while applying them during the analysis of the LOPES data.[2] However, since the software's main target is to automate the data analysis procedure as much as possible, the binning method has been chosen. Therefore the input data was divided into equi-width bins and smoothed with one of two methods:

- smoothing by bin medians – in which each bin value is replaced by the bin's median value;
- smoothing by bin boundaries – in which the minimum and maximum values in a given bin are identified as the bin boundaries and each bin value is then replaced by the closest boundary value.

In [40] one can find, however, the very important remark concerning data cleansing. The author points out, that it is not desirable to train the model using "clean" data if its performance is going to be tested using a "dirty" one. In other words removing noise from attributes for the training set gives the model no chance to find a way to "combat" with it. This paradox should be taken into account while developing any data mining model used for the analysis of noisy data.

2.1.3 Attributes selection

The practical experiments[3] revealed that data mining algorithms tend to "get confused" when they have to consider irrelevant attributes while searching for a solution. This is why the usage of methods which eliminate unused attributes is the important part of the data mining analysis. The bibliography describes plenty of possibilities which allow to perform such reduction. The group of most effective ones consists of those which use data mining methods

[2] In this case it also confirms the authenticity of the KISS principle, since the binning method is also the simplest one.

[3] Described among others in [40] and [39].

to choose the attributes which are supposed to provide the best performance.[4]

The developed software uses decision trees to reduce the amount of attributes taken into consideration during the data mining analysis. Attributes which are rejected are those which does not participate in the solution obtained with the decision tree method. The advantage of this method is the fact that such a tree can be quickly generated and greatly improve the performance of others data mining methods which use the same input data set. The other popular method to perform such a reduction is SVM, which will be described later on in this thesis. This method, however, must be mentioned now, since it also provides a set of tools which make it possible to build a hierarchy of attributes according to their relevance to the solution of a given problem. The method is nevertheless used during the analysis stage and therefore it would be purposeless to deploy the same mechanism two times.[5]

Data reduction methods

At this point the concept of the so called data reduction should be introduced When data is stored inside of a standard form there exist three simple operations which can reduce its amount:

- deleting a row which represents an instance;
- deleting a column which represents an attribute;
- reducing the number of values in a column (smoothing an attribute).

These operations attempt to reduce the dimension of the standard form and preserve the character of the original data.[6] The last two has been already described in this chapter, so only the first one needs a short clarification. Basically several methods of sampling are used to reduce the number of attributes. The simplest solution assumes using a single sample consisting of n instances. As far as LOPES data are concerned, it is reasonable to sample the input data based on the value of the selected attribute.[7]

[4] The application of the methods from the data analysis phase during the data preparation can be considered as contradictory to the data mining schema presented in the figure 1.2. As previously stated this strategy is, however, commonly used due to its effectiveness.

[5] According to [40] the combination of two methods (in this case of two mechanisms responsible for attributes scoring) provides better results than a single method used alone.

[6] Except from the third operation which goal is to perform data discretization.

[7] Such attribute should be chosen by a scientist who uses the software.

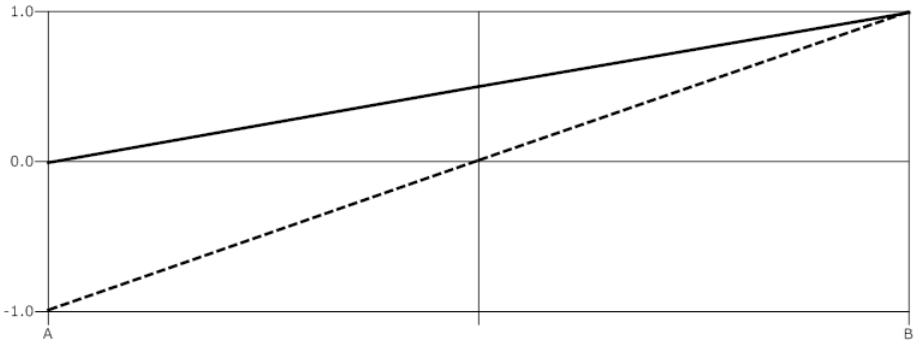

Figure 2.2: Linear transformation functions – $f_1 : [A, B] \rightarrow [-1, 1]$ (dashed line) and $f_2 : [A, B] \rightarrow [0, 1]$.

2.1.4 Normalization

The last concept concerning the data preparation, which shall be introduced in this chapter, is normalization. Some data mining methods tend to operate better when normalized data is provided as their input. The simplest normalization's approach is the conversion of the input data from interval [A, B] into other interval (often [0, 1] or [-1, 1]). The elementary approach is to apply the so called decimal scaling which is presented using the equation 2.2.

$$value'(i) = \frac{value(i)}{10^k} \tag{2.2}$$

The parameter k is the smallest value for which $max(|value'(i)|)$ is lesser than interval's boundary. The other method (which is also commonly used in the field of image processing) is to use a linear function which conducts the appropriate transform of the input values (see figure 2.2).

2.2 SVM method

SVM (abbreviation for "Support Vector Machine") is one of the data mining techniques which is commonly used both for classification and regression problems. The technique was invented by Vapnik and described in details in his book [36]. The method is often compared with the one using neural networks, hence the short comparison between those two will be presented in the section 2.2.2. Despite the fact that the method is relatively easy to use, some remarks have to be given to maximize the probability of getting the acceptable results. Those remarks concern choosing optimal parameter, proper kernel and transforming the input data.

2.2.1 The preferred approach

The commonly used (but not optimal) approach to apply the SVM method is to use it in the following way:

1. Transform the data into a standard form.
2. Choose random kernel and parameters.
3. Test.

Such application of the method, however, does not guarantee that the acceptable results will be obtained since SVM (just like any other data mining technique) is not able to provide a reasonable performance without at least the proper parameters setting. The practical and preferred usage of this method consists of the following steps:

1. Transform the data into a standard form.
2. Perform data scaling.
3. Choose the proper kernel (the RBF[8] kernel provides often good results for regression problems).[9]
4. Use the cross validation method[10] to obtain optimal parameters C and γ.
5. Apply SVM method using chosen kernel and parameters to the whole training set.

Since kernel functions are often dependent on the inner product of input vectors, the normalization of the data is the particularly important step concerning the SVM method. The large values of mentioned products can induce overflow errors or at least make the computations slower. This and

[8] Abbreviation for "Radial Based Function".

[9] Obviously the RBF kernel is not a panacea for all regression problems. Practical experiments have shown that the function tends to provide poor results when the number of attributes differs significantly from the number of instances.

[10] Cross validation is an approach to estimate the error of the data mining method. The data are dived into n parts and the training phase will be repeated n times. Each time training set contains $n-1$ parts and the test set one part. The performed n error calculations are used later on to estimate the overall method's error. Plenty of literature sources (among others the book [40]) suggest 10 as the optimal amount of folds which should be used during the cross validation.

other problems are described in details in the second part of the [29].[11] The preferred RBF kernel is the function which form is presented in the equation 2.3.[12]

$$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}, \gamma > 0 \tag{2.3}$$

The function's result is directly dependent on the mentioned parameter γ The parameter C is so called "penalty parameter". The larger the C, the more the error in the training set is penalized. This parameter should be therefore chosen with the consideration of so called overfitting phenomenon.[13] The search for the optimal parameters C and γ is, as mentioned previously, performed using the cross validation method. The set of combined pairs (C, γ) are investigated and the pair which provides best cross validation results is chosen. The practical approach which often provides acceptable results is to combine the elements from the exponentially growing sequences of C and γ. The developed software used the following sequences:[14] $C = 2^{-5}, 2^{-3}, \ldots, 2^{15}$ und $\gamma = 2^{-15}, 2^{-13}, \ldots, 2^{3}$. Such "grid search" may seem to be a naive approach and it can be considered as attractive to substitute it with a heuristic technique. However, the method's searching time is relatively short and what is more the solution can be easily parallelized, since cross validations are independent and can be performed in the same time. This makes it attractive to apply using a multicore machine.

2.2.2 Advantages of the SVM solution in comparison to the neural networks

According to [39] neural networks are theoretically capable to simulate any complex mathematical relationship between the input data. However, the book which was written in 1998 does not mention SVM method at all and therefore does not provide any useful benchmark nor comparison. Also there are no references to the SVM technique in the classical book from Berry et al. – [4] – and in the first (published in the year 2000) edition of [40]. In the authors opinion such situation was a result of the fact that the method was officially presented in the year 1995 and it was at that time perhaps

[11] Although this source concentrates primarily on the neural networks, most considerations also apply to SVM.

[12] For further reference see [22].

[13] The phenomenon is explained with the fact that the model loses its "sense of generalization" when it fits too much to the given training set.

[14] This strategy is recommended by the authors of the LIBSVM library which is integrated inside of the Orange API which is in turn directly integrated in the developed software – see section 3.3.1.

"too immature" to be seriously considered as a data mining tool. Currently it is often described as an alternative to the neural networks.[15] Interesting enough is the fact that principles of evolving of those two methods are completely different. The development of the first one followed the heuristic path with applications and extensive experimentation preceding theory. On the other hand the development of SVM involved sound theory first, afterwards implementation and experiments. As far as the performance differences are concerned, neural networks tend to suffer from multiple local minima whereas SVM is more resistant to them. Plenty of experts describes the reason why SVM often outperform neural networks in practice is the fact that it is less prone to overfitting. It would be, however, an exaggeration to state that the method in the current development state is better than the neural networks approach. It showed, however, better practical results while processing the LOPES data and was therefore integrated in the developed software.

2.3 Genetic programming

Genetic programming (GP) is integrated within the software to automate the process of finding the mathematical relationships between the input data. The following section starts with the definition of symbolic regression and genetic programming itself. Symbolic regression is the process of identification of a model which fits the input data as good as possible. For the purpose of this thesis it will be considered, however, as the process of developing the mathematical formula which tries to optimally depict the relationships between the LOPES data.[16] To simplify one can consider genetic programming[17] as the process of symbolic regression which is performed by evolutionary algorithms.[18] An important property of genetic programming is its problem independence which allows to construct the flowchart specifying the general sequence of stages. This sequence does not have to be modified

[15] This comparison concerns feedforward neural networks, using backpropagation algorithm for training, which are commonly applied for data mining purposes – for instance as a part of the popular RapidMiner solution – see section 3.3.1.

[16] This definition is somehow congruent with the one describing the sequence of activities which is called "initial analysis" in the section 1.2. The data preparation step shall be, however, not forgotten.

[17] For further reference see [41].

[18] An evolutionary algorithm is the one which bases on the Darwin's evolution's theory. Although the terms evolutionary algorithm and genetic algorithm are commonly used as synonyms (see for instance [5]) they do not have exactly the same meaning. Simply stated genetic algorithms are subset of evolutionary algorithms. For further reference see [37].

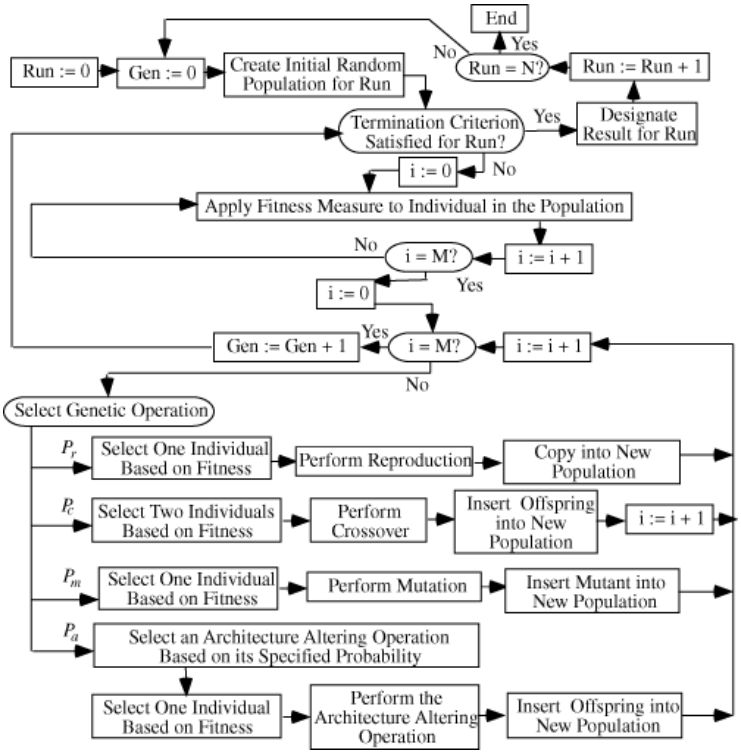

Figure 2.3: Basic flowchart for genetic programming. Source: [23].

for each new run or each new problem and is depicted in the figure 2.3.[19] To produce a mathematical formula, evolution algorithm operates on a set of terminal symbols[20] and a set of functions. For instance if a function of one independent variable x is supposed to be found, the set of teminals T consists of this variable and the numerical constant A – $T = \{x, A\}$. The set of functions contains in the simplest case four elementary mathematical operators (addition, subtraction, multiplication and division) – $F = \{+, -, *, /\}$.[21]

[19] The figure 2.3 depicts genetic operations such as crossover, mutation and reproduction. The simplified definition of crossover is creating a new program from two parent programs. Reproduction can be considered as copying the chosen individual programs to new population and mutation is creating a new program from the randomly modified old program. For further reference see [37] and [23].

[20] In the domain of computer science a terminal symbol is commonly associated with the theory of formal languages as the grammatical symbol which cannot be substituted by any production (see [1]). In the mentioned case the set of terminal symbols contains only variables and numerical constants.

[21] From the practical reason an algorithm is usually constructed in such a way that it returns 1 as the result of the division by 0. This assumption could be problematic while analyzing the solution's performance – see section 4.2.3.

The important step is to choose so called fitness function which provides the information about the quality of consecutive solutions. The smaller is the output of the fitness function, the better – the return value of 0 would indicate a perfect fit. The algorithm chooses solution's candidates according to the "survival of the fittest" principle and uses genetic operations to create the next generations of objects. The target is of course to obtain possibly the fittest (optimal) solution.

GP as a data mining method

One can discuss the affiliation of genetic programming to the set of data mining methods. The book [4] classifies evolutionary algorithms as one of them (with the remark that the method is relatively rarely used). It does not, however, mention genetic programming at all. The target of GP is nevertheless congruent with the definition of data mining provided in the chapter 1. For the purposes of this thesis it will be therefore assumed that genetic programming can be treated as one of the data mining techniques.

Chapter 3

Description of the applied solutions

As previously stated, the practical part of this thesis consist of the software which was designed and developed by the author. This software allows the application of data mining methods during the analysis of the data from the LOPES experiment. The ordering party is the KITs Institute for Nuclear Physics[1] which is responsible for the LOPES experiment. It is also highly designated that the software could be used in the future by the radio astronomy community as an universal analysis tool. Moreover the software should work together with the current programming components which are partially applied during the analysis of the data – CR-Tools.[2] At this point it needs to be clearly stated that the implemented data mining solution should not be considered as any kind of competition to the applied one, but as an alternative which can be also used parallel. Such combination can simplify the process of finding optimal results since it allows concerning the problem using different perspectives. This chapter discusses the most important decisions which had to be made during design and implementation of the mentioned software. This concerns the description of the used programming languages, third party libraries, rules and conventions. First of all, however, it summarizes the most important software requirements.

3.1 Software requirements

This section presents both functional and technical requirements concerning the implemented data mining solution. In author's opinion it is better not to describe the whole list of requirements, however, to depict details of

[1] http://www-ik.fzk.de

[2] http://usg.lofar.org/wiki/doku.php?id=software:packages:cr-tools

those which are the most important. The less important ones were subjects of informal discussions with KITs scientists during the development of the software. The presented requirements are also crucial tasks which should be accomplished.

3.1.1 Functional requirements

The KITs scientists have defined a couple of functional requirements. From those the most important are:

1. In the simplest form it should be possible to use the software without any data mining knowledge.

2. It should possible to select the best set of attributes which could build a potential relationship. For instance if y is the target attribute, the software should provide an answer if this attribute is to higher extent dependent on x_1, x_2, x_3 or x_1, x_3, x_4 (where $y, x_1, \ldots, x_4$ represents the set of chosen attributes).

3. It should be possible to compare the new method's performance[3] with the performance of the mathematical formula which has been already developed.

4. The software should provide a solution which can be understood by physicist or, if such solution cannot be provided, at least its performance should be presented in the understandable form (for example using the error's histogram).

5. The software should be designed and developed in the way which allows to configure it for the specific tasks, as well as makes eventual future extensions possible.

The discussion concerning those requirements starts with the point stating that the software could be used without any prior data mining knowledge. In principle it means that in a standard case the user employ data mining algorithms, however, he does not have to know how they work and which configuration parameters are important to obtain acceptable results – in other words, the whole data mining tier should be transparent to the user. Such requirement makes it difficult to use the existing data mining software packages in the original form.

[3] In the domain of the computer science the performance is usually associated with the algorithm's speed. In the data mining literature (also in this thesis) the term is used to describe the method's reliability.

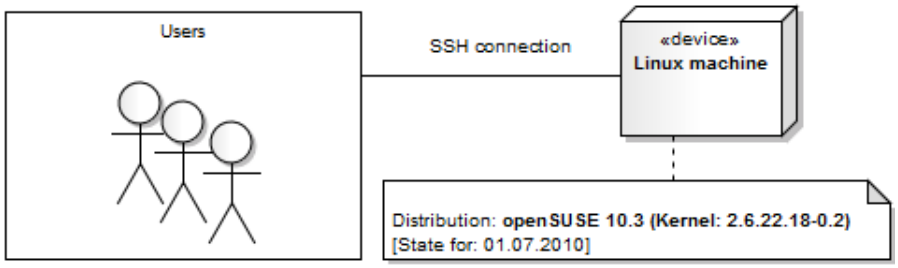

Figure 3.1: The scheme of usage of the current analysis tools.

Due to the fact that it should be possible to compare the performance of the previously developed solution with the new one, the software must be capable of evaluating any arbitrary mathematical formula and compare its performance with the performance of the newly obtained solution. The first two requirements together with the fact that the solution should be in the ideal case understandable by scientists, makes the mathematical formula the most preferable output which could be provided.

The fifth requirement states that the software can be configured for specific activities and easily extended by the physicist which theoretically contradicts the fact that its heart (which is the data mining analysis tier) should be transparent to the user. Those two items can be stipulated with the creation of the well documented interface. This interface is applied to develop the main application and it could be later used to extend the functionality of the implemented solution.[4]

The second requirement, which states that attributes could be chosen according to their relevance to the target relationship is now purposely ignored since it was discussed in details in the section 2.1.3.

3.1.2 Technical requirements

The discussion concerning the technical requirements starts with the description of the way in which the software package which is currently applied for the processing and analysis of the measurement data – CR-Toools – is used. It is installed on the Linux machine and its users log in per SSH, perform the desired type of analysis and download the results (see figure 3.1). The newly

[4] This interface is developed using Python. The advantages of applying scripting languages are described in the section 3.2.

developed solution should be capable to operate using the same principle. Therefore it would be desirable to reduce the amount of necessary libraries to the absolute minimum. Ideally the software should use those, which were already installed for the purposes of the CR-Tools.

One of the technical requirements which is directly derived from the general one, that the software can be easily extended in the future, is the choice of the programming language, which enables such development. Obviously the new software has to be first of all written with consideration of the optimal design and high quality. The scientists, however, has also explicitly declared that the language in which the software should be developed, has to be one of the languages they are familiar to.[5] Due to the fact that a lot of data mining APIs are written Java, it complicates – however makes not impossible[6] – the application of those solutions, since Java does not figure out on the mentioned list. As an example can serve of the most popular open source data mining analysis environment RapidMiner[7] or Weka[8] – the solution which was in details described (and partially recommended) in the classical data mining book [40]. The next chapters can be, however, treated as the kind of an empirical proof that it is also possible to develop the software according to the given requirements without using Java at all.

The second important precondition states that the software can contain only open source components. This is on the other hand not a very narrowing restriction due to the existence of plenty of alternatives which could be used as parts of the developed software. The both mentioned examples of popular data mining solutions are available under open source licenses.

Since LOPES is the international collaboration, both software and its documentation (which should be generated using Doxygen[9]) must be written in English.

[5] The languages prefered by the KIT-LOPES group are among others C, C++ and Python.

[6] It is often possible to write a wrapper for the necessary Java functions and call them from the other language.

[7] `http://rapid-i.com`

[8] `http://www.cs.waikato.ac.nz/ml/weka`

[9] `http://www.doxygen.org`

Communication with the user

The important technical aspect which should be clarified at the very beginning is they way in which the user can communicate with the application. After consultations with the scientists three possible forms of such communication were chosen:

- command line interface;
- graphical user interface (GUI);
- communication via configuration files.

The method using configuration files was rejected as the very first one, since it would require from the user the knowledge about the eventual structure of such file. The graphical user interface was on the other hand rejected due to the even mundane reason – the scientists have explicitly stated, that a good functioning software should have priority to a GUI. As the result the ergonomic command line interface was implemented which also allowed to invest more time in the development and testing of the software.

3.2 Choice of the programming language

The first information about the choice of the used programming languages is provided in the section 3.1.2. The final decision which has been taken assumes using Python as the software's main development language. Due to the used third party libraries also C++ had to be applied to develop a group of components which can control those libraries. The connection between C++ and Python is realized using Boost libraries – the applied ones are described in the section 3.3.

One of the most important software requirement which was described in the chapter 3.1.1 is the fact that it can be easily configured and extended. To accomplish this target the Python's class library[10] has been developed and used both for development of the main application as well as to provide a possibility of writing custom scripts. Since the main application is also written in Python, scientists can modify its source code to obtain the functionality they actually need. One cannot forget that software's target group consist both of developers as well as advanced technical users which know exactly

[10] The set of classes developed by the author is congruent with the definition of a class library described in the book [38].

what kind of expectations they associate with the application. The possibility of performing arbitrary modifications using scripting language fulfills their demands, providing the necessary degree of freedom. The main application written using script language is not a compiled black box solution but the source code which can be later on easily modified or extended.[11]

Advantages of scripting languages

The most common usage of scripting languages is the integration and eventual extension of components which are utilized by those languages. Therefore they are often called "glue languages" which is also commonly perceived as the biggest difference between them and system programming languages like C, C++ or Java. System programming languages were designed to develop algorithms and data structures literally from scratch and often using primitive types like a word of memory. The main advantages of scripting languages which allow fast and simple development of programs (scripts) is dynamic typing[12] and a relatively huge number of machine instruction per statement.[13] The comparison between system programming languages and scripting languages, based on those two categories, is depicted in the figure 3.2 and the small benchmark is presented in the figure 3.3.

Each row in the table presents an application which was implemented twice – once using a system programming language and once using the scripting language Tcl.[14] The code ratio is the ratio of lines of code (LOC) for those two implementations. If this ration is greater than 1 it means that the solution developed with system programming language needed more LOC. Effort ratio is the ratio of development times. All this information was collected

[11] The similar functionality could be achieved by the means of configuration files which would control the application's workflow. The solution was rejected due to the fact that in order to allow to control the advanced operations one would have to develop a file which had a complicated structure, which in turn would have to be learnt by scientists.

[12] The type checking and identification is performed at run-time as opposed to at compile-time.

[13] The typical statement from the scripting language executes thousands of machine Instruction, whereas the one from system programming language is translated into about five machine instructions. The part of this difference is connected with the fact that scripting languages must be interpreted which is less efficient as executing the compiled source code from the system programming language. The main reason is, however, associated with the fact that primitive statements from scripting languages provide a lot more functionality.

[14] Tcl (abbreviation for "Tool Command Language" and often called "tickle") is one of the scripting languages with syntax similar to Python.

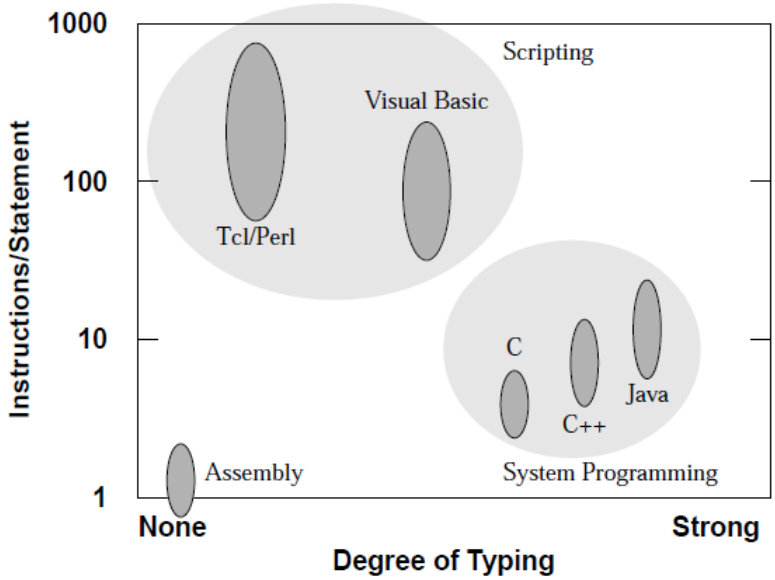

Figure 3.2: The comparison of various programming languages based on the amount of machine instructions per statement and degree of typing. Source: [26].

from various Tcl developers as answers to the article from the "comp.lang.tcl" newsgroup.[15]

The presented examples clearly present the huge potential of scripting languages which is going to be "exploited" in the developed software.

3.3 The third party components

This section describes all third party components which were integrated in the developed software. The description starts with the components used for data mining appliances.

3.3.1 Data mining software packages

One of the possibilities to use the data mining methods which were presented in the chapter 2 is to implement them from scratch. Such approach would be, however, comparable to "reinventing the wheel", since there are plenty of software packages which could be used for the specific aspects of the data

[15] `http://wiki.tcl.tk/844`

Application (Contributor)	Comparison	Code Ratio	Effort Ratio	Comments
Database application (Ken Corey)	C++ version: 2 months Tcl version:1 day		60	C++ version implemented first; Tcl version had more functionality.
Computer system test and installation (Andy Belsey)	C test application: 272 Klines, 120 months. C FIS application: 90 Klines, 60 months. Tcl/Perl version: 7.7K lines, 8 months	47	22	C version implemented first. Tcl/Perl version replaced both C applications.
Database library (Ken Corey)	C++ version: 2-3 months Tcl version: 1 week		8-12	C++ version implemented first.
Security scanner (Jim Graham)	C version: 3000 lines Tcl version: 300 lines	10		C version implemented first. Tcl version had more functionality.
Display oil well production curves (Dan Schenck)	C version: 3 months Tcl version: 2 weeks		6	Tcl version implemented first.
Query dispatcher (Paul Healy)	C version: 1200 lines, 4-8 weeks Tcl version: 500 lines, 1 week	2.5	4-8	C version implemented first, uncommented. Tcl version had comments, more functionality.
Spreadsheet tool	C version: 1460 lines Tcl version: 380 lines	4		Tcl version implemented first.
Simulator and GUI (Randy Wang)	Java version: 3400 lines, 3-4 weeks. Tcl version: 1600 lines, < 1 week.	2	3-4	Tcl version had 10-20% more functionality, was implemented first.

Figure 3.3: The comparison of system programming languages and scripting languages on the basis of LOC and time needed to develop a specific application. Source: [26].

mining analysis. The section 3.1.2 mentioned already two data mining solutions: RapidMiner and Weka. Both are implemented in Java and allow development of own programs not only using the graphical user interface but also directly with Java source code. However, as mentioned in the section 3.1.2, such solution would require an implementation of a wrapper allowing calling Java functions from Python.[16] Luckily there exists the data mining software which explicitly enables Python scripting. An example of such software is Orange which is developed by Artificial Intelligence Laboratory at the University of Ljubljana.[17] It is available under the GNU GPL[18] license,

[16] One of the solutions which could be used for this purposes is the Jython project. See `http://www.jython.org` for further reference.

[17] `http://www.ailab.si/orange`

[18] `http://www.gnu.org/licenses/gpl.html`

therefore its source code could have been carefully analyzed by the author.

One of the biggest advantages of Orange is the fact that it uses Python as the "front-end" language and all "back-end" operations are performed in C++.[19] Although the software is mainly supported for the family of Windows operating systems, the source code can be compiled manually and necessary elements applied under other operating systems. Since only the scripting part of the Orange software will be used,[20] the installation reduces itself to the proper installation of Python packages.[21] The software has, however, also some disadvantages – among others lack of documentation of various modules and not obeying the Python's code design guidelines.[22]

This thesis utilizes the assumption that genetic programming can be considered as one of the data mining methods – see section 2.3. Therefore the description of the software, which provides the implementation of the genetic programming will be hereby supplied. It is called Eureqa[23] and is developed by Cornell Computational Synthesis Laboratory at the Conrell University – see [31] for further reference. It consists both of the GUI application and the C++ API,[24] which is directly integrated in the developed software. Eureqa can be used under the terms of the BSD license.[25] In the authors opinion, one of the most important advantages of the mentioned API is the fact that its source code was written using the modern coding standards (which among others consist of using Boost libraries). Eureqa API allows searching mathematical relationships in input data using various computation servers. In the simplest case such a server should run on the local machine before the API could be used. Although the Eureqa's class library is of the type "header only"[26], the necessity of running external binary file containing server application greatly complicates its usage.

[19] The similar principle is used by the author during the software development – see section 4.2.2 for further reference.

[20] The software enables also a development of the own programs using graphical user interface.

[21] This operation can be substituted with copying files into an arbitrary directory and properly setting PYTHONPATH variable. This method can be applied if packages cannot be installed in the standard directory (for instance due to the lack of administrator privileges).

[22] See [28].

[23] `http://ccsl.mae.cornell.edu/eureqa`

[24] `http://code.google.com/p/eureqa-api`

[25] `http://www.opensource.org/licenses/bsd-license.php`

[26] It consists only of header files.

3.3.2 Boost libraries

Boost libraries are the set of modern C++ libraries based on the C++ standard. The organization responsible for developing and publishing those libraries is the Boost community. It consists of a large group of C++ developers from around the world coordinated through the website[27] and mailing lists. The mission statement of the community is to develop and collect libraries of high-quality that complement the C++ standard.[28] Those are also the part of the new C++ standard[29] which should be published in the year 2011 as the replacement of the actual one.[30] In authors opinion those libraries are next to STL (Standard Template Library)[31] the inseparable part of modern C++ programming. The source code is released under the Boost Software License[32] allowing to use, modify and distribute it for free.

Most of the libraries consist solely of header files that can be directly used, some of them require, however, prior compilation and have to be linked during the usage with external programs. The list of those used in the implemented software consists of:

- Any – allowing usage of generic container for single values of different value types;
- Exception – used for exception handling;
- Foreach – allowing the usage of foreach iterator;
- Python – serving a framework for interfacing Python and C++;
- Thread – allowing safe execution of parallel operations;
- uBLAS – allowing usage of the matrix structure;
- Utility – providing among others a possibility of a proper implementation of Singleton classes.

[27] `http://www.boost.org`
[28] Compare to [32].
[29] The standard is currently known under the unofficial name "C++0x".
[30] See [33].
[31] `http://www.sgi.com/tech/stl`
[32] `http://www.boost.org/LICENSE_1_0.txt`

3.3.3 Python packages

The two packages which are not included in the applied version of Python[33] are NumPy[34] and matplotlib[35]. The first one is necessary to perform the numerical regression using the Orange software. The second one is utilized to provide the graphical representation of obtained results.

3.3.4 ROOT library

This library is one of the most commonly used both in the LOPES experiment as well as in the other projects which are performed by the radio astronomy community. All measured data is processed using ROOT and saved as its serialized structures. The usage of this library in the developed source code will be, however, due to the following reasons reduced to the absolute minimum:

- The classes interfaces are typical examples of so called "Do-It-All Interfaces" – see [3, page 4] – what makes them unnecessary large and hard to understand.
- The class hierarchy is always designed using the "is a" principle.
- The simplest operations induce memory leaks.

To prove the correctness of the choice which was made, the class *TTree*, which structure is presented in the figure 3.6, will be used as an example. The class is derived from five classes (see figure 3.4), although it is only the special case of one of them – *TObject* – which contradicts the important design principle that aggregation should be preferred to derivation.[36] The class itself provides over one hundred methods which can be used. In addition this interface is extended due to the derivation of all public functions from parent classes and hence offers the functionality which is absolutely not correlated with its nature.

The figure 3.5 presents the results of the memory test done with the Valgrind[37] program.[38] The source code which was tested consisted only of the

[33] The software uses the version 2.6 of the interpreter.

[34] `http://numpy.scipy.org`

[35] `http://matplotlib.sourceforge.net`

[36] See [3].

[37] Valgrind is a set of tools for the dynamic error analysis of programs. See `http://valgrind.org`

[38] The used PC was Intel Core 2 Duo 2,26 GHz, 2048 MB RAM under Ubuntu 9.10 (kernel version: 2.6.31-22). If not other stated this system is assumed to be used for all such tests.

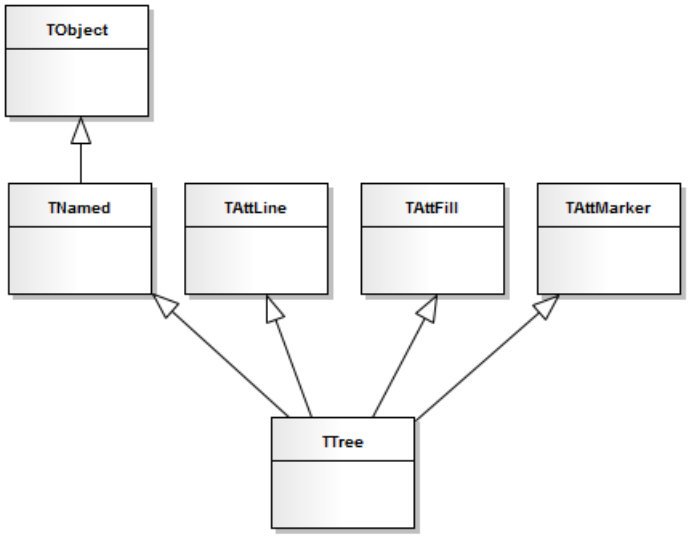

Figure 3.4: The derivation hierarchy of the *TTree* class.

statement which included one of library's headers.

Necessary evil

One can ask the question why a library which has so many disadvantages is actually used in the developed software. The simple answer is to treat it as the kind of the necessary evil which cannot be avoided. There exists absolutely no documentation which provides enough information about the serialization process of the *TTree* structure. Due to the structure's complexity (see figure 3.6) it also not trivial to obtain the necessary information using the methods of reverse engineering.[39] Since ROOT is anyway installed in the software's target environment, it would be counterproductive to spend unnecessary amount of time to achieve the software's independence from this library.

3.3.5 Summary

The chapter 3.1 states that it would be desirable to reduce the amount of third party libraries to the absolute minimum. This objective was achieved since only the Python module containing Orange has to be installed.[40] As

[39] In this case analyzing a representative set of serialized files using hexadecimal editor.

[40] The list of software necessary for the purposes of CR-Tools and therefore installed on the target machine is available at `http://usg.lofar.org/wiki/doku.php?id=software:packages:cr-tools:installation_of_cr-tools`

```
==6464== Memcheck, a memory error detector
==6464== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==6464== Using Valgrind-3.5.0-Debian and LibVEX; rerun with -h for copyright info
==6464== Command: ./root-test
==6464==
==6464==
==6464== HEAP SUMMARY:
==6464==     in use at exit: 871,511 bytes in 18,261 blocks
==6464==   total heap usage: 37,860 allocs, 19,599 frees, 1,732,632 bytes allocated
==6464==
==6464== LEAK SUMMARY:
==6464==    definitely lost: 240 bytes in 2 blocks
==6464==    indirectly lost: 120 bytes in 10 blocks
==6464==      possibly lost: 34,848 bytes in 961 blocks
==6464==    still reachable: 836,303 bytes in 17,288 blocks
==6464==         suppressed: 0 bytes in 0 blocks
==6464== Rerun with --leak-check=full to see details of leaked memory
==6464==
==6464== For counts of detected and suppressed errors, rerun with: -v
==6464== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 63 from 8)
```

Figure 3.5: Results of the the memory leak test done with the Valgrind program.

mentioned in the section 3.3.1 installation of Orange can be substituted with the proper modification of Python's environmental variable.

3.4 Principles and conventions

This section describes the most important principles which were applied during the design and development of the software. The discussion starts with the short summary of the most important properties of the quality and reliable software.

3.4.1 The most important properties of the quality and reliable software

McConnell presents in his book [24] the list of the most important principles of the high quality design which consists of the following properties:

- minimal complexity;
- ease of maintenance;
- minimal connectedness;
- extensibility;
- reusability;

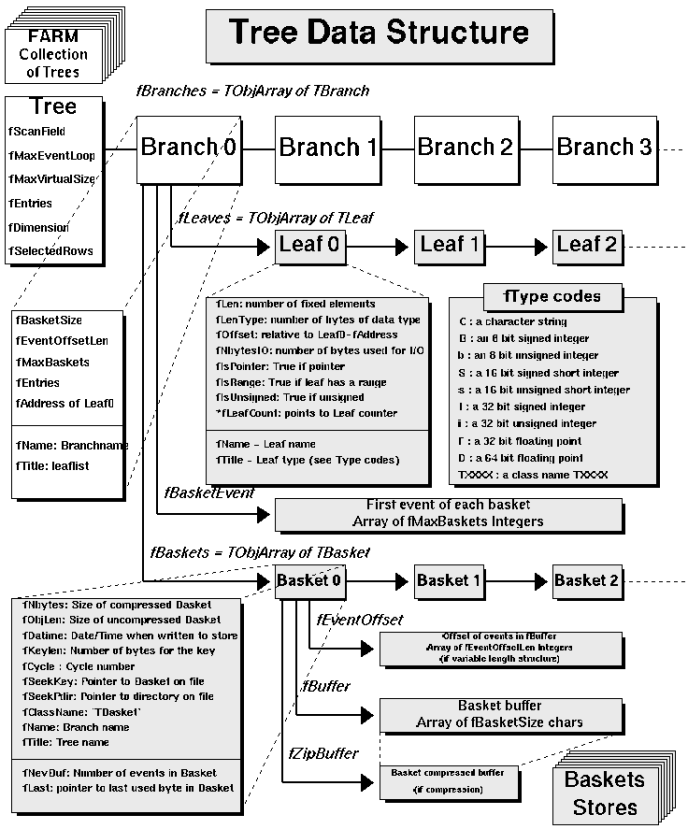

Figure 3.6: The structure of the *TTree* class. Source: `http://root.cern.ch/root/html/gif/tree_layout.gif`

- high fan-in;[41]
- low fan-out;
- portability;
- leanness;[42]
- stratification;[43]
- usage of standard techniques.

He mentions, however, that it is very difficult to combine all those goals. Nevertheless, it is the authors objective to fulfill those criteria during the

[41] Refers to the number of classes that use a given class. Its antonym is fan-out.
[42] Describes designing of the system which has no extra parts.
[43] Means that one can view the system at any single level and get a consistent view.

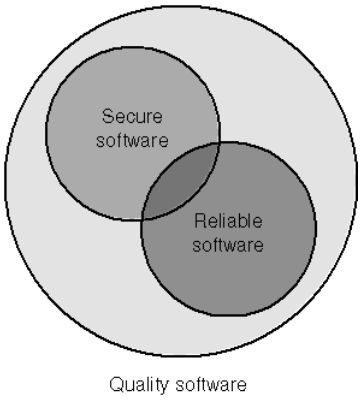

Figure 3.7: Secure software is a subset of quality software and reliable software. Source: [18].

development of the software. The software itself is developed according to object oriented paradigms and design patterns[44] are used as often as possible, as the standard technique which makes it easier to understand its structure from third parties. One of the important properties of the developed source code is the fact that is was implemented using the methods of defensive programming. The usage of those can be first of all considered as the first line of the defence against the consequences of Murphy's law.[45] The second reason is mentioned in the book from Howard et al. – [18] – and although this position focuses mainly on topics which concern the development of a secure software, the creation of the one which is resistant to various hacking techniques[46] is impossible without concerning the issues of quality and reliability (see figure 3.7). An important property of such software is for instance the fact that each input should be tested to prevent a possibility of an undefined behaviour. Howard et al. describe this consideration with one simple and accurate sentence – "all input is evil".

As far the development in C++ is concerned, the source code is written both according to the current standards [33] as well as the best practices described among others in [35], [2] and [25]. Because C++ allows dynamic memory allocation, it must be guaranteed that resources are always properly

[44] For further reference see also chapter 4, [14] and [30].

[45] Murphy's law is a commonly used name for the quotation of an american engineer describing consequences of errors in complex systems – "Whatever can go wrong, will go wrong".

[46] Buffer overrun is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory. This behaviour can be exploited to execute unwanted machine instructions.

released. In the developed software this goal is achieved using the implementation of smart pointers[47] provided by Boost libraries. The developed class collection is therefore free from memory leaks. Moreover all operators responsible for casting of undefined object are assured with the methods of dynamic type verification.

The important parts of the software development are also optimization and refactoring which are among others described in the sources [9] and [19]. The last part consists of so called "pedantic bagatelles" which do not improve significantly performance nor reliability, however, in author's opinion shall be important for any developer which tries to develop his source code as good as possible. The examples of those for C++ and Python are depicted using listings 3.1 and 3.2.

```
// Version 1
for(unsigned int i=0; i<size; ++i)
    if(i == size-1)
        g(i);
    else
        f(i);

// Version 2
for(unsigned int i=0; i<size-1; ++i)
    f(i);
g(size-1);
```

Listing 3.1: An example of the "loop partitioning" in C++.

```
# Version 1
for line in open('example.txt'):
    print line

# Version 2
with open('example.txt') as file:
    for line in file:
        print line
```

Listing 3.2: An example of usage of the Python's *with* statement.

The first example is the so called "loop partitioning" and is considered by plenty of authors as an optimization technique. For all developers with a minimal knowledge about compilation's principles it is, however, obvious

[47] Smart pointer is an abstract data type that simulates a pointer while providing additional features, such as automatic garbage collection in case of the exception or after program's normal execution.

that the first loop cannot be easily optimized by a compiler.[48] In authors opinion this method should be not considered as the optional optimization technique but as the mandatory rule which must be obeyed during the development process.

As far as the second example is concerned, plenty of Python scripts which can be found in the Internet open and read the content of a file in the way presented in first version of the listing 3.2. This can cause problems if such a file must be used "just after" the script's execution is finished. Such operation could not be completed since Python leaves the file opened for the unspecific amount of time. The second version of the source releases all resources associated with the file as soon as the *with* block is executed.

3.4.2 Conventions

Apart from the proper software design it is also important that the source code can be understood from the third parties which are supposed to read and analyze it. To facilitate this procedure, the following conventions have been applied:

- The source code is described using Doxygen comments which allows that its documentation can be generated according to the readers specific needs.
- The Python source code is written according to the official code conventions presented in [28].
- Since the official C++ standard does not define the style in which the source code should be written and the language's author himself discourages only the usage of the Hungarian notation (see [34]), the style of the source code bases on the one described in [15].

[48] Due to the condition which should be checked during each iteration.

Chapter 4

Software engineering oriented decisions

This chapter depicts the structure of the developed software as well as the most important decisions associated with its design and implementation. Due to the obvious volume's limitations of this thesis, it was not possible to describe each detail of the implemented solutions and therefore only those which were in the author's opinion considered as the important ones are hereby portrayed.

4.1 The structure of the software

The software consists of the three main groups of components:

- components developed in C++;
- components developed in Python;
- components serving as wrappers for C++ objects and making it possible to access them from Python – Python bindings.

These components, together with libraries which they utilize, are depicted in the figure 4.1. The applied global convention assumes that all names of C++ components start with "__", whereas names of their wrappers with "_". For instance *__RootTreeConverter* (C++), *_RootTreeConverter* (Python-Bindings) and *RootTreeConverter* (Python).

4.2 Software components

The discussion about the software components starts with the description of elements which are were developed using C++ programming language.

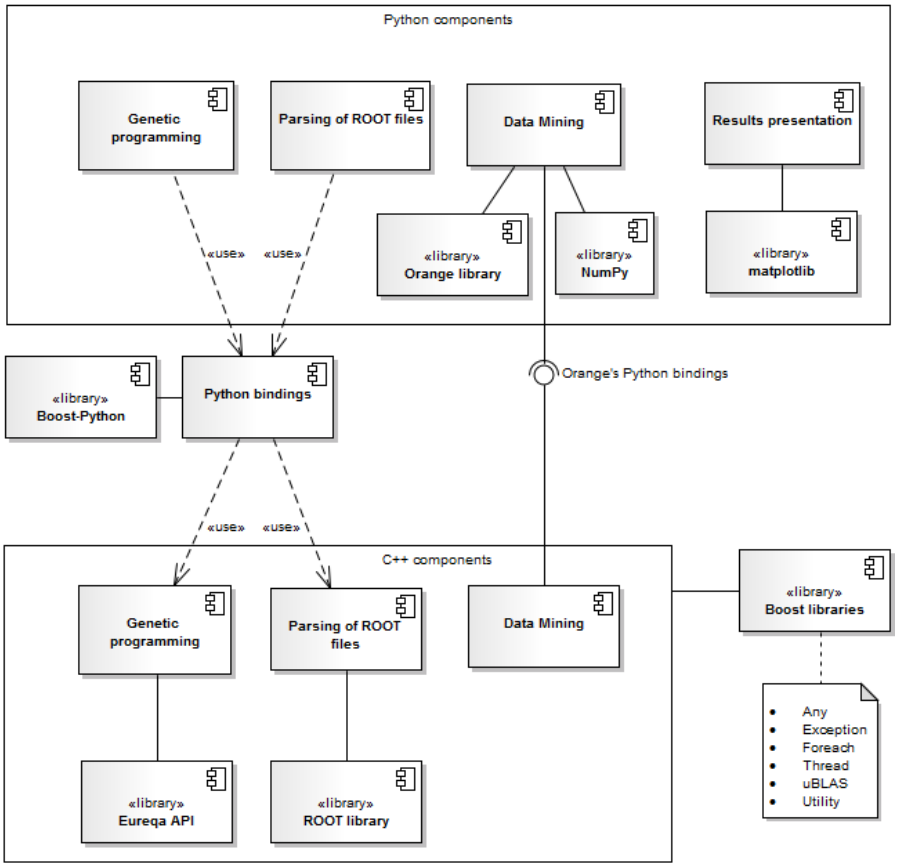

Figure 4.1: An overview of the most important software components.

4.2.1 C++ components

The part of the source code which was written purely in C++ consists of the three main groups of objects:

- the group associated with genetic programming and utilizing the functionality provided by the Eureqa API – see section 3.3.1;
- the group responsible for parsing of ROOT files;
- the group aggregating necessary data structures.

This section starts with the depiction of objects which encapsulate the functionality of the Eureqa API.

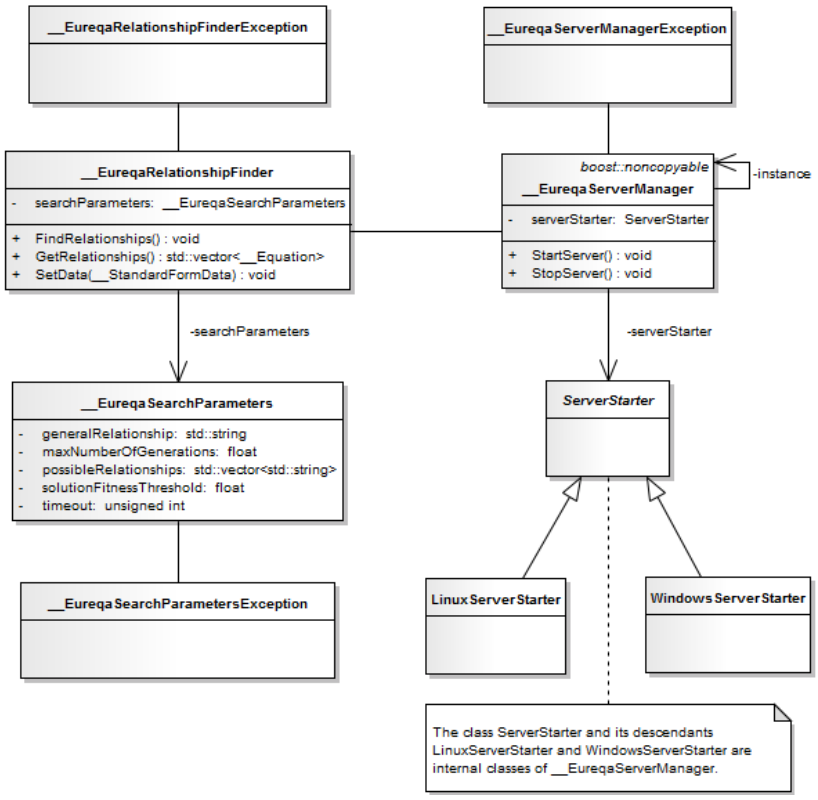

Figure 4.2: The simplified class diagram of the components utilizing the functionality of the Eureqa API.

Eureqa

The group of classes which encapsulates the functionality of the Eureqa API consists of:

- *__EureqaSearchParameters*;
- *__EureqaRelationshipFinder*;
- *__EureqaServerManager*.

The class diagram depicting the most important functionality of the mentioned classes is presented in the figure 4.2. The class *__EureqaSearchParameters* allows setting the relationships search's parameters like for instance:

- maximum searching time (*timeout*);

- maximum number of generations[1] (*maxNumberOfGenerations*);
- the general form of the target relationship which is supposed to be found (*generalRelationship*);
- the list of mathematical functions which should be used during the search (*possibleRelationships*).

The first two items from the above list are examples of terminating parameters and at least one of them must be set.

The *__EureqaSearchParameters* class will be used afterwards by the class *__EureqaRelationshipFinder* to set the parameters used in the relationship search. It can start as soon as the input data is provided using the *SetData* function.

As mentioned in the section 3.3.1, Eureqa API provides the possibility to perform the search using different computation servers which communicate with the application. In the simplest case – when the search is performed only using the single machine – the server must be, however, also started. This complicates the APIs usage, since the integrated version[2] does not provide any functionality to start the server application. It provides only two executables (one for Windows and one for Linux family of operating systems) which have to be run before any APIs searching functions are called. This was achieved using the Singleton class *__EureqaServerManager*. The class contains an instance of the *ServerStarter* class which, according to the used operating system, is initialized as *WindowsServerStarter* or *LinuxServerStarter*.[3] Since there is no requirement that the software should be used on the Windows platform, methods of *WindowsServerStarter* are currently not implemented, however, can be easily added if necessary. The class *LinuxServerStarter* provides the necessary functionality using the techniques of the Linux interprocess communication. The exact description of those methods would be, however, to comprehensive for the purposes of this thesis. Strongly simplified, the method *StartServer* from the class *__EureqaServerManager* starts and controls a new process which is responsible for execution and communication with the server application. After the search is completed the process is terminated and communication pipes are cleared and closed. For further reference see [27].

[1] See section 3.3.1 for further reference.
[2] The software utilizes the version 1.02 of the Eureqa API.
[3] Such desing choice is also congruent with the Liskov substitution principle.

Singleton's implementation in C++

The description of the *__EureqaServerManager* is the proper moment for the short discussion about the advantages and disadvantages of the C++ programming language as well as emphasizing the importance of the correct design choices as far as the development using this language is concerned. This section depicts the Singleton design pattern and its possible implementations.

Since the pattern was officially invented by the "Gang of Four", their version will be analyzed as the first one.[4] The source code (listing 4.1) seems at the very first glance to be perfectly logical, although one can say that it is far more complicated as the pattern's idea itself. This code suffers, how-

```
// Declaration
class Singleton
{
    public:
        static Singleton* Instance();
    protected:
        Singleton();
    private:
        static Singleton* instance;
};

// Definition
Singleton* Singleton::instance = NULL;

Singleton* Singleton::Instance()
{
    if(instance == NULL)
        instance = new Singleton();
    return instance;
}
```

Listing 4.1: The Singleton's C++ implementation provided by the "Gang of Four".

ever, from two serious problems hidden inside of subtle details. First of all one should notice the fact, that before calling the *Instance* function there exists no Singleton object at all. What is even worse there is no destructor implemented therefore the object will be never properly deleted. In this case

[4] The software engineering classic – [14] – is the first publication in which the pattern was described.

this is hopefully not a memory leak[5] but by all means a resource leak, since Singleton's constructor may acquire an unbound set of resources.

The another possibility to implement Singleton bases on the usage of a global variable – see lising 4.2. The solution cannot be, however, considered as a safer one, since the order of initialization of objects is unspecified and compiler dependent. Therefore there exists an unsecured possibility to access an uninitialized object. The solution suffers also from problems concerning multithreading which is, however, not relevant for this discussion. All in all Singleton's implementation in C++ is complicated enough to be considered as the separate chapter in the book [3]. The final version of the pattern (see

```
// Declaration
class Singleton : private boost::noncopyable
{
    public:
        static Singleton instance;

    private:
        Singleton();
        ~Singleton();
};

// Definition
Singleton Singleton::instance;
```

Listing 4.2: Implementation of the Singleton using a global variable.

listing 4.3) which eliminates mentioned problems was used to implement the *__EureqaServerManager* class.

One of the conclusion which can be drawn after this analysis is the fact that patterns are signs of weakness of a programming language, since there should exist transparent mechanism(s) making their usage unnecessary.[6] The implementation of such generic mechanism would be, however, extremely difficult. As far as the C++ programming language is concerned, one can, however, easily notice how treacherous it can be, and how crucial it is to make proper design decisions.

[5] Memory leaks would occur if the data is allocated and all references to it will be lost.

[6] Based on the opinion of the famous Perl programmer Mark Dominus. Source: `http://blog.plover.com/2006/09/11`

```
// Declaration
class Singleton : private boost::noncopyable
{
    public:
        static Singleton* Instance();

    private:
        Singleton();
        ~Singleton();
        // Static Initialization
        static bool initialized;
        // Dynamic Initialization
        static Singleton instance;
};

// Definition
bool Singleton::initialized;
Singleton Singleton::instance;

Singleton::Singleton()
{
    initialized = true;
}

Singleton::~Singleton()
{
    initialized = false;
}

Singleton* Singleton::Instance()
{
    return initialized ? &instance : NULL;
}
```

Listing 4.3: The final implementation of the Singleton design pattern.

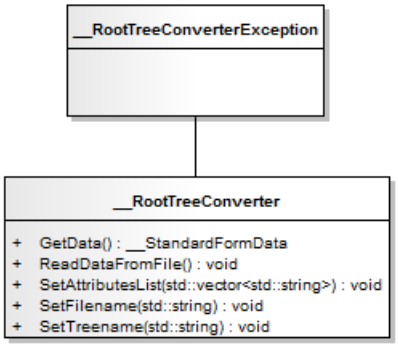

Figure 4.3: The simplified interface of the class responsible for parsing ROOT files.

ROOT library

As previously stated, ROOT is an example of the very "specific" library which does not implement plenty of good practices and standards concerning the modern C++ development – see section 3.3.4. Therefore the implemented software reduces dependencies to this library to the absolute minimum. The single class using it parses ROOT files containing serialized *TTree* structures and converts them to a standard form. Its simplified class diagram is depicted in the figure 4.3.

Data structures

The data structures which are implemented in C++ are necessary to allow the communication between the objects which are also implemented in this language. The *__StandardFormData* class is the object which stores data in a standard form and *__Equation* contains a mathematical dependency which is used to describe relations between attributes. The C++ version is quite simple since it only stores Eureqa APIs result in the form of the *std::string*. In comparison the class *__StandardFormData* is one of the most important structures concerning the whole class library. As depicted in the figure 4.4, the data is stored using the *boost::matrix* structure. Since C++ does not provide any data type which functionality could be compared with an universal "Object"[7], it is also complicated to implement a heterogeneous collection using this language. One of the possibilities would be to utilize a void pointer (*void**) along with methods of dynamic memory allocation. This solution is, however, far from being optimal since it gives the developer no possibility to identify the type of an object which is stored under

[7] The name refers to the *Object* type from C#.

the pointer's address. The better solution would be to implement a class hierarchy and use RTTI mechanism[8] to identify the object's type during the runtime. Although the solution could be considered as an acceptable one, a generic mechanism is already implemented and available as a part of Boost libraries. The so called *boost::any* object (which could be substituted with an arbitrary C++ structure) is stored inside of a *boost::matrix* to provide the mentioned functionality.

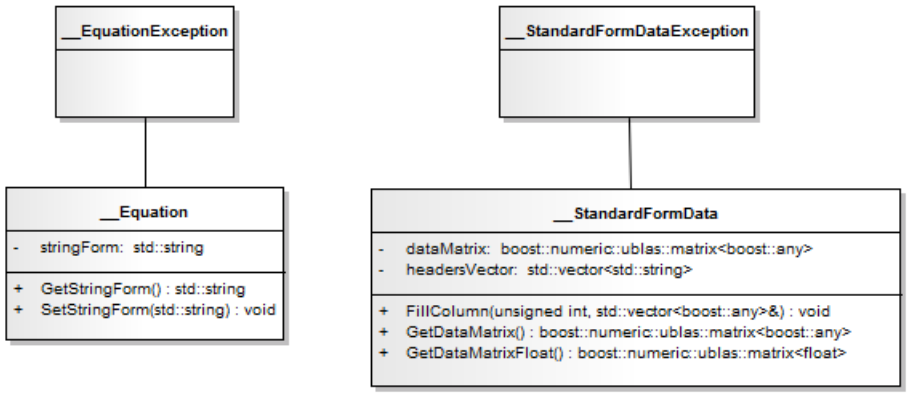

Figure 4.4: The simplified depiction of the applied data structures.

Exception handling

All exceptions which can occur in implemented C++ source code are derived from the *__ApplicationException* structure (the class diagram is presented in the figure 4.5). There exists more reasons for such a design choice and those are hereby presented. First of all this derivation provides the possibility to use only one *catch* block to catch all exceptions thrown by the developed classes. This mechanism is also utilized during the exception's translation from Python into C++ (see section 4.2.3). As one can see in the figure 4.5 all exceptions are derived from the *std::exception* and *boost::exception*. The derivation from the *std::exception* provides the possibility to define the standard *what* function which provides information about the type of an exception object. The derivation from the *boost::exception* facilitates the exception handling process even more and the in-depth description of the advantages of this mechanism is presented in the documentation of Boost libraries – see [7].

The exception's principle is common for plenty of programming languages.

[8] RTTI (abbreviation from "Run-Time Type Identification") is the C++ mechanism responsible for reflection.

There exist a *throw* block – in which an exception object is constructed and filled with data which is useful to identify the error – and the *cath* block in which the object from the *throw* block should be caught and processed. However, usually inside of the *throw* block there is not enough information to provide an accurate description of the error, at the moment an exception should be thrown. For example the function reading a file which "knows" only the pointer to the data object – *FILE** – cannot provide useful information such as file's name. One of the most common solutions to such problem, which is often applied by C++ developers with some kind of Java background, is to utilize so called "exception wrapping".[9] In the author's opinion applying this solution in C++ should be considered as a design error, since it can lead to "object slicing"[10] and loss of information which is needed to identify the type of the problem which caused the exception. The solution implemented by the author is the elegant exception handling provided by on the Boost Python library. It offers the possibility to fill an exception object with the amount of information available in a *throw* block and then supply it at any of the higher levels with the more accurate error's description.

4.2.2 Python bindings

Before a C++ object can be used inside of Python, its functions must accept arguments which are "understandable" for Python's interpreter. For instance it is impossible to use a C++ function accepting "std::vector" as its input parameter directly from Python since such type will not be identified.

At this point a short reference to the Orange library has to be made. As previously stated, the library provides the Python programming interface whereas all operations are performed using C++. All data structures are, however, pure Python ones and therefore must be translated into C++ before any operation can be performed. The implemented class library goes one step further. Not only the operations are performed in C++ but also also "whole" C++ objects are implicitly used inside of Python. One can ask the question why then a C++ data mining library has not been translated into Python using developed Python bindings. The explanation is the fact that there exist no C++ library which could be potentially used. The one

[9] It is a technique of handling exceptions by re-throwing a caught exception after wrapping it inside a new exception.

[10] Object slicing refers to a case in which a superclass instance is assigned its value from a subclass instance and ends in loss of information since the superclass has no place to store them.

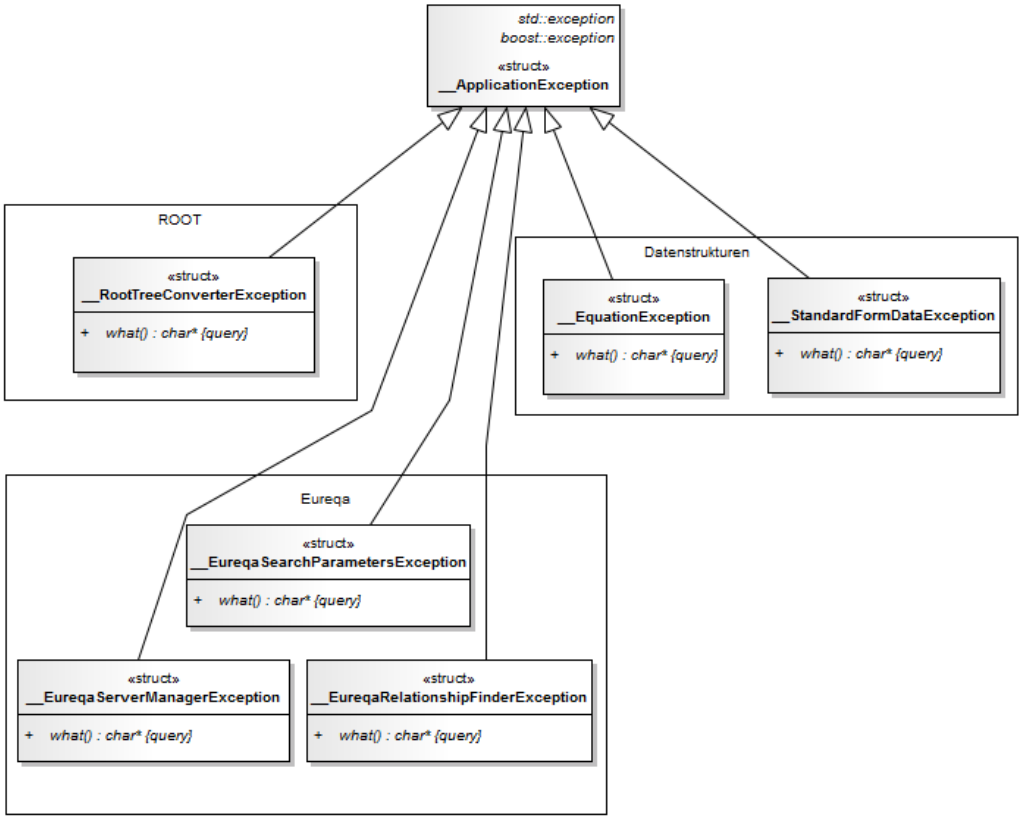

Figure 4.5: The hierarchy of C++ exceptions.

written by the famous SGI company[11] – MLC++ – was actualised for the last time in the year 1997.

The Python class *StandardFormData* will be considered as an example. The class itself is derived from *_StandardFormData* which serves as the binding object and which on the other hand aggregates the pure C++ class *__StandardFormData*. All those relations are depicted in the figure 4.6. The explanation of this design choice is the fact that *_StandardFormData* is not a object of the type of *__StandardFormData* and is therefore connected using the aggregation principle. The other implementation's possibility for this relationship, using the programming language C++, will be the private inheritance which itself is also an art of aggregation. However, according to [6] the aggregation should be used when it is possible and private inheritance only as the necessity. As far as the relationship between classes *StandardFormData* and *_StandardFormData* is concerned, the decision was taken to

[11] `http://www.sgi.com`

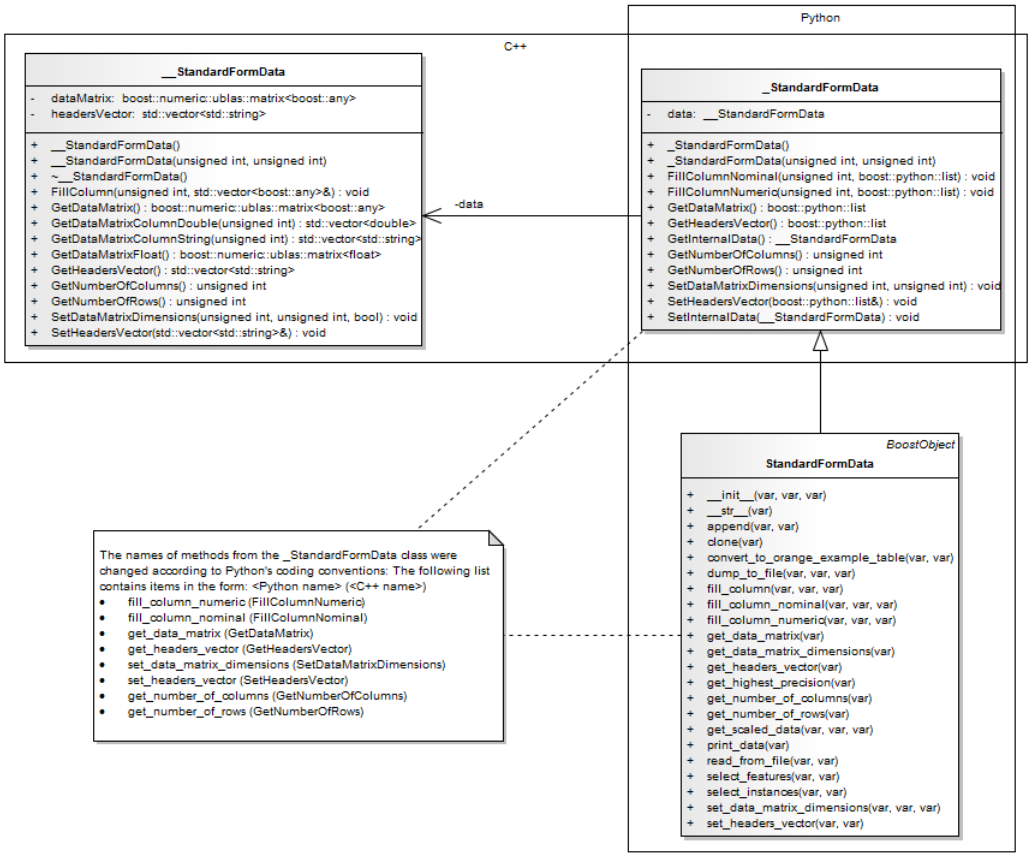

Figure 4.6: An exemplary relation beetwen Python and C++ objects.

develop them using the "is a" principle since *StandardFormData* extends the functionality of *_StandardFormData*.

Concerning the family of "StandardForm" classes, it is also important to notice the differences between their interfaces. Since Python is a dynamically typed language and data in C++ class are stored in the form of a heterogeneous matrix, the class *_StandardFormData* from Python bindings has to explicitly check the consistence of its columns which could consist of nominal or numerical elements.

Exception handling

Apart from objects, also exceptions associated with them have to be translated into Python. As presented in the figure 4.5, all C++ exceptions are derived from *__ApplicationException* which allows to perform a clean and

elegant translation. This translation is partially supported using the *RuntimeError* exception object from the Boost Python library. Exceptions which are generated by the Boost library itself are handled directly in Python. In the figure 4.6 one can notice that *StandardFormData* is derived from the *BoostObject*. This class implements mechanisms responsible for the proper handling of all described exceptions.

4.2.3 Python components

The group of Python components is the biggest one concerning the developed software and therefore only the most important elements are going to be described in this thesis. The main data structures of this group are:

- *StandardFormData*
- *Equation*
- *DMSolution*

As already mentioned the first two components are related to C++ structures: *__StandardFormData* and *__Equation*. The last one is the pure Python one and serves as the container for Orange's data mining models. All those structures can be serialized and deserialized – partially using Python's *pickle* module. The first two can be also stored in the human readable form.

Interpretation of mathematical formulas

One of the software requirements is the possibility to compare a new solution with the currently used mathematical formula (if such exists). To provide such functionality, the class *Equation* implements methods which can evaluate a given mathematical equation for a predefined set of variables.

A naive approach will be to try to implement this feature using some kind of regular expressions' based parser. The most important problem which could not be solved would be a detection of balanced braces[12] which is utterly necessary to evaluate an arbitrary mathematical formula.

[12] The proof bases on the known fact that regular expression are not powerful enough to describe a complex syntax. A regular expression R describes a set of strings of characters denoted as $L(R)$. This is the language of a deterministic state machine which on the basis of its state either accepts or rejects the given input. For the unlimited amount of brackets it is impossible to construct such set of strings and therefore to construct the language $L(R)$. For further reference see [1].

Another approach would be constructing a grammar which could interpret the "language" of mathematical formulas and then develop a compiler which could translate it to a proper set of machine instructions. It would be the only possible solution if one of compiled system programming languages would be solely used inside of the developed software. Since Python itself is a interpreted language, it is possible to call its own interpreter inside of a script and evaluate a mathematical formula using this mechanism. The semantic of the provided expression is checked before then and the statement is converted according to the rules of Python's syntax. Listing 4.4 show the

```
# Input
cos(x) + gauss(y) + 3.14

# Interpretable version
math.cos(x) + MathExtension.gauss(y) + 3.14
```

Listing 4.4: A conversion example of the mathematical expression into the Python's instruction.

example of such conversion (*MathExtension* is one of modules used in the written software containing functions which are not included in the standard math package *math* nor in *NumPy* – for example the *gauss* function which is commonly a part of solutions provided by the Eureqa API). Referring to the mathematical solutions provided by the genetic programming methods, one has to consider the problem of the division by 0. It happens quite often that inside of the test set which should evaluate the solution's performance there exists a group of values for which such a division theoretically have to be performed. As far as the genetic programming is concerned, the assumption is taken that for the sake of calculations, the fitness function should consider 1 as the result of such division. The same approach is used in the developed software.

Safety mechanisms

As previously stated each user can create his own scripts using the developed Python's class library. Although the classes documentation provides all necessary information to do this, two extra mechanisms were implemented to localize and eliminate eventual problems as fast as possible. The first one is logging which stores each function call inside of a special log file, so that the sequence of actions which led to an error could be analyzed afterwards. The second one, which serves as the support for the first mechanism, is the constant monitoring of operations performed by the selected objects and writing

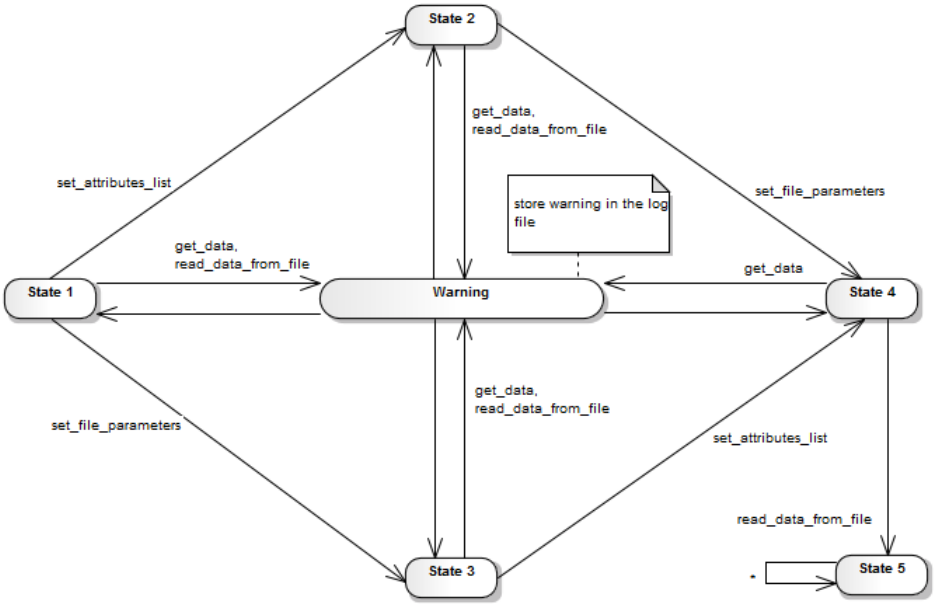

Figure 4.7: The state machine representing the activities sequence allowed for the class *RootTreeConverter*.

a warning to the a file if the operation which is going to be executed is not allowed at the given moment of time. This functionality is implemented by predefining the sequence of activities which can be performed by the selected class and store it in the form of a finite state machine. For instance the class *RootTreeConverter* parses ROOT files and stores (due to the time complexity of the read operation) the data in the operating memory. This data can be accessed afterwards using the function *get_ data*. The function call just after creating an instance of the class can be, however, treated as a potential error, since an empty structure will be returned. In this case the mechanism produces a warning which is stored inside of a log file. The state machine used for this purposes is presented in the figure 4.7.

The class which provides the above functionality was developed according to the State design pattern. The idea to save function calls bases partially on the Command design pattern. For further reference see [14] and [30].

Singleton in Python

The section 4.2.1 presents the implementation of the Singleton design pattern using C++ programming language. The implementation in Python is considerably less complicated as the one in C++, however, in spite of that

there exist at least two different possibilities to perform it. The "classical" one bases on the approach suggested by the "Gang of Four" and described in details in [30]. The alternative, which is known under the name of "Borg", simulates the Singleton's principle by creating a group of objects which share the same state. It is impossible to unambiguously determine which solution is the better one, however, the implementation of "Borg" (listing 4.5) needs considerable less of the source code – which makes it also easier to follow by third parties – and will be therefore used to develop the class responsible for the logging mechanism.

```
class Borg:
    __shared_state = {}
    def __init__(self):
        self.__dict__ = self.__shared_state
```

Listing 4.5: Python's "Borg" implementation.

4.3 Short résumé

As previously written, it would be unreasonably to present the details of the implemented software, due to the limited capacity of this thesis (depicting a little part of a full class diagram at the A4 page is already problematic). Therefore the discussion concerning the software development is hereby finished. For further reference see the source code provided as the attachement of the electronic version of this thesis.

Chapter 5

Presentation of the results

This chapter presents the two possibilities to utilize the developed class library – applying the implemented main application as well as creating custom Python scripts. Moreover the small benchmark concerning the performance of the currently applied mathematical formula and the data mining models provided by the software will be presented. The discussion starts with the demonstration of the possibilities offered by the implemented solutions.

5.1 Software's applications

The simplest method to use the developed software is to start the main application which is implemented in Python. It provides the functionality needed to load data from a ROOT file, select instances and perform a cross validation for one of two data mining methods: SVM or genetic programming. The simplified sequence diagram is presented in the figure 5.1.

The second possibility is to use the implemented class library to develop custom Python scripts. This method provides the higher degree of freedom and since Python is a very intuitive language, it can be used by almost each scientist with the minimal experience in the domain of software development. One can present now an arbitrary amount of exemplary scripts, however, for the purposes of this chapter only one of them has been chosen and hereby described. It shows the parametrized usage of the genetic programming method. The data is loaded from the ROOT file, instances are selected and the search parameters are set in the way that the search termi-

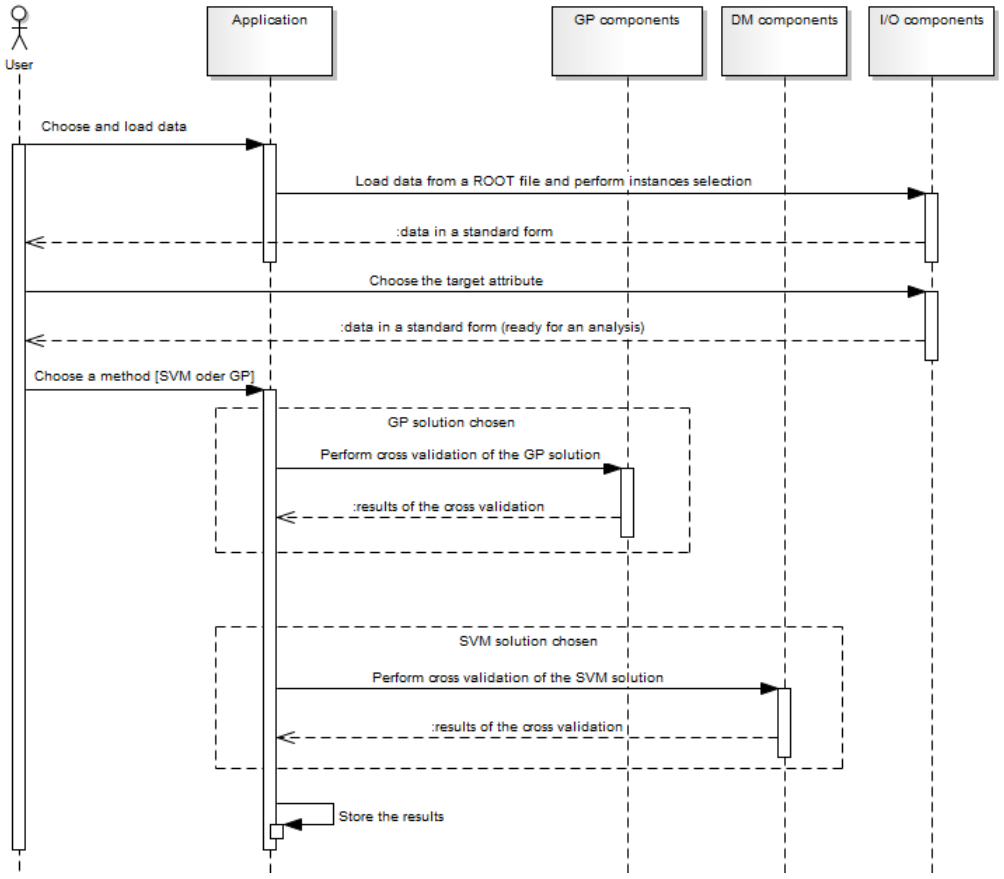

Figure 5.1: The simplified sequence diagram for the main application.

nates either after one hour, or if the solution's fitness is less than 0.1.[1] This example is depicted in the listing 5.1.[2]

5.2 Results

The software was among others already used to establish and compare the performance of the data mining solution and various mathematical formulas currently used in the experiment. The hereby presented results base on the equation 5.1 used to calculate radio pulse height on the basis of estimated

[1] The fitness of the solution is in this case the value which is the combination of the result provided by the fitness function (see section 2.3) and the solution's complexity. The fitness function used by the software aims to minimize the mean absolute error.

[2] The *import* statements are purposely omitted.

```
# Converter for ROOT files
converter = RootTreeConverter()
converter.set_file_parameters('file.root', 'T')
att_list = ['CCheight_EW', 'geomag_angle', 'Ze',
            'latMeanDistCC_EW', 'lgE']
converter.set_attributes_list(att_list)
converter.read_data_from_file()

# Data in a standard form
data = converter.get_data()
data = data.select_instances('CCheight_EW < 1e-04 and lgE > 8')

# Search parameters
search_parameters = SearchParameters()
search_parameters.set_timeout(1, 'h')
search_parameters.set_solution_fitness_threshold(0.1)
dependent_att = att_list
dependent_att.remove('lgE')
search_parameters.set_general_relationship(dependent_att, 'lgE')

# Current solution
current_solution = Equation()
string_form = 'log10(pow((CCheight_EW * pow(10, 6) / 11
              * (1.16 - cos(geomag_angle)) * cos(Ze)
              * exp((-latMeanDistCC_EW / 236))), 1/0.95)) + 8'
current_solution.set_string_form(string_form)

# Cross validation of the GP solution
gp_tester = EureqaTester()
gp_tester.test_performance(search_parameters, data, 10,
                           current_solution)
```

Listing 5.1: An example depicting the usage of the developed Python's class library.

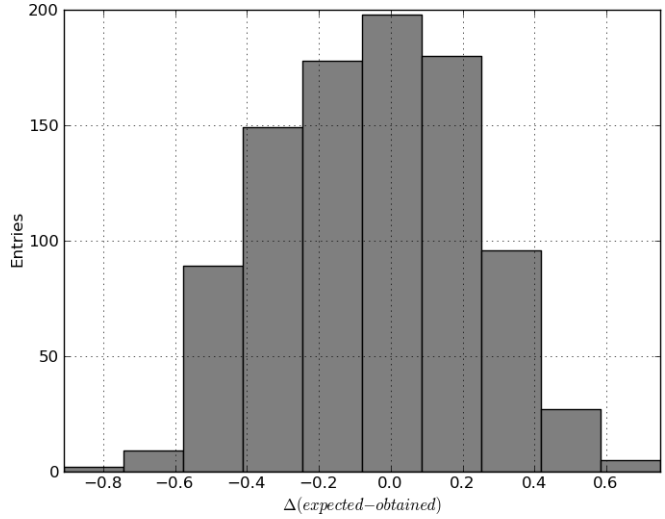

Figure 5.2: Error's histogram of the currently applied mathematical formula.

primary energy, which is described in details in [17].

$$\epsilon_{est} = 11 \cdot (1.16 - cos(\alpha)) \cdot cos(\theta) \cdot e^{\frac{-R_{SA}}{236}} \cdot \frac{E_p}{10^{17}} \quad \left[\frac{\mu\ V}{m\ MHz}\right] \tag{5.1}$$

The parameter α stands for the geomagnetic angle, θ the zenith angle and R_{SA} the mean distance of the antennas to the shower axis. The formula was transformed to calculate the primary energy and the results were compared with the reference values from KASCADE experiment.[3] The error's histogram is presented in the figure 5.2.

The same data has been also used to perform the cross validation for both SVM and the method of genetic programming. The performance results are depicted in the figures 5.3 and 5.4. Figures 5.2 and 5.4 were obtained using the source code presented in the listing 5.1.

Results interpretation

It is not possible for the author to assess the method only on the basis of its performance. Such evaluation must be performed by scientist during their everyday work analyzing the experiment's data. As far as the chosen data mining solutions are concerned, one can, however, unambiguously state

[3] About 1,000 instances were used for the testing purposes.

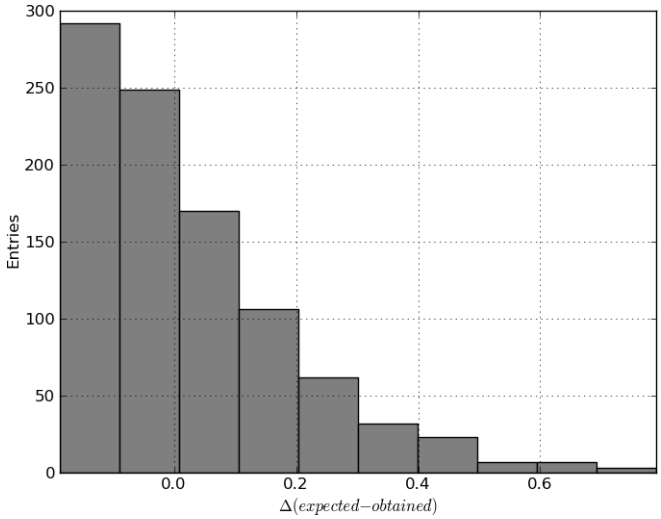

Figure 5.3: Error's histogram of the SVM solution.

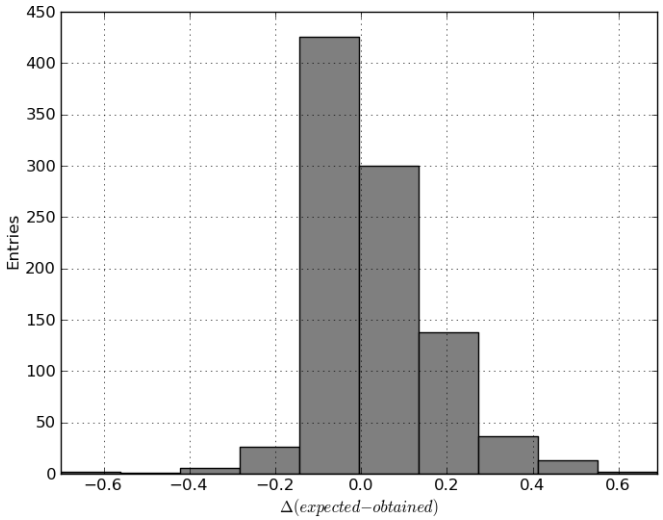

Figure 5.4: Error's histogram of the GP solution.

that provided results are on the basis of their accuracy at least comparable[4] with the results obtained using the current formula. The major difference is associated with the way in which they were obtained. The usage of SVM and GP methods involved only the development of the simple script, allowing the tedious procedure of finding the fittest solution to be substituted by the application of the data mining algorithms.

[4] In the case of the GP solution the results are even better concerning the values of the mean squared error and standard deviation.

Chapter 6

Conclusions

The aim of this thesis was to integrate the data mining methods in the LOPES experiment. First of all selected techniques were tested using the experiment's data and afterwards the most promising ones were chosen and integrated in the implemented software. The developed main application, which can be used without any additional knowledge of data mining itself, provides an attractive alternative for the current analysis approach. Apart from that arbitrary scripts can be implemented using the developed Python's class library.

The topic concerning applications of data mining methods in the domain of physics is not the author's genuine idea, however, it does not base on assumptions commonly found in plethora of online sources and publications, which state that neural networks can be considered as the universal solution for all mathematical and physical problems. This thesis is also not an example of a paper which tries to convince its audience that the method of decision trees can be successfully applied to provide a set of understandable rules for data sets containing dozens of attributes and thousands of instances.[1]

The defined software requirements were fulfilled, therefore author expects that the implemented solutions will be used by KITs scientists and at least implicitly contribute to the development in the domain of astrophysics. The tailored class library represents the highest coding standards, solve complex software engineering problems and is optimally designed concerning even "subtle"[2] details like for instance the proper implementation of the Singleton

[1] The method is used in the developed software during the data preparation phase as a tool to reduce the amount of input attributes.

[2] The adjective subtle is used to express the advanced level of the discussed topics.

design pattern. The software design, implementation as well as solving technical problems were definitely very interesting challenges – like for instance the issues directly associated with the secure interprocess communication under Linux environment or compilation and linking issues concerning deployment of the software on 64 bit platforms. The cooperation with KIT was extremely fruitful and the interdisciplinary nature of the thesis allowed the author to expand his knowledge not only in the domains of data mining and software engineering, but also in astrophysics. The project was definitely a valuable lesson for the author regarding the role of precision in design and implementation of complex software solutions.

The issue of integration of data mining methods which are capable to perform the semi-automated analysis of the astrophysical data was also presented during the conference in Słok (23-25th June 2010) and the associated article is going to be submitted for the XVII International Conference on Information Technology Systems (3-5th November 2010).[3] The part of the software which servers as the Python's facade for the C++ Eureqa API is planned to be extended and hopefully available as the internal part of the Eureqa software in the near future.[4]

[3] For further reference see [13] and [12].

[4] The issue concerns the author's response to the advertisement which can be found at the official Eureqa's website concerning the API's translation to Python.

Acknowledgments

This thesis would not have been possible to realize without the cooperation with the Institute for Nuclear Physics at Karlsruhe Institute of Technology, therefore I would like to express my gratitude to Ph.D. Andreas Haungs and Frank Schröder for the provided support. I am also heartily grateful to my supervisor – Ph.D. Andrzej Romanowski – for his constructive remarks and guidance. My sincere thanks goes out to Professor Dominik Sankowski – the Head of the Computer Engineering Department – for reviewing this dissertation. Lastly, I offer my regards to Professor Janusz Zabierowski from the Andrzej Sołtan Institute for Nuclear Studies who enabled me the cooperation with KIT.

This paper is based on my master's thesis prepared at Brandenburg University of Applied Sciences – [11], and defended there on August 2010.[5] It was later on modified to comply with relevant standards, following the instructions from the supervisors, at Technical University of Łódź and hereby submitted in this form.

[5] The procedure was formalized by the means of the Learning Agreement signed both by me and the Technical University of Łódź.

Bibliography

[1] A. V. Aho et al. *Compiler. Principles, Techniques and Tools, Second Edition*. Addison-Wesley, 2006.

[2] A. Alexandrescu. *C++ Coding Standards. 101 Rules, Guidelines, and Best Practices*. Addison-Wesley, 2004.

[3] A. Alexandrescu. *Modern C++ Design. Generic Programming and Design Patterns Applied*. Addison-Wesley, 2001.

[4] M. J. A. Berry and G. S. Linoff. *Data mining techniques. For marketing, Sales and Customer Relationship Management, Second Edition*. Wiley, 2004.

[5] M. J. A. Berry and G. S. Linoff. *Mastering Data Mining. The Art and Science of Customer Relationship Management*. Wiley, 1999.

[6] M. Cline. *C++ FAQ Lite*. 2010. URL: `http://www.parashift.com/c++-faq-lite/`.

[7] B. Daves, D. Abrahms, and R. Rivers. *Boost Library Documentation*. 2010. URL: `http://www.boost.org/doc/libs/`.

[8] H. Falcke et al. "Detection and imaging of atmospheric radio flashes from cosmic ray air showers". In: *Nature* 435 (2005). Nature Publishing Group, pages 313–316.

[9] M. Fowler et al. *Refactoring. Improving the Design of Existing Code*. Addison-Wesley, 1999.

[10] M. Franc. "Data Mining in industriellen Prozessen". Coursework (not officialy publicated). Brandenburg University of Applied Sciences, 2010.

[11] M. Franc. "Integration von Data-Mining-Methoden zur Analyse der Daten aus dem astrophysikalischen Experiment LOPES". Master's thesis. Brandenburg University of Applied Sciences, 2010.

[12] M. Franc, A. Romanowski, and K. Grudzień. "Examples of particular Data Mining techniques and software for automated analysis of scientific data". In: Łódź, Poland 2010.

[13] M. Franc, A. Romanowski, and K. Grudzień. "Przegląd technik eksplopracji danych i oprogramowania stosowanych do automatycznej analizy danych naukowych". In: Słok, Poland 2010.

[14] E. Gamma et al. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[15] T. Hoff. *C++ Coding Standard*. 2008. URL: `http://www.possibility.com/Cpp/CppCodingStandard.html`.

[16] A. Horneffer. "Measuring Radio Emission from Cosmic Ray Air Showers with a Digital Radio Telescope". PhD thesis. University of Bonn, 2006.

[17] A. Horneffer et al. "Primary Particle Energy Calibration of the EAS Radio Pulse Height". In: Merida, Mexiko 2007.

[18] M. Howard and D. LeBlanc. *Writing Secure Code. Second Edition*. Microsoft Press, 2002.

[19] P. Isensee. *C++ Optimization Strategies and Techniques*. 1998. URL: `http://www.tantalon.com/pete/cppopt/main.htm`.

[20] J. V. Jelley et al. "Radio Pulses from Extensive Cosmic-Ray Air Showers". In: *Nature* 205 (1965). Nature Publishing Group, pages 327–328.

[21] H. Jiawei and K. Micheline. *Data Mining. Concepts and Techniques*. Academic Press, 2001.

[22] S. S. Keerthi and C.-J. Lin. "Asymptotic behaviors of support vector machines with Gaussian kernel". In: *Neural Computation* 15 (2003). MIT Press, pages 1667–1689.

[23] J. R. Koza. *The home page of Genetic Programming Inc.* 2007. URL: `http://www.genetic-programming.com`.

[24] S. McConnell. *Code Complete. Second Edition*. Microsoft Press, 2004.

[25] S. Meyers. *Effective C++. Second Edition*. Addison-Wesley, 1998.

[26] J. K. Ousterhout. "Scripting: Higher-Level Programming for the 21st Century". In: *Computer* 31 (1998). IEEE Computer Society, pages 22–30.

[27] M. J. Rochkind. *Advanced UNIX Programming. Second Edition*. Addison-Wesley, 2004.

[28] G. van Rossum and B. Warsaw. *Style Guide for Python Code*. 2009. URL: http://www.python.org/dev/peps/pep-0008.

[29] W. S. Sarle. *Neural Network FAQ*. 1997. URL: ftp://ftp.sas.com/pub/neural/FAQ.html.

[30] V. Savikko. *Design Patterns in Python*. 1997. URL: http://www.python.org/workshops/1997-10/proceedings/savikko.html.

[31] M. Schmidt and H. Lipson. "Distilling Free-Form Natural Laws from Experimental Data". In: *Science* 324 (2009). American Association for the Advancement of Science (AAAS), pages 81–85.

[32] B. Schäling. *The Boost C++ Libraries*. 2010. URL: http://en.highscore.de/cpp/boost.

[33] International Organization for Standardization and International Electrotechnical Commission. *ISO/IEC 14882:2003. Programming languages – C++*. 2003. URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=38110.

[34] B. Stroustrup. *Bjarne Stroustrup's C++ Style and Technique FAQ*. 2009. URL: http://www2.research.att.com/~bs/bs_faq2.html.

[35] B. Stroustrup. *The C++ Programming Language. Third Edition*. Addison-Wesley, 2001.

[36] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[37] K. Weicker. *Evolutionäre Algorithmen*. Teubner, 2002.

[38] M. Weigend. *Python GE-PACKT. Schneller Zugriff auf Module, Klassen und Funktionen. Tkinter, Datenbanken und Internet-Programmierung. Für die Versionen Python 3.0 und 2.x*. Mitp-Verlag, 2008.

[39] S. M. Weiss and N. Indurkhya. *Data Mining. A practical guide*. Morgan Kaufmann Publishers, 1998.

[40] I. H. Witten and E. Frank. *Data Mining. Practical Machine Learning Tools and Techniques, Second Edition*. Elsevier, 2005.

[41] I. Zelinka. *Symbolic regression - an overview*. 2004. URL: http://www.mafy.lut.fi/EcmiNL/older/ecmi35/node70.html.