Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

Inductive Preprocessing Technology

by

Miroslav Čepek

A thesis submitted to
the Faculty of Electrical Engineering, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of Doctor.

PhD programme: Electrical Engineering and Information Technology
Specialization: Computer Science and Engineering

February 2013

Thesis Supervisor:

Miroslav Šnorek
Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo nám. 13
121 35 Praha 2
Czech Republic

Abstract

This thesis presents a novel approach to automate a part of the data preprocessing for the knowledge discovery. The approach is called **Inductive Preprocessing Technique**. The aim of the Inductive Preprocessing Technique is automatic selection of preprocessing methods (like non linear transformations, normalisations, etc.) and their ordering to maximise the accuracy of a data mining model. The thesis presents its application to classification models but it is possible to extend the Inductive Preprocessing Technique to regression problems in the future as well.

The data preprocessing is equally as important as a selection of the correct classifier, but in many cases the preprocessing part of the knowledge discovery process is neglected. There were some efforts to assist the data mining experts with this task. They are mostly based on ontologies, similarity to other datasets and hard-coded rules. In contrast to these approaches the Inductive Preprocessing Technique is **data-driven** and it is based on **idea of the inductive modelling** and the **optimisation approach**. No prior knowledge about the data is needed. This approach was not tested yet in this field before.

The thesis presents the cornerstones of the Inductive Preprocessing Technology – the *search method*, the *parameter value optimisation* and the *classifier*. The *search methods* automatically select data transformations, the *parameter value optimisation* adjusts parameters of preprocessing methods and the *classifier* provides the model.

In this work I have tested several methods for the search for sequences of preprocessing methods and for parameter value optimisation and in the end I have decided to use the genetic algorithm based search for sequences and the random mutation for the parameter values optimisation. The testing on real world datasets shows that the Inductive Preprocessing Method typically improves the accuracy of the classifier by **about 5% to 10%** by the transforming the data. The thesis also present a way how to speed up the Inductive Preprocessing Algorithm using meta data and information about the past datasets.

The Inductive Preprocessing Method is a great help to the data mining experts who can concentrate on other parts of the knowledge discovery process.

Keywords:

Inductive Modelling, Data Preprocessing for Knowledge Discovery, Genetic Algorithms, Meta-Learning, Machine Learning, Optimisation.

Acknowledgements

I want to thank to many students who has helped me in the evolution and implementation of the Inductive Preprocessing Technology: Dušan Zámečník, Miloslav Pavlíček, Helena Krkošková, Jitka Závadová, Aleš Pinlý, Miloš Koválčik, Kamil Kopejtko, Jan Jakeš and Petr Galík.

I also want to express my gratitude to my supervisor Miroslav Šnorek and also to Pavel Kordík for many comments and suggestions. And last but not least I want to thank to my girlfriend Marika Hrdinová and my parents for their support.

In this thesis I have used several publicly available datasets and I want to acknowledge their the authors:

- The *Brest Cancer Wisconsin* dataset was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg.
- The *Teet Age* dataset was provided by the Faculty of Science, Charles University in Prague in cooperation with the General University Hospital on Karlovo Namesti, the Motol University Hospital and the Fakultní Nemocnice Královské Vinohrady.
- The *Steel Faults* dataset was provided by Semeion, Research Center of Sciences of Communication, Via Sersale 117, 00128, Rome, Italy.
- The *Indian Liver Patient* dataset was collected and provided by following institutions: Hungarian Institute of Cardiology; University Hospital, Zurich, Switzerland; University Hospital, Basel, Switzerland; V.A. Medical Center, Long Beach and Cleveland Clinic Foundation.

Contents

List of Figures

1 Introduction and Problem Statement

There is a constant desire in data mining and machine learning community to improve accuracy of classifiers, like Decision Trees or Support Vector Machines. For many years new classifiers were developed and old ones improved. But to create and train a classifier one needs not only the training method but also a training dataset. These two cornerstones are both essential and it is not possible to create good model with inappropriate classification method or bad training dataset. There is a well known principle called *Garbage in, garbage out.* It means that if the training dataset carry little or confused information, even the best classification method fails to create any meaningful model. Even if the information is present in the data it may be presented in such way which the classifier is not able to take advantage of. To transform the data into a form suitable for classification model and its training method is a task of data preparation or data preprocessing. The data preprocessing is the most time-consuming and for above reasons probably the most important step in the data mining process. The data preprocessing mainly consists of data acquisition, feature construction and data transformation. At present the data mining expert has to preprocess data manually and based on his or hers experience and experiments.

During the past years data mining experts have created recommendations for different typical data mining tasks in different environments. For example, if one is going to predict churn in telco[1], the data miner should include in his/hers training set information like customer's spending over the past six months, type of contract, phone usage, special offers and so on [1, 2]. Similar recommendations were established also in other araes [3]. The data required to generate the recommended attributes are typically distributed all over a data warehouse in different tables and databases. So the data miner has to load and merge information from different places in the warehouse, extract the attributes and form the data matrix for the model training. This is a time-consuming work. The Jermyn, et al. in [4] estimate that the data preparation phase of the data mining process consumes about 80% of the time needed to finish the data mining project. Other authors do not estimate the same value, but it is still the longest part of the data mining process [5].

The data preparation can be basically divided into three main stages – the data acquisition, feature construction and data transformation. In the data acquisition phase the data are extracted from a warehouse. In the feature construction phase the data miner extracts information from acquired data. After this the data miner ends up with a set of input attributes and output variables. This matrix (or dataset) is generally ready to be used for training and validation of the data mining model. Usually this matrix has many problems – in the source systems there can be missing values, different ranges or strange distribution of values in attributes, data could be discretised, non-linearly transformed and so on. It is not always clear which transformations would help the model and which will not. The data miner then have to rely on his/hers experience and has to experiment. In a case when the classifier does not have satisfactory accuracy the data miner has to go back and select another transformations and/or set different parameters and create model again. The data miner has to repeat this process until he/she is satisfied with the result. But it is not certain that is it the best result the data miner can get from the data. I see a big room for

[1]If customer is going to leave his mobile phone provider.

automation and aid the data miner in this step.

There are several papers showing that the correct data preprocessing is important for the accuracy of the classifier and that it is difficult to find the correct preprocessing methods and their order. The first example presented in [6] tests different types of data normalisation for the Support Vector Machines classifier. Using the correct normalisation technique they can improve accuracy of the classifier by about 2%. The other very recent example in [7] aims to find the best practise sequence (or pipeline in their terminology) of preprocessing methods in the field of Brain Computer Interface. The authors have manually tested different signal preprocessing methods and their ordering. The problem is suitable for IPT and their approach is very similar to IPT, only done manually. Their problem could be automatically solved by IPT.

1.1 Inductive Approach

In contrast to other approaches the proposed Inductive Preprocessing Technology is **data-driven** and it is based on the **idea of inductive modelling** and the **optimisation approach**. It uses them in the field of the Data preprocessing where these approaches were not used before. In the inductive modelling the structure of the model is not predefined. The model starts from a minimal form without any knowledge of the data[2]. During the training the algorithm examines the data and improves the model's structure until it is complex enough to fit the data. Similarly to inductive modelling, the IPT approach is data-driven and the sequence of the preprocessing methods has no predefined structure. The sequnce starts from a random form **without any knowledge** about the data. As IPT progresses, it gets more information about the data and by adding, removing and modifying the preprocessing methods. It is searching for the simplest sequence of the preprocessing methods that achieves the highest accuracy of the classifier. IPT tries to find sequences only as complex (and contains only the preprocessing methods) as it is needed to preprocess the dataset correctly and to maximise the accuracy (fitness) of the classifier. The optimisation approach is used to find the correct modification of the sequences of preprocessing methods to increase the accuracy of the classifier. The modifications are to add, remove or modify the preprocessing method.

1.2 Approaches to Support Data Preprocessing

As I stated above, my approach is based on optimisation and inductive approaches. But it is not the only way to aid data miners. In the past there were also other approaches to facilitate data preprocessing.

One possible approach is the Intelligent Discovery Assistant (IDA)[11, 12]. Their approach is based on the ontology. The IDA is a framework for ontology-driven process-oriented assistants for the Knowledge Discovery in Databases (KDD) [13]. The assistant concerns about the whole KDD process not just the preprocessing. The IDA helps a user to create a valid KDD process composes of several blocks. Each block contains *pre-conditions*, *post-conditions* and *heuristic*

[2]The examples of this approach are GMHD[8] modelling method or Decision Tree Induction[9, 10]

indicators [14]. The *pre-conditions* indicate meta-features or conditions, data must fulfill before a block is applicable. For example input data may contain missing values or must be nominal values, etc. The *post-conditions* describes which meta-features data posses after this block. For example data are normalised or in One-of-N code. With *pre-conditions* and *post-conditions* the IDA indicates to user which block he/she may use and what operations are applicable to the given data. *Heuristic indicators* indicates influence of block on the KDD process. How the block affects speed, accuracy, comprehensibility of model, etc... Data reduction increases speed, pruning decreases speed but increases comprehensibility of model (examples are taken from [14]). Definition of *heuristic indicators* allows the IDA to search for the KDD sequence which fits the best to the user defined conditions.

Another possible approach represents the MiningMart project [15, 16, 17, 18]. It design a sequence of data transformation and other blocks from the database of existing sequences. The MiningMart tries to reuse successful preprocessing sequences from the past. It collects information about both data and preprocessing sequences, in the MiningMart terminology a case. After successful preprocessing user can add a case to the database. When user faces a new problem he/she may search through the database of cases and seeks for the most similar to the current problem [19].

The MiningMart leaves the building of a preprocessing sequence on a user and. But successful case is stored in database with meta-data of original dataset. When a new dataset is presented to the MiningMart, it calculates metadata of the dataset and compares them to metadata of cases stored in database and matching cases are offered to user [14].

The extension or continuation of the MiningMart Project is the myExperiment.org Project. This project allows researches to share workflows for some task or even data. The main focus is on the processing and transforming bioinformatics data but the approach in general is applicable also to the data preprocessing for data mining. The myExperiment.org does not automate the workflow creation, but researcher can search for workflow for similar or even the same data, use it and can share it with others [20, 21].

The most recent project in this field is the E-LICO project[22, 23]. This project incorporates and develops many past projects like myExperiment.org and Intelligent Discovery Assistant.

The CITRUS project uses object oriented schema to model relations between a database and models. But its main aim is to guide and support to the data miner, not the full automation. [24].

The completely different approach to automated data preprocessing is implemented in the *IBM SPSS Modeller*'s Automatic Data Preparation Node. The node is only one of the nodes in a workflow and it does not help data miner with construction of the workflow. It contains a predefined set of if-then rules and transforms data according to them. The rules were found by data mining experts. The example of a rule is *"For each continuous variable, if the number of distinct values is less than a threshold (default is 5), then it is recast as an ordinal variable."* The node handles ouliers, missing data or normalisations. For more details see [25].

The approach similar to the Modeler's is implemented in the *Oracle Data Mining Option.* It has also predefined set of rules telling when to apply data transformation [26].

1.3 Goals of the Thesis

The main goal of the thesis is to design the algorithm, called the Inductive Preprocessing Technology (IPT), that is able to improve accuracy of a classifier by transforming the data. In the past several other approaches were developed. These approaches are based on ontology, similarity to other datasets and workflows and the hard-coded rules. In contrast to these approaches IPT is based on idea of inductive modelling and on data-driven and optimisation approaches. My goal is to provide automatic way to transform the data into form suitable for given classifier.

The particular tasks:

- investigate if the data transformations have really an influence on the accuracy of the model trained using preprocessed data,
- design search methods for the best preprocessing methods and their order and to test them on artificial datasets,
- investigate if the parameters of the preprocessing methods have influence on the accuracy of the model.
- design and test the parameter optimisation algorithms,
- test the Inductive Preprocessing Technology with a selected search method and parameter optimisation method on publicly available real world datasets,
- test if it is possible to improve speed of the Inductive Preprocessing Technology by providing the search method with better starting points.

1.4 Organization of the thesis

The thesis consists of two parts – the theoretical and the experimental. The theoretical part briefly introduces machine learning and data mining concepts, including identification of data preparation methods belonging to the data transformation part of the data preprocessing, short description of used data transformation methods and used classifiers (Chapter 2). The Chapter 3 describes high level overview of the Inductive Preprocessing Technology. The Chapter 4 describes fundamental terms of the *fitness* and the *subsequence*. The Chapter 5 continues with a brief introduction to optimisation. The same chapter also describes of the search methods for the sequences of preprocessing methods and the parameter values optimisations. In the last chapter of the theoretical part (Chapter 6) I describe the use of the meta-data to build a database of data preprocessing cases and successful preprocessing methods.

The Experimental part starts with the Chapter 7 introducing artificial datasets and shows the best preprocessing methods for them. Later it demonstrates that IPT can find the sequences of the preprocessing methods for the artificial datasets. It also shows and discusses the sequences IPT has found. The Chapter 8 shows that the parameters of the preprocessing methods have an influence on the accuracy of the classification model and presents results of the parameter value optimisation methods. The Chapter 9 shows the performance of the whole the IPT process of the sequence searches and the parameter optimisations. The best combination is identified and presented. The Chapter 10 shows selected results of IPT on the real datasets and discusses the results. The last Chapter 11 shows and discusses the use of the meta data to speed up IPT for new datasets based on historical knowledge (the results for the previously processed datasets).

Part I

Theoretical Background

2 Data Mining Background

In this chapter I will briefly describe background of data mining (DM) and the knowledge discovery in databases (KDD). In fact the data mining is usually seen as a part of the knowledge discovery process. There is no single definition of the KDD. In contrast there are many different definitions saying more or less the same. I will use the one given be Fayyad in [27]:

KDD is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.

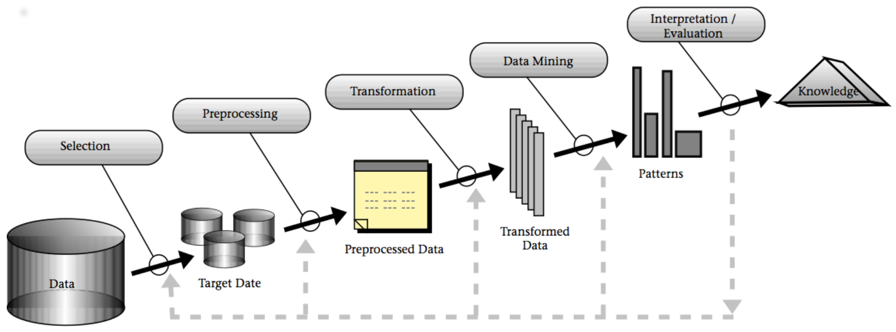


Figure 2.1: Stages of the Knowledge Discovery Process. Credit [27].

The knowledge discovery process is usually divided into several stages, see the Figure 2.1. The *Selection*, the *Preprocessing* and the *Transformation* are similar to data acquisition, feature construction and the data transformation phases as I have talked about them in the introduction. The result of these three phases is the dataset or the data matrix. These two terms are used as synonyms in the data mining and knowledge discovery community. Since data miners comes from different fields and have different backgrounds these are not the only terms to be freely mixed and used as synonyms. The other synonyms used are input or independent variables, input attributes or features meaning the inputs of the model. The same is for the outputs of a model. It is referred to as output, dependent or predicted variable. The other mixed terms are row, instance and pattern meaning information in different input attributes concerning one object. For example to describe an instance "flower" I can use input attributes like blossom colour, leaf shape, height or trunk size.

There is also controversy about the correct usage of terms – training, testing, validation sets. Some authors use training set to train a model, testing set to stop the training process and to prevent overfitting and after the training is finished, they use the validation set to validate the model. Some authors replaces the term testing set with the validation set and vice versa. In this thesis I will keep the first way.

There were several attempts to standardise the KDD process. According to [28], in academia the first attempt was in mid-1990s. The [29] introduced the model which consists of nine steps: the *Developing and understanding the application domain*, the *Creating a target dataset*, the *Data cleansing and preprocessing*, *Data reduction and projection*, *Choosing the data mining task*, *Choosing the data mining algorithm*, the *Data mining*, the *Interpreting patterns* and the last step, the

Consolidating discovered knowledge. As the thesis concentrates on the data preprocessing I will comment only on the appropriate part of the process. In the *Creating a target dataset* step data miner retrieves and merges data from primary sources and explores the data. The *Data cleansing and preprocessing* step consists of identifying and removing outliers, removing a noise and/or dealing with missing data. The last step *Data reduction and projection* consists of selecting important attributes, data reduction and projections. This methodology does not deal with the business point of view and sees the data mining as an linear process, which is usually not true. The knowledge discovery is usually an iterative process where data miner has to repeat earlier steps and examine theirs influence on the later steps.

On the basis of the previous model, the industrial standards have emerged. One of the is the CRISP DM[1] methodology [30]. The CRISP DM consists of 6 steps. Each of the steps have a number of substeps. The high level view of the CRISP DM is shown on the Figure 2.2.

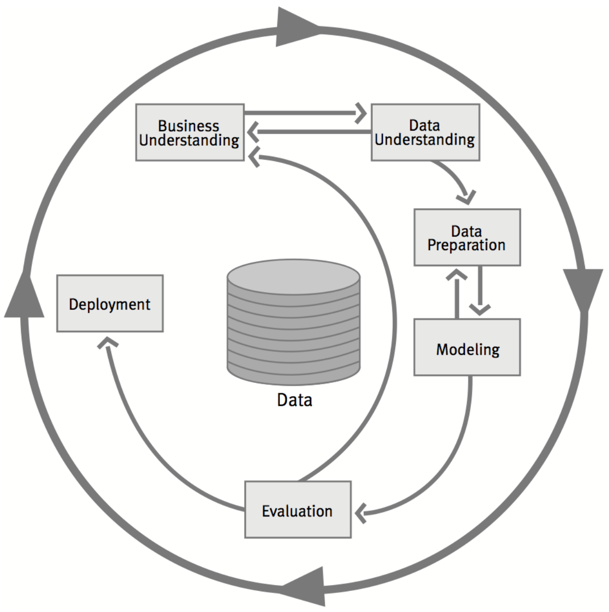


Figure 2.2: Overview of CRISP DM metodology. Credit [30].

The *Data Preproaration* step consists mainly of the *data selection*, the *data cleaning*, the *data construction*, the *data integration* and the *data formatting*. The *data selection* step identifies which of the data in the primary data sources are needed. The *data construction* step includes constructive data preparation operations such as the production of derived attributes or entire new records, or transformed values for existing attributes. The *data integration* combines multiple tables or instances to create new records and/or instances. And the *data formatting* refer to modifications made to the data that do not change its meaning, but are required by the modelling tools [30].

[1]Cross Industry Standard Process for Data Mining

2.1 Data Preprocessing and Preprocessing Methods

The previous sections presented the data preprocessing (or data preparation) as it is seen by the standard methodologies. For this thesis I have created a bit different subdivision for the data preprocessing task. The *Data acquisition*, the *Feature construction* and the *Data transformation.* These three subtasks overlaps and reorders the tasks from the CRISP DM methodology.

The data manipulation in the first two steps, the *Data acquisition* and the *Feature construction* need huge human interaction and business insight. For this reason they can be hardly fully automated. In these steps the ontology based support is the best way.

The *Data acquisition* step involves: *selecting the data* (attributes and instances) from the primary source (the database, data warehouse), *merging the primary data sources* together. To give an example I will continue to use the *churn* example from the telco industry from the introduction. This part also involve searching the company warehouse and selecting tables with customer demography information, money spend, service usage, promotions special offers and so on. In addition the dataset should contain the reasonable number of customers that have left to the another mobile services provider. The preprocessing in this phase can be imagined as the working with the warehouse and shaping the "SQL SELECT" to get the right data.

The *Feature construction* phase *removes obvious outliers*, eg. stolen mobile phones, and *replaces non-random missing data*, like amount of transferred data for users without mobile internet. The other task in this phase is to *construct new attributes*, like the region where customer lives from the call logs, and to *transform the existing ones*, eg. determine age in years from the date of birth.

These data preparation methods are **offline** – they do not need any classifier and the workflow and used methods are more or less the same no matter which classification method I will use.

The *Data transformation* mainly changes the dataset to be more suitable for the classifier. This phase primarily does not change the meaning of the data, but it mainly changes the representation. This statement is not completely the true. This part also can identify and remove not-obvious outliers, impute randomly missing data, normalisations, discretisation (or binning) an so on. The transformations does not add any new information to the dataset, but it can be essential for the classifier. The obvious example is to "straighten" the non-linear decision boundary for linear classifier. The transformations I have to do here do depend on the classifier I use, hence I call them **online**. The transformations for the Support Vector Machine are different from transformation for the Decision Trees or Back Propagation Neural Networks.

2.1.1 Online Data Preprocessing Methods in the Thesis

In this section I want to present online data preprocessing methods I use in the thesis. For purpose of my thesis, I have further divided online preprocessing methods on **local** and **global**. The local preprocessing methods are transforming only values in single attribute and works independently from other attributes. Examples of such methods normalisation, non-linear transformation or some missing values imputation methods. The global methods are, on the other hand, methods that transforms the dataset as whole and needs all attributes to work properly. The examples are outlier detection methods or data enrichment methods. In the thesis I have implemented preprocessing methods from several different fields:

- **Data Enrichment** methods – generate new instances for smaller class in imballanced datasets.
- **Data Reduction** methods – reduce number of instances in datasets.
- **Discretisation** methods – discretise the continuous attributes.
- **Non Linear Transformation** methods – transform the data
- **Missing Data Imputation** methods – replace missing values
- **Normalisation** methods – transform the range of values in an attribute
- **Outlier Detection** methods– detect and remove outliers

Data Enrichment is represented by a single method – *SMOTE*. The goal of this method is to generate artificial data points of a class that has low number of instances. The method takes two instances of from the class with less instances and generates a new instance on a line connecting the two original instances [31]. The *SMOTE* is a global method applied only on training data.

Data Reduction is a set of intelligent sampling methods who in more or less intelligent way reduces the dataset. The simplest way is the random sampling, when one removes the random portion of instances from the dataset. A bit more complicated method is stratified sampling [32], which reduces instances of different classes differently. In its way it complementary to the *SMOTE* method above as it allows one to balance number of instances for different classes by removing instances. There are also more sophisticated methods, trying to remove instances far from decision boundary, or instances in dense regions [A.2]. There is a great survey in the PhD thesis [33].

Discretisation also known as binning. According to [34] the binning is a technique of lumping small ranges of values together into categories, or bins, for the purpose of reducing the variability (removing some of the fine structure) in a data set. My implementation uses two ways to transform continuous values to discrete. One divides a value range into equal bins without regard how many instances are in them. The other divides the value range into bins with the same number of instances in each bin, but bins have different width [34]. Both of them are global methods.

Non Linear Transformation contains several transformation methods. Most of them are trivial local non-linear transformation like exponential , power and power root methods. But this group also contains the *Principal Component Analysis* global preprocessing method [35].

Missing Data Imputation contains several local and global preprocessing methods for imputing missing values. Their list and influence on a dataset can be found in [A.1].

Normalisation contains several standard – z-score, softmax or linear – normalisation methods.

Outlier Detection contains distance and density based algorithms for detecting and removing outliers [36, 37].

The list of implemented preprocessing methods with names and references is in the Appendix A.

2.2 Introduction to Used Modelling Methods

In my work I have concentrated on the classification task as described above. The selection of classification algorithms are quite wide and I have tested some of them in [A.6] but I will I my work use only two classification methods: the J48 Decision Tree and the Simple Logistic Regression Classifier. In this section I will describe them shortly.

2.2.1 J48 Decision Tree

A decision tree is a quite natural way to classify instances by series of questions. In inner nodes of a tree are questions concerning values of attributes and leaf nodes contains output classes. When a new instance arrives, the classifier starts in the root node and follows the correct answers through inner nodes to a leaf node. [38].

There is of course question how to construct such tree. In general the tree is constructed in a recursive manner, starting from the root. The attribute to put into a node is determined by its prediction power. The prediction power is determined using the information entropy, mutual information or correlation between given attribute and the output variable. The division terminates when a node contains instance from only one class or the set of instances is too small.

The unlike the typical tree training algorithm, the J48 Decision Tree can process continuous input attributes. It is an implementation of the well known C4.5 tree. More information can be found in [39, 40, 38].

2.2.2 Simple Logistic Regression Classifier

According to [41] the Linear logistic regression models the posterior class probabilities $Pr(G = j|X = x)$. The j is predicted class in condition of observed values x. The probability function for the J classes, using functions linear in x. The probability function must hold that they sum to one and remain in $[0 \ldots 1]$. The probability is calculated:

$$Pr(G = j|X = x) = \frac{e^{F_j(x)}}{\sum_{k=1}^{J} e^{F_k(x)}}$$

where $F_j(x) = \beta_j^T x$. The β_j are estimates found by iterative numeric optimisation algorithms that finds the maximum likelihood.

One such iterative method is the LogitBoost algorithm (see [42]). In each iteration, it fits a least-squares regressor to a weighted version of the input data with a transformed target variable. Here, y_{ij}^* are the binary variables which indicate if the instance x_i belongs to observed class y_i.

$$y^*_{ij} = \begin{cases} 1 \iff y_i = j \\ 0 \iff y_i \neq j \end{cases}$$

The [41] achieves the linear logistic regression behaviour by adding a constrain to use only linear functions in $F_k(x)$. In addition the [41] uses the LogitBoots to improve performance of the classifier. To get more details about Linear Logistic Regression Classifier (or SimpleLogistic), please refer to [41, 42, 43].

3 Inductive Preprocessing Technique

The **Inductive Preprocessing Technology** (IPT) is central piece and a contribution of my thesis. It is a name for my approach to the selection of the preprocessing methods. In contrast to other approaches to aid data miners in data preprocessing task IPT is based on idea similar to inductive modelling and optimisation approach.

In the inductive modelling the structure of the model is not predefined. The model starts from a minimal form without any knowledge of the data[1]. During the training the algorithm examines the data and improves the model's structure until it is complex enough to fit the data. Similarly to inductive modelling, the IPT approach is data-driven and the sequence of the preprocessing methods has no predefined structure. The sequnce starts from a random form **without any knowledge** about the data. As IPT progresses, it gets more information about the data and by adding, removing and modifying the preprocessing methods. It is searching for the simplest sequence of the preprocessing methods that achieves the highest accuracy of the classifier. IPT tries to find sequences only as complex (and contains only the preprocessing methods) as it is needed to preprocess the dataset correctly and to maximise the accuracy (fitness) of the classifier. The optimisation approach is used to find the correct modification of the sequences of preprocessing methods to increase the accuracy of the classifier. The modifications are to add, remove or modify the preprocessing method.

IPT has three cornerstones. The basic structure and the cornerstones of IPT is shown on the Figure 3. The cornerstones are the *Search for sequences of preprocessing methods*, the *Parameter values optimisation* and the *Classifier.* The cornerstones are discussed and described in separate chapters. The *Search for sequences of preprocessing methods* and the *Parameter values optimisation* are described in the Chapter 5. The *Classifer* is described in the Chapter 2. There are two other important terms – the *Sequence of the Preprocessing Methods* and the *Fitness* value. These terms are explained in the Chapter 4.

All the cornerstones are not limited to one specific method but I can easily use and test other methods. This is the most important for the classifier. This means that I can use any classification method with IPT. In my thesis I use only the Simple Logistic Regression Classifier and the J48 Decision Tree, but one can use any classification method. The only limitation is that the classifier has to process numerical data.

From the practical point of view the less important but with more scientific challenges are the *Search for sequences of the preprocessing methods* and the *Parameter value optimisation.* The Search for the sequences of the preprocessing methods finds the preprocessing methods to transform the dataset. The Parameter value optimisation optimises the values of the preprocessing method parameters. The tested search methods for the sequences and the Parameter values optimisations are presented in the Chapter 5. The output of IPT is a sequence of the preprocessing method to apply on a dataset.

[1]The examples of this approach are GMHD[8] modelling method or Decision Tree Induction[9, 10]

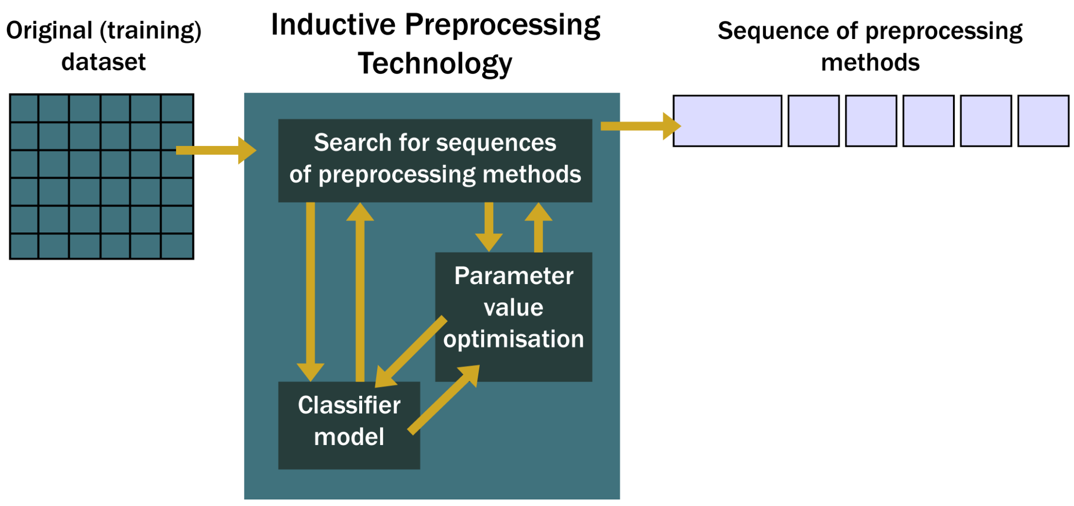

Original (training) dataset
Inductive Preprocessing Technology
Sequence of preprocessing methods
Search for sequences of preprocessing methods
Parameter value optimisation
Classifier model

Figure 3.1: Illustration of IPT's building blocks and theirs interaction.

4 Sequences of Preprocessing Methods and Fitness

In this chapter I will introduce and describe terms of *Sequences of Preprocessing Methods* and the *Fitness*. The first section of the chapter will introduce sequences of preprocessing methods. The second section will explain how the fitness is calculated.

4.1 Sequences of Preprocessing Methods

The *sequence of preprocessing methods* in short means a set of preprocessing methods applied to the dataset in given order. The ordering is quite essential and final result depends on it. To give an example, if you calculate second power and then linear normalise the dataset, you will get distinctly different result from the reverse order (linear normalisation first and then the second power).

What is equally important, are the preprocessing methods which should be applied are different for different input variables. Therefore the *sequence of preprocessing methods* must contain several *subsequences* – one for each input variable, see the Figure 4.1. Each *subsequence* is tied with one input variable. The *subsequence* contains preprocessing methods which are applied on given input variable. There are two types of subsequences – **local**, containing preprocessing methods which transform values in given input variable and should ignore all other variables. I will from time to time refer to such preprocessing methods as local preprocessing methods. Examples of such preprocessing methods are – *linear normalisation*, *N-th power calculator* or *discretisation*. The other type is a **global** subsequence. There is only one global subsequence and it contains preprocessing methods which transform whole dataset. The examples are *PCA* transformation or different types of sampling. It will refer to these preprocessing methods as global.

During search for optimal preprocessing methods there will be several sets of subsequences, each

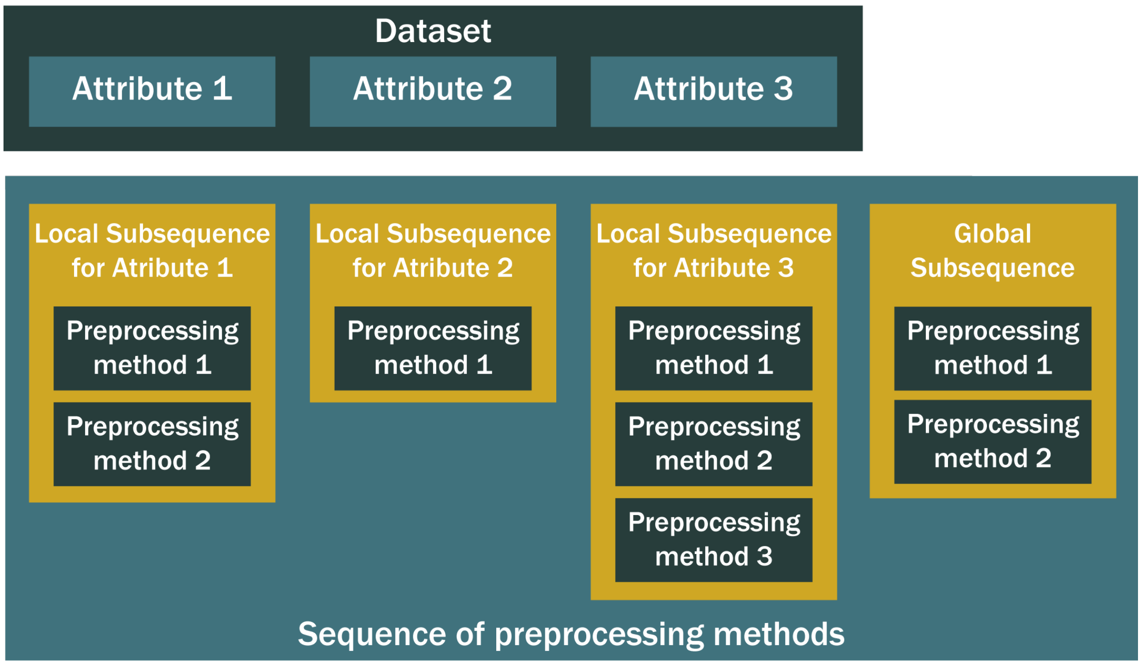


Figure 4.1: Illustration of dataset with three attributes and corresponding local and global subsequences.

set presents one possible way to preprocess the dataset. In future I will refer to such set as **sequences of preprocessing methods**.

Disabled Preprocessing Methods I have used this inspiration from [44] and I have added possibility to *disable* some preprocessing methods in the subsequences. Disabled preprocessing methods remain in the subsequences, they can take part in genetic search for the sequence of preprocessing methods (like in mutation or cross over) and a mutation can reenable them again. But they are not applied to the training nor testing datasets.

Application of Preprocessing Methods on Training and Testing Datasets A sequence of preprocessing methods is applied on the data in very simple way. First are applied all the local *subsequences* and then the *global subsequence* is applied. The local *subsequences* are applied in the same order as the attributes are stored in the dataset. In case of example dataset shown on the Figure 4.1 the *subsequence* for the Attribute 1 will be applied first, then the *subsequence* for the Attribute 2 and so on.

The sequence is applied on a training set and the testing set in a slightly different way. It makes no sense to apply some preprocessing methods when the testing set is preprocessed. For example it makes sense to reduce size of (sample) training dataset, as it results in faster training process and possibly in model that is easier to understand. However it makes no sense to reduce the testing set – I want to classify all the instances in the testing dataset, not only a few of them. There is additional possible problem, that the data reduction method could reduce the testing set to one instance and thus achieve very good accuracy.

4.2 Fitness Value and Its Calculation

This section introduces a **fitness** and explains how it is calculated for given sequence of preprocessing methods. As I have explained earlier I want to find the best sequence of preprocessing methods using the optimisation approach. The best sequence of preprocessing methods is the one which gives the most accurate classifier. In other words I am maximising the accuracy of the model by selecting and reordering data preprocessing methods in a sequence. The accuracy of a model trained with the preprocessed training set is an objective function for my optimisation or it is also in field of genetic algorithms it is also referred to as a fitness [45].

In simple terms the fitness value for given sequence of preprocessing method is an accuracy of a model trained with the preprocessed training set and tested with the preprocessed testing set. The exact fitness calculation is described in Algorithm 1.

Algorithm 1 Accuracy calculation for a sequence.

1. Divide a dataset into training and testing sets.
2. Shuffle both sets randomly.
3. Preprocess the training dataset with all the enabled the preprocessing methods recorded in the sequence. Start with the methods in the subsequence for the first attribute, continue with the subsequence for the second attribute, and so on. The last subsequence to be applied is the global subsequence.
4. The model is trained using the preprocessed training set.
5. The testing part is preprocessed in the same way as the training set. But some preprocessing methods (like sampling, see the chapter 2.1) are not applied.
6. Accuracy of the model is calculated using the testing set and becomes fitness.

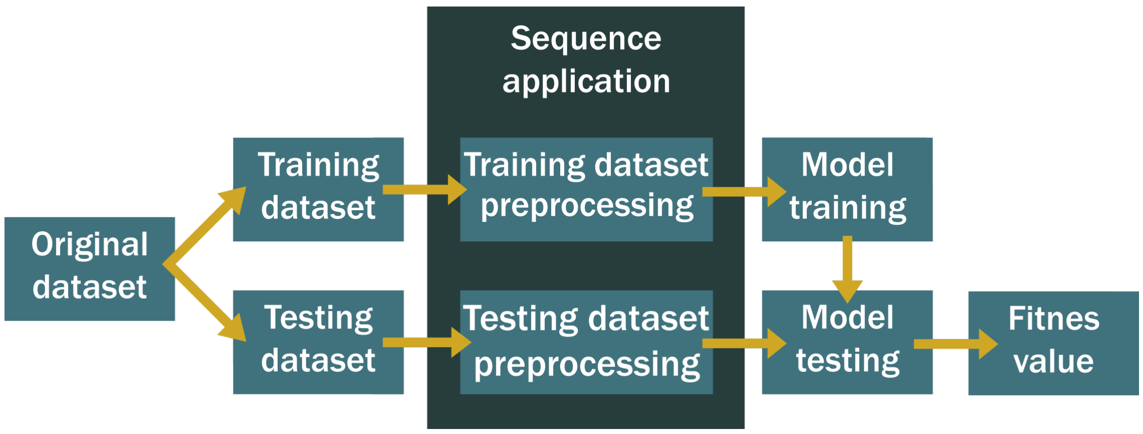


Figure 4.2: Application of sequence of preprocessing method and fitness calculation.

Although the above algorithm is quite straightforward, there are some open questions left. The first such question is the noisy fitness value [46]. The problem is that if I calculate the fitness several times, the exact value will be different. There are several reasons for this – random division of the dataset into training and testing sets, usage of random numbers in preprocessing methods and random initiation of modelling method. Now I will explain these reasons in slightly higher details.

The division into training and testing sets is done at random every time the fitness is evaluated. Therefore there is a risk, that the division will be a favourable one which adds some extra accuracy to the model. The extreme but illustrative example of this could be following: imagine that I want to classify blue and red dots. The classifier marks all dots as blue. But by chance all dots selected into the testing set are blue, therefore it has 100% on testing data. The training set has also influence on structure and parameters of the model and thus indirectly also on its accuracy. The repetitive divisions are necessary. The preliminary experiments has shown that if I provide one training and one testing datasets for whole Inductive Preprocessing Technology, the sequences tends to "overfit" on the testing set. Then if one presents the same dataset but divided into training and testing sets the accuracy of the trained model decreases dramatically.

The preprocessing methods uses random numbers to calculate the results. Therefore the preprocessed training set is and a model are slightly different in different repetitions. The examples should be, random sampling method, which randomly removes instances from training set or SMOTE data enrichment which randomly generates new instances (see sections 2.1 for details). If I preprocess the same dataset with the same preprocessing methods and use such dataset to train a model I will obtain slightly different results.

The third problem is that even a construction of a model sometimes involves some random process – like random initial values of parameters (like weights in back-propagation neural networks).

To address all these problems I have decided that the correct approach to the noisy fitness is to assume that the accuracy of the model is a random variable with normal distribution [38, 47]. The one training of a model and its testing according to Algorithm 1 means getting one sample from the random variable. To be able to sort sequences by performance I have decided to calculate a mean value of repeated accuracies as suggested in [47]. The correct way to compare if two sequences has the same mean is to use independent two-sample t-test with unequal and unknown variance, also known as Welsch's t-test [48]. But the t-test is hard to visualise and present and I need a technique that is easier to visualise. The technique is the boxplots [49]. It is harder to explain it in precisely statistical terms, but it can be easily visualised and in very natural terms indicates mean, both quartiles and outliers.

I am mailny interested in finding the best estimation of mean (μ) of the calculated accuracies. According to [48] the correct estimate of the normal distribution's mean value is a sample average. The sample consists of several fitness values calculated according to the Algorithm 1. In this way I will get several values from random distribution and I can use them to estimate the mean value – the correct mean fitness.

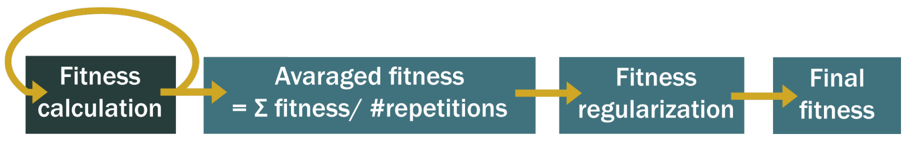


Figure 4.3: Illustration how to calculate final fitness from accuracies of several models.

I use the mean accuracy as the fitness value for given sequence. To evaluate one fitness value therefore means to train and test several models using the Algorithm 1. Apart from indicating performance of a sequence I can use the fitness also to shape the sequences as shown in the next subsection.

4.2.1 Fitness Modification

I want to keep the sequence of preprocessing methods as simple as possible. By simple I mean that the sequence and it's subsequences contains as little preprocessing methods as possible. For this reason I will employ the regularisation[50]. This means that I will penalise fitness of sequences of preprocessing methods containing too many preprocessing methods.

The regularisation works in following way: First I will calculate the mean fitness by repetitive data preprocessing, training a model and its testing as described above. After that I calculate the mean fitness. Then I apply the regularisation step. The regularisation is described by Equations 4.1 and 4.2. The first formula describes how the fitness penalty is calculated. If a subsequence contains more preprocessing methods than a certain threshold, I penalise each exceeding preprocessing method with a constant penalty. The penalties from subsequences are added together and the final penalty is calculated (as shown in the Equation 4.1).

$$\text{fitness penalty} = \sum_{\mathbf{S} \in \text{subsequences}} \max((\#\text{ methods in } \mathbf{S} - \text{threshold}) * \text{penalty}, 0) \tag{4.1}$$

After that I will subtract the penalty from the mean fitness calculated above. And correct the fitness value to 0 if the penalty is bigger that the mean fitness as shown in the Equation 4.2.

$$\text{modified fitness} = \begin{cases} \text{mean fitness} - \text{fitness penalty} & \Leftrightarrow \text{mean fitness} > \text{fitness penalty} \\ 0 & \Leftrightarrow \text{otherwise} \end{cases} \tag{4.2}$$

4.3 Conclusion

In this chapter I have introduced the sequence of preprocessing methods and the fitness value. In the next chapter I will explain how I will use them to find the best preprocessing methods for given dataset.

5 Search for Best Sequences and Optimal Parameter Values

In the previous chapter I have described the *sequences of preprocessing methods* and how to compute its *fitness*. These terms I will need in this chapter. Here I will describe algorithms I will use to reach my goal – to find the appropriate preprocessing methods for given dataset and given modelling method. Technically speaking this is an optimisation problem, in which I want to maximise the fitness value (accuracy of the model) by altering (sequences of) preprocessing methods.

In fact there are two optimisation problems – one is the **search for the preprocessing methods** and the other is **parameter values optimisation**. The search is basically the combinatorial optimisation problem [51]. The search decides which preprocessing methods – if I should use linear normalisation, discretisation, square root transformation or some other preprocessing method. The parameter values optimisation is on the edge of continuous and integer optimisation, because the parameters of the preprocessing methods are continuous, discrete and even nominal values. And is also has influence on the fitness value, as shown in the Chapter 8 and the Appendix B.

To make things more complicated these two steps can not be entirely separated. The change in sequence of preprocessing methods change parameters and the optimal values in parameters. On the other hand the change in parameter values can change the fitness of the subsequence and in this way its prospects in the next iteration of the search. The high level optimisation algorithm is shown in the Algorithm 2.

Algorithm 2 Fitness calculation algorithm.

```
while stopping criteria not met do
    do one step sequence of preprocessing methods optimisation;
    optimise parameters of preprocessing methods;
    calculate sequence fitness;
end
```

5.1 Introduction into Search and Optimisation

But before I will describe optimisation methods I have to at least briefly introduce problem of search and optimisation. Mathematically the definition is:

Find a vector $\theta \in \Theta$ that minimises (or maximises) objective function $L(\theta)$ [52].

The Θ represents the space of all possible solutions and θ is one of possible solutions, in my case all possible sequences of the preprocessing methods or all possible values in their parameters. The objective function L is fitness of a model as described in the Chapter 4. The optimisation can be constrained or unconstrained. In case of constrained optimisation, there is a set of conditions limiting values of the θ.

In my case the evaluation of the objective function with the same parameters will not yield the same value of the objective function. In this case the optimisation is called stochastic[52]. Formally the objective value is described as $L(\theta) = L_{actual}(\theta) + noise$.

There are several properties of the search spaces which selects optimisation methods. The search for the preprocessing methods is basically a discrete optimisation problem while the parameter value optimisation is mixed (real and integer) optimisation. The search for the preprocessing methods is unconstrained problem. The parameter values optimisation is constrained by a minimum and maximum values that can be assigned into the parameter.

For the parameter optimisation it is important to emphasise that the search space is not continuous and therefore the fitness (objective function) is also not continuous and thus does not have derivations.

5.2 Search for the Best Sequence of Preprocessing Methods

In this section I will describe algorithms I will use in search for sequence of preprocessing methods. The my previous work [A.3, A.4] suggests that the genetic search for the sequences of preprocessing methods is the best option. But to I want to test other approaches as well.

5.2.1 Exhaustive Search

Exhaustive or Brute Force search method examines all possible combinations and chooses the best one. This method is usable only for limited number of problems – ones with small number of inputs, limited number of preprocessing methods and limited size of sequences[1]. In all other cases the number of possible combinations is so big, that the algorithm will not finish in reasonable time. I will use this method as a benchmark in the Section 7.3 to verify that other searches are able to find correct preprocessing methods.

The Algorithm 3 summarises the Exhaustive Search. The algorithm is quite plain, it generates all possible solutions and then goes through them one by one and tests them. In the end it returns the one with the highest fitness.

5.2.2 Random Search

The Random Search is mainly yet another benchmark method. It is useful to know if other search methods are better or worse than generating solutions at random. As its name suggests it randomly generates sequences of preprocessing methods, tests its fitness and keeps the best so far found sequence [53]. In the end the best so far sequence is an output of this algorithm.

In this optimisation method there are two sequences (individuals) present at a time. One contains just produced sequence to be tested. The other contains the best sequence found so far. First the new individual is created.Then the parameter optimisation process takes place. When parameter optimisation is finished, the final fitness is calculated and compared to the fitness of the best so far sequence. If new sequences's fitness is better that the best so far sequence's, then replace the best so far sequence is replaced by the new sequence. And the algorithm continues with another loop, until predefined number of loops is finished. The Algorithm 4 summarises the Random Search.

One of advantages of the Random Search above other search methods is its simplicity and fact that I can easily control number of fitness evaluations by setting the #maxSearches parameter.

[1]Limited number of preprocessing methods in each subsequence

Algorithm 3 Exhaustive Search for optimal sequence of preprocessing methods.

```
/* Generate all possible combinations of preprocessing methods with no more than
   MAX methods in one subsequence.                                              */
allPossible ← generateAllPossibleCombinations(MAX);
bestSoFarSequence ← null;
forall the seq ∈ allPossible do
    /* Optimise parameters using methods described in the next chapter.         */
    optimiseParameters(seq);
       /* Calculate fitness for given sequence.                                 */
    calculateFitness(seq);
    if bestSoFarSequence==null then
        bestSoFarSequnce ← seq;
    else
        if bestSoFarSequnces.fitness < seq.fitness then
            bestSoFarSequnce ← seq;
        end
    end
end
return bestSoFarSequnce
```

5.2.3 Steepest Descent Search

This algorithm is inspired by the Steepest Descend method from the continuous and discrete optimisation field [51]. In the continuous optimisation the Steepest Descend uses the gradient to calculate the direction of the greatest increase/decrease of the fitness value and then makes a "step" in direction of the gradient or in direction opposite to gradient, depending if one is looking for maximum or minimum of the fitness (see for example [51, 52] for details).

In my case the fitness value has no gradient to compute hence I have to select slightly different approach, but it follows the same idea. I will start with the empty sequence, not containing any preprocessing method. Then I will try to add one preprocessing method, the one which increases the most the fitness of the sequence. Then I will find the second method causing the highest increase of fitness. And I will continue to add more and more preprocessing methods, until the fitness stop increasing or until the sequence is not too large[2].

There remains a question how to find the preprocessing method to add to the sequence and into which of its subsequences. The most straightforward way is to test all possibilities. This means that I will try to add each preprocessing method to the first subsequence[3], then I will add each preprocessing method to the second subsequence, and so on... I will calculate the fitness value for each added preprocessing method and I will keep the best one.

As you can see in the Algorithm 5, the matters are a little bit more complicated. I have found that sometimes the search get stuck in local optima and for the reasons explained in the Section 4.2 and in spite measures taken to prevent this it may happen that some preprocessing method is added although that there is some other method which performs better. For this reason I will continue the algorithm although the current step was unable to improve the fitness. But I do not

[2]Meaning contains lower number of methods than some threshold.

[3]A subsequence of preprocessing methods contains a list of methods to apply on give attribute/input variable in the dataset (see Section 4.1)

Algorithm 4 Random Search for optimal sequence of preprocessing methods.

```
/* Generates random sequence.                                              */
bestSoFarSequence ← generateRandomSequence();
optimiseParameters(bestSoFarSequence);
calculateFitness(bestSoFarSequence);
i ← 0;
while i < #maxSearches do
    newSequence ← generateRandomSequence();
        /* Optimise parameters using methods described in the next chapter.  */
    optimiseParameters(seq);
        /* Calculate fitness for given sequence.                             */
    calculateFitness(newSequence);
    if newSequence.fitness < bestSoFarSequence.fitness then
        bestSoFarSequence ← newSequence;
    end
    i ← i + 1;
end
return bestSoFarSequnce
```

want to continue with the process too long as it is waste of time, so I will perform at most two steps without fitness improvement.

There is a big disadvantages of this algorithm – the search is slow when there is a lot of preprocessing methods to test and a large number of attributes to preprocess. Also the algorithm is easily stuck in the local optima and in my case the exact found sequence is influenced by the fact that the fitness is a random variable.

5.2.4 Simulated Annealing Search

The Simulated Annealing Search is the standard simulated annealing optimisation method [54, 52]. In general it resembles the steepest descent method, but the simulated annealing tries to avoid the local optima by a possibility to accept solution with lower fitness.

In the beginning my implementation starts with a randomly generated sequence, which is also a best-so-far sequence. The best-so-far sequence is copied into a new sequence and a random change is done in the new sequence. The change is: to add a randomly selected preprocessing method, to remove from sequence a preprocessing method or to replace a preprocessing method is the sequence by another. Then the new sequence's fitness is evaluated and if its fitness is higher then the fitness of the best-so-far sequence, the best-so-far sequence is replaced by the new sequence. When the new sequence's fitness is lower than the best-so-far sequence's fitness the new sequence can still replace the best-so-far sequence with some probability Π. The probability Π decreases in each iteration of the search. The detailed version is shown in the Algorithm 6.

The simulated annealing is restarted from the beginning several times to achieve the best results.

5.2.5 Genetic Search

The last of the search methods is the genetic search. It is a standard genetic algorithm [45, 55].

Algorithm 5 Steepest Descent Search for optimal sequence of preprocessing methods.

```
bestSoFarSequence ← generateEmptySequence();
optimiseParameters(bestSoFarSequence);
calculateFitness(bestSoFarSequence);
workingBestSequence ← bestSoFar;
stepsWithoutFitnessImprovement ← 0;
while workingBestSequence.length < #maxLengthOfSequence AND
stepsWithoutFitnessImprovement < 3 do
    workingSequence ← workingBestSequence; forall the subsequence ∈ sequence do
        forall the p ∈ preprocessing methods do
            add p into subsequence;
            optimiseParameters(workingSequence);
            calculateFitness(workingSequence);
            if workingSequence.fitness > workingBestSequence.fitness then
                workingBestSequence ← workingSequence;
            end
            remove p from subsequence;
        end
    end
    if bestSoFarSequence.fitness < workingBestSequence.fitness then
        bestSoFarSequence ← workingBestSequence;
        stepsWithoutFitnessImprovement ← 0;
    else
        stepsWithoutFitnessImprovement ← stepsWithoutFitnessImprovement + 1;
    end
end
return bestSoFarSequence
```

The main loop of the genetic optimisation is shown in the Algorithm 7. It follows the loop of the standard genetic algorithm [55]. The only change is a new step `optimiseParametersIn()` which optimise parameters in the sequence. The population contains a set of sequences. Each individual is a sequence of preprocessing methods. In general the algorithm work as follows: at first, the initial population is filled with random sequences and their fitness is calculated, functions `generateRandomSequences` and `calculateFitnessInPopulation`. Then the main loop begins. It selects pairs of sequences for cross over operation. The `crossOverSequences()` function crosses over individuals in pairs. In this way the algorithm creates a new set of sequences. The new generation consists of elite sequences, sequences with the highest fitness from the current population, the crossed over sequences and several randomly generated sequences. The sequences in the new population are mutated. The only sequences excluded from mutation are elite sequences. The fitness is calculated in the new population and the last step is to replace the current population with the new one.

The selection, cross over and mutation have to be described in a greater details. In general they work as expected. The selection is a standard roulette wheel selection [56]. The cross over is a bit more interesting – the genome (sequence of preprocessing methods) is not a linear structure and different sequences has different length. For this reason the standard cross over is not usable. In the end I have decided to use a cross over operation where a subsequence for randomly selected attribute is exchanged between tow individuals.

Algorithm 6 Simulated Annealing Search for sequence of preprocessing methods. The function *random*() generates uniformly distributed random value between 0 and 1. The *initialTemperature* is set to a value between 0 and 1 and the κ is a value less then 1 but typically close to it. The number of iterations is determined by the *stoppingTemperature*.

```
bestSoFarSequence ← generateEmptySequence();
optimiseParameters(bestSoFarSequence);
calculateFitness(bestSoFarSequence);
temperature ← initialTemperature;

while temperature > stoppingTemperature do
    newSequence ← bestSoFar;
    randomChangeIn(newSequence);
    optimiseParameters(newSequence);
    calculateFitness(newSequence);
    if newSequence.fitness > bestSoFarSequence.fitness then
        bestSoFarSequence ← newSequence;

    else
        if random() < temperature then
            bestSoFarSequence ← newSequence;

        end
    end
    temperature ← temperature × $\kappa$;

end
return bestSoFarSequence
```

The mutation operation consists of two parts – the first is to mutate the structure of the sequence and the second is to change values of parameters in the sequence. To change the structure of a sequence I add a preprocessing method to a random subsequence, remove a random preprocessing method from a random subsequence or replace one preprocessing method with another. The mutation of the parameter values may be random change in a value as the standard genetic algorithm suggest (in the later text I will call this **One Random Change**) or it can be more sophisticated. In my case it is parameter optimisation as described below.

5.3 Parameter Values Optimisation

In this section I will describe algorithms used to optimise parameter values in a sequence. This step is used as a local search to improve accuracy of the fitness by tuning parameters values in preprocessing methods in the sequences. In context of the genetic algorithms it may be seen as an intelligent mutation. In contrast to the search for sequences of preprocessing methods, the parameter value optimisation is closer to the continuous optimisation. Or better mixed optimisation since some parameters contains integer values. The vector θ contains parameter values in preprocessing methods stored in the sequence. The search space (Θ) contains all the possible combinations of parameter values. The fitness value is the same as in the previous case, with the same drawbacks.

In contrast to the search for sequences of preprocessing methods one of the inputs of the optimi-

Algorithm 7 Genetic Search for sequence of preprocessing methods.

```
currentPop ← generateRandomSequences(POP_SIZE);
calculateFitnessInPopulation(currentPop);

while generation < maxGeneration do
    selectedSequences ← selectForCrossOver(currentPop);
        crossedOverSequences ← crossOverSequences(selectedSequences);
        /* Create empty population and fills it with sequences (individuals).    */
    newPop ← emptyPopulation;
        /* N sequences from the old population with the highest fitness.         */
    eliteSequences = getIndividualsWithHighestFitness(currentPop, $N_{elite}$);
        newPop ← addToPopulation(newPop, eliteSequences);
        /* Add crossed over individuals                                          */
    newPop ← addToPopulation(newPop, crossedOverSequences);
        /* Add few randomly generated sequences                                  */
    newPop ← addToPopulation(newPop, generateRandomSequences(5));
    mutateSequencesIn(newPop);
    optimiseParametersIn(newPop);
    calculateFitnessInPopulation(newPop);
    currentPop ← newPop;
    generation ← generation + 1;
end
return Sequence with the highest fitness in the currentPop
```

sation methods must be a sequence of preprocessing methods to optimise.

5.3.1 Random Optimization

The Random Optimisation is again more the benchmark method to test if other optimisation methods are at least as good as the random testing of the search space. The algorithm in this case is quite straight forward. It generates random values of parameters, evaluates fitness and remembers the best found solution. The Algorithm 8 describes the optimisation.

Algorithm 8 Random parameter values optimisation.

```
Input: Sequence of preprocessing methods
bestSoFarValues ← generateRandomValuesForParameresIn(Sequence);
calculateFitness(Sequence, bestSoFarValues);
$i$ ← 0;
while $i$ < #maxSearches do
    newValues ← generateRandomValuesForParameresIn(Sequence);
    calculateFitness(Sequence, newValues);
    if bestSoFarFitness.fitness < newValues.fitness then
        bestSoFarValues ← newValues;
    end
    i ← i + 1;
end
return bestSoFarValues
```

5.3.2 Steepest Descent Optimization

Algorithm 9 Steepest Descent Search for optimal sequence of preprocessing methods.

```
Input: Sequence of preprocessing methods
bestSoFarValues ← generateRandomValuesForParameresIn(Sequence);
calculateFitness(Sequence, bestSoFarValues);
while Number of steps < Max number of steps do
    workingBestValues ← bestSoFarValues;

    forall the parameter ∈ workingValues do
        workingValues ← bestSoFarValues;
        change parameter value in workingValues by + paramter.step;
        calculateFitness(Sequence, workingValues);
        if workingBestValues.fitness < workingValues.fitness then
            workingBestValues ← workingValues;

        end
        change parameter value in workingValues by - paramter.step;
        calculateFitness(Sequence, workingValues);
        if workingBestValues.fitness < workingValues.fitness then
            workingBestValues ← workingValues;

        end
    end
    bestSoFarValues ← workingBestValues;

end
return bestSoFarSequence
```

The steepest descend algorithm seeks a change in parameter values in which improves the fitness value the most. Since I have no gradient to guide the optimisation I have to test all the possible changes and select the best one. The change in my case is to change value in each parameter by a step in both directions. The step and value range is defined by a type of the parameter. The algorithm starts in a random point and tests all the possible steps in all the parameters and for each step evaluates the fitness and remembers the values for the best step. In the details it is described in the Algorithm 9.

5.3.3 Simulated Annealing Optimisation

The Simulated Annealing optimisation is essentially the same as in the case of the search for the best sequences. It is enhancement of the steepest descend approach which accepts with certain probability state with lower fitness value.

The simulated annealing optimisation is shown in the Algorithm 10. The simulated annealing randomly change values in parameters by a step. A size of the step is defined by the parameter. If the change can improve the fitness the change is accepted and the `newValues` are accepted. But if the fitness is not improved there is still a chance to accept the solution. The chance is expressed by a *temperature* variable in the algorithm. And it declines as the simulated annealing progresses.

Algorithm 10 Simulated Annealing for parameter value optimisation. The function *random*() generates uniformly distributed random value between 0 and 1. The *initialTemperature* is set to a value between 0 and 1 and the κ is a value less then 1 but typically close to it. The number of iterations is determined by the *stoppingTemperature*.

```
Input: Sequence of preprocessing methods
bestSoFarValues ← generateRandomValuesForParameresIn(Sequence);
calculateFitness(Sequence, bestSoFarValues);
temperature ← initialTemperature;

while temperature > stoppingTemperature do
    newValues ← bestSoFarValues;
    /* Change value in a random parameter by one step.                    */
    randomChangeIn(newValues);
    calculateFitness(Sequence, newValues);
    if newValues.fitness > bestSoFarValues.fitness then
        bestSoFarValues ← newValues;
    else
        if random() < temperature then
            bestSoFarValues ← newValues;
        end
    end
    temperature ← temperature × κ;
end
return bestSoFarValues
```

5.3.4 Differential Evolution Optimisation

The Differential Evolution (DE) [57] is the last approach to the parameter optimisation, I have tested. The DE resembles the standard genetic algorithm but uses different recombination or cross over and selection operations. The DE algorithm was developed for the real value problems, but can be extended to integer values as well [58]. I have decided to use the DE because it works well with the complicated search spaces [59].

In the DE the new generation is generated in following way. Each individual in the population is compared with a new, candidate, individual and if the candidate individual is better than the old individual, the old individual is replaced. Otherwise the old individual is retained in the population.

To generate a new candidate NC individual one needs an old individual I which may be replaced with the candidate and three other individuals P_1, P_2 and P_3 which are distinct from each other and from the I as well. The candidate NC is generated using the Algorithm 11.

Otherwise the DE optimisation works very similarly to the other optimisation methods and the DE optimisation is shown in the Algorithm 12.

Algorithm 11 Determine individual for the a generation in the Differential Evolution. The NC[i] denotes i^{th} value in the genome.

```
Input: $NC$, $P_1$, $P_2$, $P_3$, $I$
$NC \leftarrow I$;
$R \leftarrow$ random index between 0... dimension of $I$;
forall the $i \in$ 0... dimension of $I$ do
    if random() ¡ CR OR $i$ == $R$ then
        $NC$[i] = $P_1$[i] + random()($P_2$[i] - $P_3$[i]);
    end
end
if $NC$.fitness > $I$.fitness then
    return $NC$;
else
    return $I$;
end
```

Algorithm 12 Differential Evolution optmisation.

```
Input: Sequence of preprocessing methods

currentPop $\leftarrow$ generateRandomIndividuals(POP_SIZE);
calculateFitnessInPopulation(currentPop);

while generation < maxGeneration do
    /* Using the Algorithm 11 generate new population.          */
    newPopulation $\leftarrow$ generateNewPopulation(currentPop);
    mutateSequencesIn(newPop);
    calculateFitnessInPopulation(newPop);
    currentPop $\leftarrow$ newPop;
    generation $\leftarrow$ generation + 1;
end
return Individual with the highest fitness in the currentPop
```

6 Meta Data in Inductive Preprocessing Technique

The IPT Technique uses optimisation to find the best sequences of preprocessing methods. The search starts from the random points (with randomly generated sequences). This approach work but has a big drawback – it is time-consuming. I have searched the ways to improve speed of IPT and one promising way is to provide better good starting points to the search for sequences. In this case the staring points are sequences improving the dataset or at least sequences containing useful preprocessing methods.

I have started with the supposition that the similar dataset needs similar preprocessing methods. But IPT does not treat a dataset as a whole but rather treats individual attributes, so I will treat similar attributes with similar subsequences. To be able to measure similarity describe properties of attributes and then I can compare attributes. The properties are usually called meta-data.

In the past there were different projects in the field of data mining using meta-data. One of the oldest was StatLog [60]. According to [61] the aim of this project was to provide an objective assessment of the strengths and weaknesses of the various approaches to classification. The analysis of these experimental results aimed "to relate performance of algorithms to characteristics or measures of classification datasets."[60]. They used a number of dataset properties used until today.

The Statlog continued with the project called METAL. The goal of this project was to provide data mining method selection and combination algorithms to support data miner and to help him/her with the selecting appropriate classification and regression methods. There were also effort to use meta data to help the model training algorithm to select appropriate building blocks for a model [A.5] and many more. There is a great survey on the field [61]. There are even some projects in field of data preprocessing. Please see the introduction for details.

6.1 Meta Data Approach in the Inductive Preprocessing Technology

As written above I have decided to use the meta data in IPT to speed up the search for the sequences. The idea for speed up is following – if I am able to find a good starting point, it would take less steps of the search for the sequences of preprocessing methods to find the best sequence and thus the search would be faster.

To provide a good starting point means to find a good sequence of preprocessing methods or at least one that contains useful preprocessing methods. This idea is based on the supposition that similar input attributes would be preprocessed by the similar preprocessing methods [62]. To measure similarity of attributes I have measure their properties. The properties are called meta data. The similarity is then measured using the Euclidian distance. To store attribute metadata and the subsequences applied on the attributes I have created a meta database. The meta database is filled with the datasets and sequences and when a new dataset arrives, metadata for its attributes are extracted and for each attribute I find three most similar attributes in the meta database. And all the subsequences recorded for these attributes become candidate subsequences. And I will choose among them randomly when I am creating the start point sequences. The procedure is shown on the Algorithm 13. For simplicity sake I have omitted the global subsequences from the algorithm, but the steps are exactly the same as for the local subsequences shown in the Algorithm 13.

The sequences generated in the Algorithm 13 are then passed as starting points to the search for sequences.

Algorithm 13 Generating starting point sequences.

```
Input: Dataset

attributeToSequenceMap = Ø;

forall the Attribute ∈ Attributes in Dataset do
    metaData ← calculateMetaDataFor(Attribute);
        nearest ← findNearestAttributesInMetaDatabase(metaData);
        nearestSubsequences ← getSubsequencesInMetadatabaseFor(nearest);
        /* Record useful subsequences for given attribute */
    attributeToSequenceMap.add(metaData → nearestSubsequences);
end
for number of sequences to generate do
    generatedSequence ← createEmptySequence();
    forall the Attribute ∈ all attributes stored in attributeToSequenceMap do
        Subsequences ← attributeToSequenceMap.get(Attribute);
        subsequence ← get random subsequence from Subsequences;
        generatedSequence.set(Attribute, subsequence);
    end
end
return All the generatedSequences
```

6.1.1 Meta data

The meta data I have decided to extract are partially inspired by [60] and [14].

The metadata I can be separated into several groups and are following:

- Landmarking metadata:
  - *Accuracy of J48 Decision Tree* – is an accuracy of J48 Decision Tree with only given attribute and shows prediction power of the attribute.
  - *Accuracy of Simple Logistic Regression Classifier* – is an accuracy of Simple Logistic Regression Classifier with only given attribute and shows prediction power of the attribute.
  - *Average absolute correlation* – is an average correlation between the attribute and all the output variables. Since I am interested only in measure of dependence I use absolute value of correlation.
- Global dataset properties:

  - *Ratio between biggest and smallest class* – is a proportion between number of instances for the biggest and smallest class.
  - *Number of Classes* – tells how many classes are in the output variable of the dataset.

- Statistical properties in attribute: (All the statistical properties are greatly influenced by outliers in the attribute I have decided to remove values below 5-th percentile and above 95-th percentile. The exception to this rule is *Portion of Missing Values* meta datum.)
  - *Mean Value in Attribute* – tells the average value in the attribute.
  - *Value Range in Attribute* – tells the value range in the attribute.
  - *Variance in Attribute* – tells the sample variance in the attribute.
  - *Skewness in Attribute* – tells the skewness in the attribute.
  - *Portion of Missing Values* – tells portion of missing values in the attribute.
- *Information Entropy* – this is the only information theory meta datum. It indicates the information entropy in the attribute[63, 64].

Part II

Experiments and Results

7 Results of Search for Preprocessing Methods on Artificial Datasets

I will begin the experimental part of my thesis with demonstrating that the Inductive Preprocessing Technology (IPT) works with artificial datasets. The datasets are relatively simple, with only two attributes. I have decided to use these artificial dataset for two main reasons: the first is I that the dataset can be easily visualised, so I can easily demonstrate how the datasets are transformed. The second is that I know how IPT should transform the dataset in order to achieve the best accuracy of the model. This chapter will use these datasets to demonstrate that the data preprocessing can really improve the accuracy of the classification model and that IPT is able to find such preprocessing methods.

The first part of this chapter will introduce the datasets. Later I will use the brute force search to identify the preprocessing methods which transforms the datasets to the form suitable for models. In this part I will also demonstrate that the transformation of a training dataset have influence on the accuracy of a model trained with this dataset. Although the brute force search guarantees to find the best possible sequence of the preprocessing methods, it is not practically usable. In the Chapter 5 I have described several other search methods for the best sequence of the preprocessing methods. In the later part of this chapter I will test how effective they are and if they can find the same preprocessing methods as the brute force search. The latest part is dedicated to comparison of IPT and the one of the commercially available automatic dataset transformation method – the *Automatic Preprocessing* node from the IBM SPSS Modeler.

7.1 Artificial Datasets

First I want to introduce the artificial datasets I will use in this and later chapters. All four datasets are two dimensional problem. This is mainly for easier visualisation. The attributes are denoted as **A1** and **A2**. I will name the datasets according to problem they represent:

- Missing data – two U shapes and 50% of values are missing (Figure 7.1a). IPT should select a preprocessing method dealing with missing values.
- Imbalanced data – dataset where one class is has much more instances than the other (Figure 7.1b). This dataset can be repaired using the *SMOTE* enrichment algorithm.
- Non Linear – this dataset contains non-linear shape and I want to classify it using the Linear Logistic Function (Figure 7.1c). IPT should select the *Square Root Calculator* preprocessing algorithm.
- Outliers – this dataset contains some outlier which confuses the linear logistic function and it is unable to learn anything (Figure 7.1d). I expect that IPT will use the *LOF* outlier detection algorithm.

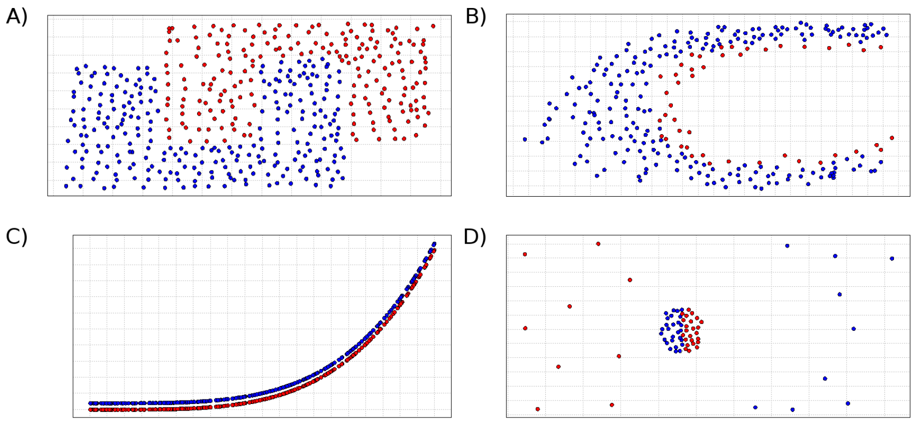


Figure 7.1: Artificial datasets used to demonstrate abilities of the Inductive Preprocessing Technology. The part **a** shows the Missing data datasets, the part **b** shows Imbalanced dataset, the part **c** shows the Non Linear dataset and the part **d** shows the Outliers dataset.

7.2 Fitness Calculation Setup

The core parameter in my work is the fitness values of selected preprocessing methods and their setup. The fitness is an accuracy of the model trained on the preprocessed dataset. To make sure, the dataset is several times divided into the training and the testing part and the accuracy of the model is calculated. The final fitness is the average accuracy of the 20 models. To make sure that the fitness does not depend on the ordering of the dataset, I will always shuffle the dataset before the part are created. For more details see the Chapter 4.

7.3 Exhaustive Search for the Best Preprocessing Method

In this section I will show the size of the search space of the preprocessing methods and their parameters and I want also to demonstrate that the there are preprocessing methods which are able to preprocess the artificial data to be suitable for the given model. And also to illustrate the influence of the preprocessing methods and their parameters on the dataset and accuracy of a model trained using such dataset.

To show that there are preprocessing method and their parameters which are able to preprocess the data, I will use the brute force search or exhaustive search, as I will continue to call it, for both – preprocessing methods and their parameters. When using the exhaustive search (testing all possible combinations of preprocessing methods and their parameters, see the Section 5.2.1) the computational complexity grows exponentially with the number of preprocessing methods and parameters. Eg. if I have 15 local preprocessing methods[1] and 18 global preprocessing methods[2] and each of them have 1 parameter with 20 different values and I will apply at most one preprocessing method on each input attribute – the number of possible combinations is $(15 \times 20)^2 \times (18 \times 20) = 32400000$. If one fitness evaluation for one combination lasts 1 second, it would take

[1]Preprocessing methods acting only on one attribute (see the Appendix A)
[2]Preprocessing methods acting with the whole dataset (see the Appendix A)

375 days to finish the calculations. This is unacceptable and for this reason I will select only a small subset of preprocessing methods, which will also contain the preprocessing methods I believe to be the best for given dataset.

For the *Missing Data* and the *Imbalanced* datasets I have selected following local methods: *Constant Missing Value Imputer*, *Another Instance Value Data Imputer*, *Missing Instances Remover* and global methods: *LOF*, *SMOTE*, *DROP3*.

For the *Non Linear* and *Outlier* datasets I have selected following local methods: *Adaptive Binning*, *N-th Power Calculator*, *Mean Value Normalizer* and *LOF*, *SMOTE*, *DROP3*. The *N-th Power Calculator* method should be used for the *Non Linear* dataset and the *SMOTE* for the Outlier dataset.

In this and all the later sections I will measure the results by achieved fitness[3] and also by accuracy of the independent model with the validation part of the dataset. The reason to use the independent model on the validation part of the dataset is to independently confirm the achieved accuracy.

7.3.1 Missing Data Dataset

The search has found following sequence of the preprocessing methods:

- for the **A1** attribute – *Missing Data Remover*.
- for the **A2** attribute – No preprocessing.
- for the **Global** attribute – *Missing Data Remover*.

The achieved fitness was 0.989 and the model trained on the preprocessed validation part has achieved 0.98 accuracy.

This dataset is quite simple for the search method – it has no parameters to set. To find the best sequence for the Missing data dataset the search method has just to select the *Missing Data Remover* method.

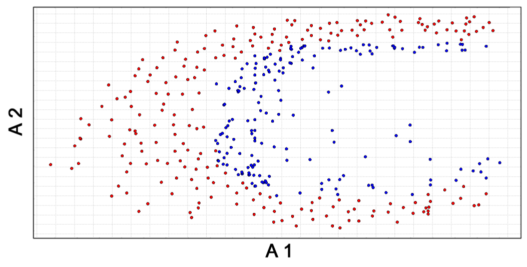
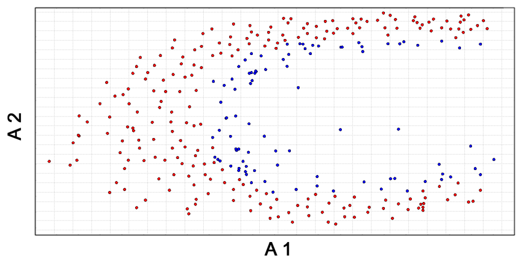
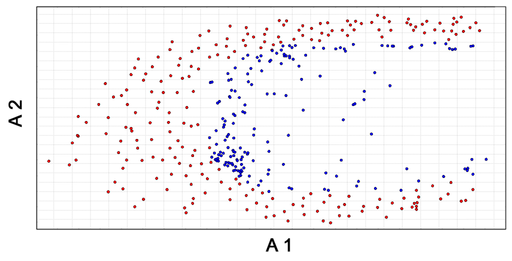
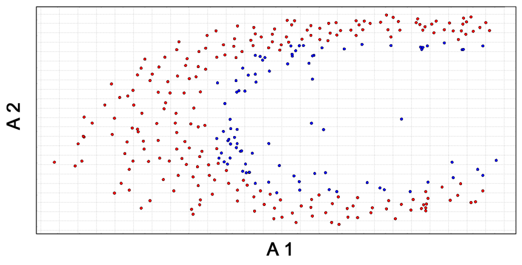
7.3.2 Imbalanced dataset

The search has found following sequence of the preprocessing methods:

- for the **A1** attribute – No preprocessing.
- for the **A2** attribute – the *Constant Missing Value Imputer* method and it imputer values -9.5. (Note that there are no missing values in the dataset, so this method does nothing.)
- for the **Global** attribute – the *SMOTE* method with – minor class enriched 3-times and uses 11 nearest neighbors.

[3]Accuracy of a model trained with the preprocessed validation part of the dataset, see the Chapter 4.

Table 7.1: Influence of the parameters (minor class enriched X times, # of nearest neighbors to use) of the *SMOTE* method on the preprocessed dataset and accuracy of the J48 Decision Tree Classifier.

 Minor class enriched = 3-times, use 11 nearest neighbors, model accuracy = 95.98%	 Minor class enriched = once, use 11 nearest neighbors, model accuracy = 91.92%
 Minor class enriched = 3-times, use 3 nearest neighbors, model accuracy = 95.48%	 Minor class enriched = once, use 3 nearest neighbors, model accuracy = 92.58%

The achieved fitness was **0.959** and the model trained with the preprocessed validation part has achieved accuracy (fitness) 0.95.

To illustrate the influence of the parameters of the *SMOTE* method, I have preprocessed the *Imbalanced* dataset with the *SMOTE* method with several different values of parameters and figures in the Table 7.1 show the results. The top left figure represents dataset preprocessed with the parameters selected by the search.

7.3.3 Non Linear Dataset

The search has found following seqence of the preprocessing methods:

- for the **A1** attribute – No preprocessing.
- for the **A2** attribute – the *N-th Power Calculator* method is used and it calculates 7^{th} power root ($\sqrt[7]{x}$).
- for the **Global** attribute – the *LOF* method with 2 nearest neighbors and $\sigma = 2$.

The achieved fitness was **0.997** and the model with the preprocessed validation part has achieved fitness equal to 1.

To illustrate the influence of the parameters of *N-th Power Calculator* on the preprocessed dataset, I have created a table 7.2. The table shows the *Non Linear* dataset preprocessed using the *N-th Power Calculator* method with the different parameters.

7.3.4 Outlier Dataset

The exhaustive search has selected following preprocessing methods for the Outlier dataset with following parameters:

- for the **A1** attribute – No preprocessing.
- for the **A2** attribute – the *Adaptive Binning* method with 2 bins and bins are coded as bin ID.
- for the **Global** attribute – the *LOF* method with 10 nearest neighbors and $\sigma = 0$.

The achieved fitness was **0.958** and the model on the validation part of the dataset achieved 0.96 accuracy.

The table illustrates influence of the parameters of the *LOF* method on the final dataset and accuracy of the Simple Logistic function. Precisely it shows selected values of σ and the *# nearest neighbors.* The dataset preprocessed by the exhaustive search is very similar to the figure in the second line, on the right in the Table 7.3.

The conclusion of this section is, that the transformation of the dataset using different preprocessing methods and different parameter values plays significant role in the accuracy of models trained with such datasets. So the selection of the proper preprocessing method and its parameters is quite an important for the accuracy of the model. Another important conclusion for the paragraphs below is that the presented datasets can be preprocessed with implemented preprocessing methods and gives estimate on accuracy of the models trained from given dataset.

7.4 IPT with Genetic Algorithm on Artificial data

The previous section illustrated the influence of the preprocessing methods on datasets and also on accuracy of models trained with the preprocessed datasets. Although the exhaustive search, as presented in the previous section, is able to find appropriate preprocessing methods and their parameters, is not usable for more complex datasets or more numerous sets of preprocessing methods. The time needed to finish the exhaustive search grows exponentially with number of input attributes, number of preprocessing methods and number of methods applied on one input attribute. It is obvious that for real-world problems the exhaustive search is not usable and I have to look for another search method. In my work I have focused on four sequence search algorithms – the genetic search, the simulated annealing, the steepest descent and the random search. The results of the later parts of this chapter shows that the genetic search is the best sequence search

Table 7.2: Influence of the N parameter of the *N-th Power Calculator* on the preprocessed dataset and accuracy of the Simple Logistic Classifier. Fraction values of N means power roots, integer values means powers.

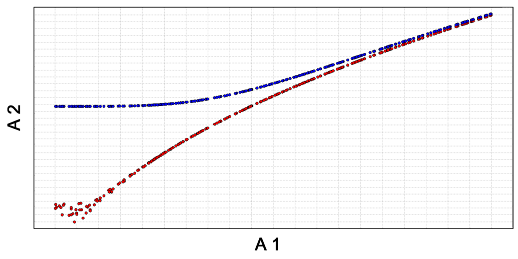
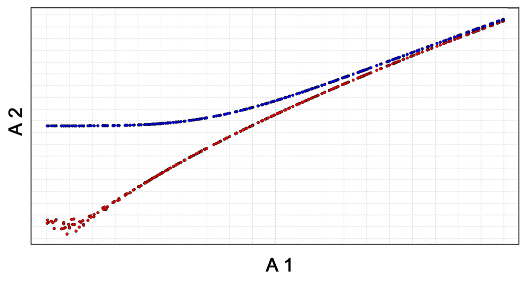
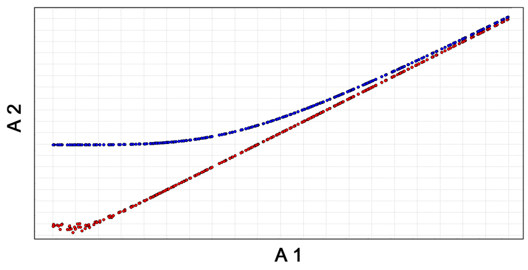
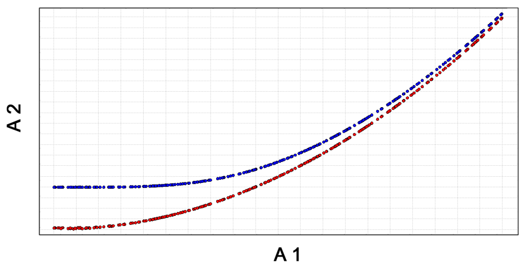
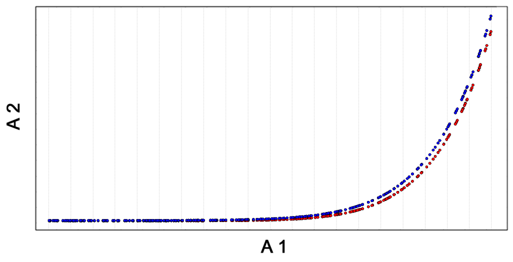
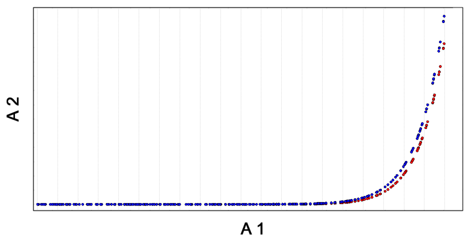
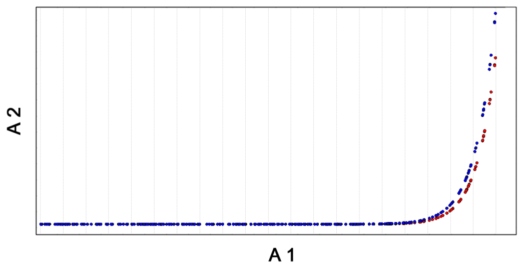
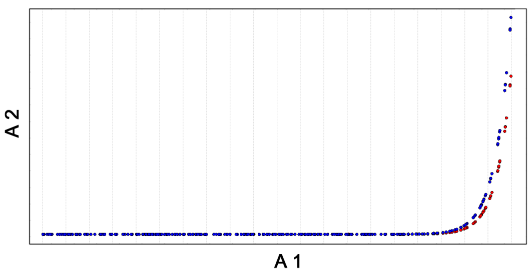
 N $= \frac{1}{8}$, model accuracy $= 99.83\%$	 N $= \frac{1}{6}$, model accuracy $= 99.83\%$
 N $= \frac{1}{4}$, model accuracy $= 93.33\%$	 N $= \frac{1}{2}$, model accuracy $= 65.17\%$
 N $= 2$, model accuracy $= 48.67\%$	 N $= 4$, model accuracy $= 48\%$
 N $= 6$, model accuracy $= 47.67\%$	 N $= 8$, model accuracy $= 48.5\%$

Table 7.3: Influence of the σ and $\#nearestneighbors$ parameters of the *LOF* method on the preprocessed dataset and accuracy of the Simple Logistic Classifier.

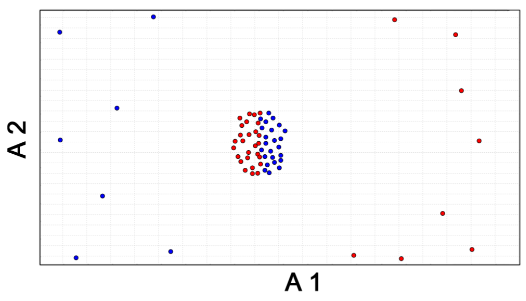
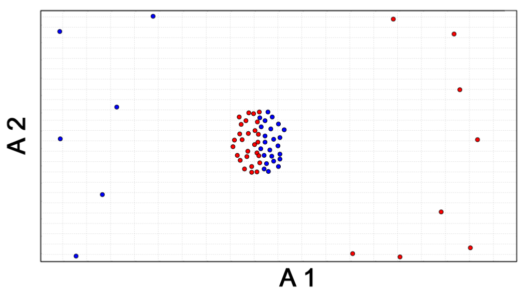
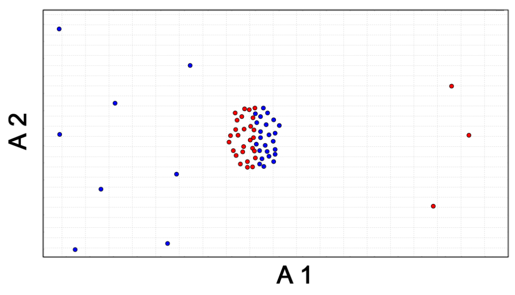
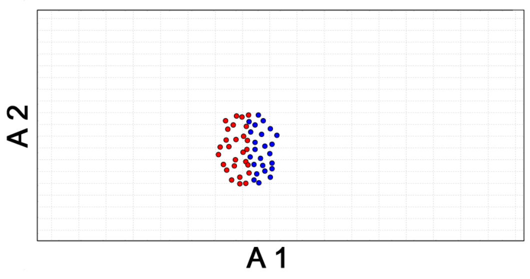
 $\sigma = 2.0$, # nearest neighbors = 2 , model accuracy = 37.62%	 $\sigma = 0.0$, # nearest neighbors = 2 , model accuracy = 45.71%
 $\sigma = 2.0$, # nearest neighbors = 10 , model accuracy = 50.95%	 $\sigma = 0.0$, # nearest neighbors = 10 , model accuracy = 98.0%

Table 7.4: Accuracy of the J48 model with the dataset preprocessed using different imputation methods.

Preprocessing method applied on A1 attribute	Missing data remover	No preprocessing	Another Instance's value	Constant Value	Median Value	Nearest Neighbor
Accuracy of the J48 Decision Tree [%]	0.98	0.88	0.838	0.877	0.877	0.825

method and for this reason I have decided to show its results separately from the other search methods whose results are presented and compared to the genetic search in the next section of this chapter. The search for the proper values of parameters will be discussed in the next chapter.

7.4.1 Missing Data dataset

The first dataset I will discuss here is the *Missing Data* dataset. The dataset contains two interwinded U shapes. From this dataset I have removed 50% values in the **A1** attribute and then I have supplied this dataset to the genetic search. I will continue to used the **J48 Decision Tree** as model for this dataset.

The genetic search has selected the *Missing data remover*, the same preprocessing method which was found by the exhaustive search in the previous section. I have visualised the first five sequences in the last generation of the genetic search. You can see it on the Figure 7.2. All the sequences contains the *Missing data remover* preprocessing method.

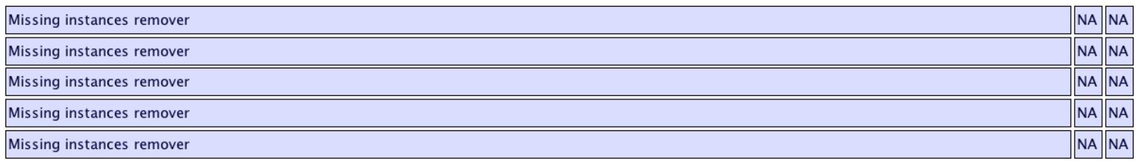

Missing instances remover
NA
NA
Missing instances remover
NA
NA
Missing instances remover
NA
NA
Missing instances remover
NA
NA
Missing instances remover
NA
NA

Figure 7.2: Five best individuals from the final generation of one of the runs.

To illustrate and compare the results of different data imputation methods I have preprocessed the original dataset with different data imputation methods and I have shown the results on the Figure 7.3a-e. The Figure 7.3a shows the original dataset – for better visualization I have replaced the missing data with 0 value, it also illustrates results of *Constant Replacer* method. Figure 7.3e shows result of *Missing Data Remover* method. The remaining figures shows result of other missing data imputing methods – Figure 7.3b *Nearest Value* imputer[4], Figure 7.3c *Median Value* imputer, Figure 7.3d *Another Instance's Value* imputer.

I have also created a J48 Decision Tree model from all above variants of the dataset and I have computed the accuracy of the model with given data. And I have calculated the accuracy of the original dataset as well. The results are presented in the Table 7.4. The *Missing data remover* method shows the best results and therefore IPT has selected the best method.

[4]Finds 5 nearest instances and replaces missing value with mean value of these 5 instances.

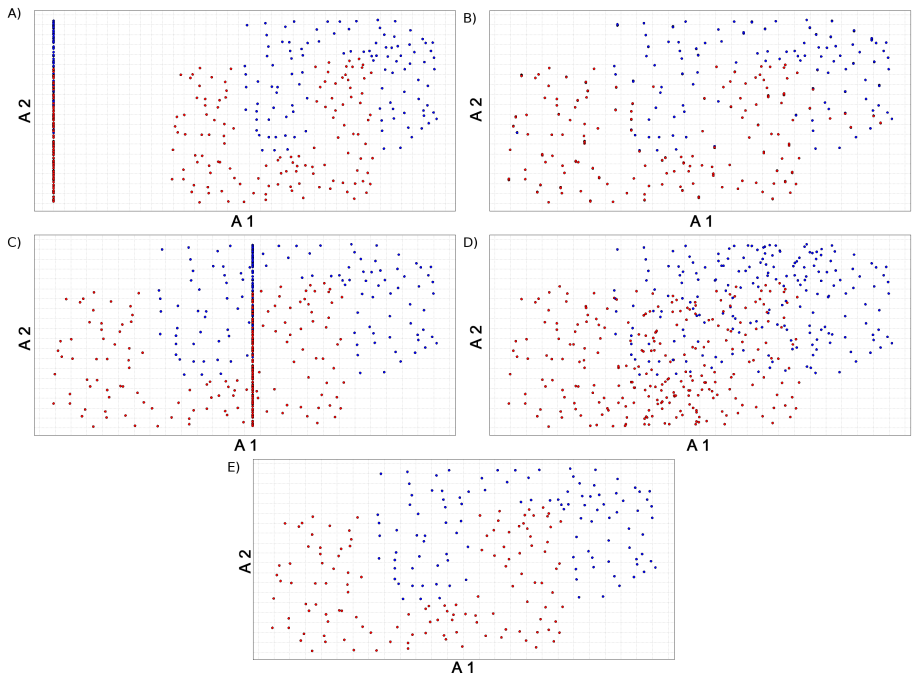


Figure 7.3: Results of imputation methods. Part **a** shows original dataset with missing data replaced with 0. It is also result of *Constant Replacer* method. Part **b** shows result of *Nearest Value* imputer. Part **c** shows result of *Median Value* imputer. Part **d** shows result of *Another Instance's Value* imputer. Part **e** shows result of *Missing Data Remover* method – selected by IPT.

7.4.2 Imbalanced dataset

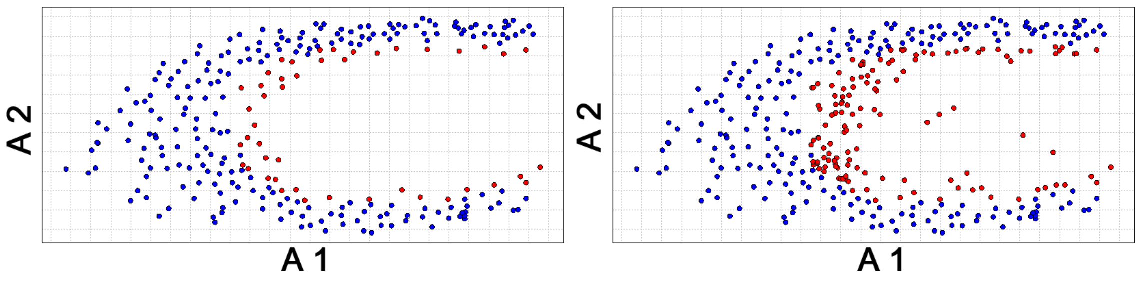


Figure 7.4: Original Imbalanced dataset on the left. Dataset preprocessed by the SMOTE method is on the right side.

This dataset contains two classes. One class is has much more instances than the other. To be precise the dataset contains 219 instance of class 1 and 45 instances of class 2. You can see this dataset on Figure 7.4 on the left. The correct solution of this problem is to use a data enrichment algorithm – in this case the *SMOTE* method. The SMOTE algorithm (see Appendix A or [31]) adds some artificial instances of minor class to the dataset. The genetic search has fulfilled my expectation and has selected the SMOTE algorithm. I have again extracted the five the best individuals and you can examine them on the Figure 7.5. All of them contains the *SMOTE* method as expected and in correspondence to the results of the exhaustive search.

Table 7.5: Accuracy of J48 model with the original validation dataset, *SMOTE* method applied 2-times (the first sequence from the top in the example population) and *SMOTE* method applied once (the last sequence in the example population).

	Original dataset	First sequence from Fig. 7.5	Last sequence from Fig. 7.5
Accuracy of J48 decision tree [%]	0.905	0.97	0.967

To confirm that applications of the *SMOTE* method brings better results, I have used the first sequence form the Figure the it to the validation dataset and the result is shown on the Figure 7.5 to the validation part of the dataset. The result is shown on the Figure 7.4 on the right side. The left side shows the original validation dataset. I have applied also other combination of preprocessing methods on the validation dataset and the results are in the Table 7.5. It show that the improvement in accuracy with preprocessed dataset in contrast to the original one is not so big as in case of the *Missing Data* dataset but is still significant. The result also shows that if the *SMOTE* method is applied several times, the results are not significantly improved.

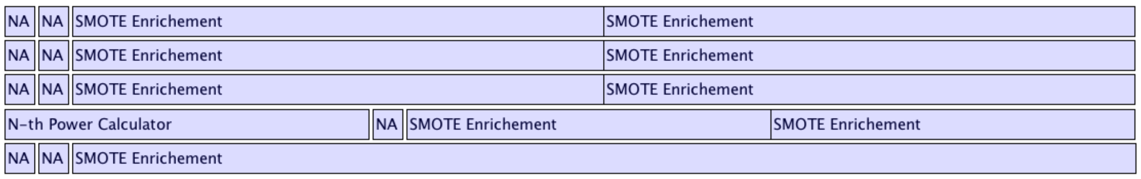

NA NA SMOTE Enrichement SMOTE Enrichement
NA NA SMOTE Enrichement SMOTE Enrichement
NA NA SMOTE Enrichement SMOTE Enrichement
N-th Power Calculator NA SMOTE Enrichement SMOTE Enrichement
NA NA SMOTE Enrichement

Figure 7.5: Sample of the best individuals in the last generation.

7.4.3 Non Linear dataset

This dataset contains non-linear decision boundary and I want to use the linear regression model to classify it. The dataset has to be preprocessed using a preprocessing method which straighten the decision boundary. To make thinks more interesting, I have omitted the *N-th Power Calculator* and I have replaced it with the *Root Square Calculator* method. In contrast to *N-th Power Calculator* the genetic search has to select the *Root Square Calculator* methods several times. And you can examine the results on the Figure 7.4.3. The left side of the Figure shows the original dataset and the right side the preprocessed dataset.

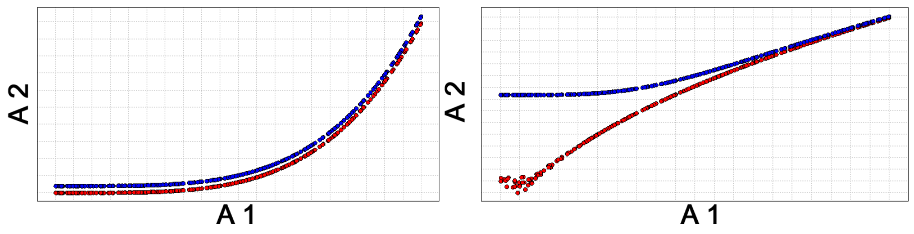


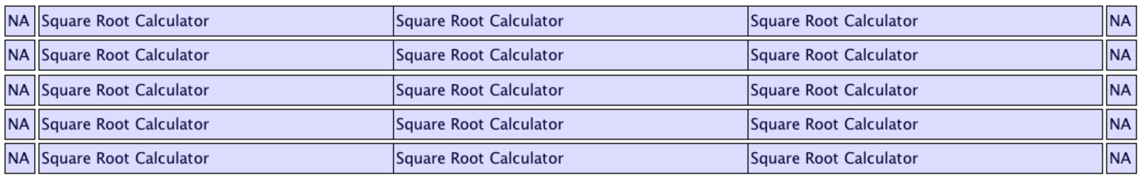
Figure 7.6: Non Linear dataset – on the left the original dataset, on the right the dataset preprocessed by several applications of *Root Square Calculator* method as found by the genetic search.

The preprocessed dataset is now easily classified by the Simple Logistic Classifer. To achieve this

Table 7.6: Error of the Simple Logistic model with the original data, data preprocessed by *Root Square Calculator* applied 3 times and data preprocessed by *Root Square Calculator* once.

	Original dataset	*Root Square Calculator* applied once	IPT preprocessed dataset (*Root Square Calculator* applied 3 times)
Error of Simple Logistic classifier [%]	0.543	0.628	0.983

the genetic search has to create a sequence containing several *Root Square Callculators.* Since the original dataset follows the x^4 function, only one *Root Square Callculator* will not do. You can examine the found sequences on the Figure 7.7. All of the uses three *Root Square Callculators.* In total they are giving 6th power root. You can examine the results of application of the other possible power root on the Table 7.2 in the previous section.



NA	Square Root Calculator	Square Root Calculator	Square Root Calculator	NA
NA	Square Root Calculator	Square Root Calculator	Square Root Calculator	NA
NA	Square Root Calculator	Square Root Calculator	Square Root Calculator	NA
NA	Square Root Calculator	Square Root Calculator	Square Root Calculator	NA
NA	Square Root Calculator	Square Root Calculator	Square Root Calculator	NA

Figure 7.7: Example population – the top 5 individuals in one of the runs.

The Table 7.6 shows errors of models with original and preprocessed data. Errors were obtained using 10 fold cross validation. Results confirms that simple logistic regression is unable to classify the separate classes properly. In fact the training algorithm completly fails to train any meaningful model. The accuracy is about 54.3% and confirms this conclusion. One application of the *Root Square Calculator* method straightens the decision boundary but not enough for simple logistic classifier and its training algorithm again fails to train any meaningful model. But repetitive applications of the *Root Square Calculator* method further straightens the boundary. After three applications of the method, the classes become linearly separable and the classification become far more accurate.

7.4.4 Outlier dataset

This last artificial dataset shows the ability of IPT to remove outliers. The original dataset is shown on the Figure 7.8a. The dataset consists of nucleus with very dense instances and a few sparse outliers. The outliers harms the capability of the simple logistic classifier to find decision boundary. The Figure 7.8b shows the dataset after application of the best individual found by the genetic search in one of the runs. The individual contained the *LOF* preprocessing method applied twice. The last Figure 7.8c shows the dataset after only one application of the *LOF* method. This dataset is much closer to the original dataset with only real outliers removed (see discussion later in this section).

The Figure 7.9 show the best sequences found by the genetic search algorithm and as was told before, the sequences mainly contains the *LOF* preprocessing method.

The Table 7.7 shows the accuracy of the models trained with the preprocessed validation part of

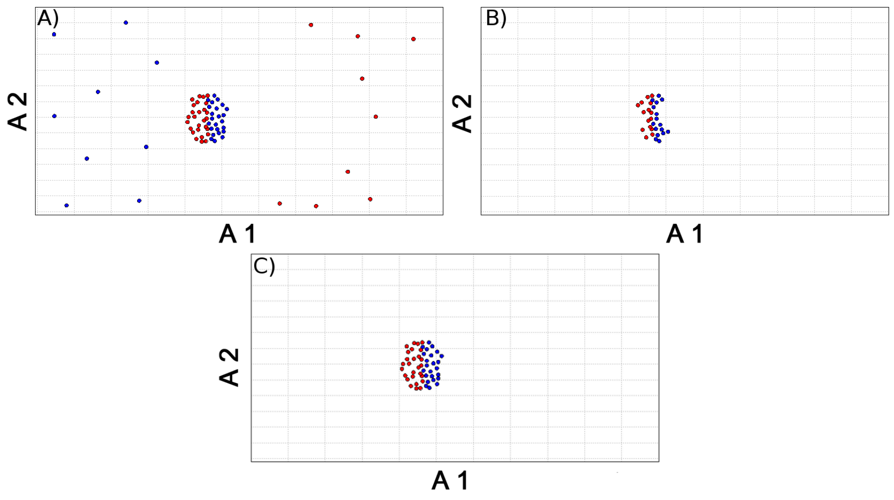
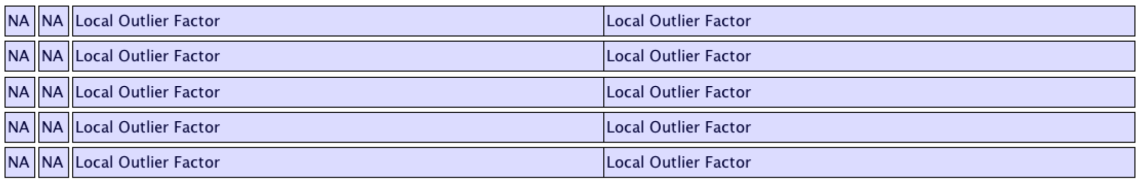


Figure 7.8: Original and preprocessed Outlier dataset. The part **a** shows the original dataset with outlier. The part **b** shows dataset after application of *LOF* method twice (as in the best sequences found). The part **c** shows the result of the one *LOF* application.



NA	NA	Local Outlier Factor	Local Outlier Factor
NA	NA	Local Outlier Factor	Local Outlier Factor
NA	NA	Local Outlier Factor	Local Outlier Factor
NA	NA	Local Outlier Factor	Local Outlier Factor
NA	NA	Local Outlier Factor	Local Outlier Factor

Figure 7.9: Example population – the top 5 individuals in one of the runs.

the dataset. The lowest accuracy shows the model with the original dataset. The reason is simple – the training algorithm of the simple logistic classifier is confused by outliers and fails to learn. The accuracy of the model with dataset preprocessed using only one *LOF* method is much better but still is higher than the accuracy for the dataset with two *LOF* methods applied. The reason lies in the way the fitness (and accuracy is calculated) – the validation dataset is randomly divided into the training and the testing part, both parts are preprocessed and the model is trained on the training part and the accuracy is computed, then the process is repeated several times and the final fitness and accuracy presented here is the average of accuracies obtained in repeated evaluations. And it may happen that in some repetitions the training and the testing sets are divided less favourable way and the *LOF* fails to remove all the outliers and the model fails to learn. In contrast if the two *LOF* methods are applied, all the outliers are removed and the model always finds the correct decision boundary.

7.4.5 Conclusion

In this section I have demonstrated the genetic search on four artificial datasets. The datasets were specially constructed to have only one data problem in it and also to be easily visualised. The problems are – missing data, different number of instances of two classes, non-linear decision boundary and outliers. The results show that the genetic search is able to find the proper data

Table 7.7: Accuracy of the Simple Logistic model with the original data, data preprocessed by *LOF* applied 2 times and data preprocessed by *LOF* once.

	Original dataset	Best sequence found (*LOF* method applied 2 times)	*LOF* method applied once
Accuracy of Simple Logistic classifier [%]	0.441	0.974	0.936

preprocessing methods to address these problems – data imputation method for missing data, data enrichment method for Imbalanced dataset, non-linear transformation for data with non-linear decision boundary and method removing outliers for the outlier dataset. Another positive outcome is that IPT works well with different modelling methods – J48 Decision Tree and Simple Logistic Function in this case.

7.5 The Genetic Search Algorithm vs Another Search Algorithms

In this section I will compare results of the genetic search algorithm from the previous section with another search algorithms. I will compare them on the same artificial datasets as in previous section and the search algorithms should find sequences very similar to ones from the previous section. I will test the following search methods:

- Exhaustive search algorithm – tries all possible combinations of the preprocessing methods (see 5.2.1).
- Random search – randomly generates sequences of preprocessing methods (see 5.2.2).
- Steepest descent search algorithm – adds preprocessing methods one by one. At first tries all preprocessing methods in all input attributes (subsequences) and selects the method and the attribute with the highest fitness. Then adds in the same way the second preprocessing method and so on (see 5.2.3).
- Simulated annealing search algorithm – classical simulated annealing algorithm (see 5.2.4).

I will compare the search algorithms using three criteria:

- How many fitness evaluations the search algorithm needs to finish the search (how long does it take to finish the search). I will also discuss the scalability of the method.
- If it is able to find combination of preprocessing methods at least as good as the genetic algorithm search from previous section.
- The last criterion will be the complexity of the best found sequence.

7.5.1 Fitness Evaluation

At first I will discuss the number of fitness evaluations to finish the search. The fitness evaluation is closely related to number of models which have to be learned. To calculate the fitness, one have to train several models. All search methods have fixed number of generations and repetitions (or at least their maximum number), I can give exact number of fitness evaluation and formula how to calculate it.

I will measure the length of the search for optimal preprocessing methods in number of fitness evaluation rather than in time. In addition to standard objections to time, there is one additional objection – that is that time needed to preprocess the data, depends on the data and also on the preprocessing methods and the model training time depends on preprocessed data.

Exhaustive Search Algorithm This algorithm generates all possible combinations of preprocessing methods, which grows at exponential rate. There is also important parameter ($N_{methods}$) which determines how many preprocessing methods at most are in each subsequence (maximal number of preprocessing methods).

The final formula to calculate number of fitness evaluation is following:

$$FE = (N_{local} + 1)^{N_{methods} \times N_{input}} \times (N_{global} + 1)^{N_{methods}}$$

where

- N_{local} – number of local preprocessing[5] methods.
- N_{global} – number of global preprocessing[6] methods.
- $N_{methods}$ – maximum number of methods in the sequence (maximum number of preprocessing methods for each attribute).
- N_{input} – number of input attributes.

The number of trained models is following:

$$N_{models} = FE \times N_{models}$$

At first I have enabled all 15 local preprocessing methods and 18 global preprocessing method. To be able to finish the calculation I have set the maximum number of methods in sequence ($N_{methods}$) to 1. Even with this limited settings the number of fitness evaluations for the artificial datasets is 4,864 fitness evaluations (145,920 models created) for each restart.

If I change the maximum number of methods in sequence ($N_{methods}$) to 2, the total number of evaluations is 23,658,496 (709,754,880 models created), which is almost unsolvable. (Even if I am able to evaluate 1 fitness in 1 second, it would take almost 274 days to finish the search).

[5]Methods working with and modifying only one attribute.
[6]Methods working with and modifying whole dataset.

Steepest descent search algorithm This search algorithm adds preprocessing methods iteratively, one by one. It starts with empty sequences (no preprocessing method applied on the data). Then it tries apply one by one all preprocessing methods on the first attribute, then tries to apply all methods on the second attribute and so on and ends with the global attribute. When it finishes, it selects the most accurate combination of the preprocessing method and the attribute. Then in the same way tries to add the second preprocessing methods to the selected and so on (see 5.2.3).

The number of evaluations depends on number of inputs, local and global preprocessing methods and maximum number of preprocessing methods to add. The formula to calculate the number of fitness evaluations is:

$$FE = (N_{local} \times N_{input} + N_{global}) \times N_{methods}$$

where

- N_{local} – number of local preprocessing methods.
- N_{global} – number of global preprocessing methods.
- $N_{methods}$ – maximum number of preprocessing methods in all sequences (maximum number of iterations).
- N_{input} – number of input attributes.

I have again used 15 local preprocessing methods and 18 global. And I have decided to try 10 preprocessing methods at the most. In this setup the number of fitness evaluations is 480. Meaning that I have to train 9600 models for each restart. This is much more reasonable number.

Simulated annealing algorithm The simulated annealing is the standard annealing algorithm with restarts. When the algorithm stops after fixed number of cooling steps or when the fitness stops to improve. After that the search start again from a new start point (restart).

The exact number of fitness evaluations is

$$FE = N_{restarts} \times N_{steps}$$

where

- $N_{restarts}$ – number of restarts of the search algorithm.
- N_{steps} – number of cooling steps.

In my case – 20 simulated annealing restarts and 500 optimisation steps, the number of fitness evaluation is 10,000 (200,000 models) for each IPT restart.

Random search algorithm This algorithm simply generates random sequences of preprocessing methods. This algorithm has very straightforward number of fitness evaluations – it is an input parameter of the algorithm.

In my case I have used 5,000 evaluations (100,000 models) for each restart.

Genetic search algorithm The number of fitness evaluations in the genetic search algorithm depends on number of repeating runs, number of generation and number of individuals. The exact formula is:

$$FE = N_{generation} \times N_{individuals}$$

where

- $N_{generation}$ – number of generations of the genetic algorithm.
- $N_{individuals}$ – number of individuals in each generation.

In case of my experiments, I am using 50 generations and 50 individuals. So there were 2,500 fitness evaluations (giving 50,000 models) for each restart.

The numbers of fitness evaluations of above search methods are summarized in the Table 7.8.

Table 7.8: Summary of number of fitness evaluations for one restart and its order of growth with respect to the number of input attributes.

Search method	# fitness evaluations	# models build	Grow order with # of input attributes
Exhaustive Search	4,864	97,280	exponential
Steepest descent search	480	9600	linear
Simulated annealing	10,000	200,000	constant
Random search	5,000	100,000	constant
Genetic search	2,500	50,000	constant

7.5.2 Best Solutions Found

Now I will demonstrate ability of different search methods to find the optimal (or nearly optimal) solution. The optimal solution for each dataset is described in the section 7.4. I have run the above search methods with parameters described above.

The aim of this section is to compare results achieved by the genetic search method and the other search methods. I will compare the best individuals found, the its fitness and the accuracy of model trained using the preprocessed dataset.

Table 7.9: Comparison of results of different search methods. The best fitness found indicates the best fitness found in all 20 runs.

Search method	Best fitness found	Accuracy on the validation set
Exhaustive Search	0.995	0.98
Steepest descent search	0.995	0.98
Simulated annealing	0.997	0.98
Random search	0.992	0.98
Genetic search	0.998	0.98

7.5.2.1 Missing Data Dataset

The results show that it is easy for all search methods to find such sequence of the preprocessing methods which produces the most accurate models. The difference in the fitness value is insignificant and is caused by random splitting of training and testing sets. The Figure 7.10 shows the best sequences found by different search methods.

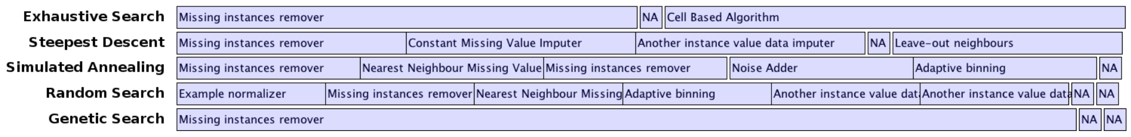

Exhaustive Search
Missing instances remover
NA
Cell Based Algorithm
Steepest Descent
Missing instances remover
Constant Missing Value Imputer
Another instance value data imputer
NA
Leave-out neighbours
Simulated Annealing
Missing instances remover
Nearest Neighbour Missing Value
Missing instances remover
Noise Adder
Adaptive binning
NA
Random Search
Example normalizer
Missing instances remover
Nearest Neighbour Missing
Adaptive binning
Another instance value dat
Another instance value dat
NA
NA
Genetic Search
Missing instances remover
NA
NA

Figure 7.10: The best sequences found for Missing Data dataset by different search methods.

To give better impression on distribution of the best fitnesses found by different search methods in different restarts I have created a boxplot shown on the Figure 7.11.

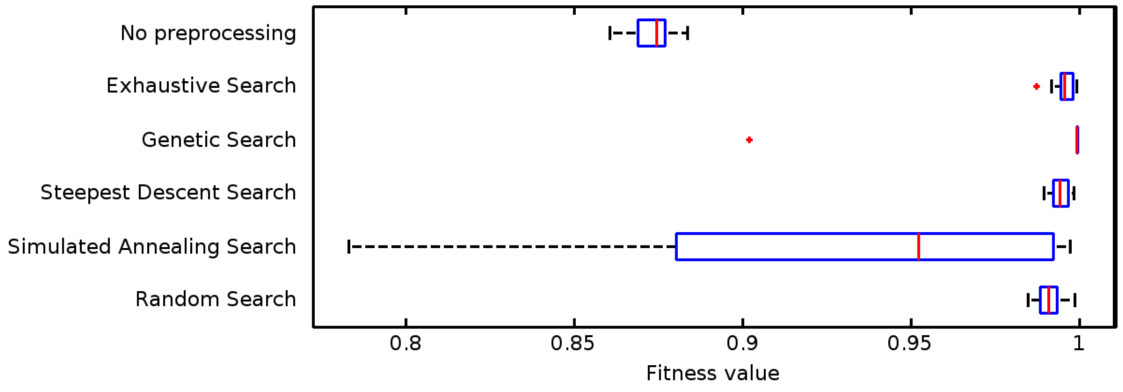


Figure 7.11: The boxplot illustrating distribution of the best fitnesses found by different search methods for the Missing data dataset. The red stripe in the middle shows the mean fitness, the blue box covers 50% of values (range between 25th and 75th percentile) and the black lines covers range between 5th and 95th percentile, covering 90% of values.

In general results show that all search methods are able to correctly add the *Missing Instances Remover* to the sequence and as the first method preprocessing the A1 attribute. All other preprocessing methods has either no or very little effect on the dataset. Eg. dealing with missing values when there are no values missing left.

The Figure 7.10 on the first line shows the best sequence found by the genetic search. Other lines shows the best sequences created by another search algorithms.

The second line on the Figure 7.10 shows the sequence created by the Simulated Annealing. It has several missing data imputation methods working with the first attribute (A1). But except the first method (*Missing Instances Remover* method) they are useless because there are no missing data. The preprocessing methods for the second attribute (A2) do some change to the dataset, but the change is not significant for the result.

The Steepest Descent algorithm has found a little bit simpler sequence then Simulated Annealing search. But there are also useless missing data imputation methods on already missing-data-free attribute A1. And a data reduction method which removes redundant data from the training set.

The Exhaustive Search algorithm has found sequence which is quite similar to the optimal sequence found by Genetic Search. The *Cell Based Algorithm* is an outlier removal method and in this particular dataset does not anything useful.

The results of the search methods with the Missing dataset data show that all the search methods are able to add the *Missing Instances Remover* preprocessing method to the final sequence for attribute A1. Search methods other than the Genetic algorithm search have also added some additional useless preprocessing methods. This is caused by fact that there are no other nearly optimal individuals (sequences) competing for the first place. So if an individual (sequence) with some useless extra preprocessing methods achieves slightly better fitness then the simpler individual (sequence) eg. by lucky selection of instances to folds in one of rounds, there is no individual to replace it in the next step (generation).

If you compare sequences found by different search methods, you will see that the sequence found by the Genetic search is the simplest and contains only the *Missing Instances Remover* method, which is the only useful preprocessing method. Results from different search methods contain number of useless preprocessing methods which make results harder to understand and explain.

7.5.2.2 Imbalanced Dataset

The table 7.10 shows results of different search methods for the Imbalanced dataset. The genetic search has achieved the best results. It is closely followed by other search methods, namely the Simulated Annealing, which is only 1% less accurate. To find the reason for better performance of the sequence found by the genetic search method, I had to examine the sequences found by different search methods. The sequences are shown on the Figure 7.12. The main reason is that the genetic search has added the *SMOTE* method twice. The others has added it only once. In the case of the Exhaustive search it is mainly due to limitations set on the search. I have allowed only one preprocessing method to be applied on each attribute. This limitation is set to finish the search in reasonable time. See the section 7.5.1 for discussion.

The other search methods has ended in local optima and were unable to add the second run of the *SMOTE* method.

To give better impression on distribution of the best fitnesses found by different search methods

Table 7.10: Comparison of results of different search methods.

Search method	Best fitness found	Accuracy on the validation part
Exhaustive Search	0.921	0.913
Steepest descent search	0.919	0.904
Simulated annealing	0.946	0.94
Random search	0.916	0.91
Genetic search	0.97	0.966

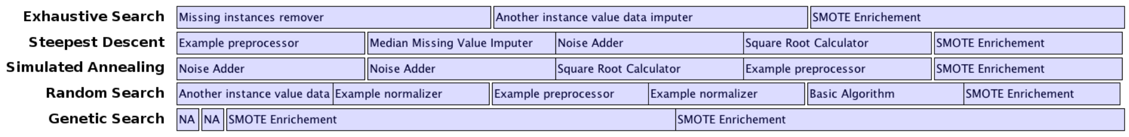


Figure 7.12: The best sequences found for Imbalanced dataset by different search methods.

in different restarts I have created a boxplot shown on the Figure 7.13.

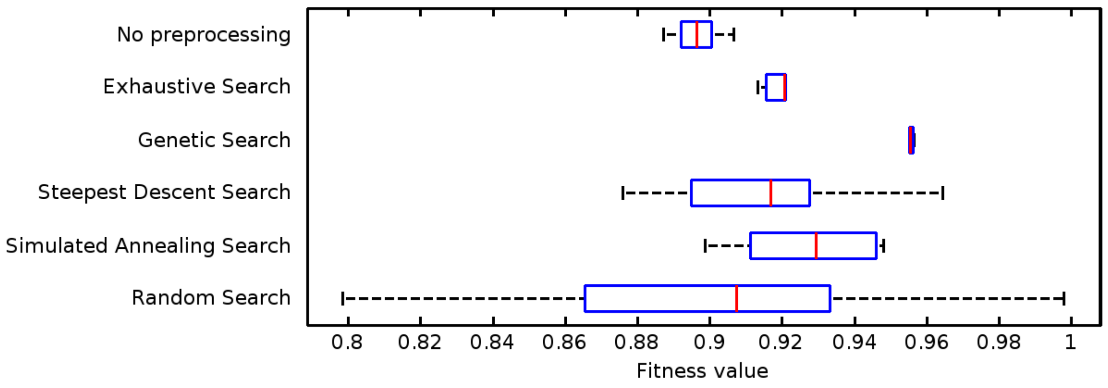


Figure 7.13: The boxplot illustrating distribution of the best fitnesses found by different search methods for the *Imbalanced* dataset. The red stripe in the middle shows the mean fitness, the blue box covers 50% of values (range between 25th and 75th percentile) and the black lines covers range between 5th and 95th percentile, covering 90% of values.

7.5.2.3 Non Linear Dataset

The results for the Non Linear dataset shows that all search methods, with exception of Exhaustive Search (see next paragraph for explanation), are again able, in terms of fitness and model accuracy, to find very good sequence of preprocessing methods, as shown in the Table 7.11. In the end all search methods have 3 times added *Square Root Calculator* method to the sequence for the **A2** attribute. The Random search added the *Square Root Calculator* only twice and this is the reason for the slightly lower accuracy.

As illustrated in Table 7.2, for successful classification at least 6th power root has to bee applied on the **A2** attribute. This means three repetitive applications of the *Square Root Calculator* method.

The Exhaustive search method is restricted to only one preprocessing method for each attribute. I have set this restriction to be able to finish the search in reasonable time as explained in 7.5.1. And this is reason for so low fitness of the Exhaustive Search. If I would soften the limitation to three preprocessing methods, the Exhaustive search would, no doubt, achieve results similar to other search methods.

Table 7.11: Comparison of results of different search methods for the Non Linear dataset.

Search method	Best fitness found	Accuracy of on the validation part
Exhaustive Search	0.748	0.73
Steepest descent search	1.00	0.991
Simulated annealing	1.00	0.99
Random search	0.947	0.98
Genetic search	1.00	0.994

The Figure 7.14 shows the best sequence of preprocessing methods found by different search methods. You can see that all search methods added the *Square Root Calculator* methods to sequence for the **A2** attribute. The sequence of preprocessing methods found by genetic search method is again the simplest and does not contain any useless preprocessing methods.

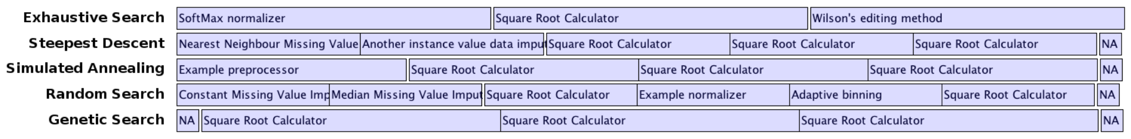


Figure 7.14: The best sequences found for Non Linear dataset by different search methods.

To give better impression on distribution of the best fitnesses found by different search methods in different restarts I have created a boxplot shown on the Figure 7.15.

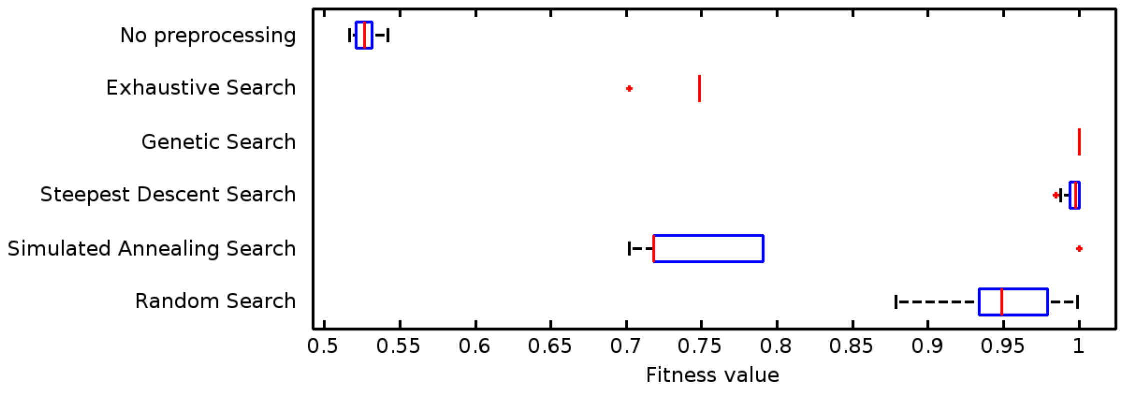


Figure 7.15: The boxplot illustrating distribution of the best fitnesses found by different search methods for the Non Linear dataset. The red stripe in the middle shows the mean fitness, the blue box covers 50% of values (range between 25th and 75th percentile) and the black lines covers range between 5th and 95th percentile, covering 90% of values.

7.5.2.4 Outlier Dataset

The results achieved for the Outlier dataset by different search methods are summarized in the Table 7.12 and the best sequences found by different search methods are illustrated on the Figure 7.16.

Table 7.12: Comparison of results achieved by different search methods for the Outlier Dataset.

Search method	Best fitness found	Accuracy of on the validation data
Exhaustive Search	0.966	0.951
Steepest descent search	0.97	0.964
Simulated annealing	0.957	0.948
Random search	0.925	0.92
Genetic search	0.991	0.982

The results shows that all five search methods are generally able to employ the *LOF* preprocessing method in the sequence and the found sequences achieved the same accuracy in the independent model. The inspection of the preprocessed data shows that all preprocessed datasets looks almost the same. The difference in the fitness has to be attributed to the random split of the data to the folds during the fitness evaluation and to the regularization penalty (namely in the case of the Random search).

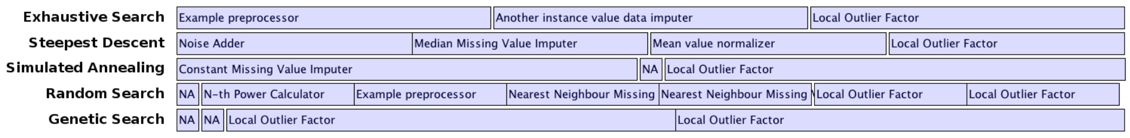


Figure 7.16: The best sequences found for Outlier Dataset by different search methods.

To give better impression on distribution of the best fitnesses found by different search methods in different restarts I have created a boxplot shown on the Figure 7.17.

7.5.3 Conclusion

In this section I have compared different search methods to find the optimal sequence of preprocessing methods. All search methods are generally able to add correct preprocessing methods to the sequence. But the genetic search generally achieves the highest fitness value.

In addition, if you inspect the best sequences shown on the figures in this section, you will see that the sequences found by the genetic search method, are generally the simplest and do not contain any useless methods. The setups found by other search methods generally contains a number of preprocessing methods which do not affect the dataset, eg. imputing missing data to attributes without missing values, or computing power roots or normalisation where it is useless.

The conclusion of this section all search methods are able to find setups of preprocessing methods which produces models with more or less comparable accuracy. To make a final choice, if you need

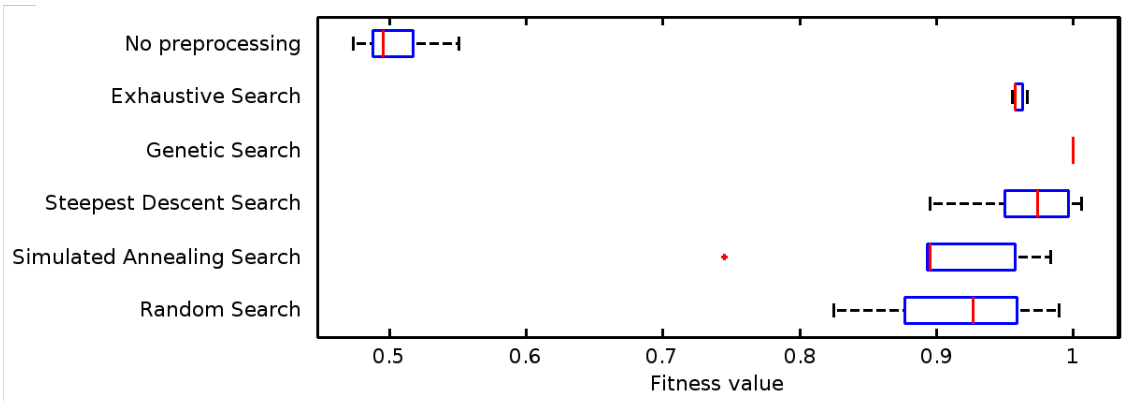


Figure 7.17: The boxplot illustrating distribution of the best fitnesses found by different search methods for the Outlers dataset. The red stripe in the middle shows the mean fitness, the blue box covers 50% of values (range between 25th and 75th percentile) and the black lines covers range between 5th and 95th percentile, covering 90% of values.

some result fast and you do not need to explain the operations in the preprocessing, you should use the **Steepest descent** search method. But if you need the best results and simple and easy to explain sequences, the **Genetic search** method is clearly the best choice.

7.6 More Complex Dataset

In the previous sections I have presented datasets which required only one preprocessing method to successfully preprocess the data. In this section I will test ability of all search methods to find solution for more complex dataset, which requires a combination of preprocessing methods to be successful.

This artificial dataset I will use here is a combination of Non Linear and Imbalanced datasets. The dataset is shown in the Table 7.13 on the lop left figure. And I will use the Simple Logistic classification algorithm. The Simple Logistic model is the most accurate when the *N-th Power Calculator* (or the *Square Root Calculator*) and the *SMOTE* methods are applied. The Table 7.13 shows the raw dataset, dataset preprocessed only by the *N-th Power Calculator*, dataset preprocessed by *SMOTE* and the dataset preprocessed by both, *N-th Power Calculator* and *SMOTE*. The table also contains accuracy of the Simple Logistic model.

The Table 7.14 shows the fitness and accuracy of independent models for the best sequences found by the search methods. It shows that in the end all search methods has found correct sequence of preprocessing methods (see the Figure 7.18 below). Almost all search methods has achieved very good accuracy.

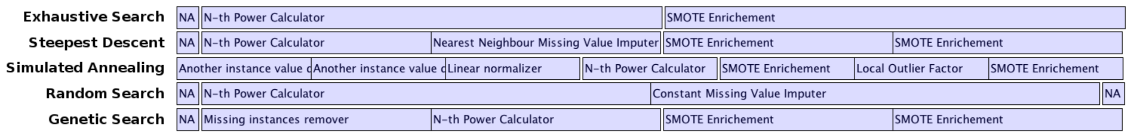


Figure 7.18: The best sequences found for Complex Dataset by different search methods.

Table 7.13: Illustration of the Complex dataset and influence of the preprocessing methods.

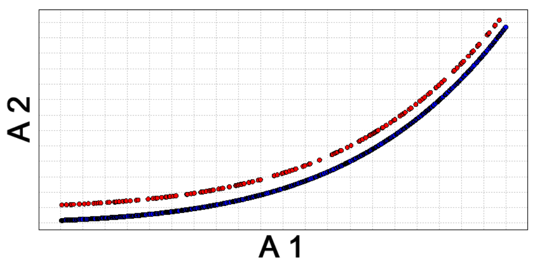
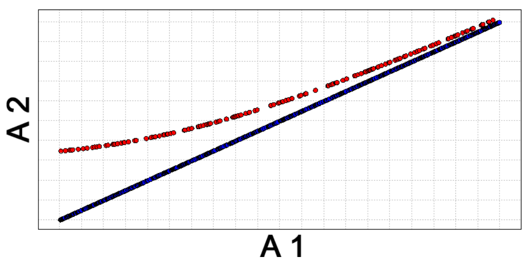
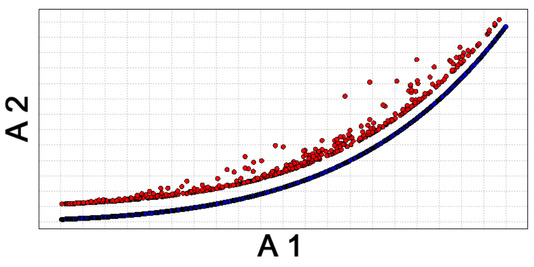
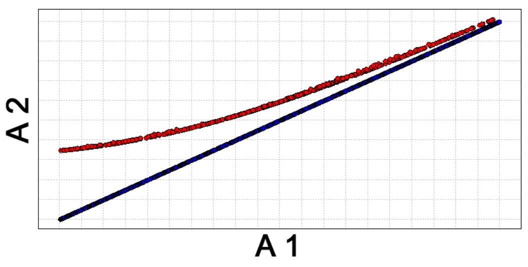
 Raw dataset without any preprocessing, accuracy of the Simple Logistic model = 84.34%	 Dataset preprocessed by the *N-th Power Calculator* calculating 8^{th} power root, accuracy of the Simple Logistic model = 98%
 Dataset preprocessed by the *SMOTE*, accuracy of the Simple Logistic model = 60.72%	 Dataset preprocessed by the *N-th Power Calculator* and the *SMOTE*, accuracy of the Simple Logistic model = 100%

Table 7.14: The fitness and errors of the independent models for the best sequences found for the Complex dataset.

Search method	Best fitness found	Accuracy with the validation part of the dataset
Exhaustive Search	0.996	0.99
Steepest descent search	1.00	1.00
Simulated annealing	0.998	0.95
Random search	0.917	0.912
Genetic search	1.00	1.00

The Figure 7.18 shows the best sequences found by different search methods. Again all the search methods have found the sequences that are very similar. All the methods has successfully added two *SMOTE* methods and the *N-th Power Calculator*. But the simplest are the sequences found by the Steepest Descent and the Genetic Search. Both sequences achieve the highest fitness as well. In this way results confirms the results from the previous section.

7.7 IPT vs IBM SPSS Modeler Automatic Preprocessing Node

The industry standard software – IBM SPSS Modeler – also contains a node for automatic preprocessing. The node uses some simple rules to preprocess data and the rules are briefly explained in the section 1.2 and in larger details in [25]. To compare the results of IPT and the SPSS Modeler's Automatic preprocessing node, I have loaded the data into the SPSS Modeler, preprocessed the data with the node and stored the result. The I have created the independent model for each dataset in the same way as in previous sections.

The results of the independent models are shown in the Table 7.15. The results show that the Modeler's automatic preprocessing node did very little to improve the accuracy. The accuracy of the dataset preprocessed by the Modeler's automatic preprocessing node is comparable with the non-preprocessed (raw) datasets.

Table 7.15: Comparison of results achieved models for the datasets preprocessed by IPT and the Automatic Preprocessing node in the IBM SPSS Modeler

Dataset	SPSS Modeler preprocessed dataset (classification error in %)	IPT preprocessed dataset (classification error in %)
Missing Data dataset	12.38	0.431
Imbalanced dataset	9.09	3.88
Non Linear dataset	45.67	0.17
Outliers dataset	57.97	1.96

8 Search for the Optimal Parameters in the Sequence of Preprocessing Methods

In the previous chapter, I have presented methods looking for the best sequence of the preprocessing methods. The first experiment in the previous chapter has given ideas that the parameters of the preprocessing methods are also quite important and I should optimise them as well. The first possible approach is to set random values of parameters for each new individual and hope that in the end I will generate correct values. I need something more sophisticated. In the Genetic Search method one solution is obvious – extend the mutation operation to parameters as well. By mutation I mean to change value of one or more parameters at random. But I am bound to test other techniques if they do not achieve better performance or if they find the optimal parameter values faster. All the approaches are described in the Chapter 5 on the page 21. In this chapter I will demonstrate and test these approaches to parameter optimisation and I will select the best one.

All the techniques share the idea of local optimisation [65]. I will generate a sequence of preprocessing methods and the I will do several steps of parameter optimisation algorithm and try to find optimal parameters for given sequence. This could shorten the overall number of steps of the search for the sequence of preprocessing methods. And thus could possibly shorten running time of the Inductive Preprocessing Technology.

To give example illustrating significance of correct values of parameters consider the Non Linear dataset from the previous chapter, one has not only select the *N-th Power Calculator* but also setup its parameter to calculate 8^{th} power root, if I will calculate, say, 2nd root, the accuracy of the model will be very low, although I have selected correct preprocessing method. To get idea how parameters of the preprocessing methods influences the accuracy of the model, please check Tables 7.1 (on page 40), 7.2 (page 42) or 7.3 (page 43). Tables shows data preprocessed with the same sequence but with the different parameters and also shows accuracy of the final model.

The problem of the parameters values optimisation can not be detached from the problem of searching optimal sequence. The choice of the best sequence depends on the values of method's parameters. As explained in Chapter 3 the search for the optimal is part of the "mutation step" of the search for the best sequences of the preprocessing methods.

There are many more or less sophisticated search methods I can choose from, but my search problem is quite different from the classical numerical and discrete optimisation:

1. First, some of the parameters are continuous numbers and some are discrete and I am unable to calculate gradient.
2. Second, the fitness is stochastic function and the exact value differs in repetitive evaluations. (See 4 for more details).
3. Third, if you examine the search spaces in Appendix B, they are not too complex.

For these reasons I have decided to implement and test the simple optimisation methods – the Random search, the Simulated annealing and the Steepest descent. I have decided also to test a genetic approach to optimisation and I have decided to to test the Differential Evolution algorithm as well. For details about the methods see the chapter 5.

This chapter is divided into four parts. The first part visualises two example space of parameters and illustrates progress of parameter optimisation methods. The second part compares ability of different search methods to find the optimal values of parameters. The experiments in the third part will try to find the minimal length of the optimal values search to find the optimal values of parameters. And the third demonstrates influence of the optimal parameters search on the search for the best sequences of preprocessing methods.

8.1 Visualisation of Search Spaces

This section visualises two example search spaces – one for the *N-Power Calculator* and the Non Linear dataset. The second example are parameters of the *SMOTE* enrichment method applied on the Imbalanced dataset.

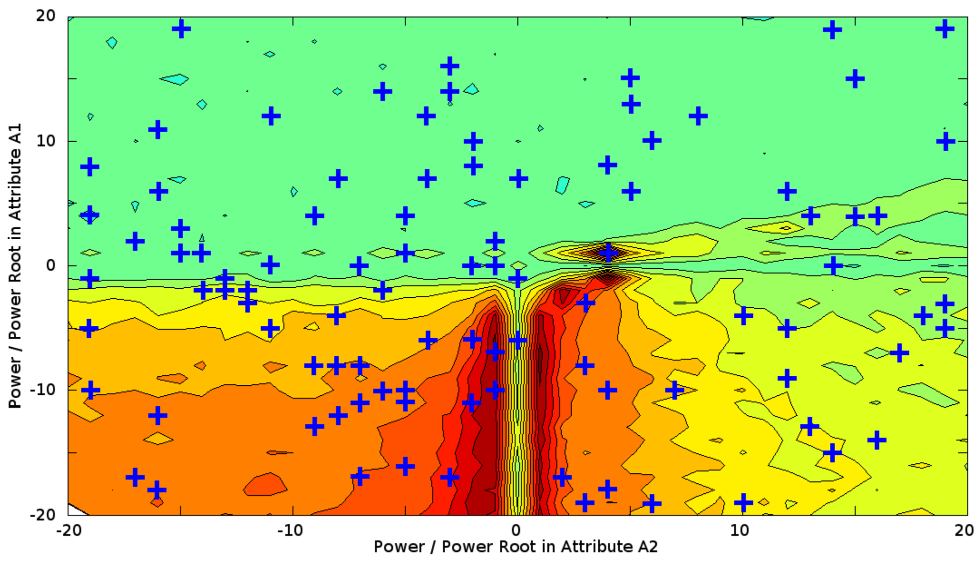


Figure 8.1: Visualisation of the parameter values search space for the *N-Power Calculator* preprocessing method and the Non Linear dataset. Blue marks shows positions examined by Random search algorithm.

At first I will show the search space for *N-Power Calculator* and the Non Linear dataset. As you can see on figures below, the parameters of the *N-Power Calculator* has very strong influence on the final fitness of the model. To be able present fitness in the 2D space I have applied the *N-Power Calculator* on both attributes – A1 and A2. The result is shown for example on the Figure 8.1. The green color represents low values of fitness and red represents the higher values.

The search space is shown on the Figures 8.1, 8.2, 8.3, 8.4. The figures shows that there is a large area with low fitness for positive values of the Power/Power root values in **A1** attribute and there are several equivalent optimal values. To be precise, there are four similar global maxima, I am looking for – the first two use 4th power root in the **A1** attribute and 1st power or 1st power root (both mean identity) in the **A2** attribute. The third and fourth extremes are for the 4^{th} power in the attribute **A2** and the identity (1^{st} power or 1^{st} power root) in the **A1** attribute. If you think about it, these situations are equal and both leads to the linear separation border between classes.

The Figure 8.1 shows parameter values in the search space examined by the Random search. It shows almost uniform spread of examined points and you can see that the search method hit one of the global optima.

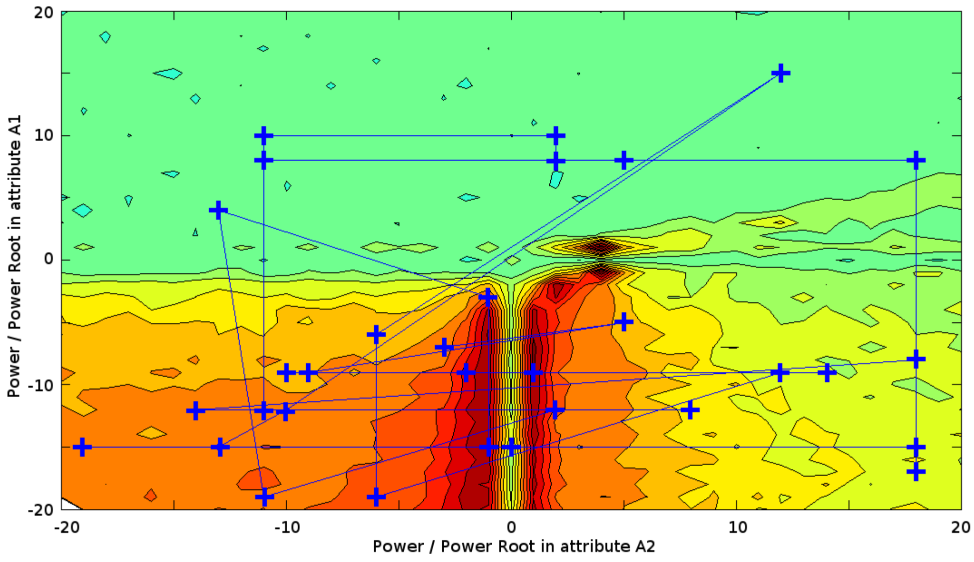


Figure 8.2: Visualisation of the parameter values search space for the *N-Power Calculator* preprocessing method and the Non Linear dataset. Blue marks shows positions of the best-so-far solution of the Simulated annealing search algorithm.

The Figure 8.2 shows the progress of the Simulated annealing. The dots now shows the positions of the best-so-far solution in the Simulated annealing. The line connecting the positions of the best-so-far individual shows its the movements during the progress of the algorithm. Note that in the beginning of the Simulated annealing, it is probable that the best-so-far solution is moved into areas with lower fitness value.

The Figure 8.3 shows the progress of the best-so-far solution in the Steepest descent search algorithm. The dots again represents best-so-far solution and lines represent its movement. In contrast to the Simulated annealing the Steepest descent shows lower number of movements of the best-so-far solution, this is caused by fact, that the best-so-far solutions are moved only if the new states with higher fitness.

The Figure 8.4 illustrates individuals in different generations of the Differential Evolution. In the beginning of the differential evolution, the individuals are uniformly distributed in the search space. After 50 generations, the individuals are more close together and all are in the area with higher fitness. Individuals in the last generation are all quite close to one of the optimal solutions.

The search space presented on the Non Linear dataset and two *N-Power Calculator* methods is an example of well defined search space with distinct and well defined optima. The second example presented here is a Imbalanced dataset with single *SMOTE* method applied. The *SMOTE* has

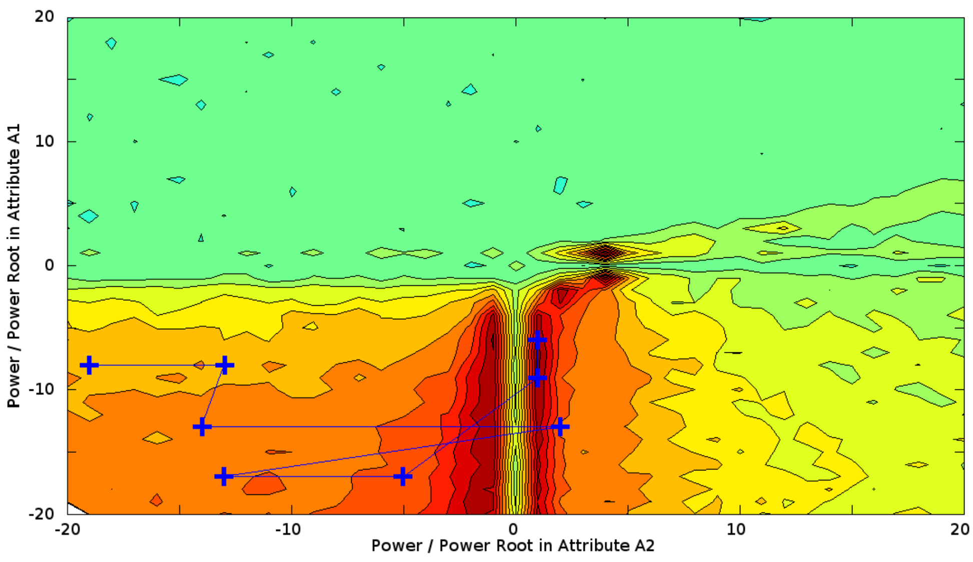
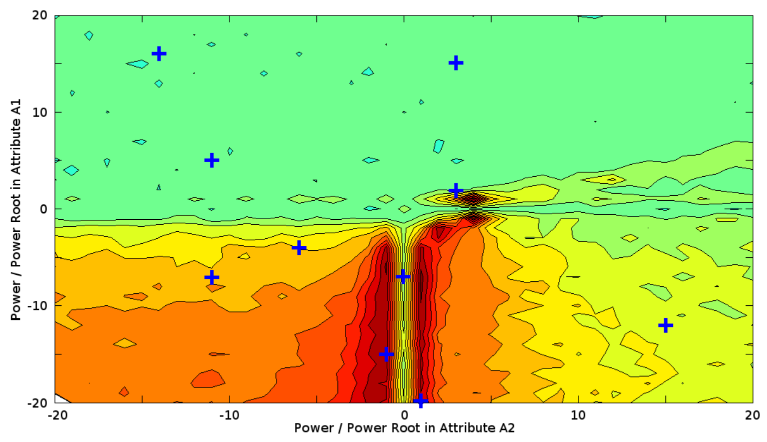


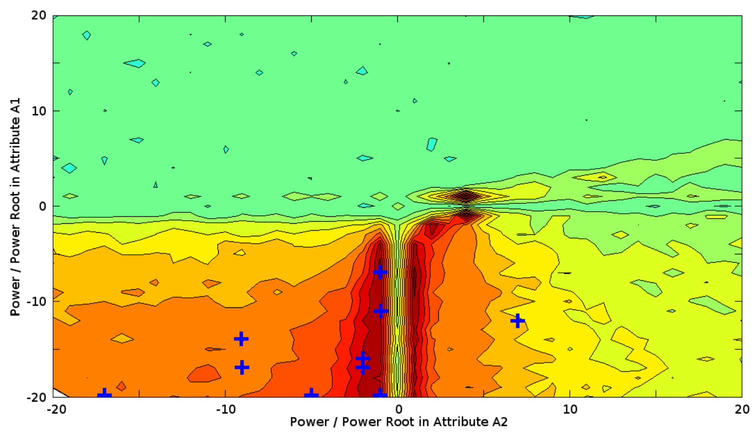
Figure 8.3: Visualisation of the parameter values search space for the *N-Power Calculator* preprocessing method and the Non Linear dataset. Blue marks shows positions of the best-so-far solution of the Steepest descent search algorithm.

two parameters, so I can produce a visualisation and are displayed on the Figure 8.5.

The search space in this case is complete mess. There are no regions of the high fitness and low fitness and the local optima are occurring at random – to be more precise they occur in places where the *SMOTE* algorithm has enriched the dataset in favourable way. This means that the fitness depends more on randomly generated new instances then on the exact value of the parameters. To confirm this conclusion, I have repeated the fitness calculation for all points in the search space 10 times and I have averaged obtained fitnesses. You can examine the three examples of the original search space and averaged space on the Figure 8.5. The single search space seems to be complete mess, but if you take a look on the averaged figure, some interesting patterns appears. In the first place the fitness value seems to be almost constant in the whole search space. On the left margin (for value 0,1,2,3 of the *How many times enrich...* parameter), the fitness is the lower than in the rest of the search space. The similar, but less significant, pattern appears for the *Number of nearest neighbours...* parameter. The fitness is lower for low, less than 10, values of this parameter and rises slowly with higher values of the parameter. The general conclusion of this figure for the *SMOTE* and the Imbalanced dataset is that the only fact which influences fitness in any larger degree, is if I do the data enrichment or not. The exact values of the parameters does not influence the fitness value and the fitness is much more influenced by the random process of enriching the dataset than by the parameters.

I have added visualisation of the progress of the parameter optimisation method. The left top image shows places visited by the Random search parameter optimisation. The top right image shows progress of the Simulated Annealing and the bottom left shows progress of the Steepest descent.





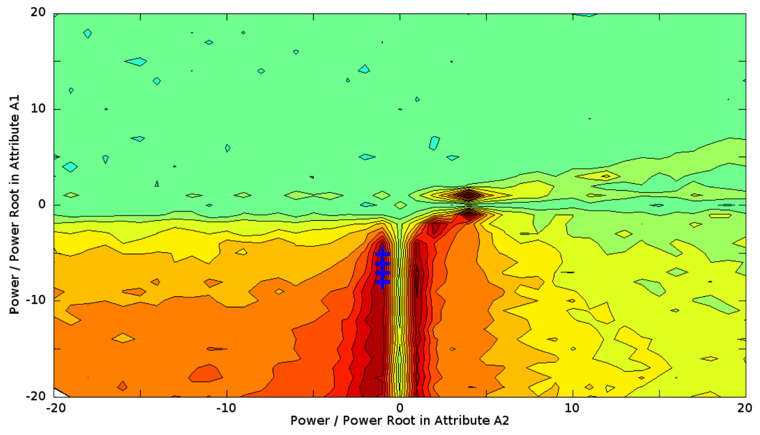


Figure 8.4: Visualisation of the parameter values search space for the *N-Power Calculator* preprocessing method and the Non Linear dataset. Blue marks shows positions of the best-so-far solution of the Differential Evolution search algorithm.

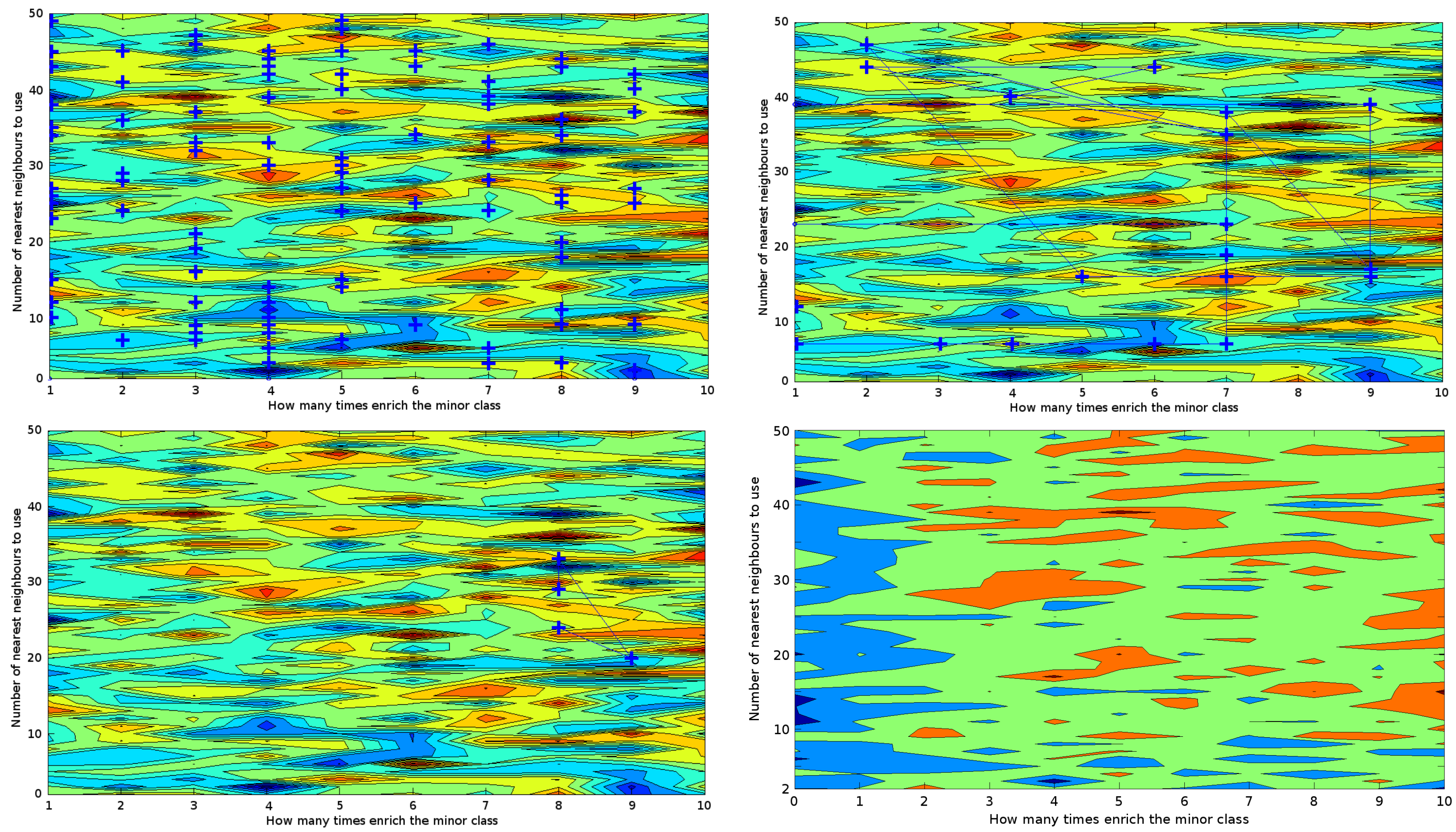


Figure 8.5: Three different visualisations of the fitness for the *Imbalanced* dataset and the *SMOTE* preprocessing method in the repetitive runs and the average of 10 search spaces on the bottom right. The non-average examples of the search spaces also shows the progress of the parameter optimisation methods. The top left image shows points visited by the Random Search algorithm. The top right image shows progress of the Simulated Annealing and the bottom left shows progress of the Steepest descent.

I have investigated more search spaces and you can examine the visualisations of search spaces of two parameters in the Appendix B. As you may examine there, some of the search spaces are quite well defined, with single optimum (like B.14), others has several local optima (e.g. B.32, B.31). In yet other cases, the parameters has no effect on the fitness value (as in B.10, B.7 or most of figures for the Missing dataset – B.1). For each figure in the Appendix B I have written the global minimum fitness and the global maximum fitness. And you can see that in a lot of cases the difference between minimal and maximal fitness value is significant. This means that the random generation of values for parameters is not enough. I can, for example, remove promising sequence of preprocessing methods just because I have randomly generated wrong values for parameters. Therefore I need to incorporate some search for the optimal parameter values to IPT.

8.2 Comparison of the Optimal Parameter Values Search Methods

To find out the best algorithm for the job, I will compare their performance on the artificial datasets. In this section I will demonstrate ability of optimal parameter value search methods to find optimal values for parameters of the preprocessing methods.

8.2.1 Setup of the Experiments

For the experiments I will use the same artificial dataset as in the previous chapter. For the experiment I will randomly select some sequences I have saved in the previous chapter during the search for optimal sequence of preprocessing methods. And I will search for optimal parameter values 10 times in each sequence and then I will compare performance of the search methods on the sequnces.

I will examine the ability of the search methods to achieve the best fitness and at least 95% or 99% of the overall best fitness[1]. I will also examine average improvement of the fitness by the parameter optimisation. This will show how much the optimisation method has improved the fitness of given sequence. The last examined parameter is the average difference between overall best fitness achieved by any optimisation method and the best fitness achieved by given optimisation method. This shows how much is the optimisation method able to get near the optimum.

At this moment I do not know how many optimisation steps I need to finish the search. So I will let all search method to make 100 optimisation steps and I will restart all search methods 5 times. More detailed discussion about this will be in the next section.

8.2.2 Results

In this subsection I will present results of the optimisation method for artificial datasets.

Missing Data dataset The Table 8.1 shows how many times each parameter optimisation search method has won. The most successful method is the Differential Evolution, which has

[1]Note that the selection of the training/testing set and the model learning process contains random part in it, so it is unlikely, that I will get exactly same numbers.

achieved the best fitness in 297 (62.3%) cases. The second is the Steepest Descent, which achieved the best fitness in 90 cases. If count how many times each method has achieved nearly-best solution – 99% respectively 95% of the best fitness, the gap in performance between Steepest Descent and the Differential Evolution vanishes and both are able to achieve high fitness in almost equal number of cases – 408 for the Differential evolution versus 405 for the Steepest Descent. Other two methods achieve much worse results.

Table 8.1: Shows how many time given parameter optimisation method has found the best parameter values for random sequences found for the *Missing Data* dataset and how many times is was able to get anywhere near to the best values.

Search method	Best fitness achieved times	Achieved at least	
		95% of best fitness times	99% of best fitness times
Differential Evolution	297x (62.3%)	408x	429x
Random Search	41x (9%)	316x	410x
Simulated Annealing	27x (5.9%)	326x	397x
Steepest Descent	90x (19.8%)	405x	442x

The Table 8.2 shows the improvement of the fitness during the parameter optimisation. The first column (*Average fitness improvement*) shows the average difference between the fitness in the first optimisation step and fitness in the last optimisation step. In other words it shows the average improvement of the fitness during the parameter optimisation. The higher value, the higher improvement was achieved during the optimisation. The second column (*Average difference between method's and global best fitness*) shows how close the optimisation method is to the best found fitness. The lower value the better. It means that the parameter optimisation method is able to produce setup that is closer to the most optimal setup found.

The highest average improvement in the Table 8.2 shows the Random Search method, but it also shows higher value in the average difference between method's and global best fitness. It means that the Random search in average has improved fitness value the most, but still it is not enough. In average the Differential Evolution and the Steepest Descent methods are able to get very close to the best found value[2].

Table 8.2: Average improvement for random sequences for the *Missing Data* dataset of the fitness during the search for the optimal parameter values and average difference between the search method's best fitness and the best fitness found by any search method.

Search method	Average fitness improvement	Average difference between method's and global best fitness
Differential Evolution	0.022	0.005
Random Search	0.042	0.015
Simulated Annealing	0.037	0.021
Steepest Descent	0.038	0.006

[2]As an reminder – the improvements are calculated for the best so far setup of parameters in each step of parameter optimisation. But for example in Differential evolution I pick the best so far setup among 10 different individuals, in Random Search and Simulated Annealing, there is only one individual to pick from and in Steepest Descent I have as many individuals as there is dimensions.

Imbalanced dataset Table 8.3 shows data for the optimal parameter search in the *Imbalanced* dataset. It shows that the Differential evolution was able to find the best values for parameters in about 60% of cases. The Steepest Descent was successful in about 183 cases, that is about 31%. The remaining two methods has found the best values of parameters in about 8% both. On the other hand, all search methods were able to get over 95% threshold in almost all cases. But this may be explained by small average improvement, as shown in the Table 8.4.

Table 8.3: Shows how many time given parameter optimisation method has found the best parameter values for random sequences found for the *Imbalanced* dataset and how many times is was able to get anywhere near to the best values.

Search method	Best fitness achieved times	Achieved at least	
		95% of best fitness times	99% of best fitness times
Differential Evolution Optimisation	355x (60.7%)	456x	516x
Random Search Optimisation	26x (4.4%)	265x	503x
Simulated Annealing Optimisation	21x (3.6%)	247x	502x
Steepest Descent Optimisation	183x (31.3%)	377x	495x

For this dataset the Table 8.4 shows that the average difference between method's best fitness and the best found fitness is the same for all parameter value search methods. The explanation is, that preprocessing methods used for this dataset depends very little on parameters of preprocessing methods (see the Appendix B). The column *Average fitness improvement* is a little bit more interesting. The it shows the biggest improvements for the Simulated Annealing. But it is caused by the fact that in the beginning the Simulated annealing generates very poor values for parameters and then it has big room for improvement.

Table 8.4: Average improvement for random sequences for the *Imbalanced* dataset of the fitness during the search for the optimal parameter values and average difference between the search method's best fitness and the best fitness found by any search method.

Search method	Average fitness improvement	Average difference between method's and global best fitness
Differential Evolution	0.021	0.034
Random Search	0.037	0.046
Simulated Annealing	0.047	0.046
Steepest Descent	0.038	0.046

Non Linear dataset The results for the *Non Linear* dataset in the Table 8.5 shows again the Differential Evolution as the best optimisation method. It wins in more the 65% of cases. The next in a row is the Steepest Descent but the difference is huge. Steepest descent has won only in 20% of runs. The remaining methods – the Random search and Steepest descent shows minor win rate.

The Table 8.6 shows the improvement of fitness of the best so far individuals. The fitness improve-

Table 8.5: Shows how many time given parameter optimisation method has found the best parameter values for random sequences found for the *Non Linear* dataset and how many times is was able to get anywhere near to the best values.

Search method	Best fitness achieved times	Achieved at least	
		95% of best fitness times	99% of best fitness times
Differential Evolution Optimisation	304x (65.4%)	401x	434x
Random Search Optimisation	42x (9%)	155x	370x
Simulated Annealing Optimisation	23x (4.5%)	154x	341x
Steepest Descent Optimisation	96x (20.6%)	260x	435x

ments are almost equal for the Random, Simulated Annealing and Steepest Descent parameter optimisation methods. The Differential Evolution shows a little bit lower average improvement, but it may be attributed to the fact, that there are several individuals in each optimisation step (generation).

The third column (the Average difference between method's and global best fitness) only confirms results concluded from the Table 8.6. The lowest difference is for the Differential evolution, the second lowest is for the Steepest descent, followed by the Random Search and the Simulated Annealing.

Table 8.6: Average improvement for random sequences for the *Non Linear* dataset of the fitness during the search for the optimal parameter values and average difference between the search method's best fitness and the best fitness found by any search method.

Search method	Average fitness improvement	Average difference between method's and global best fitness
Differential Evolution	0.044	0.009
Random Search	0.052	0.025
Simulated Annealing	0.055	0.032
Steepest Descent	0.053	0.013

Outlier dataset The Table 8.7 shows that the Differential Evolution search method is again the best. Although not so superior to others as in previous datasets (won only in 54% cases), it is still far the best method. It is again followed by the Steepest Descent, who won in 28% of cases.

The Table 8.4 again shows the average fitness improvements for all methods and also difference between method's and global best fitness. The highest improvement was achieved by the Differential evolution and it is in average the closest to the best found solution. Also the Steepest Descent method is in average quite close to the best found solution. Remaining two methods are quite behind.

The conclusion of this part is that the Differential evolution is the best in finding optimal solution. In all datasets the Differential Evolution is able to find the best setup of parameters in more the

Table 8.7: Shows how many time given parameter optimisation method has found the best parameter values for random sequences found for the *Outliers* dataset and how many times is was able to get anywhere near to the best values.

Search method	Best fitness achieved times	Achieved at least	
		95% of best fitness times	99% of best fitness times
Differential Evolution Optimisation	352x (54.1%)	458x	519x
Random Search Optimisation	75x (11.5%)	113x	395x
Simulated Annealing Optimisation	42x (6.5%)	133x	389x
Steepest Descent Optimisation	181x (27.8%)	291x	519x

Table 8.8: Average improvement for random sequences for the *Outlier* dataset of the fitness during the search for the optimal parameter values and average difference between the search method's best fitness and the best fitness found by any search method.

Search method	Average fitness improvement	Average difference between method's and global best fitness
Differential Evolution	0.12	0.021
Random Search	0.072	0.045
Simulated Annealing	0.074	0.046
Steepest Descent	0.071	0.024

half of cases. Therefore from the highest fitness point of view the Differential Evolution is clearly the best choice. The second in row is the Steepest Descent which is also quite successful in finding optimal or nearly optimal setup of parameters.

8.3 Minimal Length

The previous section I have examined the ability of the optimal parameter search methods to find the best possible solution. In this section I will examine number of optimisation steps needed to achieve the best setup of parameters. The parameter optimisation is quite time consuming and I want to make as little optimisation steps as possible.

To examine the number of steps needed, I will examine results of the same experiment as in previous section.

8.3.1 Results

The Table 8.9 shows results for the *Missing Data* dataset. In the table you can examine the average, median and 75^{th} percentile number of steps needed to achieve the best solution. Eg. the median for Differential Evolution of 52 steps means that 50% of optimal parameter searches has found the best setup in less than 52 steps and 50% of parameter searches needed more the 52 steps to find the optimal values of parameters. The value of median of 52 steps is close to average number of steps (51.3 steps). Similarly for the 75^{th} percentile, the 75% of the parameter searches have found the best setup in less then 78 steps. The similar conclusions can be drawn for all the parameter optimisation methods in the Table 8.9. To tell the truth, these results are in a bit disappointing. I have expected that the search algorithm will find the best setup in some number of steps and then it will keep searching but not finding the better and better setups. But the results in the Table 8.9 show that the sequences keep finding new and better setups.

Table 8.9: Average, median and 75th percentile number of optimisation steps to achieve the best fitness for the *Missing Data* dataset.

Search method	Average steps to achieve best	Median steps to achieve best	75 percentile of steps to achieve best
Differential Evolution	51.3	52	78
Random Search	49	49	75
Simulated Annealing	49.2	53	74
Steepest Descent	47	49	71

Table 8.10: Average, median and 75th percentile number of steps to achieve the best fitness for the *Imbalanced* dataset.

Search method	Average steps to achieve best	Median steps to achieve best	75 percentile of steps to achieve best
Differential Evolution	47.29	47	75
Random Search	46.87	44	69
Simulated Annealing	44.92	48	75
Steepest Descent	44.84	43	69

Table 8.11: Average, median and 75th percentile number of steps to achieve the best fitness for the *Non Linear* dataset.

Search method	Average steps to achieve best	Median steps to achieve best	75 percentile of steps to achieve best
Differential Evolution	50.56	51	74
Random Search	45.79	47	70
Simulated Annealing	41.8	39	69
Steepest Descent	44.8	41	72

Non dataset The similar conclusions may be drawn from the Tables 8.10 and 8.11 for the *Imbalanced* and the *Non Linear* datasets. There is a small exception in the Simulated Annealing for the *Non Linear* dataset. For the Simulated Annealing, the 50% of the best parameters are found in the 39 optimisation steps. But it is connected to the fact that the Simulated Annealing gets stuck in some local optima and can not find any better parameter values.

The only dataset where the median number of steps needed to find the best values of the parameters is the *Outlier* dataset. The median for Random Search, Simulated Annealing and the Steepest Descend is quite low as well as the 75^{th} percentile. The reason is the shape of the parameter search space, which is very flat and these methods tends to quickly end in some local optima. One of the search spaces was presented on the Figure B.26. But the Differential Evolution though it is improving constantly is in the end able to find better solution than the other search methods.

Table 8.12: Average, median and 75th percentile number of steps to achieve the best fitness for the *Outlier* dataset.

Search method	Average steps to achieve best	Median steps to achieve best	75 percentile of steps to achieve best
Differential Evolution	51.9	52	78
Random Search	30.19	20	58
Simulated Annealing	27.99	10	58
Steepest Descent	32.18	24	59

To put all above results into one big picture, the Simulated Annealing finds the optimal solution in the lowest number of optimisation steps, but also achieves the lowest performance. On the other hand the Differential Evolution needs the highest number of optimisation steps to find the best values for parameters. This is the most visible in case of the Outlier dataset. All methods have median number of steps less then 25 steps and the Differential needs in average more than 50 steps. This suggests that longer the Differential evolution runs, the better solution it produce and the Differential Evolution should run as long as possible.

One possible explanation is that the Random Search, Simulated Annealing and Steepest Descend are stuck in local optima and are unable to get out of it because they try only positions in neighbourhood of current setup. While the Differential Evolution still generates new setups and from time to time finds slightly better solution. If this is true, the results of the Differential Evolution will improve greatly in the beginning and then, after some point, the fitness improves only little. I will test this hypothesis in the next section.

8.4 Performance of Shorter Parameter Optimisation

As stated above, I presume that the improvements in parameter setups after some point are mainly due to random initiation of model and selection of training and testing sets. Since the parameter optimisation is only supplement to the search for optimal preprocessing methods and will run many times, I want to keep the parameter optimisation as short as possible. To confirm or reject my hypothesis, I will do only 1, 10, 25 and 50 optimisation steps and I supplement results presented above.

In this section I will not be focused on exact numbers but more on trends and I will present the results mainly in forms of graphs.

As usually I will begin with the Missing data dataset. The figure 8.6 shows number of cases in which given optimisation method is able to achieve the highest fitness. Eg. after 10 steps the Differential Evolution is able to find the setup of parameters with highest fitness in 228 cases, the Random Search is able to do the same in 176 cases, the Steepest descend in 55 times and the Simulated Annealing won only 8 times. The figure shows the following picture: in the beginning the Random

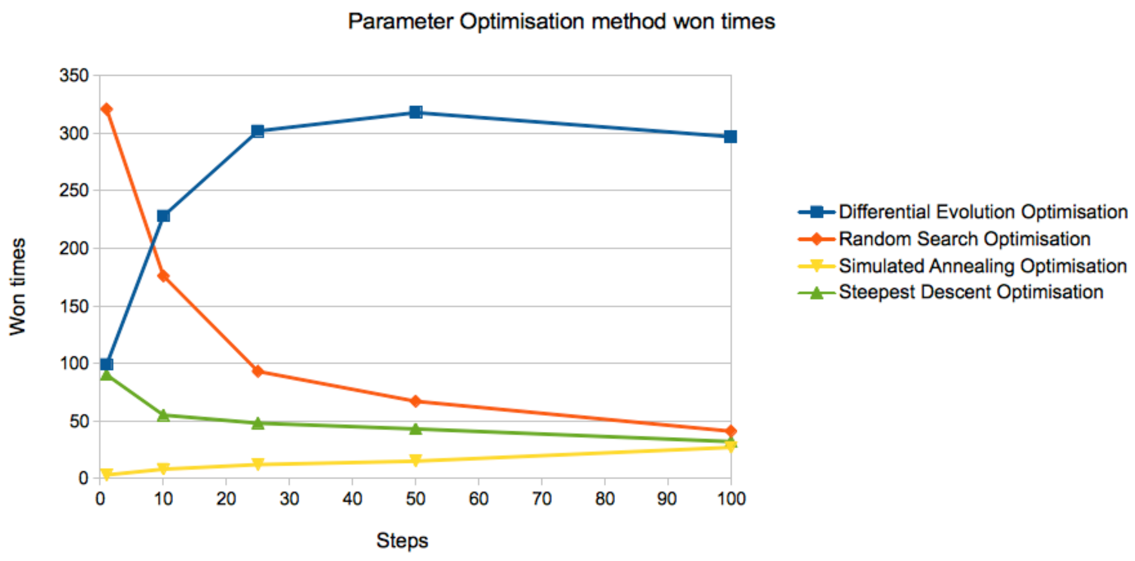


Figure 8.6: Illustrates number of cases in which the given parameter optimisation method is able to find the best setup.

Search is the most successful method. As the optimisation process progresses the Differential Evolution quickly becomes the most successful. The highest point of the most successful count is at 50 optimisation steps. The Simulation Annealing begins to gain a little, but the difference between these two is enormous.

The Figure 8.7 shows progress of the fitness value in relation to the initial fitness and to the best found fitness.

On the left side it is illustrated average difference between found fitness and the best fitness value found in all 100 optimisation steps by any method for given setup of preprocessing methods. The ideal value is 0.0. This value means that the parameter optimisation method is able to find the best fitness every time. The values higher than zero indicates that the method sometimes failed to find the best fitness.

The left part of the Figure 8.7 shows very important thing, that the highest improvement is in the beginning – in the first 10 or 20 steps. After that point the improvement rate fells. In rough terms about a half of improvement is done in the first 20 steps.

The right side of the Figure 8.7 is another progress measure. It shows the improvement from start. In the beginning (the first step) is zero and as the optimisation progresses and finds better and better setups, it rises until it reaches the best fitness. All curves in the figure are averages.

The conclusions from this figure is pretty much the same as for the left part. About a half of improvement from the start is done in first 10-20 optimisation steps. The other half of fitness improvement is done in the remaining 80 optimisation steps.

Very similar conclusions can be stated for all remaining datasets. You can examine the graphs on the Figures C.3, C.4, C.5, C.6, C.7, C.8, C.9,C.10 and C.11 in the Appendix C on page 162.

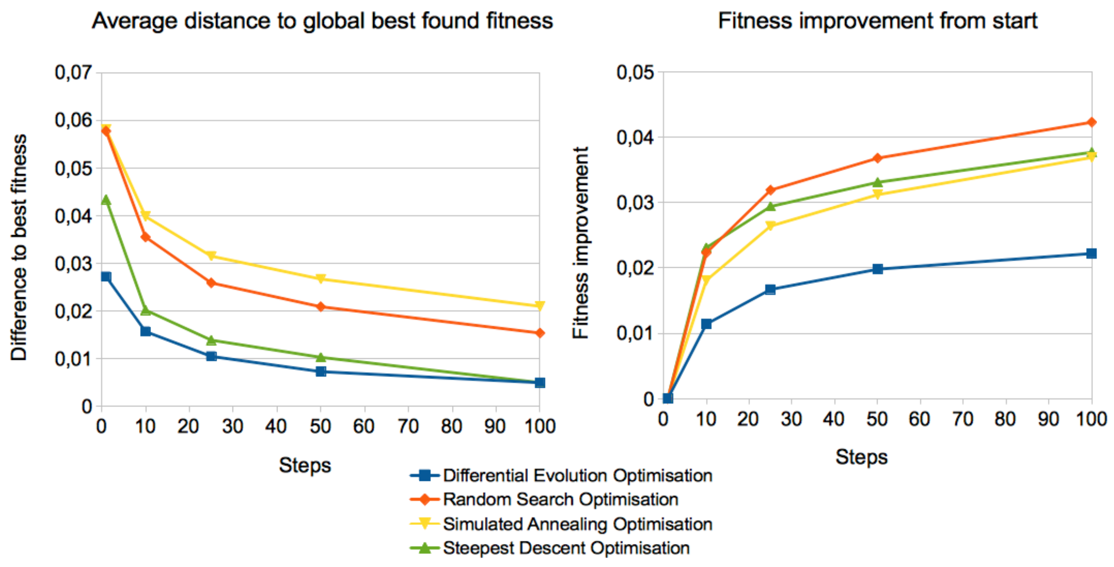


Figure 8.7: Fitness improvement for the Missing Data dataset. On the left is the difference between average fitness in given optimisation step and the best fitness ever found (Lower values are better). On the right is the improvement of fitness from the initial value in the first step (Higher value is better).

8.5 Conclusion

The important conclusion of this chapter is that the parameters of preprocessing methods can affect performance of the model and so affects the fitness value.

I have tested several parameter optimisation methods and the Differential Evolution able to find the best solution in vast majority of cases. It is not the fastest method but produce the best results, in terms of improvement of model accuracy and in its ability to find the best solutions.

9 Search for Sequences with Parameter Optimisation

In the previous chapter I have tested and verified the very important fact, that the parameters of data preprocessing methods influence resulting preprocessed data and in this way influence the fitness value – in other words they influence the accuracy of the modelling method.

The Chapter 7 presented how the search for the preprocessing methods work. In this chapter I will put these together and I will test an ability of Inductive Preprocessing Technology to find the best preprocessing methods for data as well as optimal parameters.

9.1 Experimental Setup

It is very time consuming to compute results for the all combinations of parameter optimisation methods and searches for optimal preprocessing methods. Therefore I will use only Differential Evolution for parameter optimisation and compare it to Random parameter optimisation. For search of preprocessing methods I will use the Genetic search, Steepest Descent search and Random search.

In contrast to the experiment conducted in the Chapter 7, values of the parameters are not hard coded to the correct values but IPT has to find the correct parameter values itself. In the beginning of the search for sequence of preprocessing methods, all the parameters are set to random values. Below is description how exactly incorporate the parameter optimisation into different search methods.

As explained in the Chapter 8 the parameter optimisation can be relatively short, because the biggest improvement is in the beginning of the parameter optimisation. Therefore I have decided to make only a limited number of parameter optimisation steps. This number slightly differs in different situations and is explained below.

To test which combination works the best I will measure the achieved fitness (model accuracy) as well as a number of the preprocessing method search steps needed to achieve the best fitness. I will repeat each combination 10 times to confirm the results.

To demonstrate that the parameter optimisation has any real influence in IPT, I will compare above results with an experiment, where there will be no parameter optimisation and values of the parameters will be random constant value. Its value will be determined at moment, when the preprocessing method and its parameters are added to the sequence of preprocessing methods.

9.1.1 Parameter Optimisation in Genetic Search

The parameter optimisation can be very naturally fitted in the mutation step of the genetic algorithm. The only drawback is the time needed to finish the optimisation, therefore it is not possible to optimise parameters for all sequences of preprocessing methods in all generations.

I have decided to use following rules:

- For all **Elite individuals**[1] I will conduct 5 steps of the parameter value optimisation.
- For newly added individuals I will conduct 20 parameter optimisation steps.
- The parameter optimisation is conducted only if the given individual is selected for mutation. In this case I will conduct 10 steps of the parameter optimisation.
- I will conduct 10 parameter optimisation steps for all individuals in the initial population.

9.1.2 Parameter Optimisation in Steepest Descent Search

In this search method, I have decided to conduct 5 parameter optimisation steps every time the Steepest descent test a new preprocessing method. And after the Steepest Descent decides which preprocessing method to use, it does 10 steps of the parameter optimisation.

9.1.3 Parameter Optimisation in Random Search

I will make 10 optimisation steps in every step of search for the sequences of preprocessing methods.

9.1.4 One Random Change

Apart of the parameter optimisation as described above and in the previous chapter I have decided to test one more "optimisation method". It changes a randomly values of a selected parameters. Although it is not an optimisation method in common sense it is fast, since there are no additional fitness evaluation, and in case of the Genetic Search very natural approach.

9.2 Results

In this section I will present the results. There are two principal questions I want to answer by these results. First I want to know if the accuracy of the models with randomly initialised parameters and parameter values optimisation is comparable to the accuracy of the models with preset parameters as in the Section 7.4. The second question is speed (number of generations) of the search for the preprocessing methods and which parameter values optimisation methods is the fastest or produces the most reliable results.

9.2.1 Missing Data Dataset

As usual at first I will describe the results for the Missing Data dataset. The Table 9.1 summarises the results. The table shows the achieved accuracy as well as a number of search steps needed to find it. The ideal solution is to find the combination of the search for optimal sequence and parameter setup optimisation method which achieves the highest fitness (accuracy) in the smallest number of steps.

[1]First 5 individuals with the highest fitness. See 5.2.5 for details.

The results in the Table 9.1 contains a mean value for each variable and a range given by 3 times standard deviation.

As you can see in the Table 9.1 the best accuracy is achieved by the Genetic Search. The same results as was obtained the Table 7.9 (page 53). The other two search methods had achieved slightly worse fitness. The Steepest descend achieves fitness about 0.015 worse than the Genetic search, that is the models achieve about 1.5% worse accuracy. The Random search is even worse. Its fitness is lower by 0.035 or 3.5% in accuracy. The values for the Random Search with the Differential Evolution and the Random Search with the Steepest Descent are missing because IPT has not finished in reasonable amount of time.

The parameter optimisation methods with the Genetic Search does not affect the fitness too much. The difference in fitness is less than 0.005. The highest fitness is achieved by Genetic Search in combination with the Differential Evolution parameter optimisation – 0.993 and the lowest is in combination with the Random parameter optimisation – 0.99.

Table 9.1: *Missing dataset* – accuracy of the model on validation part of the dataset and estimated number of fitness evaluations needed to find the best sequence.

Search method	Parameter Optimisation Method			
	Differential Evolution	Random Parameter Search	One random change	Steepest Descent
	Best fitness [Steps]	Best fitness [Steps]	Best fitness [Steps]	Best fitness [Steps]
Random Search	NA NA	0.955 ± 0.044 [193.8 ± 72.6]	0.955 ± 0.034 [160.6 ± 229]	NA NA
Steepest Descend	0.976 ± 0.048 [197.6 ± 333]	0.976 ± 0.011 [176.8 ± 159]	0.977 ± 0.011 [208 ± 279]	0.978 ± 0.011 [197.6 ± 116.7]
Genetic Search	0.993 ± 0.01 [600 ± 925]	0.99 ± 0.006 [470 ± 1125]	0.992 ± 0.011 [530 ± 505]	0.991 ± 0.007 [820 ± 1725]

I have some comments on number of steps needed to find the best sequence and optimal values of parameters. Judging from values is a little bit tricky. For the Random Search, Simulated Annealing and the Steepest Descent searches the number of steps correspond to a number of fitness evaluations before the best parameters are found. For the Genetic Search the Table 9.1 presents a number of generations needed to find the sequence with the highest fitness value. But each generation consists of 50 sequences of preprocessing methods. To get comparable numbers I have multiplied the number of generations needed to find the best setup by the number of individuals. The number of generations are summarised in the Table 9.2. As you can see the number of individuals evaluated by the Genetic Search is enormous and the number of steps does not involve the number of fitness evaluations needed to optimise the parameters.

Table 9.2: Number of generations in Genetic Search needed to find the optimal setup for the Missing data dataset.

Search method	Parameter Optimisation Method			
	Differential Evolution	Random Parameter Search	One random change	Steepest Descent
	Generations	Generations	Generations	Generations
Genetic Search	12 ± 18.5	9.4 ± 22.5	10.6 ± 10.1	16.4 ± 34.5

Before I will pronounce my conclusion, I have to take into account yet another two factors. The first is the complexity of the created optimal sequence. As shown in the Section 7.5 the Genetic Search produces the sequences with the lowest number of preprocessing methods. The other factor is that the Random Search and in lesser degree the Steepest Descend sometimes get stuck – precisely sometimes they produce sequences which takes a very long time to preprocess the data. For this reason takes much longer time to finish than the Genetic Search. For example they enrich dataset several times and then applies for example the LOF outlier detection method with $O(N^2)$ running time.

For these reasons I pronounce the Genetic search as the best method for the *Missing Data* dataset. The results of the parameter optimisation methods are quite similar. But they differ in number of generations needed to find the best sequence. The fastest is the Random Search followed by the One Random Change.

9.2.2 Imbalanced dataset

The results for the Imbalanced dataset are shown in the Table 9.3. As above, the Genetic Search has achieved the highest fitness or the best accuracy. The Random search has achieved fitness lower in average by 0.02 and the Steepest Descend in average lower by 0.01 or by 2% in accuracy for the Random Search and by 1% for the Steepest Descend.

Table 9.3: *Imbalanced dataset* – accuracy of the model on validation part of the dataset and estimated number of fitness evaluations needed to find the best sequence.

Search method	Parameter Optimisation Method			
	Differential Evolution	Random Parameter Search	One random change	Steepest Descent
	Best fitness [Steps]	Best fitness [Steps]	Best fitness [Steps]	Best fitness [Steps]
Random Search	0.913 ± 0.007 [83.8 ± 85.7]	0.918 ± 0.005 [126.6 ± 150]	0.907 ± 0.014 [33.2 ± 69.8]	0.911 ± 0.008 [81 ± 251]
Steepest Descend	0.918 ± 0.013 [135.2 ± 289.3]	0.92 ± 0.014 [218.4 ± 182]	0.917 ± 0.008 [145.6 ± 286]	0.922 ± 0.023 [197.6 ± 229]
Genetic Search	0.94 ± 0.014 [940 ± 1575]	0.937 ± 0.008 [490 ± 1100]	0.936 ± 0.002 [210 ± 305]	0.936 ± 0.009 [580 ± 1710]

To get better impression about distribution of best fitnesses achieved by different combinations of the sequence search methods and the parameter optimisation methods I have created boxplots presented on the Figure 9.1. It clearly shows that the best results are achieved by the Genetic Search method for the best sequence of preprocessing methods.

The Table 9.3 also shows the number of sequences needed to examine before the search method finds the best setup. The observations are quite similar to the Missing data dataset presented above. The number of sequences sequences of preprocessing methods tested and examined is the lowest for the Random Search and the highest for the Genetic Search. In the case of the Genetic Search the Table Table 9.3 contains estimated number of the sequences tested before the Genetic Search finds the best sequence. The numbers were obtained as in the previous case by multiplying

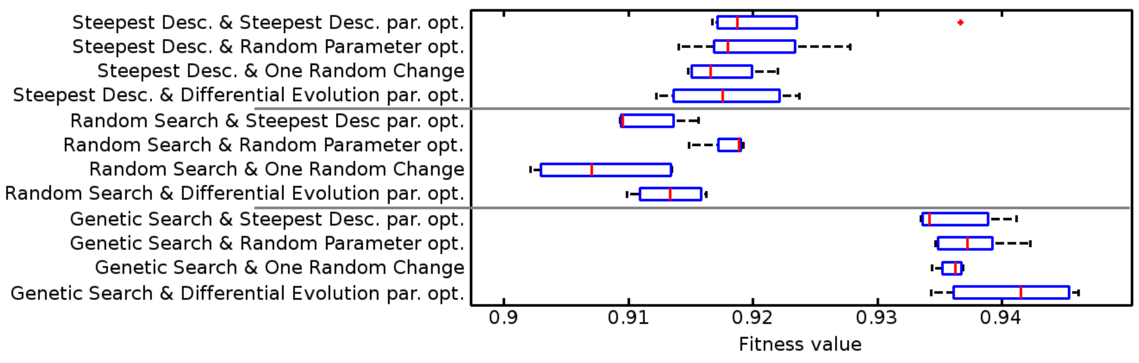


Figure 9.1: The boxplot illustrating distribution of the best fitnesses found by different search methods for the *Imbalanced* dataset.

the number of generations by number of individuals in the population (in this case 50). The number of generations is shown in the Table 9.4.

Table 9.4: Number of generations in Genetic Search needed to find the optimal setup for the Imbalanced dataset.

Search method	Parameter Optimisation Method			
	Differential Evolution	Random Parameter Search	One random change	Steepest Descent
	Generations	Generations	Generations	Generations
Genetic Search	18.8 ± 31.5	9.8 ± 22	4.2 ± 6.1	11.6 ± 34.2

Again the conclusions is similar to the previous section. The highest fitness (or the most accurate) is the Genetic Search, but it needs the most steps or sequence testings to find the best sequence. The other two search methods usually need less then a half number of steps to finish. On the other hand the Random Search and the Steepest Descend Search sometimes get stuck and needs a very long time to finish. Therefore the best though not the fastest method is the Genetic Search.

9.2.3 Non Linear dataset

The Table 9.5 presents results for the *Non Linear* dataset. The best accuracy of the Simple Logistic Regression Classifier with validation part of the dataset is achieved by the sequence found by the Genetic Search in combination with the Differential Evolution parameter optimisation and the Random Parameter Optimisation. The One Random Change parameter optimisation with the Steepest Descent achieves slightly lower accuracy. The big number of cases where IPT did not finish in a reasonable time (two days) is caused by the fact that the Simple Linear Regression classifier can not fit the data. Therefore the training algorithm has to finish all the training iterations and due to enormous number of models that has to be trained IPT takes very long time to finish.

The Table 9.6 shows number of generations needed by the Genetic Search to find the best sequence and its parameters.

Table 9.5: *Non Linear dataset* – accuracy of the model on validation part of the dataset and estimated number of fitness evaluations needed to find the best sequence.

Search method	Parameter Optimisation Method			
	Differential Evolution	Random Parameter Search	One random change	Steepest Descent
	Best fitness [Steps]	Best fitness [Steps]	Best fitness [Steps]	Best fitness [Steps]
Random Search	NA	0.873 ± 0.084 183 ± 90.4	0.898 ± 0.112 [130 ± 262]	NA
Steepest Descend	NA	0.965 ± 0.017 [176 ± 124.8]	0.988 ± 0.016 [280 ± 233.4]	NA
Genetic Search	0.986 ± 0.028 [720 ± 955]	0.986 ± n0.02 [690 ± 695]	0.985 ± 0.006 [1025 ± 1430]	0.984 ± 0.03 [1070 ± 920]

Table 9.6: Number of generations in Genetic Search needed to find the optimal setup for the Non Linear dataset.

Search method	Parameter Optimisation Method			
	Differential Evolution	Random Parameter Search	One random change	Steepest Descent
	Generations	Generations	Generations	Generations
Genetic Search	14.4 ± 19.1	13.8 ± 13.9	20.4 ± 28.6	21.4 ± 18.4

9.2.4 Outlier dataset

The Table 9.7 summarises the results for the Outlier dataset. Again the highest fitness (or accuracy) is achieved by the Genetic Search, its outperform the Steepest Descend by about 1% to 4% in accuracy and the Random Search by 4% to 8%.

Table 9.7: *Outlier dataset* – accuracy of the model on validation part of the dataset and estimated number of fitness evaluations needed to find the best sequence.

Search method	Parameter Optimisation Method			
	Differential Evolution	Random Parameter Search	One random change	Steepest Descent
	Best fitness [Steps]	Best fitness [Steps]	Best fitness [Steps]	Best fitness [Steps]
Random Search	0.935 ± 0.04 [110.2 ± 219]	0.93 ± 0.054 [151.4 ± 228.5]	0.892 ± 0.092 [98.6 ± 158.5]	0.92 ± 0.033 [114.8 ± 269.3]
Steepest Descend	0.951 ± 0.057 [187.2 ± 425.5]	0.963 ± 0.069 [270 ± 268]	0.932 ± 0.036 [218.4 ± 182]	0.948 ± 0.012 [260 ± 382]
Genetic Search	0.967 ± 0.036 [1320 ± 1735]	0.964 ± 0.04 [880 ± 1500]	0.973 ± 0.055 [880 ± 1275]	0.988 ± 0.022 1075 ± 1555

To get better impression about distribution of best fitnesses achieved by different combinations of the sequence search methods and the parameter optimisation methods I have created the boxplot presented on the Figure 9.2. In this case the Genetic Search is clearly better than the Random Search, but in the case of the Steepest Descent the situation is not so clear and in many cases the Steepest descent is able to find fitness value as high as the Genetic Search.

The fastest method method to find its top fitness is in general the Random Search. But its best

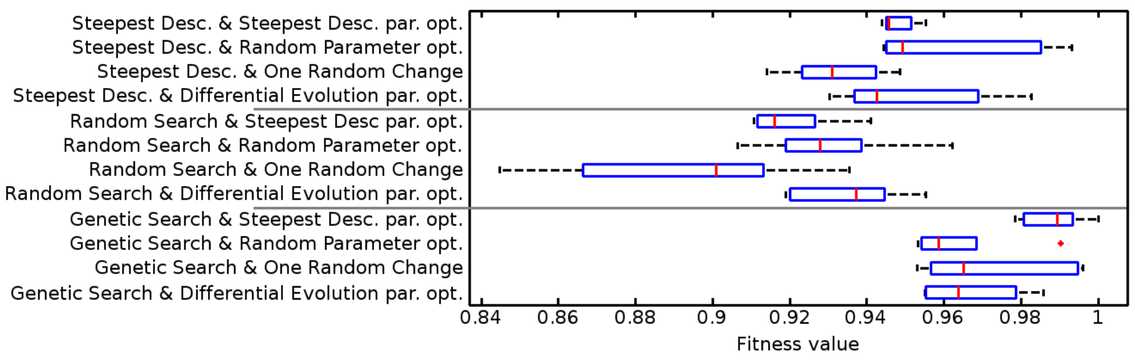


Figure 9.2: The boxplot illustrating distribution of the best fitnesses found by different search methods for the *Outlier* dataset. The red stripe in the middle shows the mean fitness, the blue box covers 50% of values (range between 25th and 75th percentile) and the black lines covers range between 5th and 95th percentile, covering 90% of values.

sequences achieves the lowest fitness in comparison to sequences found by the Genetic Search and the Steepest descend. On the other hand the Genetic Search has found the sequences with the highest fitness (best accuracy) but needs the most number of evaluations to find it. In contrast to other datasets the difference in fitness is relatively high, more than 0.08 (or 8% in accuracy).

Table 9.8: Number of generations in Genetic Search needed to find the optimal setup for the Imbalanced dataset.

Search method	Parameter Optimisation Method			
	Differential Evolution	Random Parameter Search	One random change	Steepest Descent
	Generations	Generations	Generations	Generations
Genetic Search	26.4 ± 34.7	17.6 ± 30	17.6 ± 25.5	21.6 ± 31.1

In spite of the number of steps to finish I favour again the Genetic Search. It achieves the highest fitness and more it finds the sequences with lower number of preprocessing methods. The other drawback of the Random and Steepest Descent searches is that they get stuck and are not able to finish in a reasonable time.

9.2.5 Selection of the Parameter Value Optimisation method

As shown above, there is a clear answer to the question about the best sequence search method. In case of the parameter value optimisation method the answer is not so clear. The number of steps and achieved fitness for all parameter value optimisation methods and all datasets are summarised in the Table 9.9. In the Table I have highlighted the most accurate method and the one needing the least number of individuals to optimise. The best optimisation method seems to be the Differential Evolution and the Random Parameter Search. The fastest are the One Random Change and the Random Parameter Search.

The thing I have completely omitted from my descriptions and calculations in this chapter is a number of fitness evaluations and number of models trained needed to finish the parameter optimisations. In the Random Search and Steepest descend parameter optimisation one step means

Table 9.9: Comparison of parameter optimisation methods performance in different datasets.

Search method	Parameter Optimisation Method			
	Differential Evolution	Random Parameter Search	One random change	Steepest Descent
	Best fitness [Steps]	Best fitness [Steps]	Best fitness [Steps]	Best fitness [Steps]
Missing Data Dataset	**0.993 ± 0.01** [600 ± 925]	0.99 ± 0.006 [470 ± 1125]	0.992 ± 0.011 **[530 ± 505]**	0.991 ± 0.007 [820 ± 1725]
Imbalanced dataset	**0.94 ± 0.014** [940 ± 1575]	**0.937 ± 0.008** [490 ± 1100]	0.936 ± 0.002 **[210 ± 305]**	0.936 ± 0.009 [580 ± 1710]
Non Linear Dataset	0.986 ± 0.028 [720 ± 955]	**0.986 ± 0.02** **[690 ± 695]**	0.985 ± 0.006 [1025 ± 1430]	0.984 ± 0.03 [1070 ± 920]
Outlier Dataset	0.967 ± 0.036 [1320 ± 1735]	0.964 ± 0.04 [880 ± 1500]	0.973 ± 0.055 **[880 ± 1275]**	**0.988 ± 0.022** 1075 ± 1555

250 fitness evaluation that corresponds to training 5000 models. In the Differential Evolution one step means 100 fitness evaluations[2] or training of 1000 models. All this takes a lot of time. In contrast the One Random Change does no optimisation and just randomly changes the values of the parameters. And it is still able to find setup of parameters which is quite close to the best found fitness. Therefore I will use this parameter "optimisation" method from now on.

9.3 Conclusion

In this chapter I have tested the selected combinations of the parameter optimisation methods with the best sequence search methods. In the end the **Genetic Search** shows ability to achieve the highest fitness. But the Random Search and the Steepest Descend are able to find their best fitness in lower number of steps. The difference in fitness is not big. Usually it falls between 0.02 and 0.08 in accuracy of the model trained with the preprocessed validation dataset. In a view of such a big difference in number of steps needed to find its best fitness between the Random Search and the Steepest Descend on one side and the Genetic Search on the other, it seems reasonable to decide to use the Random Search or the Steepest Descend and let them run longer. But there are two powerful reasons to reject the Random Search and the Steepest Descend and favour the Genetic Search. The first is complexity of the found sequences of the preprocessing methods. The sequences found by the Genetic Search are much simpler than the sequences found by the Random Search or the Steepest Descend. The difference is illustrated for example on the Figures 7.10 or 7.14. The second reason is the speed of the search. The Random Search and the Steepest Descend tends to generate complex sequences containing a large number of the preprocessing methods and in some cases such sequences takes a long time to preprocess the dataset. Especially when the dataset contains a lot of instances. The Genetic Search adds or removes preprocessing methods one by one and if a combination of preprocessing methods does not work well, it is not used anymore. And in this way the Genetic Search can finish faster, even though it uses higher number of the fitness evaluations.

The second concern is about the parameter value optimisation methods. As shown in the results

[2]10 individuals in 10 generations

Tables above, the Differential Evolution and the Random Parameter Search achieves the highest fitness. On the other hand, they needs a lot of steps of the optimisation algorithm to finish. The One Random Change "optimisation" method achieves only a bit worse results and it does not need any more fitness evaluations and thus increases the speed of IPT.

So to summarise my selections – the best search method for the best sequence is the Genetic Search and the One Random Change for the parameter optimisation method. And I will use them in the remaining part of my thesis.

10 Real World Datasets for Classification

So far I have presented results the Inductive Preprocessing Technique (IPT) has achieved with the artificial datasets. In this chapter at last I will test the technique with the publicly available real world datasets. The reason to use the artificial datasets in the previous part of my thesis is that I exactly know the best solution for the datasets and the datasets can be easily visualised as well. In the case of the real world dataset the correct preprocessing as well as the visualisation can not be always easily obtained. It is also hard to demonstrate basic properties of my technique when it is not clear what I want to achieve.

10.1 Selected Datasets

Through this chapter I will mainly use following publicly available datasets from the UCI Machine Learning [66] repository.

- **Bank** – Or the Bank Marketing dataset is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be (or not) subscribed [67]. The dataset can be downloaded from [68].

- **Breast-cancer-wiskonsin** – the dataset was introduced by Dr. William H. Wolberg of the University of Wisconsin Hospitals, Madison. This dataset contains Dr. Wolberg's medical cases. Each patient is represents one row in the dataset. Input variables represent characteristic of a cell nucleus from the mammography image. The output is a result from long term survey and indicates if the tumour is benign or malignant. For more information see [69, 70]. Or the web page for this dataset in the repository is [71].

- **CTG** – The dataset consists of measurements of fetal heart rate (FHR) and uterine contraction (UC) features on cardiotocograms classified by expert obstetricians[72]. Web page is [73].

- **Ecoli** – the aim of this dataset is to localise original place in cell for proteins. More information about this dataset can be obtained in [74]. Web page for this dataset at the repository is [75].

- **Glass** – the task is to distinguish type of glass from the chemical analysis [76], web page in the UCI Machine Learning repository is [77].

- **Ionosphere** – this radar data was collected by a system in Goose Bay, Labrador. The targets to track by the system were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere. Instances in this database are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal [78], for web see [79].

- **Parkinson** – This dataset is composed of a range of biomedical voice measurements from 31 people, 23 with Parkinson's disease (PD). Each column in the table is a particular voice characteristic, and each row corresponds to one of 195 voice recording from these individuals. The main aim of the data is to discriminate healthy people from those with PD [80], information online [81].
- **Satellite** – The database consists of the multi-spectral values of pixels in 3x3 neighbourhoods in a satellite image, and the aim is to classify type of ground in the image. Web page [82]
- **Segment** – Or the Image Segmentation dataset contains 3x3 pixels areas from 7 different outdoor images. The task is to identify type of texture in the pixel. The types of texture in the images were manually assigned. See [83].
- **Spambase** – The collection of spam e-mails came from a postmaster and some users who had filed spam. The collection of non-spam e-mails came from filed work and personal e-mails, and hence the word 'george' and the area code '650' are indicators of non-spam. These are useful when constructing a personalised spam filter. One would either have to blind such non-spam indicators or get a very wide collection of non-spam to generate a general purpose spam filter. See web page [84].
- **Steel Faults** – Identifies the common problems in steel surface from the images. More details on the dataset can be found in [85, 86]. It may be downloaded from [87].
- **Wine** – this dataset contains 13 chemical characteristics of different wines. The goal is to determine into which of 3 cultivars given wine belongs [88]. Web page in UCI Repository is [89].

There is one more dataset I will use – the *Teeth Age* dataset. The goal is to estimate the real age by the state of teeth mineralisation. The input attributes are development stages of the teeth on the both sides of the mandible [90, 91]. The dataset was provided by the Faculty of Science, Charles University in Prague in cooperation with the General University Hospital on Karlovo Namesti, the Motol University Hospital and the Fakultní Nemocnice Královské Vinohrady.

The presented datasets are quite well preprocessed. Some are well known for years and were selected for being well transformed and suitable for the most of the modelling methods and for benchmarking purposes. This is not true for the most datasets in the real world praxis. But it is hard to obtain permission to publish real world datasets and obtained results. For this reason I have examined several real world datasets I have been working with. Their common problems were missing values, big difference in number of instances in different classes, random outliers and non-standard distribution of values.

To simulate problems the above mentioned problems in my work I have introduced these problems into the datasets. To be more specific I have artificially damaged randomly selected attributes in the datasets by combination of:

- adding up to 40% of missing values,
- adding outliers,

- non-linear transformations (powers, power roots, logarithms).

The reason to combine the problems is that they do not occur separately in the commercial praxis but always in combination.

10.2 Experimental Setup

To be able to prove correctness of the best found sequence of preprocessing methods I will divide each dataset into two parts – the **training part** and the **validation part**. These has the same reason as in machine learning field. The training part is used to find the best sequence by IPT and the validation part is to check that the found sequence really improves accuracy of the model even on the unknown data with the same properties and the sequence is not "overfitted".

I will continue to use both models – the logistic regression classifier and the J48 decision tree – and I will demonstrate the difference in the preprocessing methods needed to find the most accurate model. I will repeat the search for the best sequences of preprocessing methods 20 times and I will choose the best one among them.

To prevent the classifier overfitting I have set the *splitmin* parameter of the J48 Decision Tree to 10% of each training dataset before preprocessing.

10.2.1 Post Processing

As shown in the chapter 7, the Genetic search produces fairly clean sequences, but still there can be preprocessing methods that has only a small effect on accuracy of the classifier or the sequence preprocesses the attribute that is not used by it. This may happen because of cross over or mutation operations in the few last generations. In this case the genetic search has no time to get rid of these preprocessing methods. Such methods then complicate the understanding what the sequence really does. For this reason I have implemented a simple post processing function to test if it is possible to remove a preprocessing method from the sequence. The post processing tries to disable one preprocessing method after another and tests if the fitness value of the sequence fells below a threshold. If it does not, the preprocessing method does not help to improve the accuracy and remains disabled.

10.3 Results

In this section I will present the results achieved by IPT on real world datasets. In the beginning I will present an accuracy of the models trained on the original (not preprocessed) datasets and I will compare their the accuracy to the models trained with the data preprocessed by the best sequences found by IPT. Later I will describe the best found sequences and I will show how the sequences improved the models.

The Table 10.1 presents accuracy of models for all real world datasets. The accuracies are measured on both – datasets preprocessed with the best sequences found by IPT and original datasets without preprocessing. The accuracy for the original dataset was obtained by 20 repetitive training

Table 10.1: Comparison of an accuracy of the models on (original) not preprocessed dataset and preprocessed by the best sequences found in repetitive runs of IPT.

Dataset name	J48 Decision Tree		Simple Logistic Regression Classifier	
	Not Pre-processed Dataset	Best Pre-processed Dataset	Not Pre-processed Dataset	Best Pre-processed Dataset
Bank	0.884	0.884	0.883	0.884
Breast Cancer Wisconsin dataset	**0.913**	**0.975**	0.92	0.928
CTG dataset	0.81	0.838	0.887	0.905
Ecoli dataset	0.854	0.884	**0.818**	**0.885**
Glass dataset	0.43	0.487	0.554	0.603
Ionosphere dataset	**0.763**	**0.814**	0.885	0.895
Parkinson dataset	0.769	0.781	0.795	0.826
Satellite dataset	**0.498**	**0.695**	**0.728**	**0.76**
Segmentation dataset	**0.401**	**0.791**	0.686	0.702
Spambase dataset	**0.734**	**0.8**	0.848	0.864
Steel Faults dataset	0.437	0.498	0.639	0.654
Teeth Age dataset	**0.689**	**0.747**	**0.648**	**0.751**
Wine dataset	**0.842**	**0.913**	**0.787**	**0.926**

of the models. The accuracy for the preprocessed dataset is an average of fitnesses of the best sequences found in 20 repetitive runs of IPT. To give better impression about the differences in accuracy between the model with original and preprocessed datasets I have created a boxplot for each dataset and model. The boxplots for all datasets and the J48 Decision Tree are shown on the Figure 10.1. The Figure 10.2 shows the boxplots for the Simple Logistic Regression Classifier with the preprocessed and the original datasets.

The Table 10.1 and the Figures 10.1 and 10.2 show that IPT is able to find sequences which transforms a dataset in a form that is easier for the modelling methods.

The improvement in accuracy of the model typically lies somewhere between **5% and 10 %**. But exceptions exists in both directions. The worst performance IPT shows on the *Bank* dataset. In this case the accuracy is not improved at all. IPT is unable to find sequence which improves the accuracy neigther for the J48 model nor for the Simple Logistic Regression model, this suggest that the models perform with the original (not preprocessed) *Bank* dataset in the best possible way. The accuracy improvement for the *Parkinsons* dataset and the J48 classifier is also minimal but at least the average accuracy of the model with preprocessed dataset is a bit higher than the accuracy for the original dataset. The last case with the minimal improvement is the *Ionosphere* dataset with the Simple Logistic Regression classifier – in this case IPT has found a sequence of the preprocessing methods which improve accuracy of the model by about 1.5%.

The opposite extreme represents the *Segment* dataset with the J48 Decision Tree. In this case the accuracy was improved by about 39%. The second highest improvement was achieved for the *Wine* dataset with the Logistic Regression Classifier. The accuracy improvement is about 15%. The accuracy improvement for other datasets is lower than 10%.

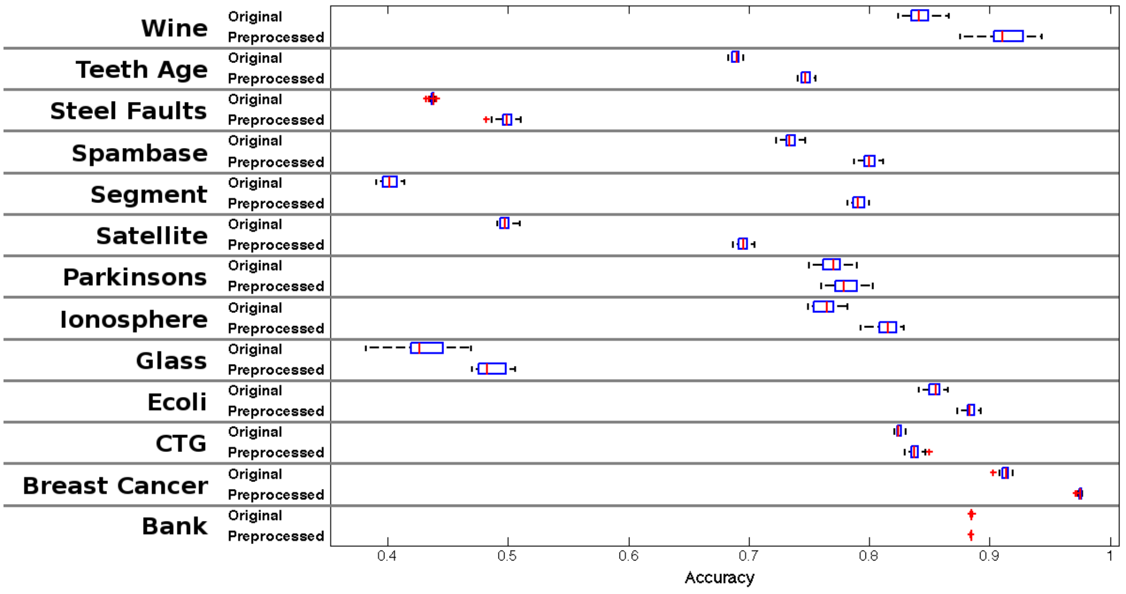


Figure 10.1: Comparison of the accuracies of the J48 Decision Tree. There are two boxes for each dataset. One is marked as *Original* and represents accuracy with original (not preprocessed) dataset. The second is *Preprocessed* and shows accuracy of models with dataset preprocessed with the best sequences found by IPT.

10.4 Results for Selected Sequences

In the previosous text I have illustrated the ability of IPT is find the sequences of preprocessing methods that transform the datasets in such way that model trained with the transformed datasets are more accurate than the models with the original datasets. In this section I will discuss results on the selected datasets and sequences in grater details. The sequences found by IPT for remaining datasets are presented in the Appendix D on the page 168.

10.4.1 Glass dataset and J48 Decision Tree

The first dataset I want to present is the *Glass* dataset. Accuracy of the J48 Decision Tree model with the non preprocessed dataset is about 43%. The improvement in accuracy of the model preprocessed by the best sequences found by IPT (Seq 05) is about 5% – that is the model with preprocessed dataset has accuracy slightly higher than 48%[1].

The Figure 10.3 shows the boxplots for the all 20 best sequences found in all repetitive runs. The label on the left side of the figure indicates the ID of the run in which the sequences were found. The sequence with the highest fitness **Seq05** indicates that the fifth run of IPT has found the best sequence with the highest fitness among all best sequences found in all repetitive runs of IPT. The best sequences found in the individual runs on the Figure 10.3 are sorted according to their mean fitness, sequences with higher fitness are at the bottom. The value at the very bottom represents an accuracy of the J48 model with the original dataset and is here mainly for easier comparison.

[1]There are six classes in the dataset.

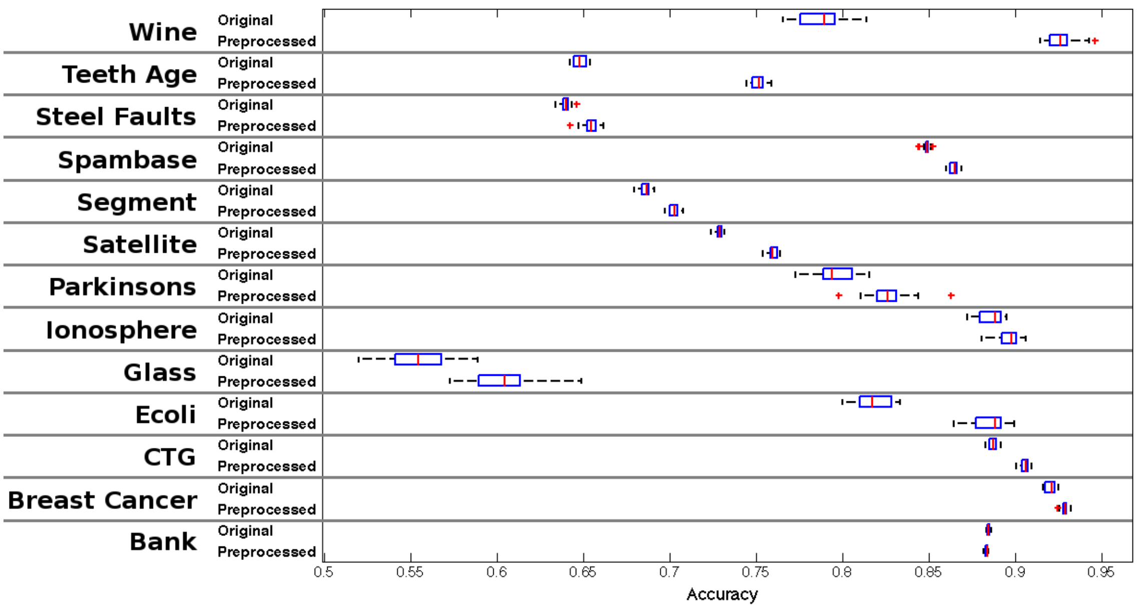


Figure 10.2: Comparison of the accuracies of the Simple Logistic Regression. There are two boxes for each dataset. One is marked as *Original* and represents accuracy with original (not preprocessed) dataset. The second is *Preprocessed* and shows accuracy of models with dataset preprocessed with the best sequences found by IPT.

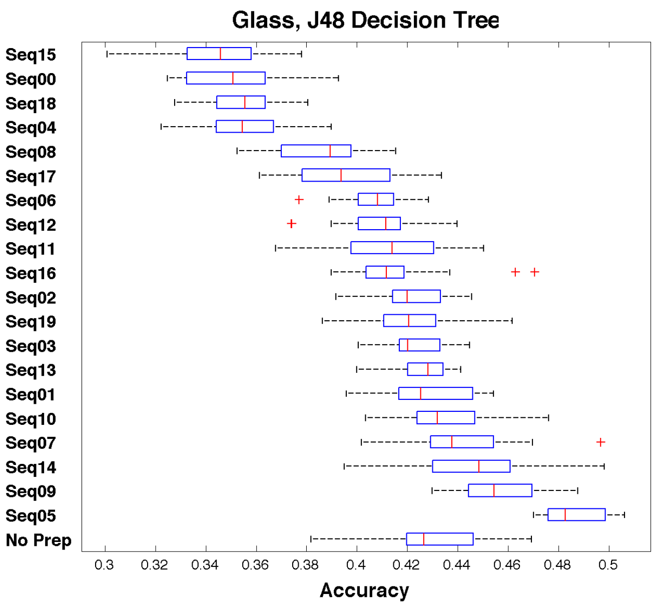


Figure 10.3: Accuracies of J48 Decision Tree classifiers trained on validation *Glass* dataset. The validation dataset was preprocessed by the best sequences of preprocessing methods found in 20 repetitive runs of IPT. Sequences are sorted by mean accuracy of ”their” models. For easier comparison the boxplot representing accuracies of model trained with original validation dataset is on the bottom of the figure.

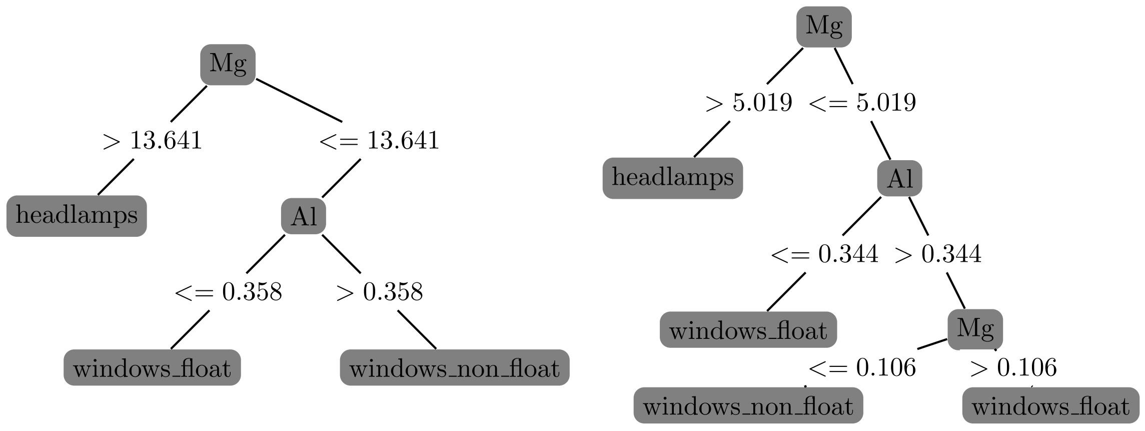


Figure 10.4: Decision trees created for classification of the original *Glass* dataset on the left and for the preprocessed *Glass* dataset by the sequence **Seq05** on the right.

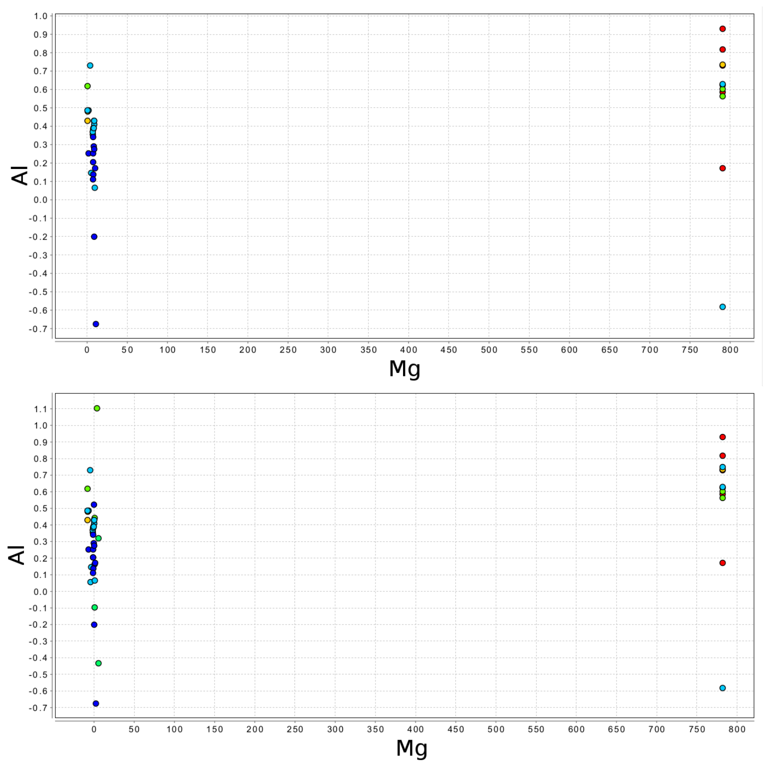


Figure 10.5: Scatter plot for the original *Glass* dataset at the top and preprocessed (in the bottom) *Glass* dataset dataset.

The sequence **Seq05** with the highest fitness is quite lonely in its performance, the other runs of IPT has failed to find sequences which could improve the accuracy of the model over the accuracy of the model with the original dataset.

You can examine all the best sequences found in repetitive runs on the Figure 10.6. The sequences are again sorted according to their fitness, but in this case the sequences with the higher fitness are at the top. The comparison of the two sequences with the highest fitness shows that the transformation of the third attribute – the **Mg** – may be important. Both sequences use the

Nearest Neighbour Missing Value Imputer preprocessing method which replaces missing values with a mean of values from nearby instances. The both sequences significantly differ in a number of nearest instances to use. The sequence **Seq05** uses 10 nearest instances and the **Seq09** uses 3 nearest. I believe that this is a property of this particular dataset and if I add some new instances, the fitness will be lower. But nevertheless it is a property of the dataset and it leads to the improvement of the fitness. It also shows that the one random mutation approach to the optimisation of parameters does not guarantee that the best parameters of preprocessing methods are always found.

The Figure 10.4 shows two J48 Decision Trees. The tree on the left side is generated from the original, not preprocessed, dataset and the tree on the right was generated from the dataset preprocessed by the **Seq05**. Both trees have the **Mg** attribute in the root, but the tree for the preprocessed dataset on the right uses different values and the learning algorithm then divides the rightmost note in the tree by the values of the **Mg** attribute.

The last Figure 10.5 shows the scatter plot of the attributes used in the decision tree. The upper scatter plot represents the original dataset without any preprocessing, while the bottom scatter plot represents the preprocessed dataset. The plot for the original dataset is missing some dots – the ones with missing values in at least one of the **Mg** or **Al** attributes. This is also the reason why the decision tree for the original dataset is one node shorter. The decision tree for the dataset preprocessed by the sequence **Seq05** has missing values replaced. And the imputed values, thanks to other values in the dataset, puts the instances into places where they may be correctly classified.

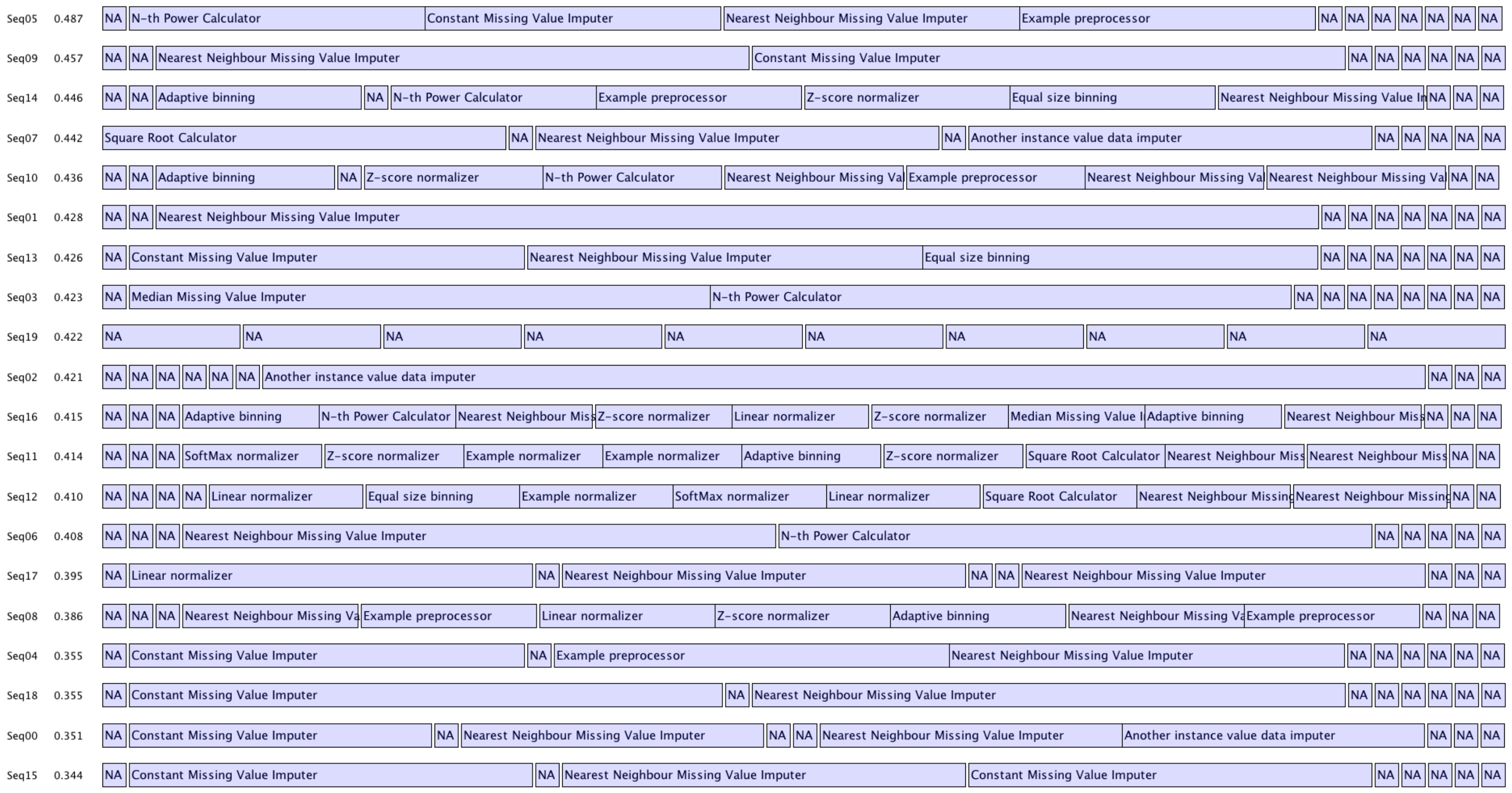


Figure 10.6: The best sequences found in repetitive runs of IPT for the *Glass* dataset and the J48 Decision Tree.

10.4.2 Segment dataset and J48 Decision Tree

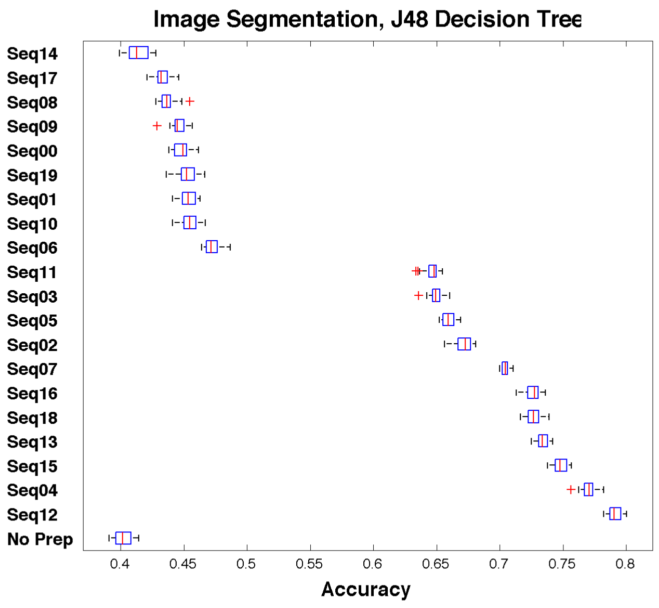


Figure 10.7: Accuracies of J48 Decision Tree classifiers trained on the validation *Segment* dataset. The validation dataset was preprocessed by the best sequences of preprocessing methods found in 20 repetitive runs of IPT. Sequences are sorted by mean accuracy of "their" models. For easier comparison the boxplot representing accuracies of model trained with original validation dataset is on the bottom of the figure.

This subsection discusses the *Segment* dataset with the J48 Decision Tree. I will again start with the Figure 10.7. This figure illustrates fitness of all the best sequences found by all 20 repetitive runs of IPT. The sequences with the highest fitness are again in the bottom. On the very bottom, denoted as **No Prep**, is accuracy of the J48 model trained with the original dataset. The situation here is better and in the case of the *Glass* dataset. The sequence with the highest fitness is closely followed by other sequences with lower but similar fitness. It seems that the sequences up to **Seq16** should be similar, at least they achieve very high fitness.

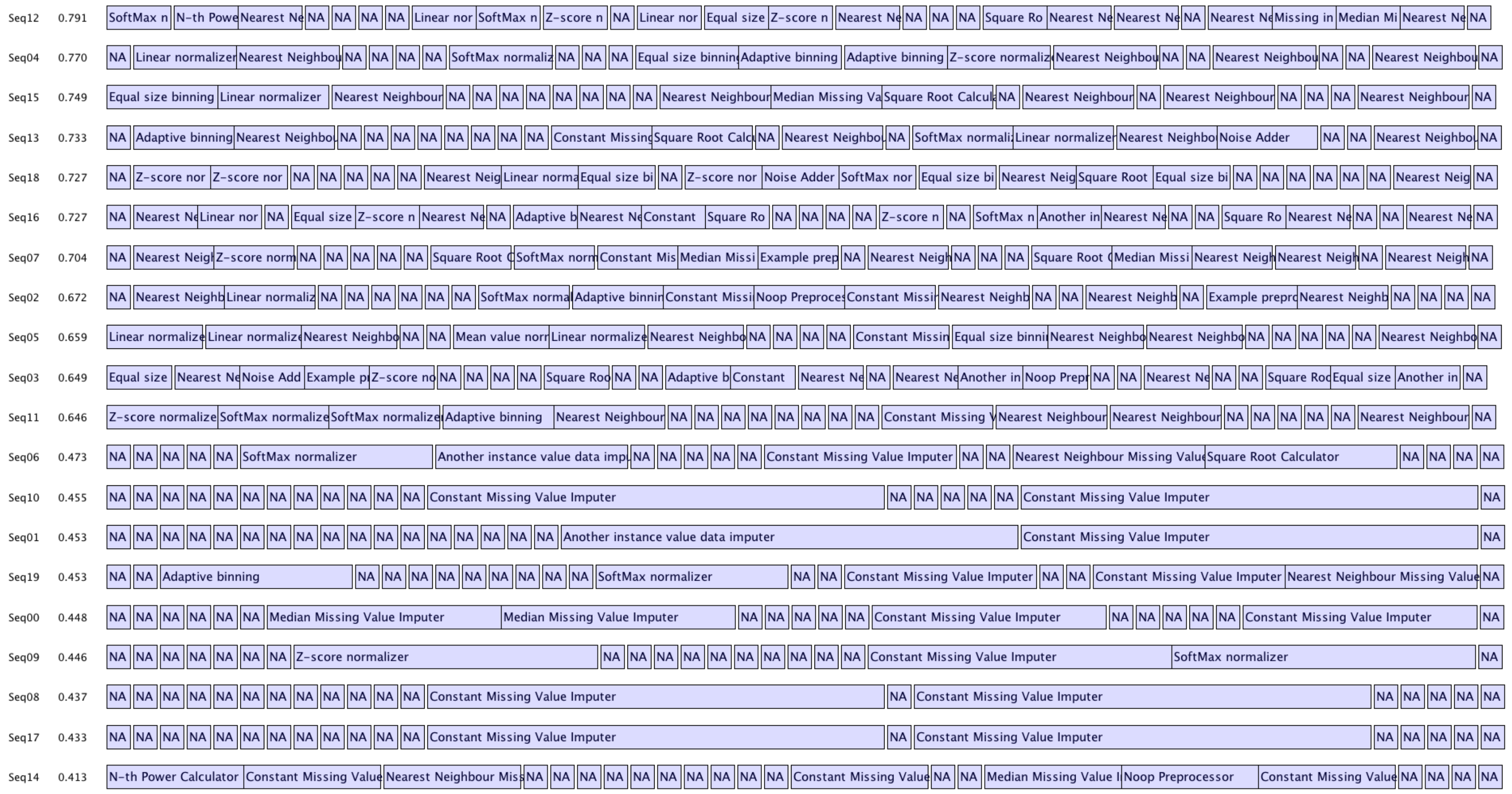


Figure 10.8: The best sequences found in repetitive runs of IPT for the *Segment* dataset and the J48 Decision Tree.

The Figure 10.8 shows the sequences in details. In the number of attributes and subsequences in the sequences, it is hard to find similar preprocessing methods. But at least the first two sequences, the **Seq12** and the **Seq04**, with the highest fitness value transform the **hue_mean** attribute (second from the right), which is important for the construction of the decision tree. Both sequences uses the *Nearest Neighbour Value Imputation* method, but they differ in parameters. The **Seq12** uses also another preprocessing methods to transform the **hue_mean** attribute. Unfortunately they are all treating missing values which have been removed by the first of the preprocessing methods, therefore they have no effect on the dataset. The main difference between the first two sequences is the treating the **rawblue_mean** attribute. While the **Seq12** is transforming this attribute with the *Nearest Neighbour Value Imputation* preprocessing method the **Seq04** is not transforming it at all and is transforming another attributes.

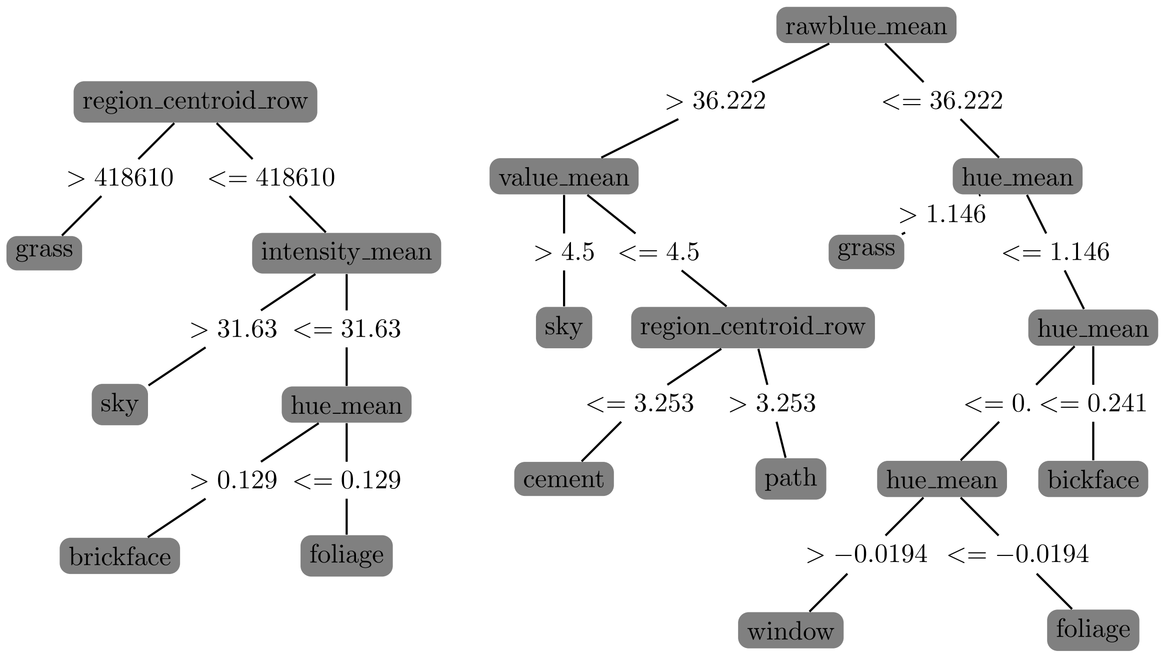


Figure 10.9: Decision trees created for classification of the original *Segment* dataset on the left and for the preprocessed *Segment* dataset by the sequence **Seq12** on the right.

The Figure 10.9 shows the decision trees for the original dataset (on the left) and the dataset preprocessed by the **Seq12**. The treating of missing values in the **rawblue_mean** and the change in mean and scale in the *hue_mean* attribute allows the training algorithm of the J48 Decision Tree create much more accurate tree.

The Table 10.2 shows the confusion matrix for the model trained with the original, not preprocessed, dataset on the left and the model trained with the dataset preprocessed by the sequence **Seq12** on the right. The rows represent real classes and the columns predicted classes. The intersection of a row and a column then show the number of instances of one of real classes which were classified as a class. For example the number 178 in the top left cell tells that the 178 instances which belong in reality to *brickface* class were (correctly) classifier as *brickface* class. The number 55 just left to the 178, tells that the 55 instances of the *brickface* class were (incorrectly) classified as *sky* class.

Table 10.2: Confusion matrices for the *Segment* and the J48 decision tree.

Original dataset Classified as								**Preprocessed dataset** Classified as						
brickface	sky	foliage	cement	window	path	grass	In reality	brickface	sky	foliage	cement	window	path	grass
178	55	0	0	75	0	0	brickface	295	0	0	6	7	0	0
1	236	74	0	0	0	0	sky	0	311	0	0	0	0	0
12	59	179	0	53	0	0	foliage	10	0	207	16	70	0	0
45	199	46	0	6	3	4	cement	5	2	3	237	26	29	1
68	50	119	0	65	0	0	window	58	0	45	10	189	0	0
16	97	13	0	4	76	100	path	0	0	0	71	4	231	0
82	17	1	0	26	98	80	grass	2	0	1	0	0	0	301

If you compare the both sides of the confusion matrix, the difference is enormous. The confusion matrix on the left side for the original dataset shows very many misclassifications and the *cement* class is completely ignored. The confusion matrix on the right side look much better. The most of the instances are correctly classified and the *cement* class is classified.

The last thing I want to show here are the scatter plots for the original and the preprocessed datasets. The scatter plots are shown on the Figure 10.10. The decision tree for the original dataset uses mainly the *Intesity_mean* and *Region_centroid_row* which do not discriminate the classes well and the use of the *hue_mean* attribute can not save the situation. On the other hand the transformed attributes of the preprocessed dataset allows the training algorithm to use attributes which discriminate the classes much more clearly.

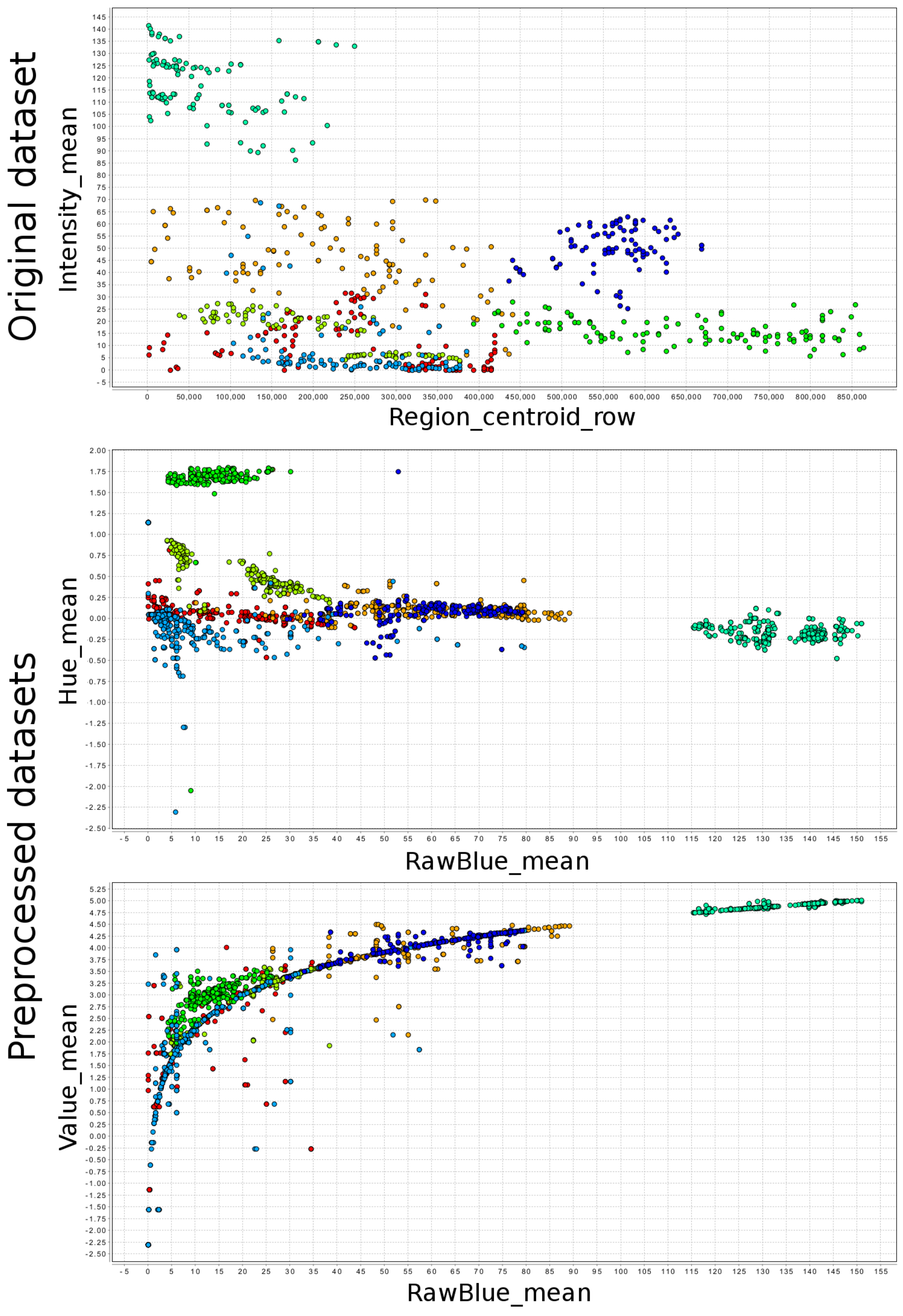


Figure 10.10: Scatter plot for the original (on the left) *Segment* dataset and preprocessed (on the right) *Segment* dataset dataset.

10.4.3 Spambase dataset and J48 Decision Tree

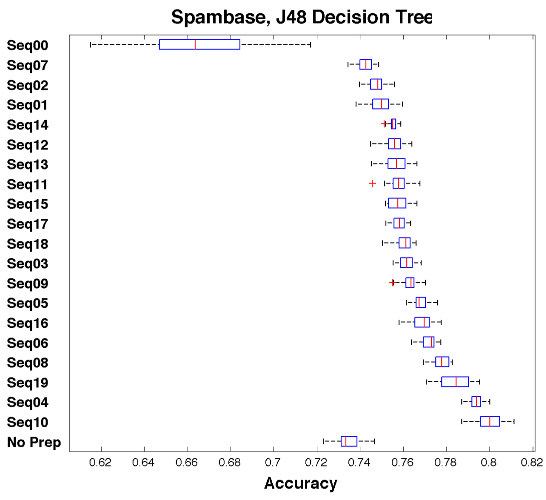


Figure 10.11: Accuracies of J48 Decision Tree classifiers trained on validation *Spambase* dataset. The validation dataset was preprocessed by the best sequences of preprocessing methods found in 20 repetitive runs of IPT. Sequences are sorted by mean accuracy of "their" models. For easier comparison the boxplot representing accuracies of model trained with original validation dataset is on the bottom of the figure.

The last dataset to describe in combination with the J48 Decision Tree is the *Spambase* dataset. This dataset present a typical situation where the accuracy of the J48 model is improved only by 7%. Again at first the Figure 10.11 shows fitness values for all the best sequences found in all 20 runs of IPT and at the bottom shows the accuracy of the J48 model for the original dataset. The best of sequences is the **Seq10**. The model trained from the dataset preprocessed by this **Seq10** achieves accuracy about 7% higher than the model with the original dataset. The Figure 10.14 shows the best sequences of found in repetitive runs of IPT.

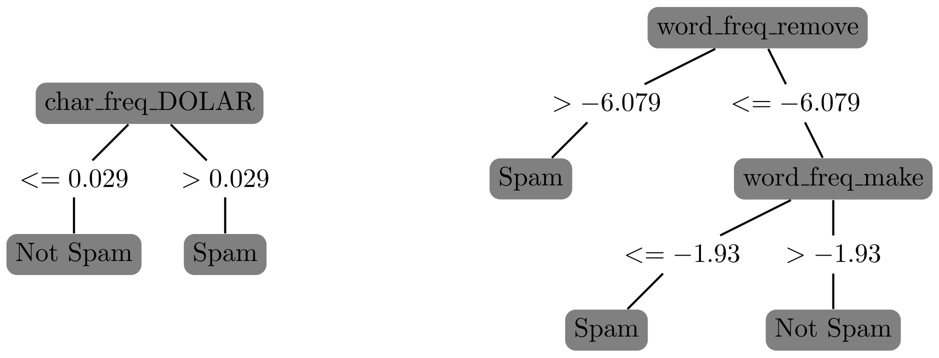


Figure 10.12: The J48 JDecision trees created for classification of the original *Spambase* dataset on the left and for the preprocessed *Spambase* dataset by the sequence **Seq10** on the right.

The Figure 10.12 shows difference between the decision tree for the original dataset on the left and the decision tree for the preprocessed dataset on the right. For the original dataset the J48 training algorithm is able to use only the *char_freq_DOLAR*. After the preprocessing the decision tree is much bigger. The accuracy is not improved too much, but still the more complex decision tree improves the accuracy by 7%. The decision tree for the preprocessed dataset uses frequencies of words *remove* and *make* to achieve better fitness.

Table 10.3: Confusion matrices for the *Spambase* and the J48 decision tree.

Original dataset			**Preprocessed dataset**	
Classified as			Classified as	
Spam	Not spam	In reality	Spam	Not spam
641	1036	Spam	1249	456
136	2436	Not spam	385	2239

The Table 10.3 shows the confusion matrix for both the original and the preprocessed dataset. The decision tree for the original dataset is very bad for classifying spam. The almost two thirds of spam is classified as normal emails. Only few normal emails are classified as spam. The picture for the preprocessed dataset is different. The spam and normal emails are generally far better recognised. But the number of emails incorrectly classified for the spam and normal emails is almost equal.

The 10.13 shows the scatterplot in the most important input attributes and you may check how the decision trees divide the space.

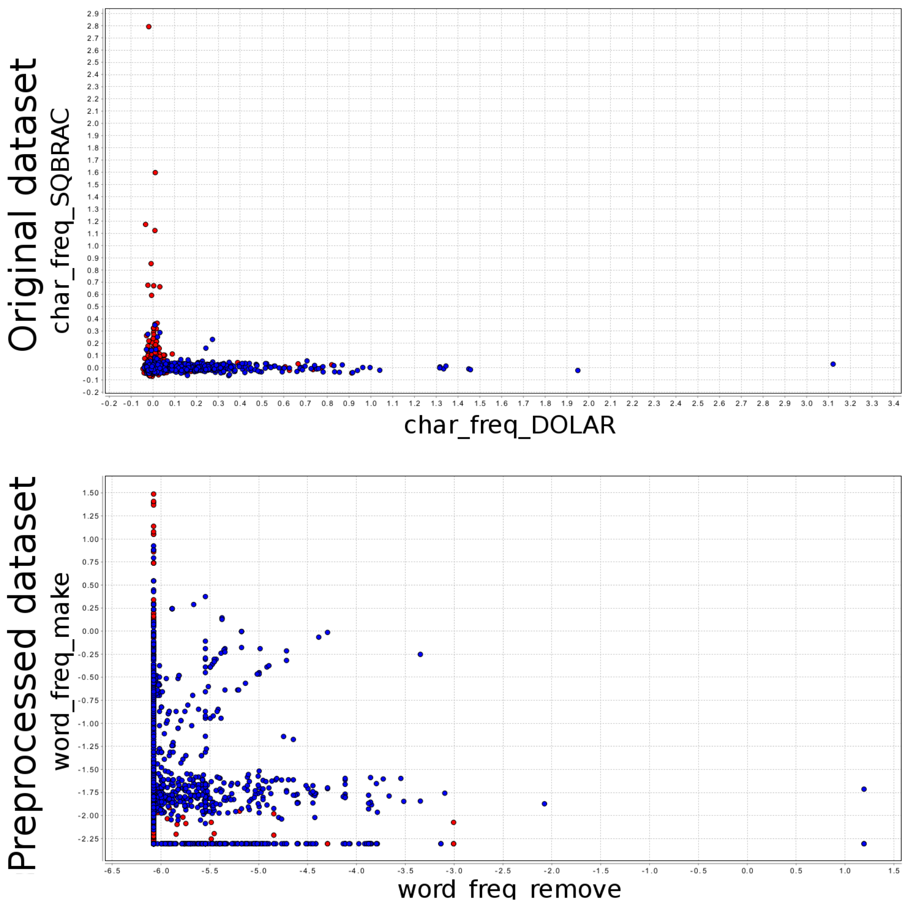


Figure 10.13: Scatter plot for the original (on the left) *Spambase* dataset and preprocessed (on the right) *Spambase* dataset dataset.

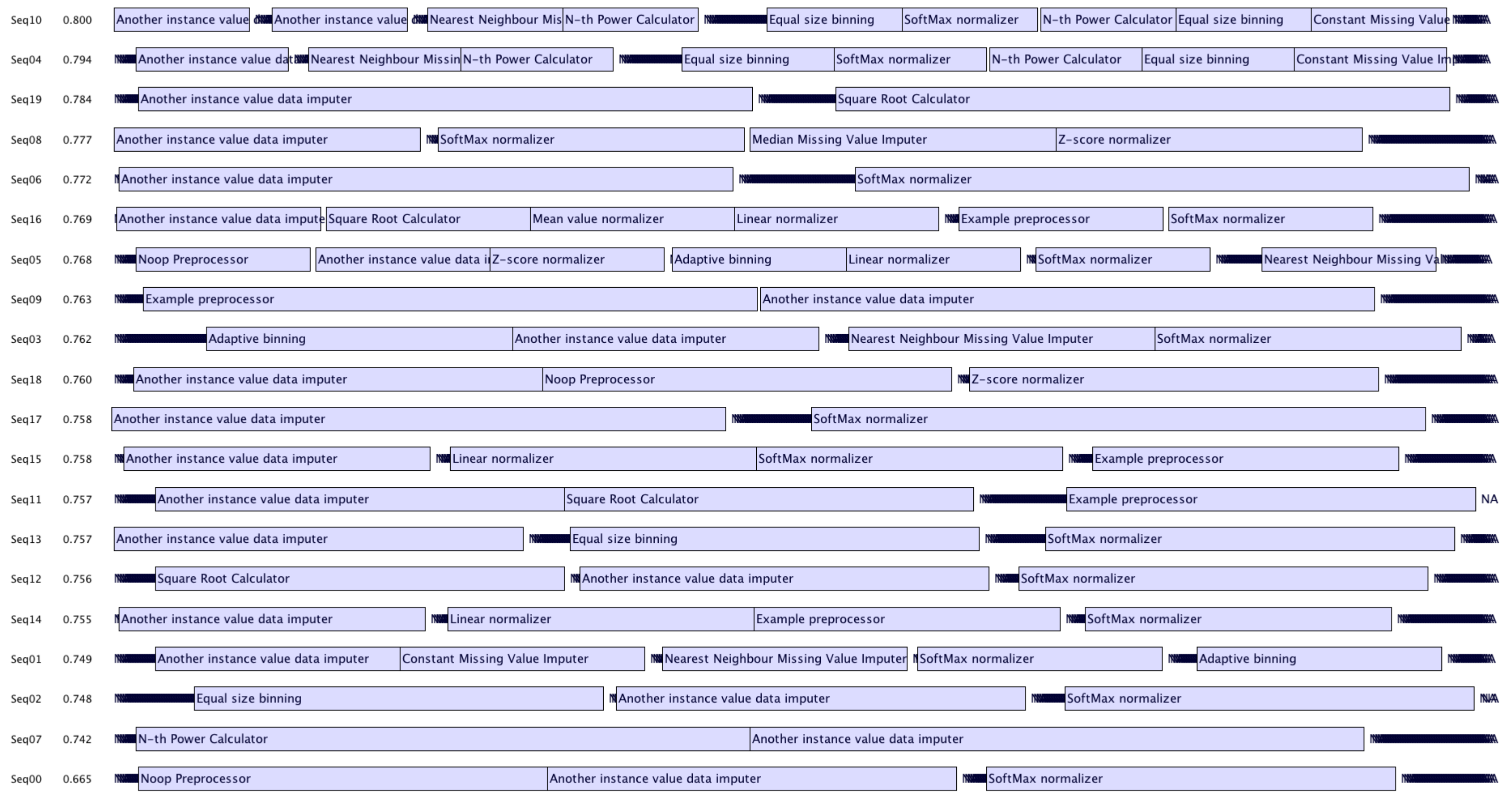


Figure 10.14: The best sequences found in repetitive runs of IPT for the *Spambase* dataset and the J48 Decision Tree.

10.4.4 CTG and the Simple Logistic Regression Classifier

This section open description of the datasets with the Simple Logistic Regression Classifier. The first to describe is the *CTG* dataset. As shown on the Figure 10.15 more than a half of the sequences found by IPT are better than the accuracy for the original dataset. But the best of them **Seq05** improves the accuracy only by about 2%. This is not much but the difference is statistically significant.

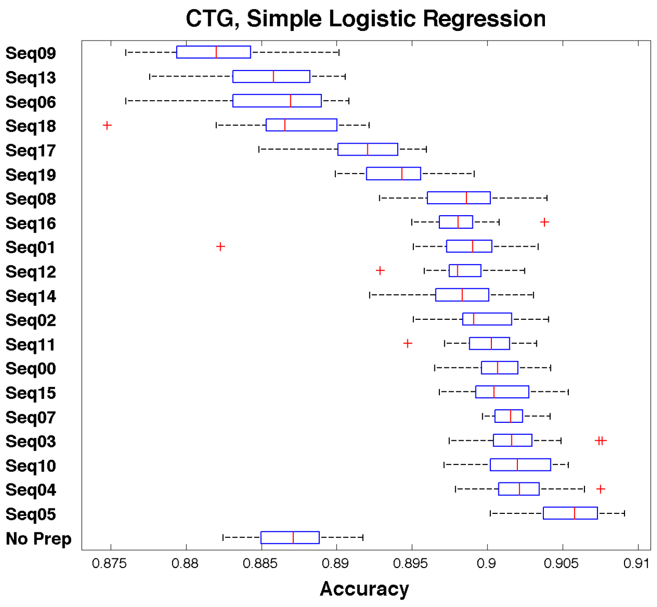


Figure 10.15: Accuracies of Simple Logistic classifiers trained on validation *CTG* dataset. The validation dataset was preprocessed by the best sequences of preprocessing methods found in 20 repetitive runs of IPT. Sequences are sorted by mean accuracy of ”their” models. For easier comparison the boxplot representing accuracies of model trained with original validation dataset is on the bottom of the figure.

The Figure 10.17 shows all the best sequences found by IPT. The sequence **Seq05** does not transforms too many attributes. The only ones the **Seq05** preprocess are the **Width**, the **AC** and the **UC**.

The Figure 10.16 shows equations representing the models for original dataset on the left and for the preprocessed dataset on the right. In contrast to the J48 models presented above, it is much harder to analyse changes in the Simple Logistic Classifiers. There is only some comments I can make in connection with the analysis of **Seq05** sequence. The **UC** does not appear in any of formulas representing the model (see the Figure 10.16) and I regard it as useless. The **Width** attribute does appear in the formula for the Class 2. But the most important of these three is the attribute **AC**. This attribute is not present in the formulas for the original dataset and is present in all formulas for the preprocessed model. More the **Seq05** applies non-linear transformation on the **AC** attribute. Since it had non-linear character, it was useless for the Simple Logistic Classifier, but after transformation it introduce new information which add some percentages to accuracy.

Class 0 : $-2105.79-1.21[ASTV]+1.2[MSTV]-0.99[DP]+0.68[Width]+425.53[A]+1.38[D]+2.18[AD]+0.3[DE]-4.81[FS]$

Class 1 : $-3055216869166.34-0.93[LB]+1.29[ASTV]-0.7[MSTV]+0.02[MLTV]+0.15[DP]+3866482157.21[DR]-0.27[Width]+4.2[Median]-369.41[A]-2.53[D]+2.03[E]-0.75[AD]-13.88[FS]$

Class 2 : $635.81+[ASTV]-0.23[MSTV]-0.15[MLTV]+0.52[DP]-0.04[Mean]-10.68[Median]-113.9[A]-4.87[AD]-4.43[DE]-0.02[LD]+45.48[FS]$

Class 0 : $-1188.29-0.94[LBE]+0.52[AC]-1.19[ASTV]+1.24[MSTV]-1.03[DP]+0.53[Width]+0.01[Mean]+239.62[A]+0.59[D]-0.26[E]+2.53[AD]+0.87[DE]$

Class 1 : $3372.23+3.16[LBE]-1.85[LB]-0.62[AC]+1.1[ASTV]-0.13[MSTV]+0.36[DP]-0.39[Width]+1.67[Median]-677.18[A]-2.43[D]+1.47[E]-0.26[DE]+0.01[LD]-11.19[FS]$

Class 2 : $1903.31-0.52[AC]+0.94[ASTV]-0.17[MLTV]+0.54[DP]-0.01[Mean]-8.48[Median]-370.69[A]-4.29[AD]-4.73[DE]-0.03[LD]+37.98[FS]$

Figure 10.16: Equations created for classification of the original *CTG* dataset on the left and for the preprocessed *CTG* dataset by the sequence *CTG* on the right.

Table 10.4: Confusion matrices for the *CTG* and the Simple Logistic classifier.

Original dataset Classified as				**Preprocessed dataset** Classified as		
Normal	Suspect	Patologic	In reality	Normal	Suspect	Patologic
1464	28	5	Normal	1466	24	7
98	165	3	Suspect	47	210	9
18	21	114	Patologic	17	15	121

The Table 10.4 shows the confusion matrices. For the model with the original dataset on the left and for the preprocessed dataset on the right. The differences are not big. The biggest improvement is in the *suspect* class where additional 45 instances is correctly classified. In total there are 54 more instances correctly classified. This is not big number but since here we talk about application in medicine, specifically about delivering babies, every small improvement in accuracy helps.

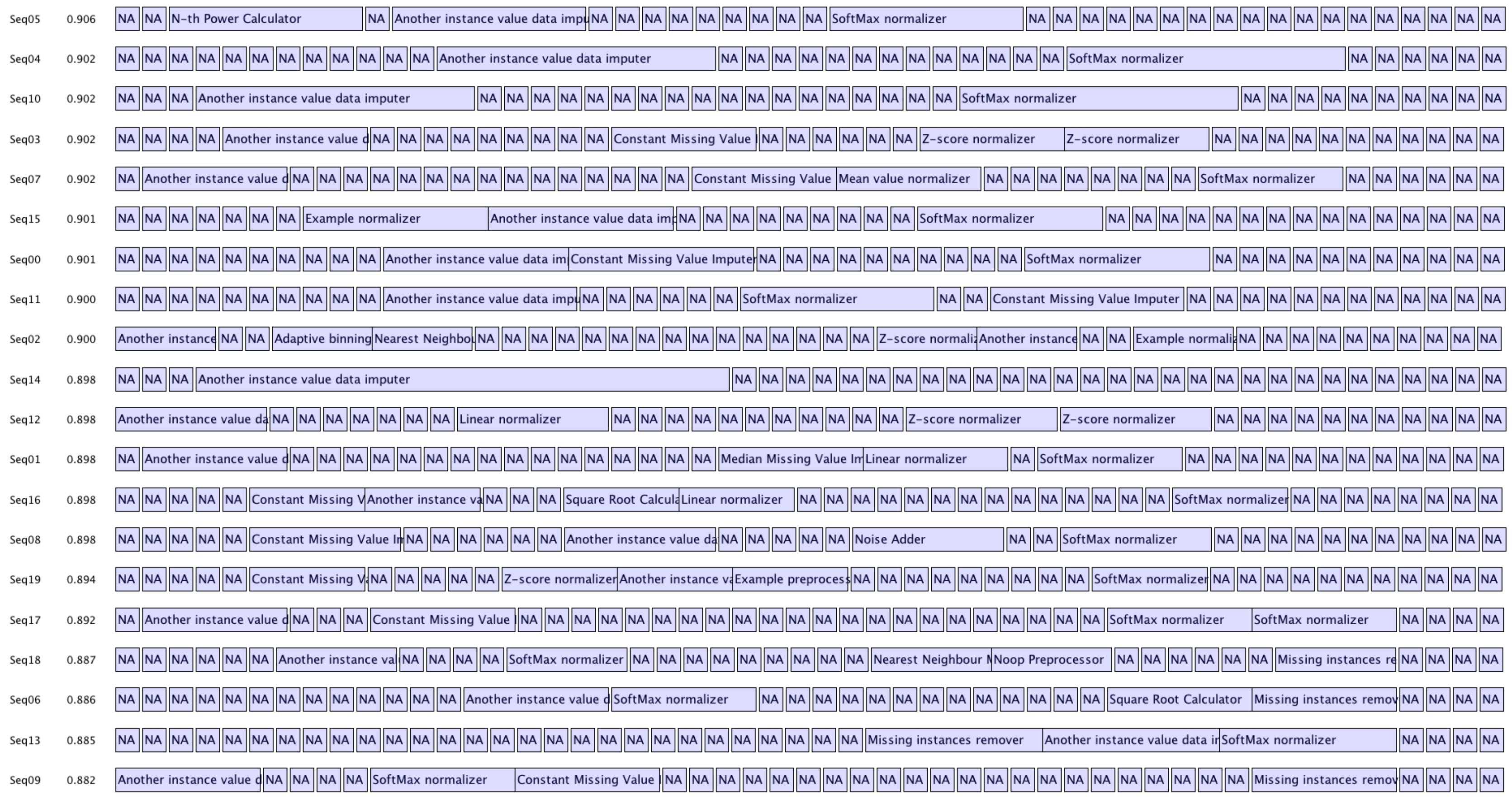

Seq05 0.906 N-th Power Calculator, Another instance value data impu, SoftMax normalizer
Seq04 0.902 Another instance value data imputer, SoftMax normalizer
Seq10 0.902 Another instance value data imputer, SoftMax normalizer
Seq03 0.902 Another instance value d, Constant Missing Value, Z-score normalizer, Z-score normalizer
Seq07 0.902 Another instance value d, Constant Missing Value, Mean value normalizer, SoftMax normalizer
Seq15 0.901 Example normalizer, Another instance value data im, SoftMax normalizer
Seq00 0.901 Another instance value data im, Constant Missing Value Imputer, SoftMax normalizer
Seq11 0.900 Another instance value data impu, SoftMax normalizer, Constant Missing Value Imputer
Seq02 0.900 Another instance, Adaptive binning, Nearest Neighbou, Z-score normaliz, Another instance, Example normaliz
Seq14 0.898 Another instance value data imputer
Seq12 0.898 Another instance value da, Linear normalizer, Z-score normalizer, Z-score normalizer
Seq01 0.898 Another instance value d, Median Missing Value Im, Linear normalizer, SoftMax normalizer
Seq16 0.898 Constant Missing V, Another instance va, Square Root Calcula, Linear normalizer, SoftMax normalizer
Seq08 0.898 Constant Missing Value Im, Another instance value da, Noise Adder, SoftMax normalizer
Seq19 0.894 Constant Missing Va, Z-score normalizer, Another instance va, Example preprocess, SoftMax normalizer
Seq17 0.892 Another instance value d, Constant Missing Value I, SoftMax normalizer, SoftMax normalizer
Seq18 0.887 Another instance val, SoftMax normalizer, Nearest Neighbour N, Noop Preprocessor, Missing instances re
Seq06 0.886 Another instance value d, SoftMax normalizer, Square Root Calculator, Missing instances remov
Seq13 0.885 Missing instances remover, Another instance value data in, SoftMax normalizer
Seq09 0.882 Another instance value d, SoftMax normalizer, Constant Missing Value I, Missing instances remov
NA

Figure 10.17: The best sequences found in repetitive runs of IPT for the *CTG* dataset and the Simple Logistic.

10.4.5 Ecoli and the Simple Logistic Regression Classifier

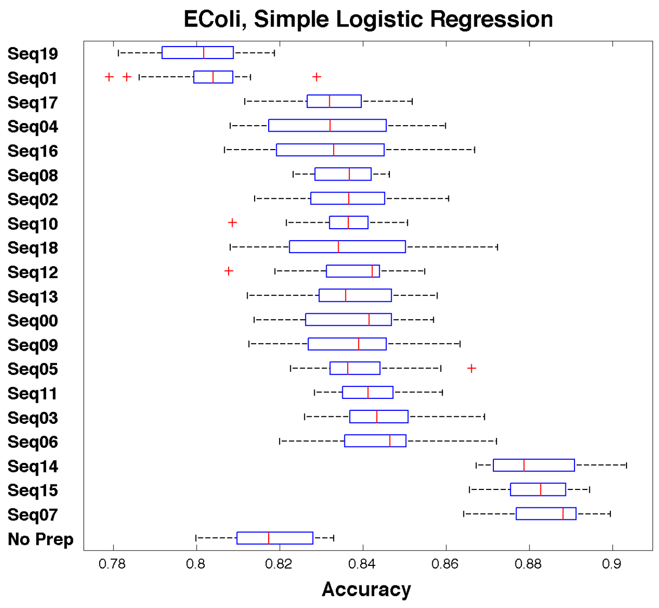


Figure 10.18: Accuracies of Simple Logistic classifiers trained on validation *Ecoli* dataset. The validation dataset was preprocessed by the best sequences of preprocessing methods found in 20 repetitive runs of IPT. Sequences are sorted by mean accuracy of "their" models. For easier comparison the boxplot representing accuracies of model trained with original validation dataset is on the bottom of the figure.

The EColi dataset with the Simple Logistic Classifier represents another case with typical improvement in model accuracy. In this case the improvement is about 6%. You can examine the performance of the sequences found by IPT on the Figure 10.18. The best of the is **Seq07**. But the **Seq15** and **Seq14** are very close to the **Seq07** and infact it seems that the differences between them are statistically insignificant.

The Figure 10.20 shows the found sequences in details. The sequences with the highest fitness – the **Seq07**,the **Seq15** and the **Seq14** are shown at the top. Their common sign is the transformation of the **alm1** attribute, the third from the right. All three sequences use methods treating missing data but since there are no missing values, they have no effect. What has effect, is the *N-th Power Calculator* method, the non-linear transformation. The sequences calculates 14^{th}, 15^{th} and 17^{th} power roots which makes them successful.

The Figure 10.19 shows equations for the original and the preprocessed datasets. But the changes between them are only small and it is hard to identify ones caused by different initiation and the ones caused by the transformed dataset. The biggest change is in the equation for the Class 2. This is probably caused by the dataset transformations. I have two reasons for this – the first one is that the structure of the equation has changed and now uses the transformed **alm1** attribute. The other reason lies in the confusion matrix in the Table 10.5. The equation for Class 2 assigns instances into the *IM* class which shows the biggest change in numbers – see the Table 10.5.

Class 0 : $5.67 - 1.59[mcg] - 11.28[gvh] - 12.38[aac] + 0.44[alm1]$

Class 1 :$-1.53 + 2.51[mcg] + 6.33[gvh] - 1.14[chg] - 8.53[aac] - 0.26[alm1]$

Class 2 :$-7.72 + 6.15[chg] + 4.71[aac] - 0.36[alm1] + 14.86[alm2]$

Class 3 :$4.58+5.49[mcg]+3.62[lip]+11.24[aac]-12.62[alm2]$

Class 0 :$-62.22 - 6.45[mcg] - 3.03[gvh] - 19.99[aac] + 62.07[alm1] + 3.82[alm2]$

Class 1 :$-1.77 + 3.47[mcg] + 5.46[gvh] - 1.78[chg] - 13.61[aac] + 2.88[alm1] - 3.02[alm2]$

Class 2 :$88.99 + 6.53[chg] - 95.82[alm1]$

Class 3 :$49.81 + 6.22[mcg] + 17.52[aac] - 46.33[alm1] - 15.43[alm2]$

Figure 10.19: Equations created for classification of the original *Ecoli* dataset on the left and for the preprocessed *Ecoli* dataset by the sequence *Ecoli* on the right.

Table 10.5: Confusion matrices for the *Ecoli* and the Simple Logistic classifier.

Original dataset Classified as					**Preprocessed dataset** Classified as			
CP	PP	IM	OM	In reality	CP	PP	IM	OM
68	1	1	1	CP	70	1	0	0
3	19	2	1	PP	5	18	2	0
6	2	45	0	IM	0	3	49	1
2	4	0	7	OM	0	4	2	7

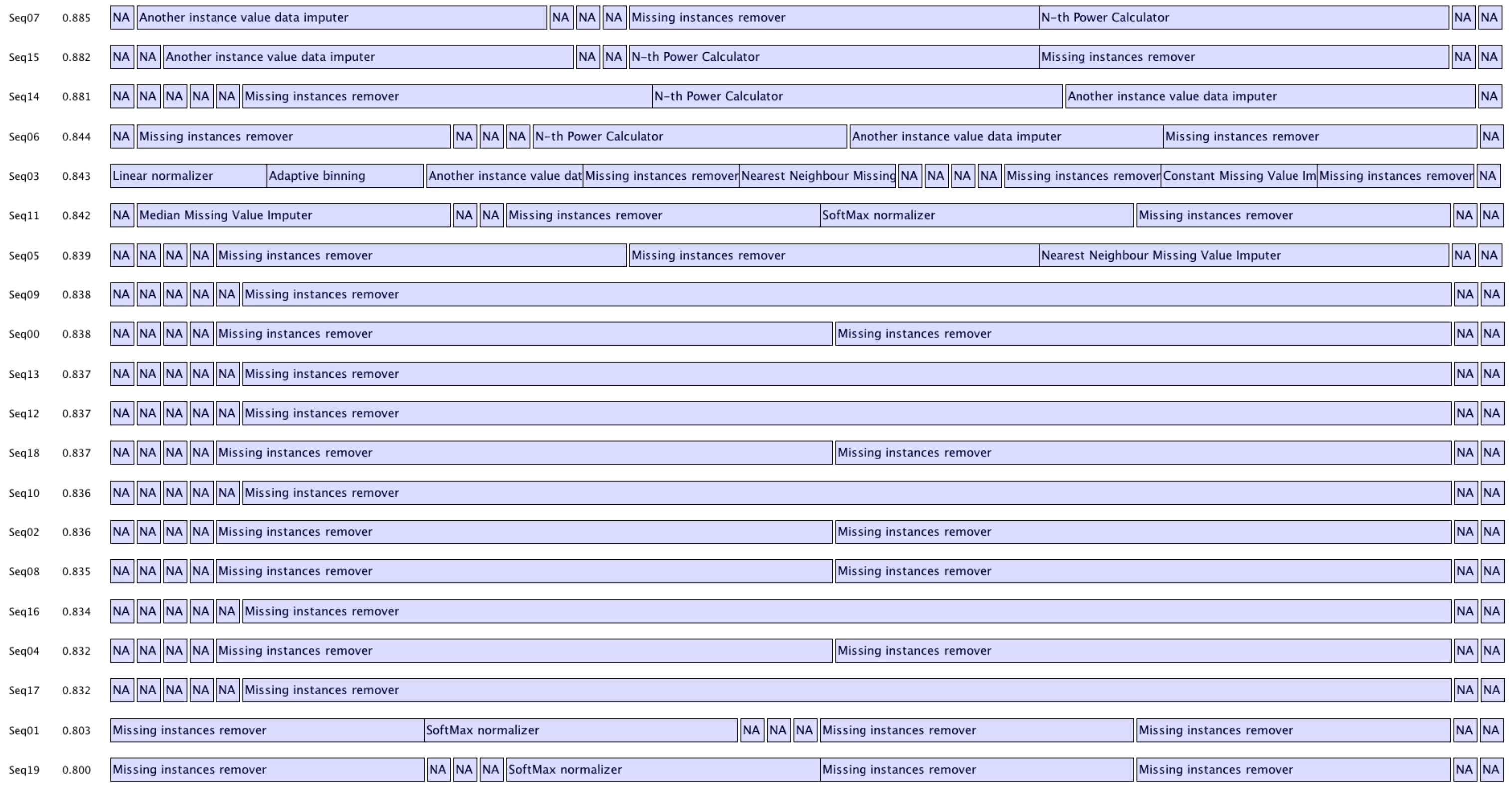

Seq07 0.885 NA | Another instance value data imputer | NA | NA | NA | Missing instances remover | N–th Power Calculator | NA | NA
Seq15 0.882 NA | NA | Another instance value data imputer | NA | NA | N–th Power Calculator | Missing instances remover | NA | NA
Seq14 0.881 NA | NA | NA | NA | NA | Missing instances remover | N–th Power Calculator | Another instance value data imputer | NA
Seq06 0.844 NA | Missing instances remover | NA | NA | NA | N–th Power Calculator | Another instance value data imputer | Missing instances remover | NA
Seq03 0.843 Linear normalizer | Adaptive binning | Another instance value dat | Missing instances remover | Nearest Neighbour Missing | NA | NA | NA | NA | Missing instances remover | Constant Missing Value Im | Missing instances remover | NA
Seq11 0.842 NA | Median Missing Value Imputer | NA | NA | Missing instances remover | SoftMax normalizer | Missing instances remover | NA | NA
Seq05 0.839 NA | NA | NA | NA | Missing instances remover | Missing instances remover | Nearest Neighbour Missing Value Imputer | NA | NA
Seq09 0.838 NA | NA | NA | NA | NA | Missing instances remover | NA | NA
Seq00 0.838 NA | NA | NA | NA | Missing instances remover | Missing instances remover | NA | NA
Seq13 0.837 NA | NA | NA | NA | NA | Missing instances remover | NA | NA
Seq12 0.837 NA | NA | NA | NA | NA | Missing instances remover | NA | NA
Seq18 0.837 NA | NA | NA | NA | Missing instances remover | Missing instances remover | NA | NA
Seq10 0.836 NA | NA | NA | NA | NA | Missing instances remover | NA | NA
Seq02 0.836 NA | NA | NA | NA | Missing instances remover | Missing instances remover | NA | NA
Seq08 0.835 NA | NA | NA | NA | Missing instances remover | Missing instances remover | NA | NA
Seq16 0.834 NA | NA | NA | NA | NA | Missing instances remover | NA | NA
Seq04 0.832 NA | NA | NA | NA | Missing instances remover | Missing instances remover | NA | NA
Seq17 0.832 NA | NA | NA | NA | NA | Missing instances remover | NA | NA
Seq01 0.803 Missing instances remover | SoftMax normalizer | NA | NA | NA | Missing instances remover | Missing instances remover | NA | NA
Seq19 0.800 Missing instances remover | NA | NA | NA | SoftMax normalizer | Missing instances remover | Missing instances remover | NA | NA

Figure 10.20: The best sequences found in repetitive runs of IPT for the *Ecoli* dataset and the Simple Logistic.

10.4.6 Wine Dataset and the Linear Logistic Regression

The last dataset I want to present here is the *Wine* dataset. Ihe IPT has improved the accuracy by about 12%. The absolute value of the improvement is a bit tricky because there are not that many instances in the validation part of the dataset, see the confusion matrix in the end of this section. But first, the Figure 10.21 show the fitness values for all the sequences found by IPT in all 20 runs. The best of them is the **Seq06**. All other sequences improve the accuracy of the model as well but in lesser degree.

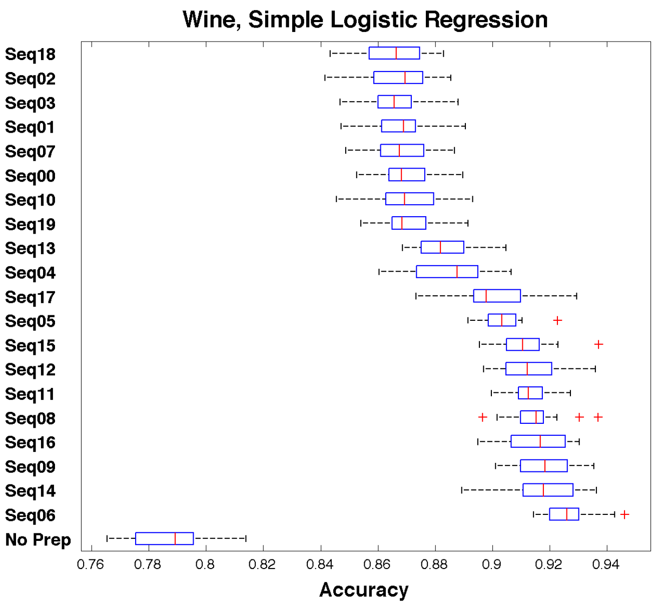


Figure 10.21: Accuracies of Simple Logistic classifiers trained on validation *Wine* dataset. The validation dataset was preprocessed by the best sequences of preprocessing methods found in 20 repetitive runs of IPT. Sequences are sorted by mean accuracy of "their" models. For easier comparison the boxplot representing accuracies of model trained with original validation dataset is on the bottom of the figure.

Class 0 : $-29.19 + 1.95[Alcohol] - 1.1[Nonflavanoid\_phenols] + 0.92[OD280\_OD315]$

Class 1 : $70.09 - 4.34[Alcohol] - 12.47[Ash] - 0.41[Color\_intensity]$

Class 2 : $4.62 - 2.21[Proanthocyanins] + 710.33[Hue] - 0.57[OD280\_OD315]$

Class 0 : $-52.81 + 4.09[Alcohol] - 1.15[OD280\_OD315\_bin\_0] + 1.16[OD280\_OD315\_bin\_1]$

Class 1 : $101.6 - 7.06[Alcohol] - 13[Ash]$

Class 2 : $5.85 - 1.1[Flavanoids] + 200.28[Hue] - 1.21[OD280\_OD315]$

Figure 10.22: Simple Logistic classifier created for classification of the original *Wine* dataset on the left and for the preprocessed *Wine* dataset by the sequence **Seq06** on the right.

All the sequences of the preprocessing methods found by IPT are shown in the Figure 10.23. All the sequences with the highest fitness value transforms the *Flavanoids* attribute. The transformation is a non linear *Square Root Calculator* or *N-th Power Calculator* respectively. Both of them do similar thing – calculate the power roots. The *Square Root Calculator* calculates only the second power root. The *N-th Power Calculater* has a parameter which tells which power root to calculate. In case of this particular dataset, it calculates power roots between 3^{rd} and 6^{th}. The other preprocessing methods are not present in all the sequences and therefore they do not have so strong influence on the accuracy of the model.

The Figure 10.22 shows the models for the original dataset and the dataset preprocessed by the **Seq06**. The biggest change in the structure of the model is in the equation for the Class 0 and Class 2. The transformed *Flavanoids* attribute is used in the equation for the Class 2. Also the equation for the Class 0 uses the transformed attribute – the discretised *OD280_OD315* attribute to be specific.

Table 10.6: Confusion matrices for the *Wine* and the Simple Logistic classifier.

Original dataset Classified as				**Preprocessed dataset** Classified as		
Class 1	Class 2	Class 3	In reality	Class 1	Class 2	Class 3
29	1	0	Class 1	30	0	0
4	35	3	Class 2	2	39	1
2	6	13	Class 3	1	1	19

The Table 10.6 shows the confusion matrix for Simple Logistic Classifier trained with the original and transformed datasets. The differences between the matrices are small in absolute numbers but because the size of the dataset itself is small, the small changes mean the big relative improvement. The highest improvement in the number of correctly classified values is in the Class 2. Equation for this class is the one which uses the transformed *Flavanoids* attribute and in this way confirms the supposition that the non linear transformation of the *Flavanoids* dataset leads to the improved accuracy.

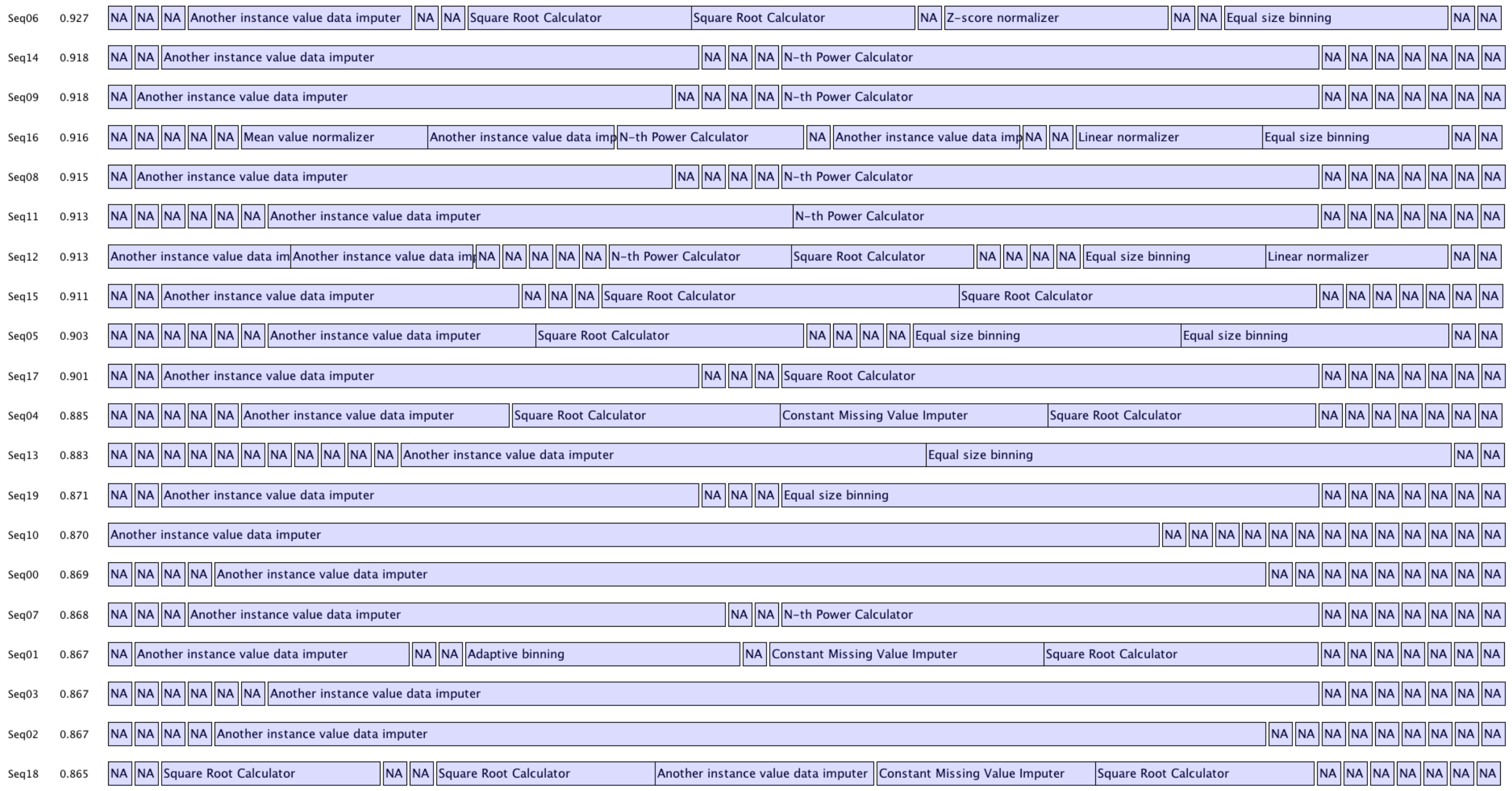


Figure 10.23: The best sequences found in repetitive runs of IPT for the *Wine* dataset and the Simple Logistic.

10.5 Conclusion

This chapter presents core results of my thesis – the performance of IPT with the real world data. I have used 13 publicly available datasets from the UCI Machine Learning Repository. Unfortunately these datasets are very well prepared for the data mining and many people made their best to transform so the data mining methods has the highest possible accuracy. For this reason I have damaged the datasets with additional transformations. These transformations have introduced missing values, different ranges, outliers and non-linear characteristics of the data. These misshapen datasets are referenced throughout this chapter as the original or not preprocessed datasets. This misshapen datasets are then used as the input for the Inductive Preprocessing Technology.

In almost all cases I am able to improve the accuracy of models only by the transforming the misshapen datasets. The typical improvement in accuracy of the model is about 5% to 10%, but there are exception in both directions. The worst dataset is the *Bank* dataset which I am unable to improve at all. The other extreme it the *Segment* dataset with the J48 Decision Tree. For selected datasets I have also discussed the sequences found by IPT. In addition I have compared the models for the original, not preprocessed, dataset with model for the transformed (preprocessed) dataset and I have tried to identify preprocessing methods which improved the accuracy of the model.

The sequences found IPT and the boxplots showing their fitness for the remaining datasets and models are shown in the Appendix D on page 168.

11 Use of Meta Data to Speedup the Search

Until now, when the Inductive Preprocessing Technique started with a new dataset, it has to start with a set of random sequences as it has no knowledge about the dataset. It has to undergo the whole process of search of the best sequence of the preprocessing methods as described in previous chapters. When I have preprocessed some datasets, I have gained some knowledge about the dataset properties (meta data) and used preprocessing methods. I suppose that for similar datasets, similar preprocessing methods will be useful. And I can use this information to guide or speedup IPT. Therefore I will store and reuse the knowledge about the datasets I have preprocessed. In other words, when IPT preprocesses a dataset, it also store its meta data (dataset properties) and used preprocessing methods to the meta database (See the Chapter 6 for exact algorithm).

At first I will discuss following – if I can find rules when to preprocess an attribute or even which preprocessing method to used. If I am able to find at least one positive answer, it could mean a great speed up of IPT. If I am able to find which attributes I should not preprocess, I can omit these attributes for IPT and in this way I can limit the search space for the search for preprocessing sequences. If can even create successful rules identifying when to use specific preprocessing method, I can put such method to a subsequence of even I can omit IPT entirely and I can use the rules to directly generate the best sequences of preprocessing methods.

The second experiment will show different approach. I will use the sequences stored in the meta database to generate a new initial population for IPT. I expect that IPT should take an advantage of these *generated sequences* and should find the best sequence in lower number of steps.

11.1 The Experiments and their Setup

I have build the meta database from the training parts of the same datasets I have used in the previous chapter. For each attribute I have added to the meta database three subsequences from the sequences with the highest fitness. To describe the dataset, or more precisely its attributes, I will use the meta data described in the Chapter 6. Shortly the meta data are following:

- information entropy in the attribute,
- value range in the attribute,
- number of output classes,
- ratio between the most and the least numerous output class,
- portion of missing values in %,
- sample mean in the attribute,
- sample variance in the attribute,
- sample skewness in the attribute,

- average absolute correlation of the attribute to the classes,
- accuracy of the J48 decision tree classifier using only the attribute,
- accuracy of the simple logistic regression classifier using only the attribute.

There are the four experiments I would like to present in this chapter. In the **first experiment** I will try to find out if I can find rules when to use some preprocessing method.

The **second experiment** should demonstrate that the selected meta data and distance measure are able to define distance of attributes properly. The next step in this experiment will be to generate new sequences of preprocessing methods from the meta-database and I will calculate accuracy of the generated sequences and to find out if these sequences are comparable in accuracy of models to the sequences found by IPT.

To do the **third experiment** I will again transform the original data with different setup and I will try to find out how many attributes are similar to their previous versions. And again I will also generate sequences of preprocessing methods from the meta database. Later I will calculate accuracy of the sequences and I will compare them to accuracy of models with the original dataset.

The **fourth experiment** is about the ability of the meta database to generate sequences for datasets that were not seen before. I will introduce yet new datasets – the Credit Approval [92, 93], Heart Disease [94, 95], Indian Liver Patient [96, 97] and Pima Indian Diabetes [98, 99]) datasets from the UCI Machine Learning repository [66]. I will again add some missing values and outliers and so on and I will test if the meta database can supply sequences of preprocessing methods which works well with completely unknown datasets. And in the end I will test if the sequences produced by the meta database can speed up IPT, that is if IPT with generated initial population can achieve the similar fitness on lower number of steps. I will produce 10 sequences and I will put them into the initial population of the genetic search of IPT. I will IPT to run for 10 generations and I will repeat 20 runs as in the previous chapter. And I will compare the achieved results with the results achieved by IPT starting from the random initial population and running for 50 generations.

11.2 Building Rules for Selecting Preprocessing Methods

In this section I will try to find the rules when to apply a preprocessing method based on the attribute's metadata. The reason to try this is further possible speed up of IPT or even possibility to skip it. If I could predict which attribute IPT should concentrate on and which to ignore, it would narrow the search space and would be great help to the search method for sequences of preprocessing methods and the resulting sequence of preprocessing methods will be even simpler.

11.2.1 Clustering Analysis of Meta Database

At first I will present results of a clustering analysis. The reason to use the clustering analysis is to test if the meta data has sufficient power to discriminate attributes with different properties. I will use the SOM map [100] as a clustering analysis.

The visualisation of the clustering analysis, SOM map, is shown on the Figure 11.1 on the left side. The similar regions in the map are blue and the boundaries between similar regions are yellow or even red. There is a way how to divide the map into crisp cluster and they are shown on the same figure on the right side. The black dots in the U-Matrix on the Figure 11.1 marks where are the preprocessed attributes in all the datasets. In the end to so many attributes were preprocessed. From 312 attributes only 52 were transformed.

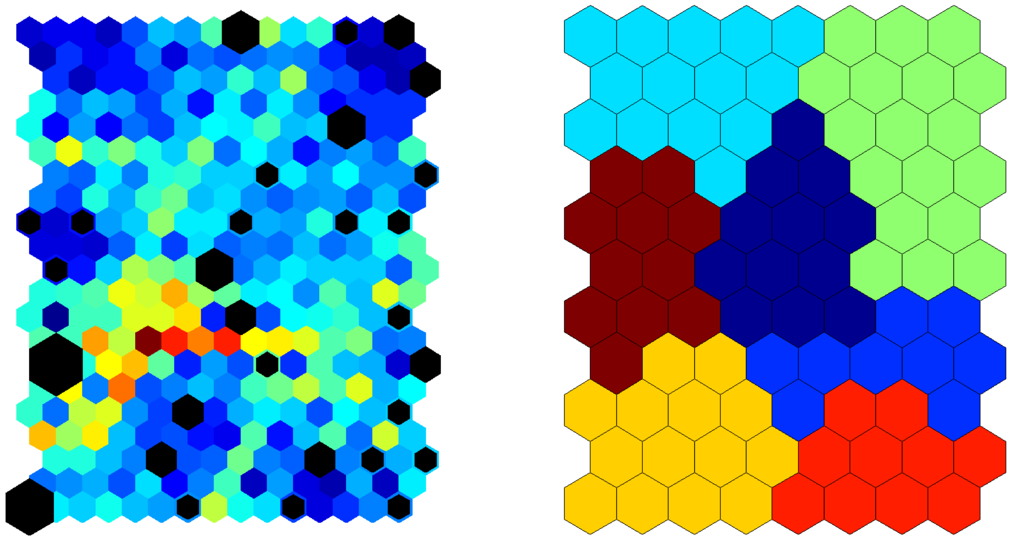


Figure 11.1: The U Matrix visualisation of the SOM clustering on the left and the map divided into clusters on the right. The picture gives ideas about distances in the SOM map. The blue regions contains rows from meta database that are more similar and the yellow or even red regions shows dissimilar regions. The regions with the different colours on the right side represents different clusters.

The SOM map on the left side of the Figure 11.1 is divided nicely into distinct clusters. This shows that there are different groups with different typical properties. It is important to know the typical values for the individual SOM nodes (neurons). In the SOM map it can be obtained using so called feature plots. The feature plot displays input variables separately and shows regions in the map with higher (white) or lower (dark) values in the given variable. The Figure 11.2 shows feature plots for selected metadata.

The feature plots shows for example that the attributes with more output classes should belong into lower part of the map. The attributes with the high predictive power for the single-node decision tree belongs to right top corner and the attributes from datasets with high inequality in number of instances for different output classes are clustered on the right side of the map and so on. The rules about regions with low or high values can not be used strictly. Sometimes other meta data values forces an attribute for example with low mean into a region with high mean value.

The next visualisation on the Figure 11.3 shows distribution of two datasets – the *Bank* and the *Breast Cancer* – in the SOM map. The SOM map is the same and the positions of the attributes in the map are shown by the black marks. The map shows that attributes for the *Bank* dataset are mainly in the left-bottom part of the map, but there is a big number of exceptions, while

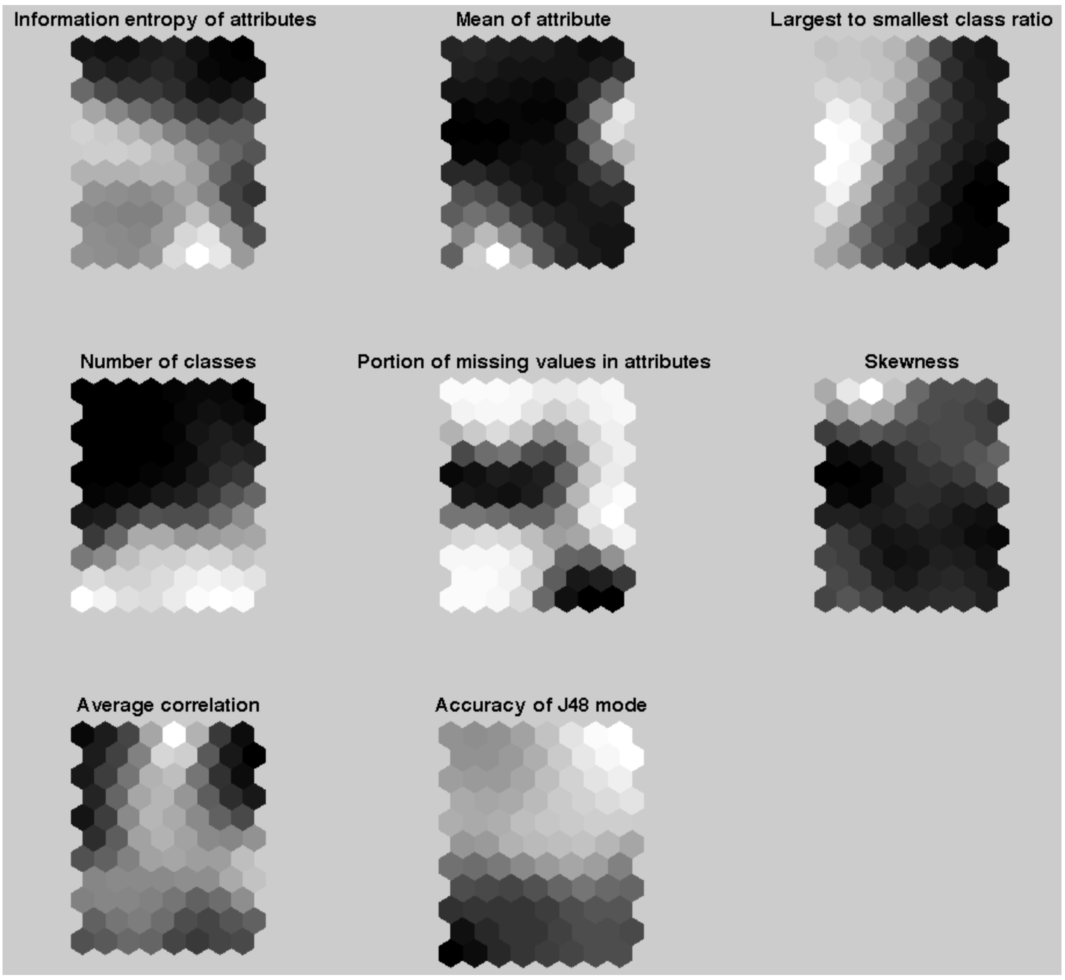


Figure 11.2: The feature plot is showing the distribution of the typical metadata values in the map.

attributes for the *Breast Cancer* dataset are generally more in the right half. But the big picture is that the attributes are distributed all over the map. This shows that the attributes, though they are from one dataset, do not have the same values in meta data.

The Figure 11.4 represents the last clustering visualisation. It shows attributes preprocessed by the *N-th Power Calculator* preprocessing method on the left and by the *Another Instance Value Imputer* on the right. Positions of the preprocessed attributes are shown by the black marks.

The first impression is that there are only few attributes preprocessed by these two preprocessing methods. The *Another Instance Value Imputer* is used to preprocess 20 attributes and the *N-th Power Calculator* is used 6 times out of 312 attributes. The *Another Instance Value Imputer* is the most used preprocessing method but the typical number of method usages is closer to the *N-th Power Calculator.*

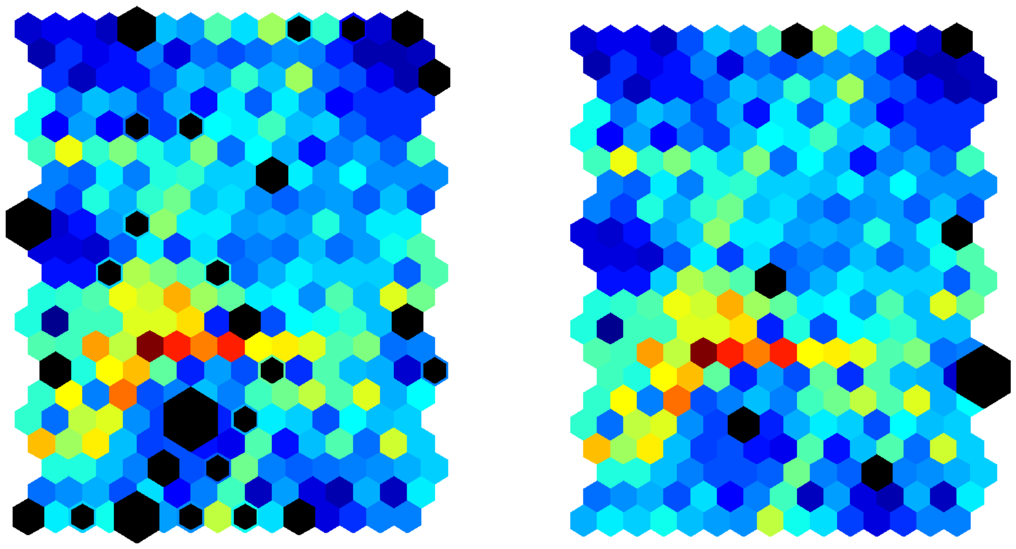


Figure 11.3: The distribution of attributes of the Bank on the left and the Breast Cancer on the right Datasets in the SOM map.

11.2.2 Finding Rules

In this section I will try to extract some rules when to preprocess an attribute and which preprocessing method to use.

The Figure 11.5 shows the text representation of the best found decision tree describing if to preprocess an attribute or not. The input variables of the tree are the metadata values about the individual attributes and the output is information if any preprocessing method was applied or not. The not preprocessed attributes are more numerous than the preprocessed ones. For this reason I have created a training set using about a half metadata for the preprocessed attributes and the corresponding number of not preprocessed attributes. The remaining part of the meta database is used for testing.

Table 11.1: Confusion matrices for model predicting I should apply any preprocessing method on an attribute or not.

	Classified as	
In reality	Preprocess	Not Preprocess
Preprocess	17	9
Not Preprocess	42	210

The results for the testing set are shown in the Table 11.1 in form of the confusion matrix. The overall accuracy is about 81.6%, but usage of the results in IPT is questionable. The aim of this experiment was to identify attributes, that should not be preprocessed and therefore left untouched by IPT. This would allow the search for the preprocessing methods to concentrate on the attributes that can be successfully preprocessed. But the confusion matrix shows that the decision tree makes mistakes for both classes. About a third of *should-be* preprocessed attributes are marked as *not-to-be-preprocessed* and about 16% of *not-to-be-preprocessed* attributes are marked as to *should-be*

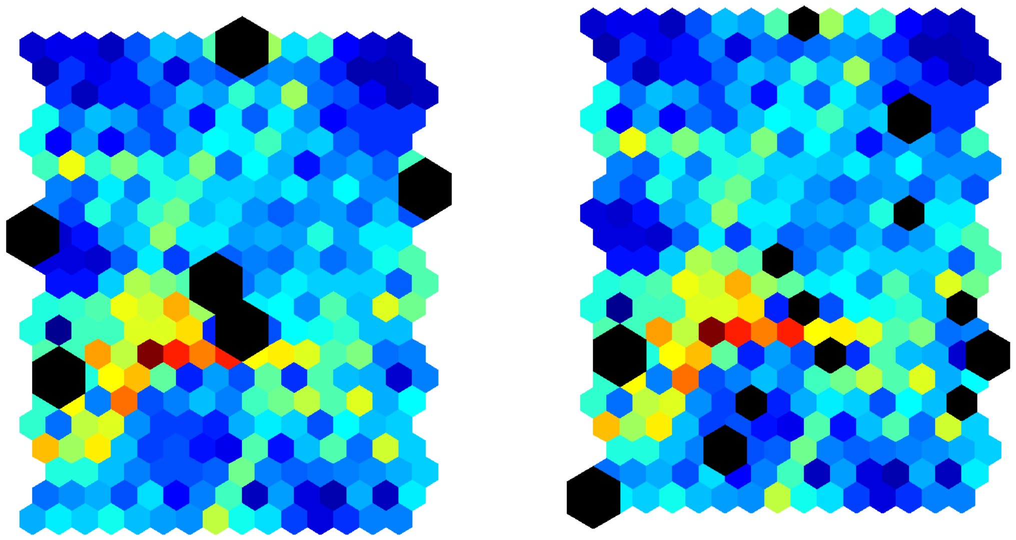


Figure 11.4: U Matrix with marks showing attributes preprocessed with the *N-th Power Calculator* on the left and with the *Another Instance Value Imputer* on the right.

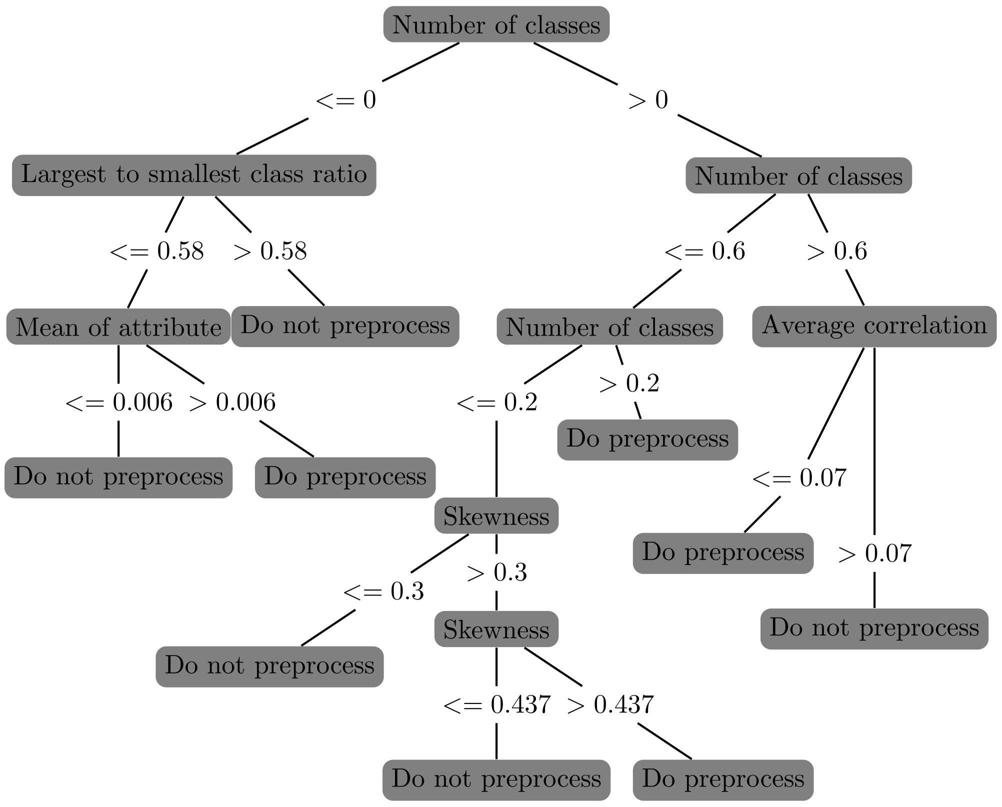


Figure 11.5: Decision trees classifying if I should preprocess an attribute or not. Nodes in the tree use the meta data from the database.

preprocessed. The second type of error – misclassification of the *not-to-be-preprocessed* attributes – is not so serious, it would just slow down IPT a little. The other type of error is much worse, because it means that I would omit some important attributes from IPT.

I have tested a decision tree for predicting if I should apply the *Another Instance Value Imputer* on an attribute. But the results are even less encouraging than in the case presented above. The bias in the number of instances in classes is in this case enormous – 20 instances that should be preprocessed using *Another Instance Value Imputer* versus 292 that should not be preprocessed. I have tested several classifier but the bias in numbers is too big for all of them and none was able to distinguish the attributes to be preprocessed. I have tested the stratified sampling and data enrichment to balance the number of instances, but the results are equally as bad.

11.3 Building the Meta database and Its Performance on Validation Data

In this section I will describe how to use of the meta database in the other way – to generate meaningful sequences for the initial population on IPT. The generated sequences then can improve results and mainly the speed of IPT. To generate the sequences of preprocessing methods I will use nearest neighbours approach. I will find the most similar attributes and I will use them to generate sequences in the initial population of IPT.

The meta database contains the metadata for each attribute and the sequences from the three most accurate sequences of preprocessing methods as presented in the previous chapter and the Appendix D. I have selected the three sequences because in general the first three sequences are comparable in fitness (accuracy) to each other and therefore they contain useful preprocessing methods. The attribute properties in the meta database are calculated from the training parts from the previous chapter. The properties or meta data to calculate are discussed earlier in this chapter and also in the chapter 6.

The very first part of this experiment verifies that the meta data describes the datasets properly. I have calculated the meta data from the validation parts of the datasets and I will check if in terms of meta data the validation attributes are similar to their training counterparts. I can not expect the 100% match nor that the attribute from the validation part will always be the closest match to its training part. The reasons for differences come from the fact, that the training parts are in general shorter than validation parts and in case of outliers present in the validation part of a dataset, the characteristics may change a lot. Especially the *value range in the attribute*, the *sample skewness* and also the *accuracy of the J48 decision tree classifier using only the attribute.* For this reason I will not concentrate only on the nearest (most similar) attribute but the three nearest attributes. The other reason to use more than the nearest attributes is that more attributes gives me more potentially useful sequences I can combine to create initial population for IPT.

The results are shown in the Table 11.2. The *Number of attributes* column shows the total number of input attributes in each dataset. The *Number of attributes within 3 most similar* column shows number of attributes in the validation part of a dataset that has the corresponding attribute from the training part of the dataset among the three most similar attributes.

The Table 11.2 shows a bit disappointing results. Some datasets like *Parkinsons* or *Wine* works well and the attributes from validation part are alike the attributes in the training part. But, for example, the validation parts of the *Steel Faults* or *Teeth Age* datasets are not similar to the

Table 11.2: Shows how many times is an attribute from the validation part of the dataset nearest in terms of metadata to its training part counterpart.

Dataset	Meta database for J48 decision Tree Classifier		Meta database for Logistic Regression Classifier	
	Number of training part attributes within 3 most similar to the validation part	Number of attributes	Number of training part attributes within 3 most similar to the validation part	Number of attributes
Bank	27	49	28	49
Breast Cancer Wisconsin	9	10	7	10
CTG	32	33	30	33
Ecoli	8	8	8	8
Glass	8	10	10	10
Ionosphere	14	35	8	35
Parkinsons	23	24	19	24
Satellite	30	37	29	37
Segment	2	20	1	20
Spambase	42	57	42	57
Steel Faults	0	28	8	28
Teeth Age	0	17	0	17
Wine	12	13	10	13

training parts. The both meta databases performs more or less the same. So if attributes in validation part of a dataset are similar to their training counterparts in one meta database, they are also similar in the other meta database.

To discuss the difference between attributes in the training and the validation parts I have created the Table 11.3 showing the attribute meta data for one selected typical situation. The shown attribute is the *CLDx* attribute of the Teeth Age dataset. The training part of the *CLDx* attribute is not among the 3 most similar attributes. The differences in distance are not big, but still there two other CTG attributes who are closer to the *CLDx*. The main reason for this result are the values in *Portion of Missing Values*, the *J48 Decision Tree accuracy*, the *Log Number of Instances* and the *Sample Variance*. Do not forget that the values displayed in the Table 11.3 are not used directly for the distance calculation, but they are normalised. So if there is a big difference shown in the Table between values in the *Number of Attributes*, its contribution to the distance is much smaller than is suggested here.

The difference in the number training part attributes among the three nearest to the validation part attribute varies between the J48 meta database and the Simple Logistic Regression meta database mainly due to *J48 Decision Tree accuracy* and the *Logistic Regression accuracy* meta data. The difference in values is caused by the accuracy calculation process and model initiation as discussed in this thesis many times before.

Table 11.3: Example of metadata values for the *CLDx* attribute of the validation part of the Teeth Age dataset. The table compares the meta data for validation part of the *CLDx* attribute to training part and the training part of the *DL* attribute of the CTG dataset.

	Teeth Age, V, *CLDx*	Teeth Age, T, *CLDx*	CTG, T, *DL*
Distance	—	0.10769	0.095978
Information Entropy	2.95	2.86	2.03
Sample Mean	0.947	1.17	0.681
Log Value Range	0.845	0.845	0.903
Most to least numerous output class ratio	0.0780	0.0806	0.149
Number of Classes	5.00	5.00	3.00
Portion of Missing Values	0.0051	0.0139	0.00
Sample Variance	0.634	0.563	0.638
Sample Skewness	-0.487	-0.421	1.91
ø Abs Correlation to Classes	0.0360	0.0519	0.110
J48 Decision Tree accuracy	0.623	0.579	0.726
Logistic Regression accuracy	0.365	0.526	0.733

11.3.1 Performance of the Generated Sequences

So far, I have created the meta database and I have generated the sequences of the preprocessing methods based on the properties of the validation part of the datasets. To generate new sequences I will first select the three most similar attributes to each attribute in the new dataset. The meta database for each of these attributes contains three subsequences[1] attached to these attributes. This selects the 9 subsequences to be used for generation. Then I will randomly select subsequences for each attribute among the 9 subsequences found in the meta database. In this way I will generate as many new sequences as needed. In this experiment I have generated 10 new sequences. Plus I will combine a new sequence in following way: for each attribute I will select the most similar attribute from the meta database. This most similar attribute in the meta database has 3 subsequences and one of them is a part of the sequence with the highest fitness value. And I will use this subsequence as a part of the new, generated sequence.

Now is the time to test the performance of the generated sequences. I will calculate the fitness value (and the accuracy of the models) using the same algorithm as in previous chapter (see the Chapter 4). And as I use the same datasets as in the previous sections I can directly compare the accuracy of the resulting models with the accuracy of the sequences found by IPT.

The results for the J48 Decision Tree model and corresponding sequences are shown on the Figure 11.6. The Figure compares accuracy for the original, not-preprocessed, dataset – denoted on the Figure as **No Prep** – the accuracy of the generated sequence – denoted on the Figure as the **metaDB** – and the sequence found by IPT – denoted as the **Best IPT**.

The Figure 11.6 shows that for four datasets (CTG, Glass, Satelite and Steel Faults) the best generated sequences are comparable to the sequences generated by IPT. In other cases the fitness of the generated sequences are better, in most of the cases even statistically better, than the

[1]The subsequence is a part of sequence of preprocessing methods and contains preprocessing methods which should be applied to given attribute.

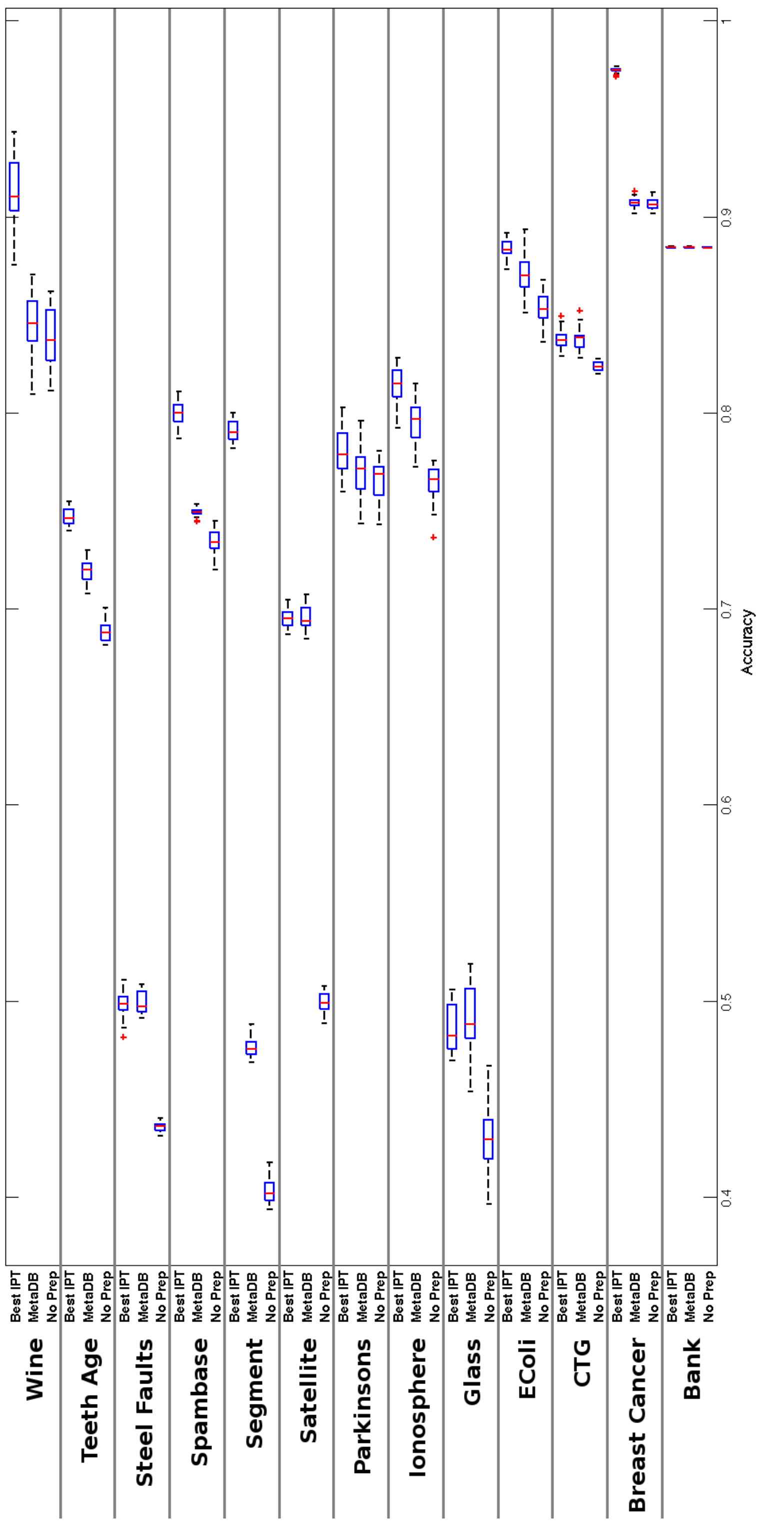


Figure 11.6: Comparison of the fitness (accuracy) of J48 Decision Tree model on original datasets, sequences generated from the J48 meta database and the sequences generated by IPT.

accuracy of the models with the original, not preprocessed, datasets, but they achieve lower fitness than the best sequences found by IPT. Though it is not the best possible result it shows a very good starting point. These generated individuals will be useful for the initial population and will should provide additional information and guidance for IPT.

The fact that the training part of the attributes were not among the nearest attributes for validation parts of the *Teeth Age*, *Steel Faults* or the *Segmentat* datasets obviously does not harm the accuracy of the generated sequences. The generated sequences for the *Steel Faults* dataset is comparable to the best sequence found by IPT and the generated sequence for the *Teeth Age* and *Segmentation* achieve better accuracy of trained models than the models with not preprocessed datasets.

The sequences generated from the Simple Logistic meta database shows the same results and it would be waste of space to discuss them further. But you can examine its performance the Figure E.1 on the page 182.

11.3.2 The Generated Sequences

In this subsection I will present some of the generated sequences and I will discuss their differences to the best sequences found by IPT.

11.3.2.1 J48 Meta Database and the Generated Sequence for the Glass Dataset

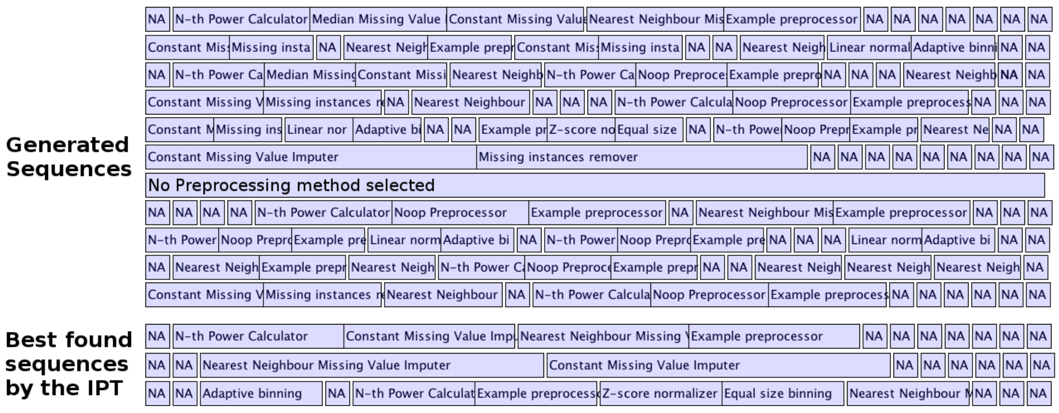

Generated Sequences
NA N−th Power Calculator Median Missing Value Constant Missing Valu Nearest Neighbour Mis Example preprocessor NA NA NA NA NA NA NA
Constant Mis Missing insta NA Nearest Neigh Example prep Constant Mis Missing insta NA NA Nearest Neigh Linear normal Adaptive binni NA NA
NA N−th Power Ca Median Missing Constant Missi Nearest Neighb N−th Power Ca Noop Preproce Example prepro NA NA NA Nearest Neighb NA NA
Constant Missing V Missing instances r NA Nearest Neighbour NA NA NA N−th Power Calcula Noop Preprocessor Example preprocess NA NA NA
Constant M Missing ins Linear nor Adaptive bi NA NA Example pr Z−score no Equal size NA N−th Powe Noop Prepr Example pr Nearest Ne NA NA
Constant Missing Value Imputer Missing instances remover NA NA NA NA NA NA NA NA NA
No Preprocessing method selected
NA NA NA NA N−th Power Calculator Noop Preprocessor Example preprocessor NA Nearest Neighbour Mis Example preprocessor NA NA NA
N−th Power Noop Prepr Example pre Linear norm Adaptive bi NA N−th Power Noop Prepr Example pre NA NA NA Linear norm Adaptive bi NA NA
NA Nearest Neigh Example prepr Nearest Neigh N−th Power C Noop Preproc Example prepr NA NA Nearest Neigh Nearest Neigh Nearest Neigh NA
Constant Missing V Missing instances r Nearest Neighbour NA N−th Power Calcula Noop Preprocessor Example preprocess NA NA NA NA NA NA
Best found sequences by the IPT
NA N−th Power Calculator Constant Missing Value Imp Nearest Neighbour Missing Example preprocessor NA NA NA NA NA NA NA
NA NA Nearest Neighbour Missing Value Imputer Constant Missing Value Imputer NA NA NA NA NA NA
NA NA Adaptive binning NA N−th Power Calculat Example preprocess Z−score normalizer Equal size binning Nearest Neighbour M NA NA NA

Figure 11.7: Generated sequences for the Glass Dataset from the J48 Meta Database.

The first set of the sequences I want to present are the sequences generated for the *Glass* dataset and the J48 Decision Tree. The seqquences are shown on the Figure 11.7. The figure shows 11 generated datasets and the three best sequences found for the *Glass* dataset by IPT in the previous chapter. Both, the generated sequences and the best sequences found by IPT are sorted by the their fitness value. The fitnesses of the both best sequences – generated and found by IPT – are shown on the Figure 11.6 below and they are comparable.

The close examination of the sequences even show that the both best sequences are exactly the same. The Figure 11.8 shows boxplots for the fitness values of the generated sequences. The remaining sequences are more or less comparable to the to the original, not preprocessed, dataset.

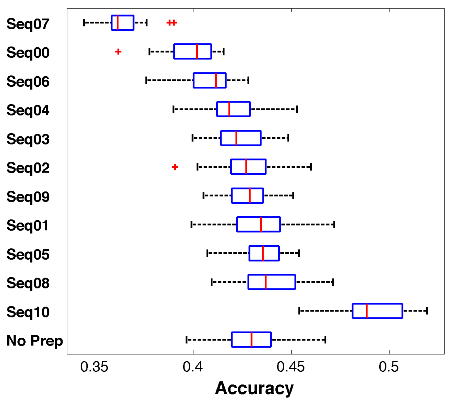


Figure 11.8: Fitness values for the sequences generated for the validation part of the *Glass* dataset from the J48 meta database.

Based on the fitness values and the generated sequences, I can conclude that the preprocessing methods leading to better accuracy of the resulting model are the *N-th Power Calculator* and the *Median Missing Value Imputer* applied on the second attribute and the *Nearest Neighbour Missing Value Imputer* applied on the third attribute. The *Constant Missing Value Imputer* has no effect on the second attribute since there are no missing values left after the *Nearest Neighbour Missing Value Imputer* is applied. The *Example Preprocessor* adds only a constant value to each value in the attribute thus it also has very little practical importance.

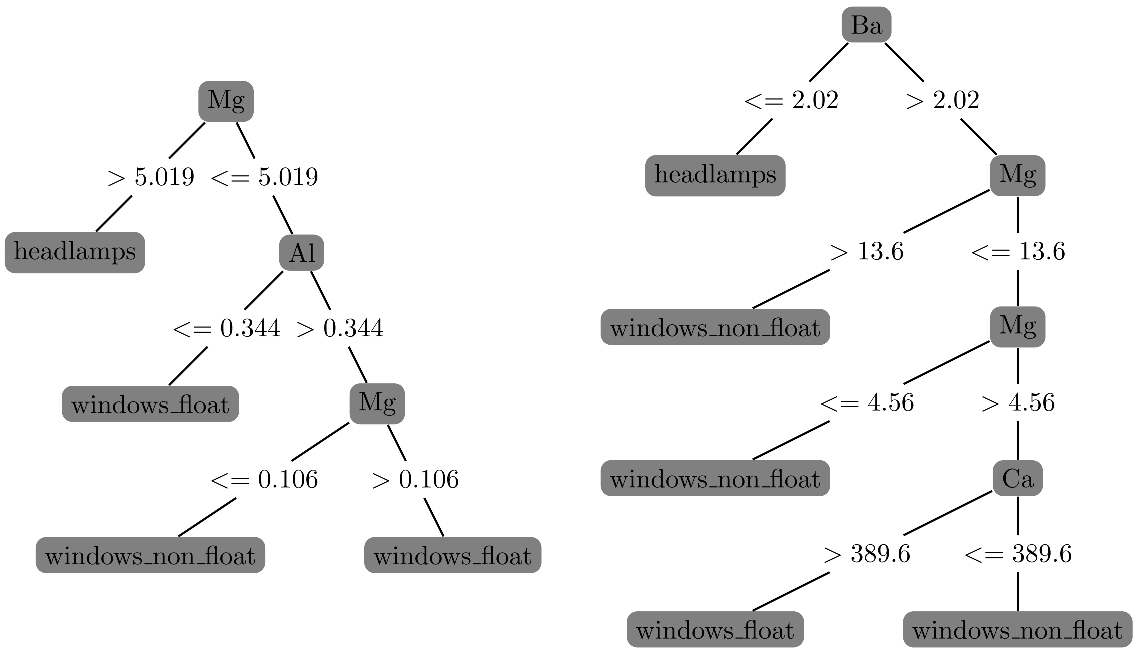


Figure 11.9: The example J48 Decision Trees for the *Glass* dataset. The left tree was created with the dataset preprocessed by the best IPT sequence, the right tree was created from the dataset preprocessed by the best generated sequence.

The third generated sequence from the top is quite similar to the best generated sequence but has some preprocessing methods added. Although it may be seen as and advantage in fact it is not.

The more preprocessed attributes causes the J48 training algorithm to select another attributes for decision and in this way to harm the classification accuracy. The Figure 11.9 shows the decision trees generated from the data preprocessed by those two sequences. On the left is the tree for the data preprocessed by the first generated sequence and on the right the tree for the third sequence.

11.3.2.2 J48 Meta Database and the Generated Sequence for the Teeth Age Dataset

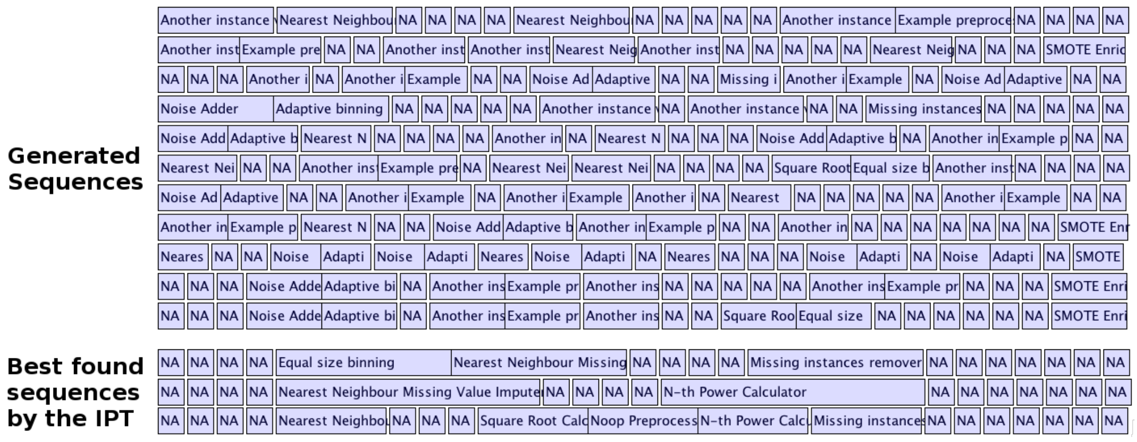

Generated Sequences
Another instance | Nearest Neighbou | NA | NA | NA | NA | Nearest Neighbou | NA | NA | NA | NA | NA | Another instance | Example preproce | NA | NA | NA | NA
Another inst | Example pre | NA | NA | Another inst | Another inst | Nearest Neig | Another inst | NA | NA | NA | NA | NA | Nearest Neig | NA | NA | NA | SMOTE Enric
NA | NA | NA | Another i | NA | Another i | Example | NA | NA | Noise Ad | Adaptive | NA | NA | Missing i | Another i | Example | NA | Noise Ad | Adaptive | NA | NA
Noise Adder | Adaptive binning | NA | NA | NA | NA | NA | Another instance | NA | Another instance | NA | NA | Missing instances | NA | NA | NA | NA | NA
Noise Add | Adaptive b | Nearest N | NA | NA | NA | NA | Another in | NA | Nearest N | NA | NA | NA | Noise Add | Adaptive b | NA | Another in | Example p | NA | NA
Nearest Nei | NA | NA | Another ins | Example pre | NA | Nearest Nei | Nearest Nei | NA | NA | NA | NA | Square Root | Equal size b | Another inst | NA | NA | NA | NA
Noise Ad | Adaptive | NA | NA | Another i | Example | NA | Another i | Example | Another i | NA | Nearest | NA | NA | NA | NA | NA | Another i | Example | NA | NA
Another in | Example p | Nearest N | NA | NA | Noise Add | Adaptive b | Another in | Example p | NA | NA | Another in | NA | NA | NA | NA | NA | NA | NA | SMOTE Enr
Neares | NA | NA | Noise | Adapti | Noise | Adapti | Neares | Noise | Adapti | NA | Neares | NA | NA | NA | Noise | Adapti | NA | Noise | Adapti | NA | SMOTE
NA | NA | NA | Noise Adde | Adaptive bi | NA | Another ins | Example pr | Another ins | NA | NA | NA | NA | NA | Another ins | Example pr | NA | NA | NA | SMOTE Enri
NA | NA | NA | Noise Adde | Adaptive bi | NA | Another ins | Example pr | Another ins | NA | NA | Square Roo | Equal size | NA | NA | NA | NA | NA | NA | SMOTE Enri
Best found sequences by the IPT
NA | NA | NA | NA | Equal size binning | Nearest Neighbour Missing | NA | NA | NA | NA | Missing instances remover | NA | NA | NA | NA | NA | NA | NA
NA | NA | NA | NA | Nearest Neighbour Missing Value Impute | NA | NA | NA | NA | N-th Power Calculator | NA | NA | NA | NA | NA | NA | NA
NA | NA | NA | NA | Nearest Neighbo | NA | NA | NA | Square Root Calc | Noop Preprocess | N-th Power Calc | Missing instance | NA | NA | NA | NA | NA | NA | NA

Figure 11.10: Generated sequences for the *Teeth Age* Dataset and the J48 Meta Database.

I will not discuss other sequences in that much details as the previous case. All the generated sequences for the Teeth Age dataset are shown on the Figure 11.10. Again I have added the three sequences with the highest fitness found by IPT. The accuracy of the J48 model with the training data preprocessed by the best generated sequence is in between the accuracy of the J48 trained with the non preprocessed and the accuracy of the model with the data preprocessed by the best sequence found by IPT.

The generated sequences are completely different from the best sequences found by IPT. The reason is in the fact that the meta data failed to match the training and the testing attributes of the *Teeth Age* dataset. And in this case the supposition that the similar meta data values brings the same preprocessing methods is not completely true. The generated sequence improves accuracy of the model but the sequence found by IPT performs much better.

11.3.2.3 Logistic Regression Classifier Meta Database and the Generated Sequence for the Breast Cancer Wisconsin Dataset

The generated sequences for the *Breast Cancer Wisconsin* dataset are shown on the Figure 11.11. You can compare the performance of the best generated sequence and the best sequence found by IPT on the Figure E.1 on the page 182. The figure shows that the performance of the fitness value of the generated sequence is even slightly higher than the fitness of the best sequence found by IPT.

The reason for the better performance lies in the *N-th Power Calculator* preprocessing method applied to the 6^{th} attribute and the absence of the *Mean Value Normalizer* and the *Another*

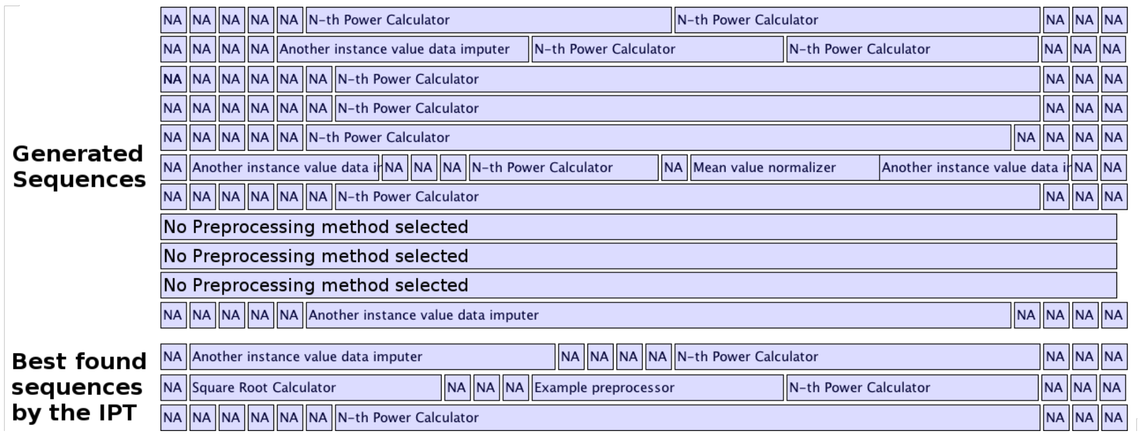


Figure 11.11: Generated sequences for the *Breast Cancer* Dataset and the Logistic Regression Classifier database.

*Instance Value Imputer*preprocessing methods. But the difference in the accuracy is not too big, just about a 1%.

11.3.2.4 Logistic Regression Classifier Meta Database and the Generated Sequence for the Satellite Dataset

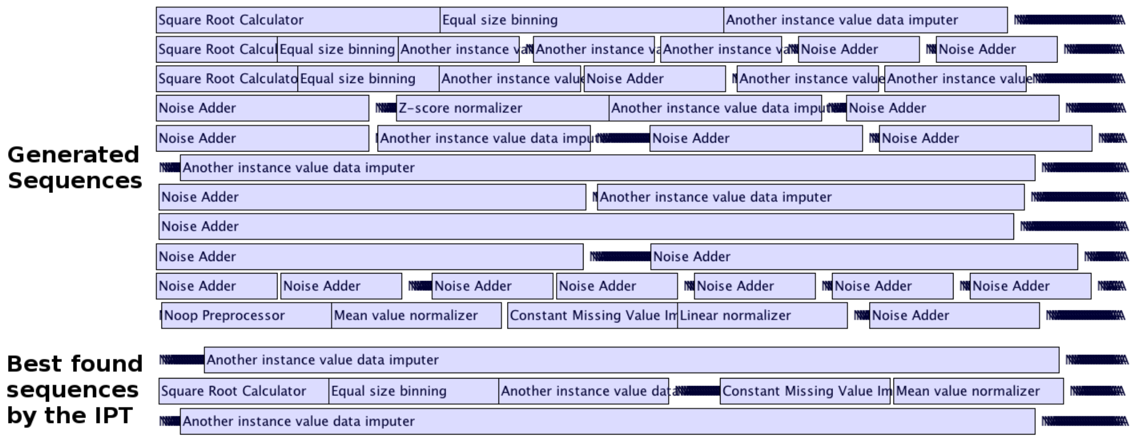


Figure 11.12: Generated sequences for the Satellite Dataset from the Logistic Regression Classifier database.

The last set of sequences is shown on the Figure 11.12. The best generated sequences are quite different from the best sequences found by IPT. But in terms of the fitness the generate sequences are slightly better, see the Figure E.1 on the page 182.

The reason for the better performance lies in the fact that the best generated sequence preprocess the first attribute. In fact the first three sequences from the top, the ones with the highest fitness among the generated sequences, have comparable accuracy and all of them treat the first attribute in the same way.

11.4 Meta Database on Modified Datasets

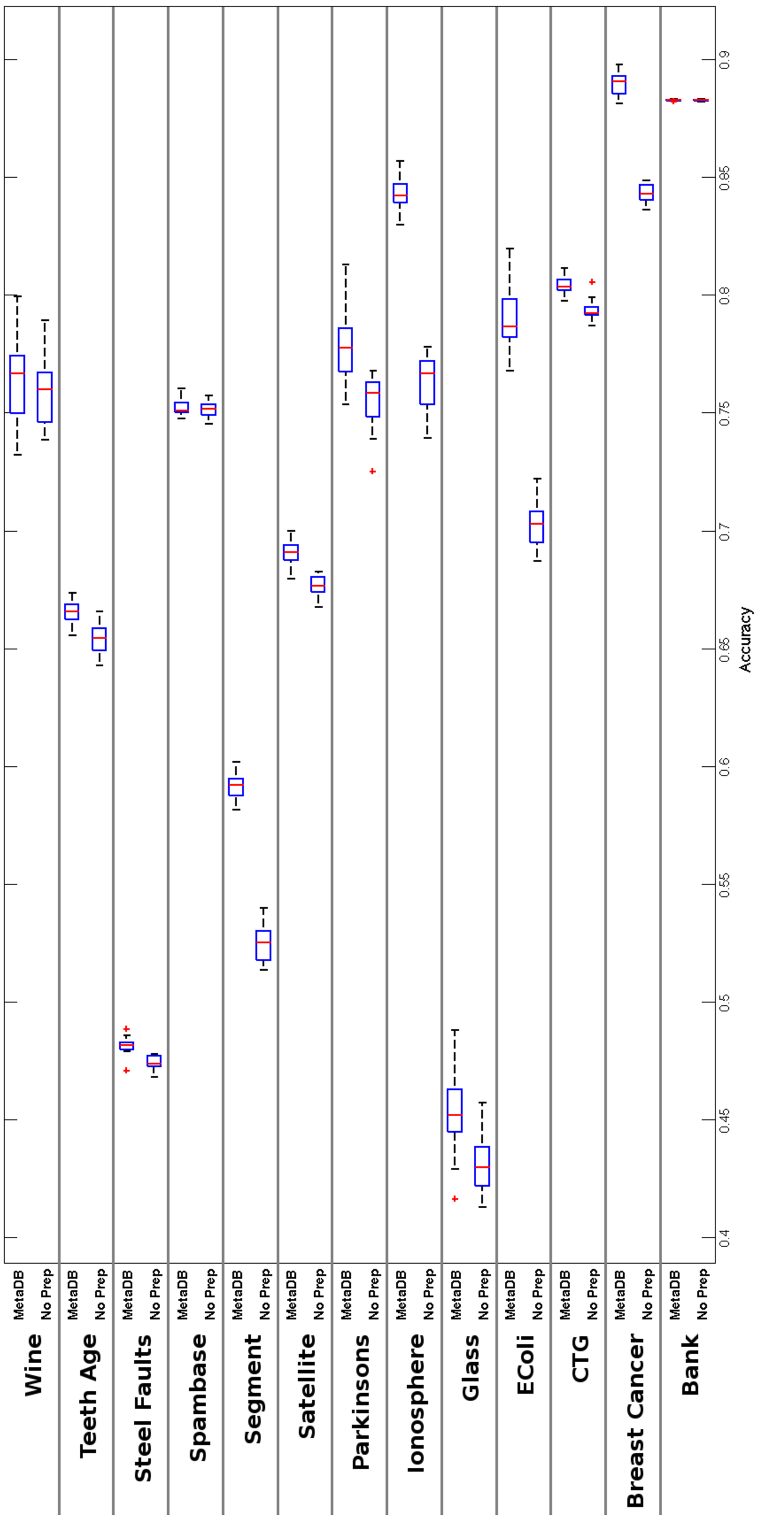


Figure 11.13: Comparison of the fitness (accuracy) of original datasets and datasets preprocessed by sequences generated from the meta database.

In this section I will show results for the generated sequences on the modified datasets. The aim of this experiment is to test generalisation capabilities of the meta database. In other words I want to test if a relatively small change in properties of the datasets will result in a very different generated sequences. And also if the generated sequences are able preprocess a dataset in a way that it will achieve better fitness than the original, not preprocessed, dataset.

As I have said in the previous chapter I the real world datasets I use through this and previous chapter are known to have quite good accuracy with many models. To give some room for improvement of the accuracy I have transformed the datasets from the UCI (see 10.1 for details). This gives me an opportunity to transform the datasets with different parameters, eg. introducing different amount of missing values, doing different non-linear transformations and so on.

I have used the same meta database as before to generate the sequences. And then I have calculated fitness of the generated sequences. The results of the best sequences in comparison to the non-preprocessed dataset are shown on the Figure 11.13. I have not ran IPT for the datasets therefore I compare only the generated sequences to the non preprocessed datasets. The results are for the J48 Decision tree meta database and the J48 models.

The Figure 11.13 shows that even though I have changed the properties of the datasets, the generated sequences are able to preprocess the changed dataset effectively and the models trained with the preprocessed datasets in all cases achieve better fitness than the not preprocessed datasets. The improvement in the accuracy is usually a bit smaller than in the previous case but still if there was improvement in the accuracy of model in the previous section, the accuracy of the model with the modified dataset is improved as well. Do not forget that the datasets here are different from the datasets used in the previous section and even accuracy of models with the not preprocessed can not be directly compared.

The results for the Simple Logistic Regression Classifier are very similar to the results presented here. You can examine the results in the appendix on the Figure E.2 on the page 183.

11.4.1 Generated Sequences

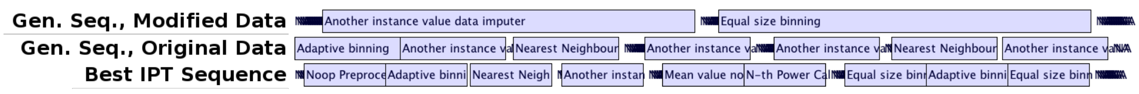


Figure 11.14: The best sequences generated from the J48 meta database for the modified and the original *Ionosphere* dataset.

Here I want to present and describe selected generated sequences and find its differences to the sequences from the previous section. In contrast to the previous section I will compare only the best generated sequences for the modified datasets, the best generated sequence for the original dataset and the best sequence found by IPT.

I will present generated sequences for the *Ionosphere* and the *Segment* datasets were generated from the J48 Meta Database and the generated sequences for the *Breast Cancer* and the *Steel Faults* datasets.

The generated sequences are shown on the Figures 11.14, 11.15, 11.16 and 11.17. All the Figures

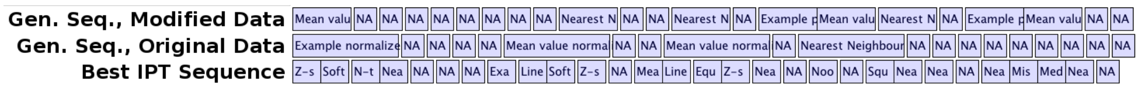

Gen. Seq., Modified Data
Mean valu NA NA NA NA NA NA NA NA Nearest N NA NA Nearest N NA Example p Mean valu Nearest N NA Example p Mean valu NA NA
Gen. Seq., Original Data
Example normalize NA NA NA NA Mean value normal NA NA Mean value normal NA Nearest Neighbour NA NA NA NA NA NA NA NA NA
Best IPT Sequence
Z-s Soft N-t Nea NA NA NA Exa Line Soft Z-s NA Mea Line Equ Z-s Nea NA Noo NA Squ Nea Nea NA Nea Mis Med Nea NA

Figure 11.15: The best sequences generated from the J48 meta database for the modified and the original *Segment* dataset.

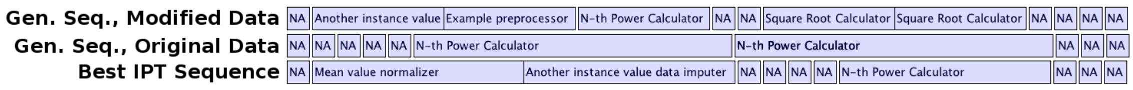

Gen. Seq., Modified Data
NA Another instance value Example preprocessor N-th Power Calculator NA NA Square Root Calculator Square Root Calculator NA NA NA
Gen. Seq., Original Data
NA NA NA NA NA N-th Power Calculator N-th Power Calculator NA NA NA
Best IPT Sequence
NA Mean value normalizer Another instance value data imputer NA NA NA NA N-th Power Calculator NA NA NA

Figure 11.16: The best sequences generated from the Simple Logistic Regression meta database for the modified and the original *Breast Cancer* dataset.

shows that the generated sequences for the modified datasets are not too similar to sequences for the original datasets, not mentioning the sequences found by IPT. The sequences for the modified datasets use the same subsequences of the preprocessing methods but for different attributes.

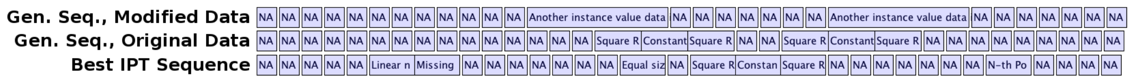

Gen. Seq., Modified Data
NA NA NA NA NA NA NA NA NA NA NA NA Another instance value data NA NA NA NA NA NA NA Another instance value data NA NA NA NA NA NA NA
Gen. Seq., Original Data
NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA Square R Constant Square R NA NA Square R Constant Square R NA NA NA NA NA NA NA NA NA
Best IPT Sequence
NA NA NA NA NA Linear n Missing NA NA NA NA NA NA NA Equal siz NA Square R Constan Square R NA NA NA NA NA NA NA N-th Po NA NA NA NA

Figure 11.17: The best sequences generated from the Simple Logistic Regression meta database for the modified and the original *Steel Faults* dataset.

The conclusion is that the generated sequences of the preprocessing methods looks differently for the modified datasets. But the models trained with the preprocessed modified datasets achieve better accuracy than the models with the not preprocessed datasets. And in some sequences the difference between accuracy of the model with non preprocessed dataset and the preprocessed dataset is even bigger than in the case of the original datasets presented above. Thus the meta data show at least in some degree ability to generalise.

11.5 Meta Database on Unknown Datasets

The last experiment demonstrates the ability of the meta databases to find the generated sequences on completely unknown datasets and ability to speedup IPT. I have downloaded from the UCI Machine Learning Repository four datasets – the Credit Approval, the Heart Disease, the Indian Liver Patient and the Pima Indian Diabetes. I have transformed the datasets with parameters yet different from the previous sections.

The main aim of this experiment is to test the influence of the improved initial population to IPT. Also I want to improve speed of IPT. To do this I will compare following values:

- the accuracy of a model with a non preprocessed dataset,
- accuracy of a model with dataset preprocessed with the best of the generated sequences,
- accuracy of model with the dataset preprocessed with the best sequence found by IPT after 10 generations from the random initial population,

- accuracy of model with the dataset preprocessed with the best sequence found by IPT after 10 generations from the initial population with the generated sequences,
- accuracy of model with the dataset preprocessed with the best sequence found by IPT after 50 generations from the random initial population.

The results for the experiments are shown on the Figures 11.22 and 11.23. The Figures show very similar story. The lowest accuracy of models achieve datasets without preprocessing, denoted on the figures as *No Preprocessing*. The second worst result is achieved by the sequences generated from the meta database. They are denoted on the figures as the *Meta DB Sequences*. The accuracy of models trained with the dataset preprocessed with the generated sequences is only a small bit better than the not preprocessed datasets. This shows that the new datasets are too different from the datasets used to create the meta database. On the other hand, the small improvement gives hopes that the generated sequences directs the search in the correct way.

Now I come to boxplots that interests me very much – the sequences found by IPT. The first boxplot shows the best sequence found by IPT in only 10 steps of the sequence search algorithm, with generated sequences in the initial population. This is denoted in the Figures 11.22 and 11.23 as *IPT 10 Steps, Meta DB Init*. The second boxplot shows the accuracy of the model trained with the dataset preprocessed with the best sequence found in 10 steps of the sequences search algorithm. But in contrast to previous boxplot the initial population for the search was randomly generated. The boxplot is denoted as the *IPT 10 Steps, Random Init*. The last boxplot shows the result of the full IPT with 50 steps of the sequence search algorithm. The boxplot is denoted as *IPT 50 Steps, Random Init*. The initial population for this was again randomly generated.

The results show that when I add the generated sequences into the initial population of IPT, it is able to find a good solutions faster. The generated sequences proved good starting point and even though the generated sequences are not well performing, they can be easily improved in a few search steps. For all datasets and both modelling methods, IPT with the generated sequences in the initial population is in 10 steps able to find sequences comparable in fitness to the sequences found in by IPT in 50 steps with the random initial population.

11.5.1 Selected Sequences

In this section I will present one set of the best generated sequences and compare them to the best sequences found by the all three variants of IPT – that is sequences found in 10 and 50 steps with random initial population and sequences found in 10 steps with generated sequences in the initial population. And I will discuss the differences.

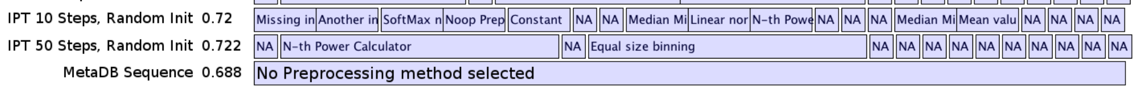

IPT 10 Steps, Meta DB Init 0.728
NA | N-th Power Calculator | NA | SoftMax normalizer | N-th Power Calculator | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA
IPT 10 Steps, Random Init 0.72
Missing in | Another in | SoftMax n | Noop Prep | Constant | NA | NA | Median Mi | Linear nor | N-th Powe | NA | NA | NA | Median Mi | Mean valu | NA | NA | NA | NA
IPT 50 Steps, Random Init 0.722
NA | N-th Power Calculator | NA | Equal size binning | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA
MetaDB Sequence 0.688
No Preprocessing method selected

Figure 11.18: The best sequences found by different variants of IPT for the Heart Disease dataset and the Simple Logistic Regression Classifier. The number on the left indicates the fitness value of the sequence.

The second set of sequences is for the *Heart Disease* dataset and the Simple Logistic Regression Classifier. The best sequences for all four situations are shown on the Figure 11.18. The differences in accuracy between the best sequences are small, still the sequence found by IPT with the generated sequences in the initial population achieves the highest fitness. What differs more in the best sequences is the structure of the sequences. The best sequences found by IPT in 50 steps with random initial population and by IPT in 10 steps with generated sequences in the initial population are less complex than best sequence found by IPT in 10 steps with the random initial population.

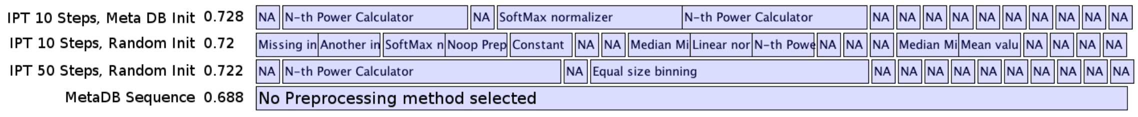


Figure 11.19: All generated sequences for the Heart Disease dataset for the Simple Logistic Regression. The first two sequences at the top are empty – they do not use any preprocessing method.

The best generated sequence does no preprocessing and returns the original dataset but the remaining generated sequences contains some useful data preprocessing methods, see the Figure 11.19. The *Nth Power Calculator* preprocessing method used un the best sequence found by IPT with generated sequences in the initial population is repeated several times in the generated sequences.

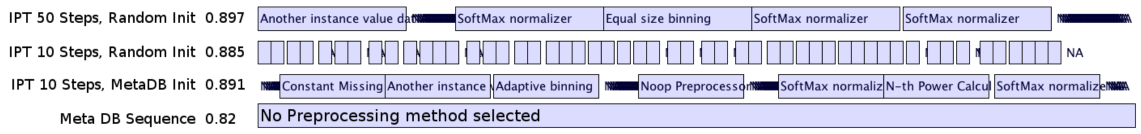


Figure 11.20: The best sequences found by different variants of IPT for the Credit Approval dataset and the Simple Logistic Regression Classifier. The number on the left indicates the fitness value of the sequence.

The best sequences for the *Credit Approval* dataset shows less favourable story. You can examine the best sequences on the Figure 11.20. The sequence for IPT with generated individuals in the initial population (denoted as *IPT 10 steps, metaDB init*) and the and the sequence for the full IPT (denoted as *IPT 50 steps, random init*) are simple and preprocess only few attributes. On the other hand the best sequence found by the shorter IPT in 10 steps with only random sequences in the initial population is a complete mess and applies among useful preprocessing methods also a lot of rubbish.

So far the results are nice, but when I start to examine the generated sequences (shown on the Figure 11.21) I came to difficulties. Almost all the generated sequences are empty – only two of the sequences contain some preprocessing method. So the high fitness and the simple sequences are more the found by a chance than with help of the generated sequences. The most similar attributes to the ones from the *Credit Approval* are the attributes from the *Spambase* dataset. This dataset contains a lot of attributes but only few of them are preprocessed. The generated sequences forces IPT to use only simpler sequences – the search algorithm can add only a limited

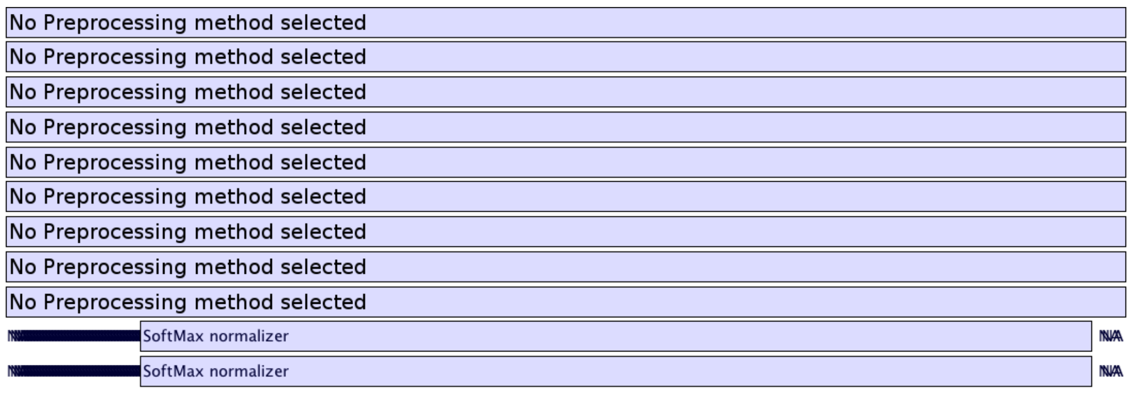

No Preprocessing method selected
No Preprocessing method selected
No Preprocessing method selected
No Preprocessing method selected
No Preprocessing method selected
No Preprocessing method selected
No Preprocessing method selected
No Preprocessing method selected
No Preprocessing method selected
SoftMax normalizer
SoftMax normalizer

Figure 11.21: All generated sequences for the Credit Approval dataset for the Simple Logistic Regression. Almost all sequences are empty – they do not use any preprocessing method.

number of the preprocessing methods on each generation and in this way the sequences are kept simple.

11.6 Conclusion

In this chapter I have experimented with the use of the properties of attributes or meta data to speed up IPT. At first I have used the SOM clustering to show that the meta data I am calculating have capability to separate different types of attributes. The results show that the clustering is able to distribute attributes with different properties to different clusters.

The next experiment was to find rules to indicate if I should preprocess an attribute or not. The results show that I am unable to find such rules. The problem is not in the overall accuracy of the classifier, but in the fact that one third of attributes that should be preprocessed are misclassified as not to be preprocessed. This would be problem when I remove such attributes from IPT and IPT would fail to find meaningful sequences of preprocessing methods. I have also tested if I can create rules saying when to use one specific preprocessing method. I have again failed. The reason is a ratio between number of all attributes and number of attributes preprocessed using given preprocessing method. To improve the results I have to collect metadata and find sequences of preprocessing methods for more datasets.

In the next section I have tested the algorithm for generating sequences using the meta data. The generated sequences for the set of datasets used to create the meta database are able to compete with the best sequences found by IPT. The sequence generation for the initial population of IPT is working. The generated sequences work well even though the datasets were modified, so the meta data and the generated sequences seems to be robust enough. Then I have used the same meta database to generate sequences for the completely new datasets, the performance were not so great. Then I have mixed the generated sequences into the initial population and ran IPT in with only 10 steps. The results of this shorter run are comparable to the results of IPT found in 50 steps with random initial population. Although the generated sequences do not perform well they have given IPT useful starting points to use and IPT has found the great results in one fifth of the time needed with random initial population.

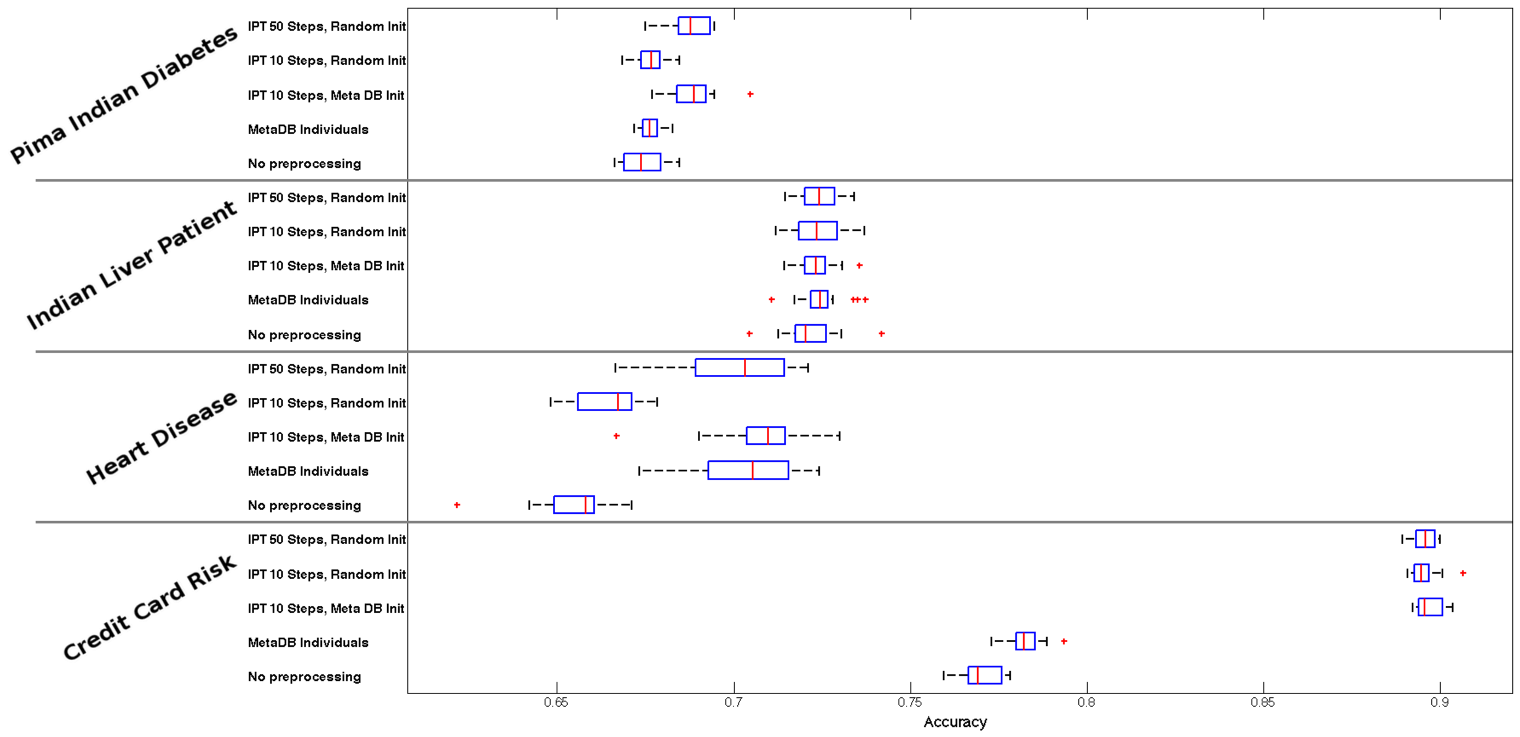


Figure 11.22: Comparisons of the accuracy of the J48 Decision Tree Classifier with the training data preprocessed by: sequences generated from the meta database, sequences found by IPT in 10 steps with generated sequences in the initial population, sequences found by IPT in 10 and 50 steps with random initial population.

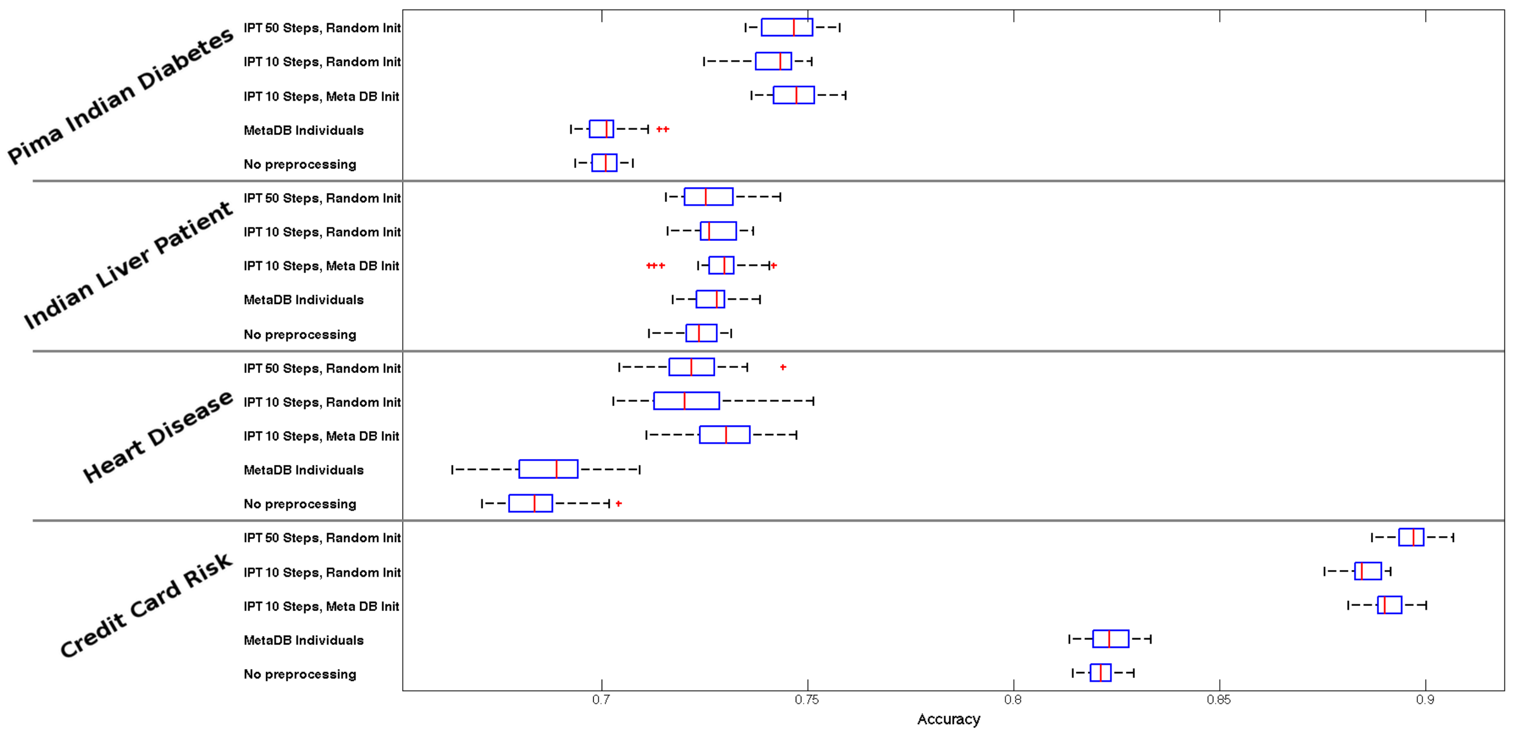


Figure 11.23: Comparisons of the accuracy of the Simple Logistic Regression Classifier with the training data preprocessed by: sequences generated from the meta database, sequences found by IPT in 10 steps with generated sequences in the initial population, sequences found by IPT in 10 and 50 steps with random initial population.

12 Thesis Conclusions, Contributions and Suggestions for Future Work

12.1 Achieved Results

In the thesis I have presented novel approach to data preprocessing called the **Inductive Preprocessing Technology**. The Inductive Preprocessing Technology (IPT) automates a part of the data preparation for data mining. There are several other approaches to aid data miners with the data preprocessing. The other approaches are based on ontology, hard coded rules or similarity to a previously transformed dataset. The novelty of IPT lies in novel combination of idea of the inductive modelling, data driven-approach and optimisation approach to the data preprocessing field.

Similarly to the inductive modelling, IPT starts with no prior knowledge about the data. As IPT progresses, it gets more information about the data and by adding, removing and modifying the preprocessing methods, it is searching for the simplest sequence of the preprocessing methods that achieves the highest accuracy of the classifier. IPT tries to find the sequence only as complex (and contains only the preprocessing methods) as it is needed to preprocess the dataset correctly and to maximise the accuracy of the classifier. The optimisation approach is used to find the correct modification of the sequences of preprocessing methods to increase the accuracy of the classifier.

The main goal of this thesis has been accomplished. IPT is able to automatically find the best sequences of the preprocessing methods for tested datasets. The accuracy of the classifier for real world datasets is improved by 5% to 10%. The detailed experimental results are shown in the next section.

12.2 Experimental Results

To test if IPT fulfils the goals, I have presented in the introduction, I have done following steps:

- in the Chapter 7 I have investigated that the preprocessing of a training set has really an influence on the accuracy of the trained model. To test this I have created four artificial datasets and showed accuracy of models with datasets preprocessed with different preprocessing methods. Later I have tested ability of different search methods to find the expected sequences of preprocessing methods and accuracy of the classifiers. The results shows that the preprocessing really affects the accuracy of a classifier. Also they show that all the tested search methods can find the expected preprocessing methods for all the artificial datasets. **But the genetic search regularly finds sequences with the lowest number of preprocessing methods**.
- in the Chapter 8 I have investigated the parameter values optimisation methods. Again at first I investigated if at least some parameters of preprocessing methods have influence on the accuracy of the trained classifier. The results show that at least in some cases the parameters and their values have influence on accuracy of the model. Later I have tested performance – the best achieved accuracy of the model and a number of iterations – of the optimisation

methods on randomly selected sequences. The Differential Evolution optimisation is clearly the best optimisation method.

- the Chapter 9 is the last chapter to work with the artificial datasets. It tests the performance of the whole IPT – the search for the best sequences and the parameter optimisation. The result is that the best sequences are found by the **genetic search** for the sequence search method and the **one random change** for the parameter optimisation. The *one random change* is a standard mutation in the genetic algoritm and randomly changes values of the parameters.
- the Chapter 10 finally tests IPT with *genetic search* with the *one random change* and their performance on the 13 publicly available real world datasets. The result shows that the best found sequences are able to improve the accuracy of classifiers by **about 5% to 10% on validation parts** of the datasets. The base for improvement is accuracy of the model achieved on the original, not preprocessed, validation part of a dataset. This is quite interesting especially when you take into account that it is achieved only by transformations of a dataset.
- the Chapter 11 looks for a way how to decrease a time needed to finish IPT. The typical search for the best sequence lasts several hours so it would be great to find a way to reduce time needed to finish. One of the ways is the supposition that similar attributes of datasets should be preprocessed using similar preprocessing methods. I have decided to find similarity of attributes using meta-data. I have created the meta database using results and the best sequences found in the Chapter 10. When a new dataset arrives, the meta database is used to generate initial population (sequences) of IPT. In this way I can use the past information. I have used 4 more real world dataset to test the performance of the meta database. **The results show that with the generated initial population IPT in 10 steps can find sequences with comparable accuracy to IPT with random initial population in 50 steps. I have also say that IPT in 10 steps with random initial population achieves worse results that IPT in 10 steps with generated initial population.**

12.3 The Contributions

The presented IPT shows great potential for real-world commercial applications. The data preprocessing is the most time-consuming phase of the knowledge discovery process. It is estimated that it takes about 80% of the whole data mining process. A significant part of the data preprocessing – the data transformation – can be automated.

There are two commercially available approaches, but they are based on simple rules and do very simple transformations. IPT is able to find preprocessing methods for much more complex datasets. IPT allows the data miners to concentrate on the other stages of the knowledge discovery process where the human expert's insight and expertise is needed.

From the scientific point of view the thesis combines the inductive modelling and the optimisation approach and applies them in the data preprocessing field. IPT also has potential to find new "best-practises" for data preprocessing in different fields.

12.4 The Future Work

I see several directions to continue with the research and development of IPT.

- To extend the use of IPT to regression problems.
- to extend the use of IPT to feature extraction from time signals. The classification of the time signals are yet more complicated by the fact that a data mining expert has to extract features from the signal. It is not clear which features of the signal to use. In many fields there are gold-standard features, but for a new problems there are no guidelines how to describe the signal and its properties properly. IPT could find the best combinations of the feature extraction methods.
- to preprocess more real world datasets using IPT to obtain better and wider meta database that can be used to generate rules if to preprocess attributes and even which preprocessing methods to use.
- to polish the IPT implementation and to incorporate it into a data mining software like Rapidminer. This would test IPT in much larger scale and for wide range of problems. The results would be great for improving precision and accuracy IPT and to collect the real world datasets for the meta database.

13 Bibliography

[1] Wai-Ho Au, Keith C. C. Chan, and Xin Yao. A novel evolutionary data mining algorithm with applications to churn prediction. *Evolutionary Computation, IEEE Transactions on*, 7(6):532–545, 2003.

[2] Chih-Ping Wei, I Chiu, et al. Turning telecommunications call details to churn prediction: a data mining approach. *Expert systems with applications*, 23(2):103–112, 2002.

[3] Robert Nisbet, John Fletcher Elder, and Gary Miner. *Handbook of statistical analysis and data mining applications*. Academic Press, 2009.

[4] Paul Jermyn, Maurice Dixon, and Brian J Read. Preparing clean views of data for data mining. *ERCIM Work. on Database Res*, pages 1–15, 1999.

[5] N.R. Pal and L. C. Jain, editors. *Advanced techniques in knowledge discovery and data mining*. Springer, 2005.

[6] A. Stolcke, S. Kajarekar, and L. Ferrer. Nonparametric feature normalization for svm-based speaker verification. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 1577 –1580, 31 2008-april 4 2008.

[7] Jason Farquhar, N Jeremy Hill, et al. Interactions between pre-processing and classification methods for event-related-potential classification: best-practice guidelines for brain-computer interfacing. *Neuroinformatics*, 2013.

[8] Petr Buryan and Godfrey C Onwubolu. Design of enhanced mia-gmdh learning networks. *International Journal of Systems Science*, 42(4):673–693, 2011.

[9] Rodrigo C Barros, Márcio P Basgalupp, André CPLF de Carvalho, and Alex A Freitas. Towards the automatic design of decision tree induction algorithms. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 567–574. ACM, 2011.

[10] Rodrigo Coelho Barros, Márcio Porto Basgalupp, ACPLF de Carvalho, and Alex A Freitas. A survey of evolutionary algorithms for decision-tree induction. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(3):291–312, 2012.

[11] A. Bernstein and F. Provost. An intelligent assistant for the knowledge discovery process. *Proceedings of the IJCAI-01 Workshop on Wrappers for Performance Enhancement in KDD*, 2001.

[12] A. Bernstein, F. Provost, and S. Hill. Towards intelligent assistance for a data mining procesr a data mining proces. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):503–518, 2005.

[13] Bernstein A., Provost F., and Hill S. Intelligent assistance for the data mining process: An ontology-based approach. *Information Systems Working Papers Series*, 2002.

[14] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta. *Metalearning, Applications to Data Mining.* Cognitive Technologies. Springer Berlin Heidelberg, 2009.

[15] T. Euler, K. Morik, and M. Scholz. Miningmart: Sharing successful kdd processes. *LLWA 2003 – Tagungsband der GI-Workshop-Woche Lehren – Lernen – Wissen – Adaptivitat*, 2003.

[16] K. Morik and M. Scholz. The miningmart approach to knowledge discovery in databases. In N. Zhong and J. Liu, editors, *Intelligent Technologies for Information Analysis.* Springer, 2004.

[17] T. Euler. Publishing operational models of data mining case studies. *Proceedings of the ICDM Workshop on Data Mining Case Studies*, 2005.

[18] T. Euler and M. Scholz. Using ontologies in a kdd workbench. *Proceedings of the ECML/PKDD Workshop on Knowledge Discovery and Ontologies*, 2004.

[19] Miningmart internet case base, available http://mmart.cs.uni-dortmund.de/end-user/casebase.html.

[20] Carole A Goble, Jiten Bhagat, Sergejs Aleksejevs, Don Cruickshank, Danius Michaelides, David Newman, Mark Borkum, Sean Bechhofer, Marco Roos, Peter Li, et al. myexperiment: a repository and social network for the sharing of bioinformatics workflows. *Nucleic acids research*, 38(suppl 2):W677–W682, 2010.

[21] D. De Roure, C. Goble, and R. Stevens. The design and realisation of the myexperiment virtual research environment for social sharing of workflows. *Future Generation Computer Systems*, 25:561–567, 2009.

[22] J Kietz, Floarea Serban, Abraham Bernstein, and Simon Fischer. Towards cooperative planning of data mining workflows. In *Proceedings of the Third Generation Data Mining Workshop at the 2009 European Conference on Machine Learning (ECML 2009)*, pages 1–12, 2009.

[23] Jörg-Uwe Kietz, Floarea Serban, Abraham Bernstein, and Simon Fischer. Data mining workflow templates for intelligent discovery assistance and auto-experimentation. *Third-Generation Data Mining: Towards Service-Oriented Knowledge Discovery SoKD*, 10, 2010.

[24] Rüdiger Wirth, Colin Shearer, Udo Grimmer, Thomas Reinartz, Jörg Schlösser, Christoph Breitner, Robert Engels, and Guido Lindner. Towards process-oriented tool support for knowledge discovery in databases. *Principles of Data Mining and Knowledge Discovery*, pages 243–253, 1997.

[25] IBM, 233 South Wacker Drive, 11th Floor, Chicago, IL 60606-6412. *IBM SPSS Modeler 14.2 Algorithms Guide*, 2011.

[26] Kathy L. Taylor. *Oracle Data Mining Concepts.* Oracle, Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

[27] Usama Fayyad, Gregory Piatetsky-shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996.

[28] Krzysztof J Cios, Witold Pedrycz, Roman W Swiniarski, and Lukasz Andrzej Kurgan. *Data mining: a knowledge discovery approach.* Springer Publishing Company, Incorporated, 2010.

[29] Usama Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, et al. Knowledge discovery and data mining: Towards a unifying framework. *Knowledge Discovery and Data Mining*, pages 82–88, 1996.

[30] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rudiger Wirth. Crisp-dm 1.0 step-by-step data mining guide. 2000.

[31] N Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:341–378, 2002.

[32] Tantan Liu, Fan Wang, and Gagan Agrawal. Stratified sampling for data mining on the deep web. *Frontiers of Computer Science*, 6(2):179–196, 2012.

[33] Marıa Teresa Lozano Albalate, J Salvador Sánchez Garreta, and Filiberto Pla Banón. *Data reduction techniques in classification processes.* PhD thesis, Ph. D. dissertation, Universitat Jaume I, 2007.

[34] Dorian Pyle. *Data preparation for data mining*, volume 1. Morgan Kaufmann, 1999.

[35] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[36] Edwin M Knorr, Raymond T Ng, and Vladimir Tucakov. Distance-based outliers: algorithms and applications. *The VLDB Journal*, 8(3):237–253, 2000.

[37] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM Sigmod Record*, volume 29, pages 93–104. ACM, 2000.

[38] R. Duda, Peter Hart, and David Stork. *Pattern Classification.* Wiley-Interscience, 2001.

[39] John Ross Quinlan. *C4. 5: programs for machine learning*, volume 1. Morgan kaufmann, 1993.

[40] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.

[41] Marc Sumner, Eibe Frank, and Mark Hall. Speeding up logistic model tree induction. In AlípioMário Jorge, Luís Torgo, Pavel Brazdil, Rui Camacho, and João Gama, editors, *Knowledge Discovery in Databases: PKDD 2005*, volume 3721 of *Lecture Notes in Computer Science*, pages 675–683. Springer Berlin Heidelberg, 2005.

[42] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[43] Niels Landwehr, Mark Hall, and Eibe Frank. Logistic model trees. In Nada Lavrač, Dragan Gamberger, Hendrik Blockeel, and Ljupčo Todorovski, editors, *Machine Learning: ECML 2003*, volume 2837 of *Lecture Notes in Computer Science*, pages 241–252. Springer Berlin Heidelberg, 2003.

[44] Kenneth O. Stanley, Risto Miikkulainen, et al. Competitive coevolution through evolutionary complexification. *J. Artif. Intell. Res. (JAIR)*, 21:63–100, 2004.

[45] Keigo Watanabe and M. A. Hashem. *Evolutionary Computations.* Studies in Fuzzyness and Soft Computing. Springer, 2004.

[46] Kay H Brodersen, Cheng Soon Ong, Klaas E Stephan, and Joachim M Buhmann. The balanced accuracy and its posterior distribution. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 3121–3124. IEEE, 2010.

[47] Yaochu Jin and Jürgen Branke. Evolutionary optimization in uncertain environments-a survey. *Evolutionary Computation, IEEE Transactions on*, 9(3):303–317, 2005.

[48] Peter Bickel and Kjell Doksum. *Mathematical Statistics, Basic Ideas and Selected Topics*, volume 1. Prentice Hall, 2001.

[49] Robert McGill, John Tukey, and Wayne Larsen. Variations of box plots. *The American Statstician*, (1):12–16, 1978.

[50] Nikolay Y Nikolaev and Hitoshi Iba. Regularization approach to inductive genetic programming. *Evolutionary Computation, IEEE Transactions on*, 5(4):359–375, 2001.

[51] J. Nocedal and S. Wright. *Numberical Optimisation.* Series in Operational Research. Springer, 2006.

[52] James C. Spall. *Introduction to Stochastic Search and Optimisation.* Series in Discrete Mathematics and Optimisation. Wiley-Interscience, 2003.

[53] Z. B. Zabinsky. *Wiley Encyclopedia of Operations Research and Management Science*, chapter Random Search Algorithms. Wiley & Sons, 2009.

[54] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[55] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* Springer, 2003.

[56] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *Urbana*, 51:61801–2996, 1991.

[57] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[58] Jouni Lampinen and Ivan Zelinka. Mixed integer-discrete-continuous optimization by differential evolution. In *Proceedings of the 5th International Conference on Soft Computing*, pages 71–76. Citeseer, 1999.

[59] Jakob Vesterstrom and Rene Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1980–1987. IEEE, 2004.

[60] Donald Michie, David J Spiegelhalter, Charles C Taylor, and John Campbell. *Machine learning, neural and statistical classification.* Ellis Horwood London, 1994.

[61] Kate A Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):6, 2008.

[62] Floarea Serban, Joaquin Vanschoren, Jorg-Uwe Kietz, and Abraham Bernstein. A survey of intelligent assistants for data analysis. *ACM Computing Surveys*, 2012.

[63] Shunsuke Ihara. *Information theory for continuous systems*, volume 2. World Scientific Publishing Company Incorporated, 1993.

[64] Claude Elwood Shannon, Warren Weaver, Richard E Blahut, and Bruce Hajek. *The mathematical theory of communication*, volume 117. University of Illinois press Urbana, 1949.

[65] Holger H Hoos and Thomas Stützle. *Stochastic local search: Foundations & applications.* Morgan Kaufmann, 2004.

[66] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[67] S. Moro, R. Laureano, and P. Cortez. Using data mining for bank direct marketing: An application of the crisp-dm methodology. In *Proceedings of the European Simulation and Modelling Conference - ESM'2011*, pages 117–121, 2011.

[68] Bank marketing data set, available at http://archive.ics.uci.edu/ml/datasets/bank+marketing.

[69] O. L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News*, 23(5):1, 18, 1990.

[70] William H. Wolberg and O.L. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences*, 87:9193–9196, 1990.

[71] Breast cancer wisconsin (original) data set, available at http://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+

[72] Diogo Ayres-de Campos, Joao Bernardes, Antonio Garrido, Joaquim Marques-de Sa, and Luis Pereira-Leite. Sisporto 2.0: a program for automated analysis of cardiotocograms. *Journal of Maternal-Fetal and Neonatal Medicine*, 9(5):311–318, 2000.

[73] Ctg dataset, available at http://archive.ics.uci.edu/ml/datasets/cardiotocography.

[74] Paul Horton and Kenta Nakai. A probablistic classification system for predicting the cellular localization sites of proteins. *Intelligent Systems in Molecular Biology*, pages 106–115, 1996.

[75] Ecoli dataset, available at http://archive.ics.uci.edu/ml/datasets/ecoli.

[76] Ian W. Evett and E. J. Spiehler. Rule induction in forensic science. In *KBS in Goverment*, pages 107–118. Online Publications, 1987.

[77] Glass identification dataset, available at http://archive.ics.uci.edu/ml/datasets/glass+identification, 2012.

[78] V. G. Sigillito, S. P. Wing, L. V. Hutton, and K. B. Baker. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10:262–266, 1989.

[79] Ionosphere dataset, available at http://archive.ics.uci.edu/ml/datasets/ionosphere, 2012.

[80] M. A. Little, P. E. McSharry, S. J. Roberts, D. A. E. Costello, and I. M. Moroz. Exploiting nonlinear recurrence and fractal scaling properties for voice disorder detection. *BioMedical Engineering OnLine*, 2007.

[81] Parkinsons dataset, available at http://archive.ics.uci.edu/ml/datasets/parkinsons.

[82] Statlog (landsat satellite) dataset, available at http://archive.ics.uci.edu/ml/datasets/statlog+

[83] Statlog (image segmentation) dataset, available at http://archive.ics.uci.edu/ml/datasets/statlog+

[84] Spambase dataset, available at http://archive.ics.uci.edu/ml/datasets/spambase.

[85] M Buscema, S Terzi, and W Tastle. A new meta-classifier. In *Proceedings of NAFIPS 2010*, 2010.

[86] M Buscema. Metanet: The theory of independent judges. In *Substance Use & Misuse*, pages 439 – 461, 1998.

[87] Steel faults dataset, available at http://archive.ics.uci.edu/ml/datasets/steel+plates+faults.

[88] S. Aeberhard, D. Coomans, and O. de Vel. Comparison of classifiers in high dimensional settings. Technical Report 92-02, Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland, 1992.

[89] Wine dataset, available at http://archive.ics.uci.edu/ml/datasets/wine, 2012.

[90] Miroslav Cepek, Ales Pilny, Radka Kubelkova, Jana Veleminska, and Miroslav Snorek. Modelling of age from the teeth development. In *Proceeding of the 7th EUROSIM Congress on Modelling and Simulation*, 2010.

[91] Coenraad Moorrees, Elizabeth Fanning, and Edwawd Hunt. Age variation of formation stages for ten permanent teeth. *Journal of Dental Research*, 42(1490-1502), 1963.

[92] J.R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.

[93] Credit approval dataset, available at http://archive.ics.uci.edu/ml/datasets/credit+approval.

[94] R. Detrano, A. Janosi, W. Steinbrunn, M. Pfisterer, J. Schmid, S. Sandhu, K. Guppy, S. Lee, and V. Froelicher. International application of a new probability algorithm for the diagnosis of coronary artery disease. *American Journal of Cardiology*, 64:304–310, 1989.

[95] Heart disease dataset, available at http://archive.ics.uci.edu/ml/datasets/heart+disease.

[96] Bendi Venkata Ramana, M. S. Prasad-Babu, and N. B. Venkateswarlu. A critical comparative study of liver patients from usa and india: An exploratory analysis. *International Journal of Computer Science Issues*, 2012.

[97] Indian liver patient dataset, available at http://archive.ics.uci.edu/ml/datasets/ilpd+

[98] J.W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R.S. Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *In Proceedings of the Symposium on Computer Applications and Medical Care*, pages 261–265. IEEE Computer Society Press, 1988.

[99] Pima indian diabetes dataset, available at http://archive.ics.uci.edu/ml/datasets/pima+indians+diabetes.

[100] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[101] Ivan Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, (6):448–452, 1976.

[102] CH Chen and Adam Jóźwik. A sample set condensation algorithm for the class sensitive artificial neural network. *Pattern Recognition Letters*, 17(8):819–823, 1996.

[103] D Randall Wilson and Tony R Martinez. Reduction techniques for instance-based learning algorithms. *Machine learning*, 38(3):257–286, 2000.

[104] P.E. HART. The condensed nearest neighbour rule. *IEEE Transactions on Information Theory*, 1968.

[105] David W Aha, Dennis Kibler, and Marc K Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.

[106] Daeryong Lee, SeongJoon Baek, and Koengmo Sung. Modified k-means algorithm for vector quantizer design. *Signal Processing Letters, IEEE*, 4(1):2 –4, jan. 1997.

[107] G.W. Gates. The reduced nearest neighbour rule. *IEEE Transactions on Information Theory*, 1972.

[108] JS Sánchez. High training set size reduction by space partitioning and prototype abstraction. *Pattern Recognition*, 37(7):1561–1564, 2004.

[109] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *Systems, Man and Cybernetics, IEEE Transactions on*, (3):408–421, 1972.

[110] Kevin Gimpel and Noah A Smith. Softmax-margin crfs: Training log-linear models with cost functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 733–736. Association for Computational Linguistics, 2010.

14 Refereed publications of the author

[A.1] Čepek, M. and Šnorek, M. and Kordík, P. Missing Data Imputation and the Inductive Modelling. In *Proceedings of the 6th EUROSIM Congress on Modelling and Simulation*, ARGESIM, 2007, 978-3-901608-32-2

[A.2] Čepek, M. and Šnorek, M. Basic Data Reduction Techniques and Their Influence on GAME Modeling Method. In *Proceedings of UKSIM Tenth International Conference on Computer Modelling and Simulation*, IEEE Computer Society, pp 138–143, 2008, 0-7695-3114-8

[A.3] Čepek, M. and Kordík, P. and Pavlíček, M. and Šnorek, M. Automatic Method for Data Preprocessing for the GAME Inductive Modelling Method. In *Proceedings of the 2nd International Conference on Inductive Modelling*, Ukr. INTEI, pp 207–212, 2008, 978-966-02-4889-2

[A.4] Čepek, M. and Kordík, P. and Šnorek, M. Testing of Inductive Preprocessing Algorithm. In *Proceedings of the 3rd International Workshop on Inductive Modelling 2009*, Ukr. INTEI, pp 13–18, 2009

[A.5] Kordík, P. and Koutník, J. and Drchal, J. and Kovářík, O. and Čepek, M. and Šnorek, M. Meta-learning approach to neural network optimization. *Neural Networks*, 23:568–582, 2010.

[A.6] Čepek, M. and Kordík, P. and Šnorek, M. The Effect of Modelling Method to the Inductive Preprocessing Algorithm. In *Proceedings of 3rd International Conference on Inductive Modelling 2010*, Ukr. INTEI, pp 131–138, 2010

A Implemented Preprocessing Methods

In this appendix you can find a list of implemented preprocessing methods, their names and references to literature. Methods written in *italics* are global.

- Data Enrichment
  - *SMOTE Enrichement* – is implementation of standard enrichment algorithm [31].
- Data Reduction[1] (For details see [A.2]).
  - *All-KNN editing method* [101]
  - *Chen's condensing method* [102]
  - *DROP3 reduce method* [103]
  - *CNN - Hart's condensing method* [104]
  - *IB3 reduce method* [105]
  - *KMeans data replacer* – runs standard K-Means clustering algorithm to do the vector quantisation [106] and then replaces instances in dataset by the centroids.
  - *RNN condensing method* [107]
  - *RSP3 condensing method* [108]
  - *Wilson's editing method* [109]
  - *Random data reducer* – does random and stratified sampling.
- Discretisation
  - Adaptive binning – unequal bin sizes, equal number of instances in each bin [34].
  - Equal size binning – equal bin sizes, unequal number of instances in each bin [34].
- Non Linear Transformation
  - N-th Power Calculator – according to parameter N calculates the power or power root. If the N positive integer, the method calculates N^{th} power and if N is negative integer, the method calculates N^{th} power root.
  - Square Root Calculator – calculates the square power root of a value.
  - Exponential Calculator – calculates exponential for a value (e^{value}).
  - *Principal Component Analysis* – standard principal component analysis [35]. The original dataset is replaced by the transformed dataset.
- Missing Data Imputation (for more details see the [A.1].)
  - Another Instance Value Data Imputer – replaces missing value by a value from the nearest instance in the dataset.
  - Constant Missing Value Imputer – replaces missing value by a predefined constant.

[1]All the data reduction methods work with training set only.

  - Median Missing Value Imputer – replaces missing values in attribute by a mean of no-missing values in the attribute.
  - Nearest Neighbour Missing Value Imputer – finds K-nearest instances, according to non-missing values, and replaces them by the mean value from values in attribute in nearest instances.
  - *Missing instances remover* – removes instances with missing values in attribute. Applied on training data only.

- Normalisation
  - Cut off values – uses thresholding
  - Linear normalizer – linearly normalises the values into specified range using following formula: $y = \frac{x - min(\chi)}{max(\chi) - min(\chi)}$, where x is actual values and χ represents the attribute.
  - Mean value normalizer – shifts the dataset to have 0 mean.
  - SoftMax normalizer – normalises values to given range using softmax function [110]
  - Z-score normalizer – transforms attribute to have mean equal to 0 and standard deviation equal to 1.

- Outlier Detection
  - *Local Outlier Factor* – a standard LOF algorithm implementation [37].
  - *Basic Algorithm* [36]
  - *Cell Based Algorithm* [36]

B Parameter Search Space Examples

This appendix supplements the section 8.1 and contains more examples of the parameters search spaces of sequences of the preprocessing methods.

All example search spaces given here are generated for the sequences of preprocessing methods with mainly two parameters. This limits the set of examples and sequences that can be presented here, but on the other hand, there is no hidden parameter to influence the search space visualisation.

For each parameter search space I will present example of 1 individual search space and the average of 10 search spaces. The reason is following – the fitness value is an accuracy of the model learned from the preprocessed data. This accuracy is not always constant – for the training and testing of model the different instances are use, the model is initialised in different way, and so on. Therefore the averaged search space is useful to give better idea how the search space actually look like.

To give idea about global minimum and global maximum I have written them for each figure. Please remember, that the fitness value can be zero. This does not mean that the underlying model outputs exactly opposite prediction with the 100% accuracy. This means that the underlying model can not be learned from the data preprocessed by given sequence of preprocessing methods with give parameters (e.g. all instances were removed from the training dataset).

B.1 Missing Data dataset

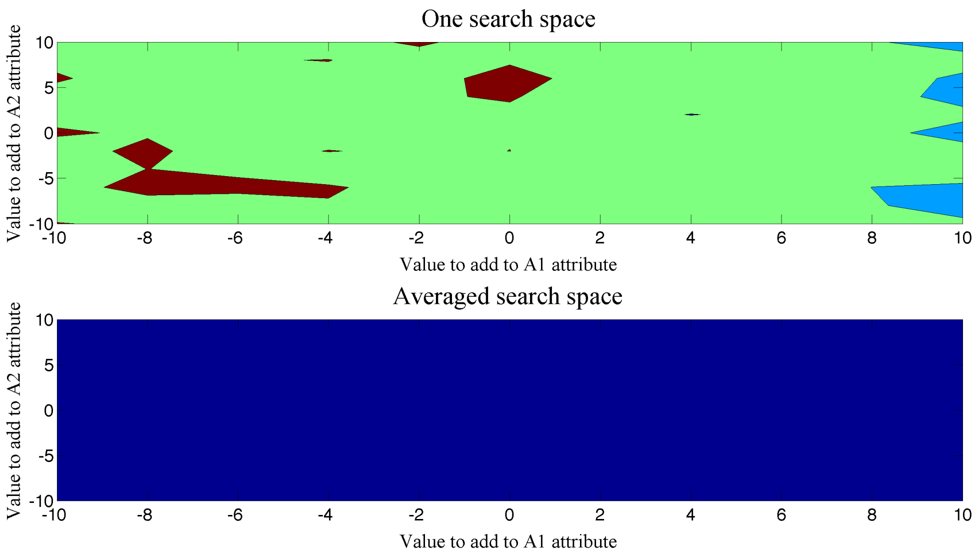


Figure B.1: Search space example, Missing Data dataset

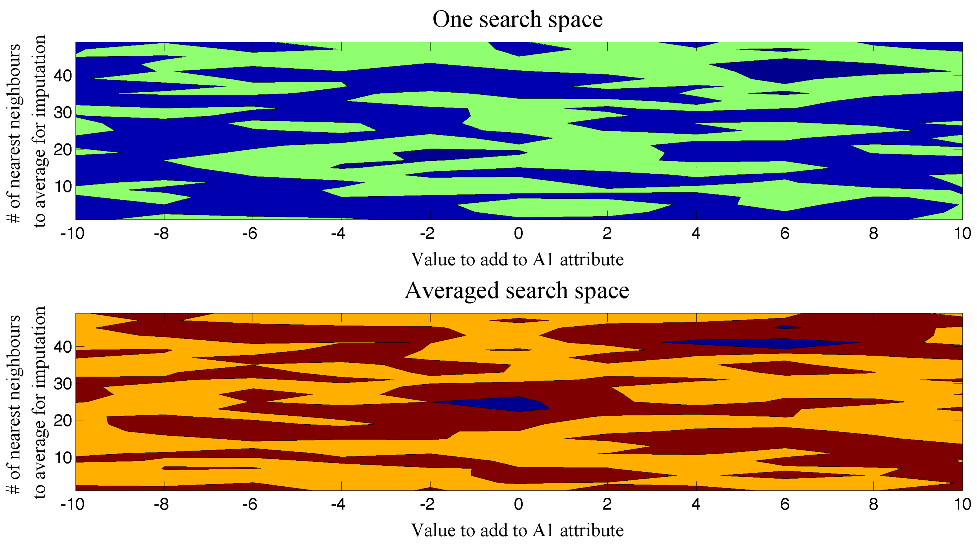


Figure B.2: Search space example, Missing Data dataset

Figure B.3: Search space example, Missing Data dataset

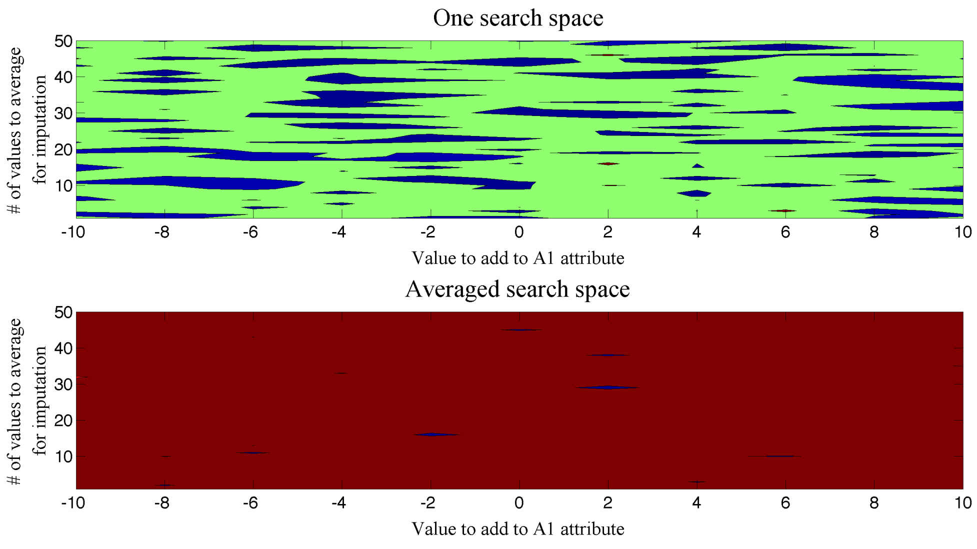


Figure B.4: Search space example, Missing Data dataset

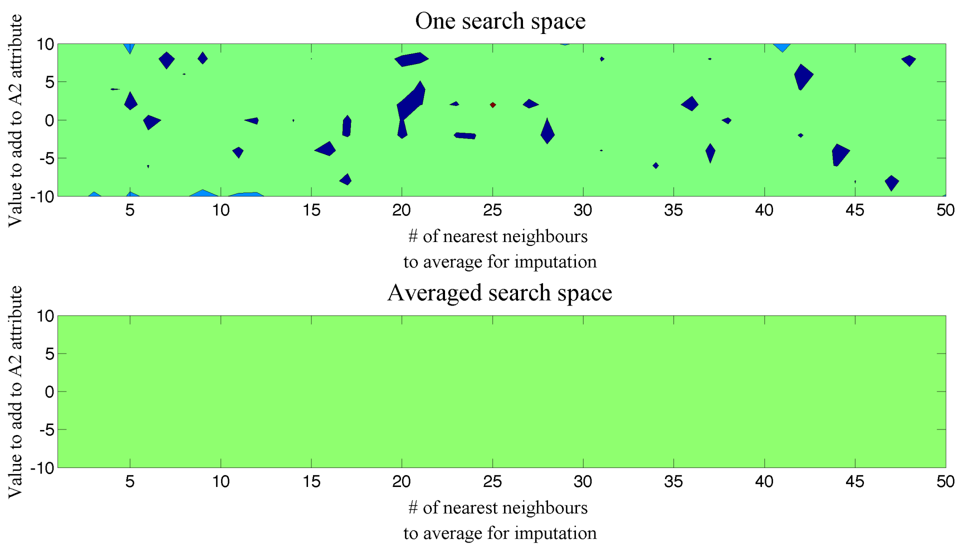


Figure B.5: Search space example, Missing Data dataset

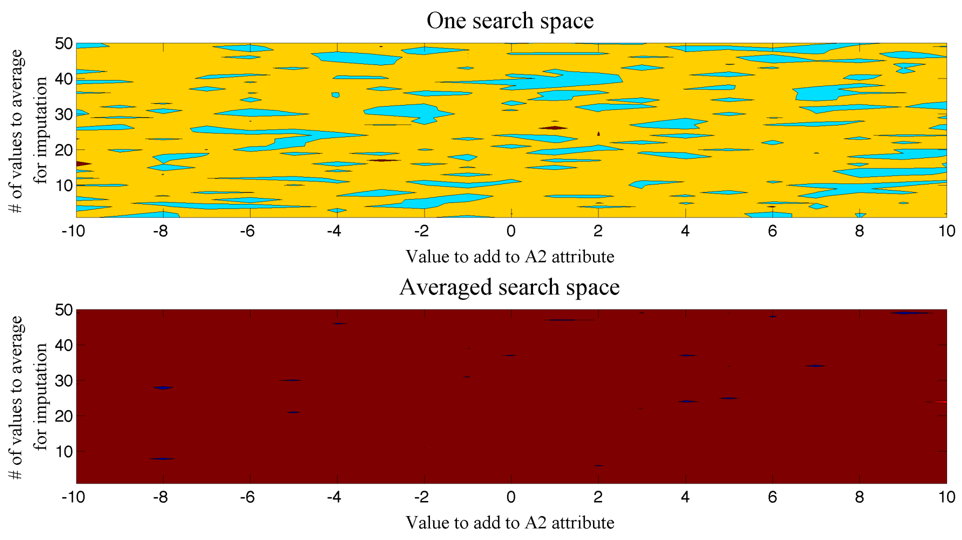


Figure B.6: Search space example, Missing Data dataset

B.2 Imbalanced dataset

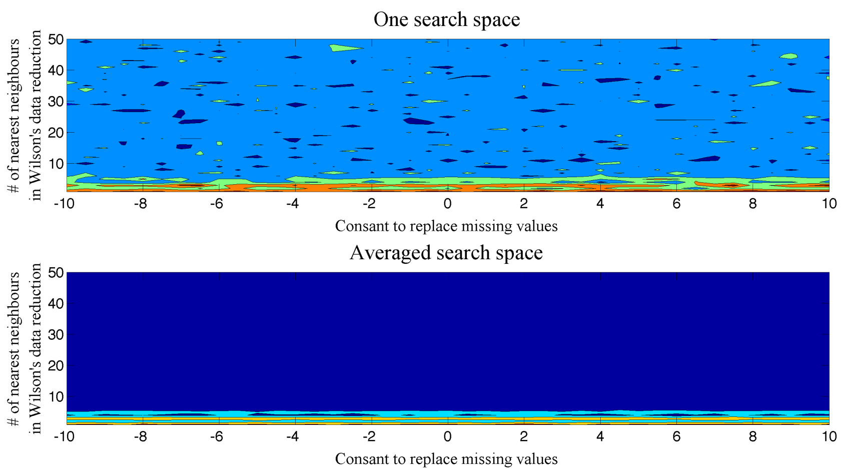


Figure B.7: Search space example, Imbalanced dataset

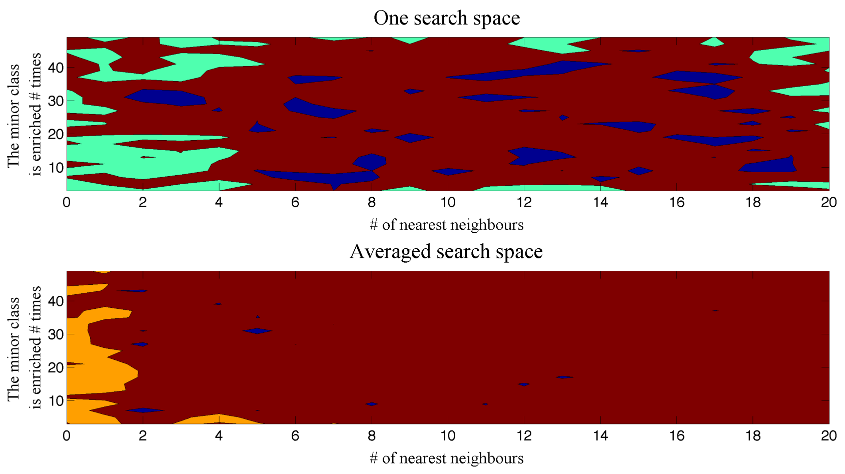


Figure B.8: Search space example, Imbalanced dataset

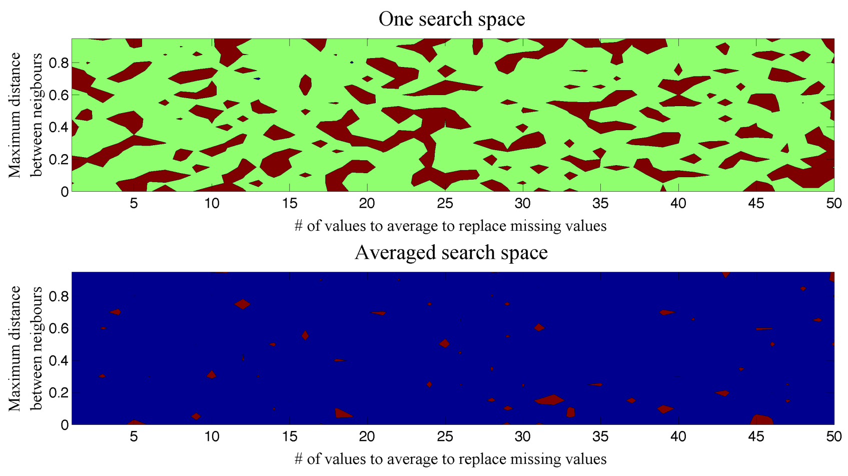


Figure B.9: Search space example, Imbalanced dataset

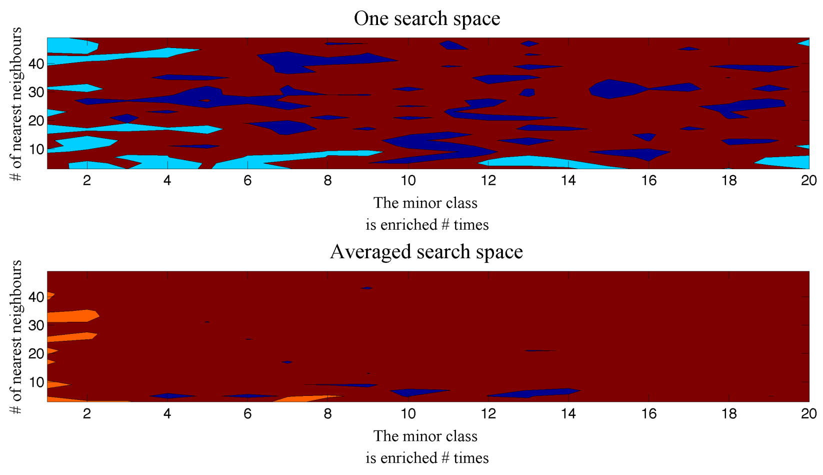


Figure B.10: Search space example, Imbalanced dataset

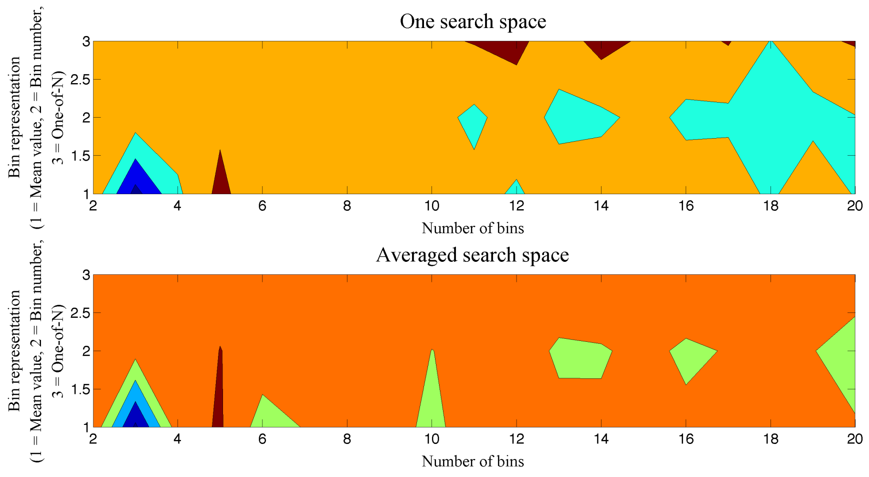


Figure B.11: Search space example, Imbalanced dataset

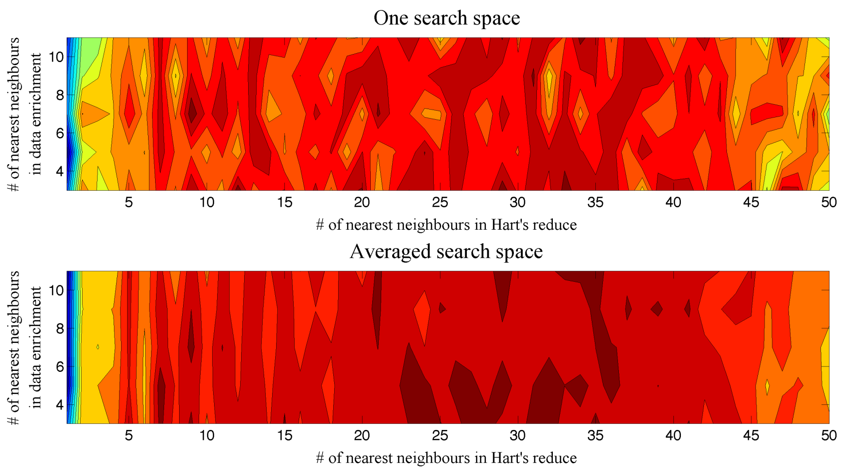


Figure B.12: Search space example, Imbalanced dataset

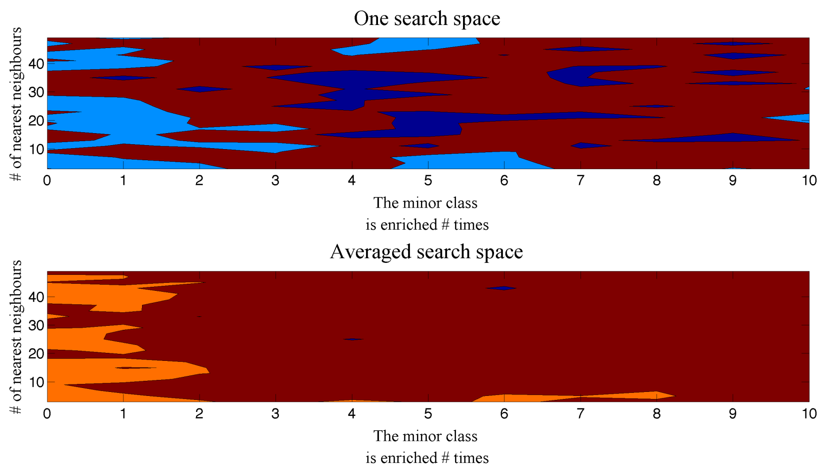


Figure B.13: Search space example, Imbalanced dataset

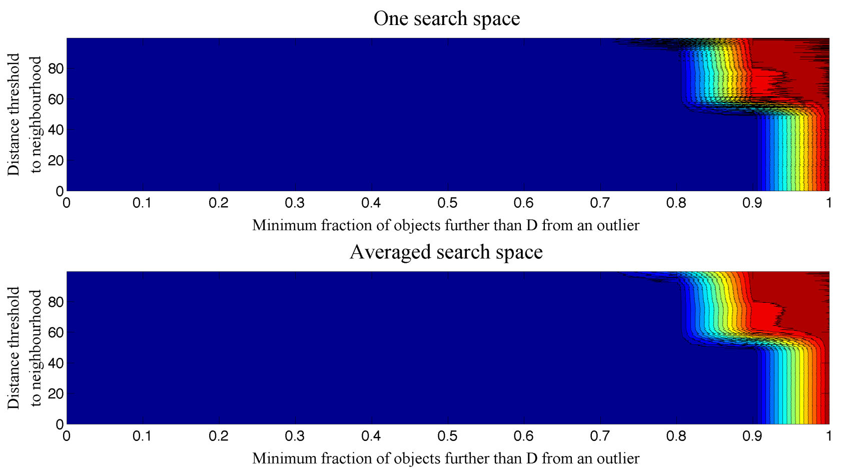


Figure B.14: Search space example, Imbalanced dataset

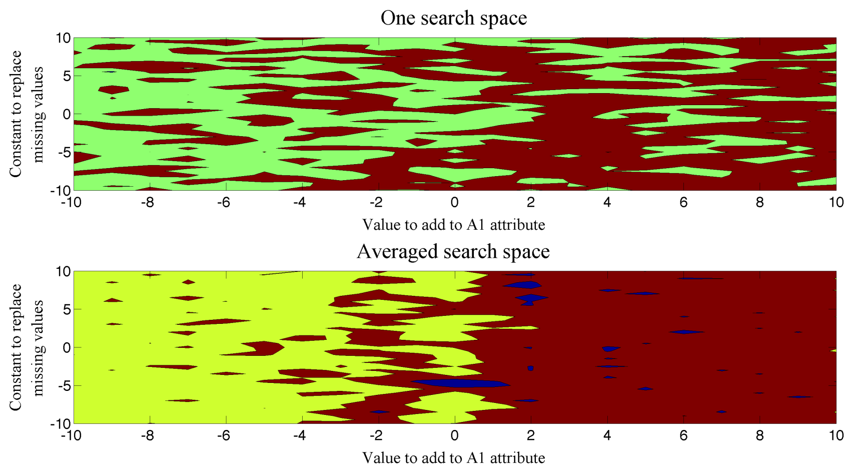


Figure B.15: Search space example, Imbalanced dataset

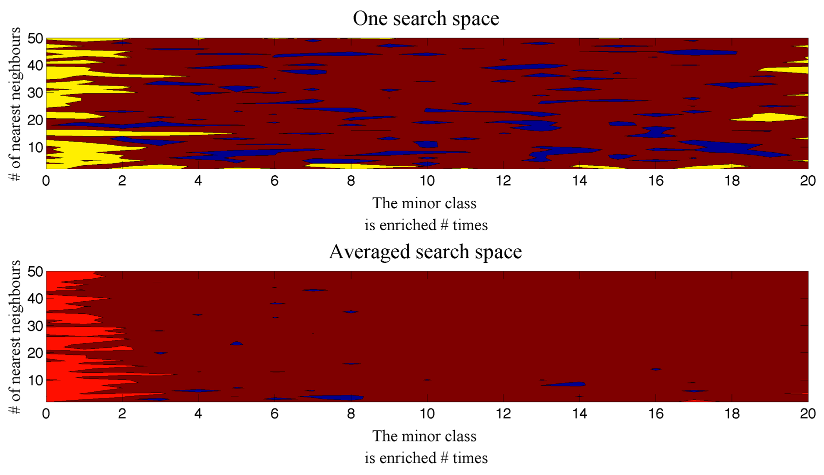


Figure B.16: Search space example, Imbalanced dataset

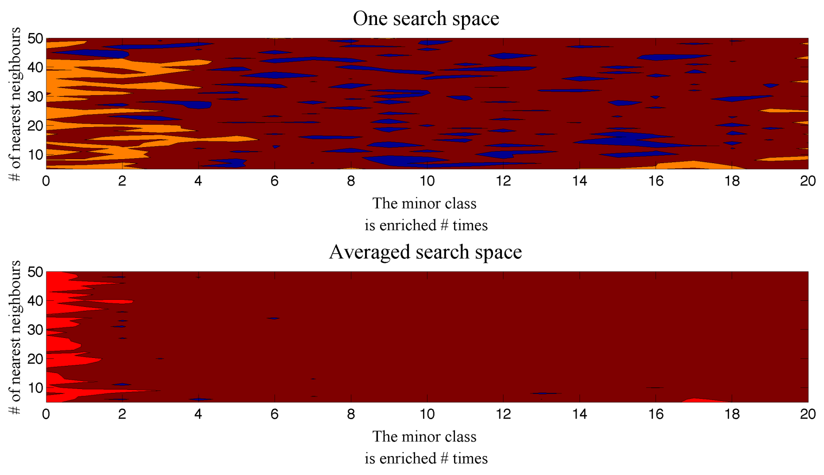


Figure B.17: Search space example, Imbalanced dataset

B.3 Non Linear dataset

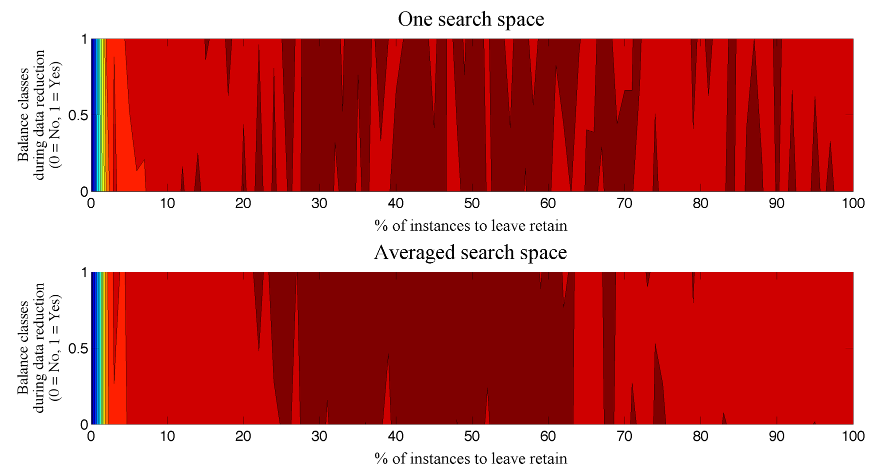


Figure B.18: Search space example, Non Linear dataset

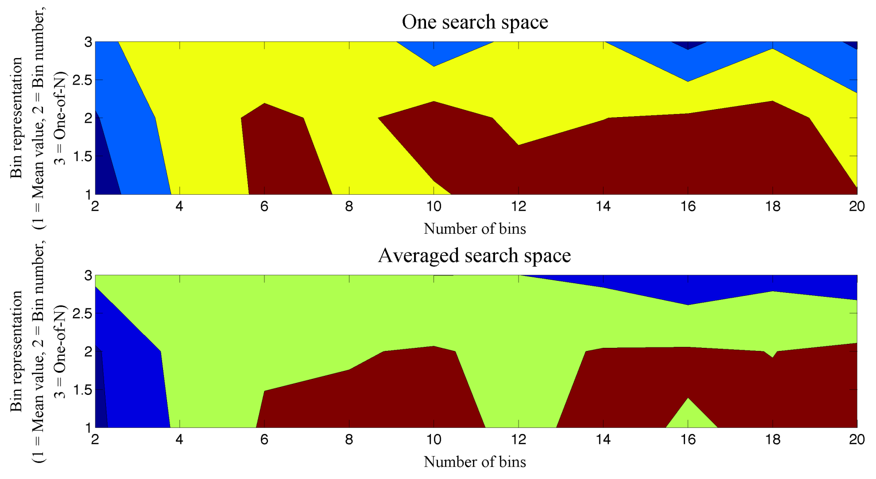


Figure B.19: Search space example, Non Linear dataset

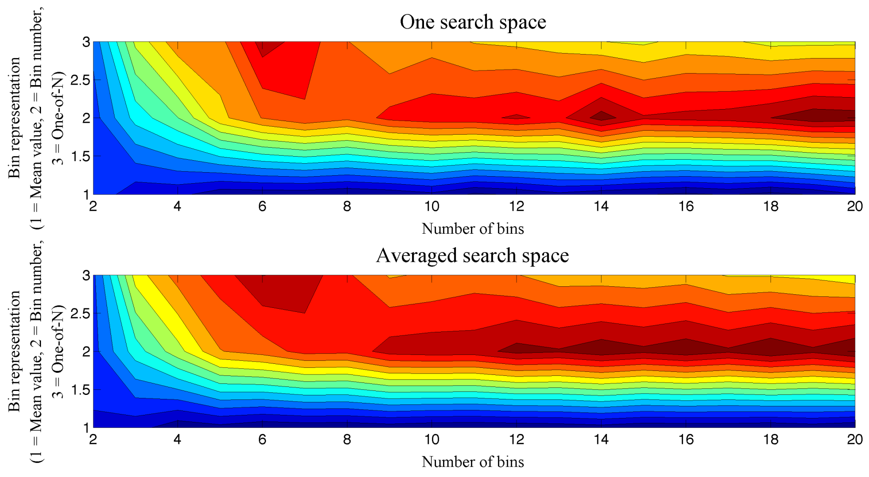


Figure B.20: Search space example, Non Linear dataset

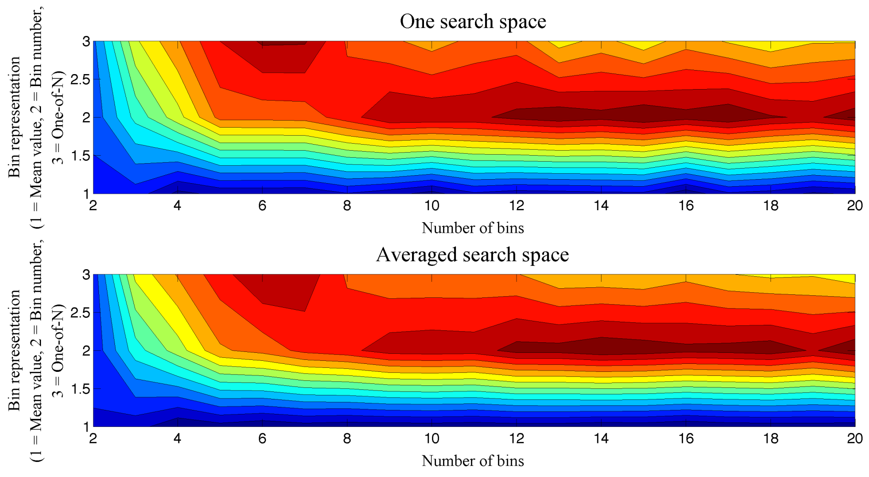


Figure B.21: Search space example, Non Linear dataset

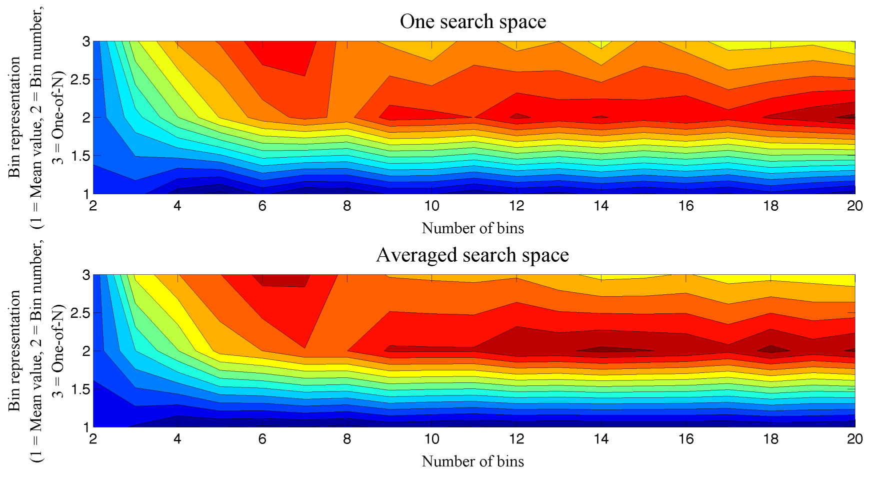


Figure B.22: Search space example, Non Linear dataset

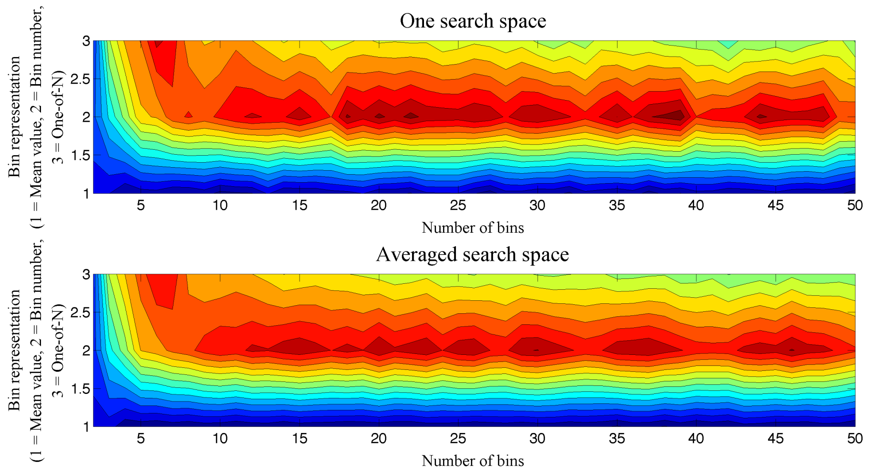


Figure B.23: Search space example, Non Linear dataset

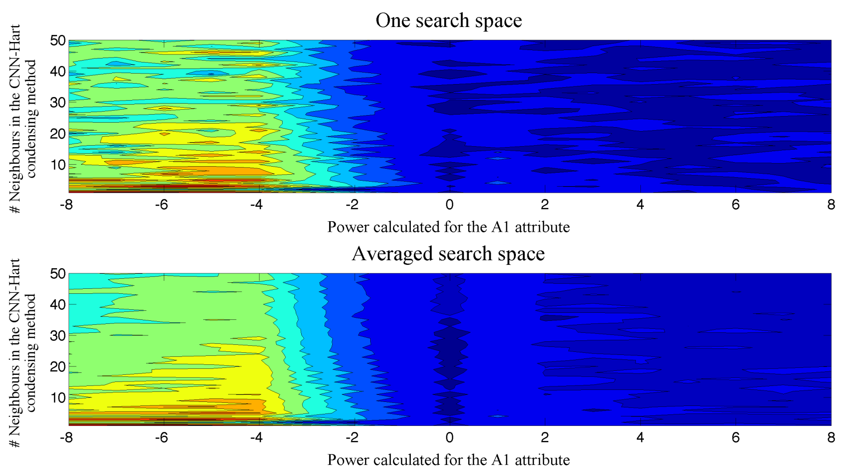


Figure B.24: Search space example, Non Linear dataset

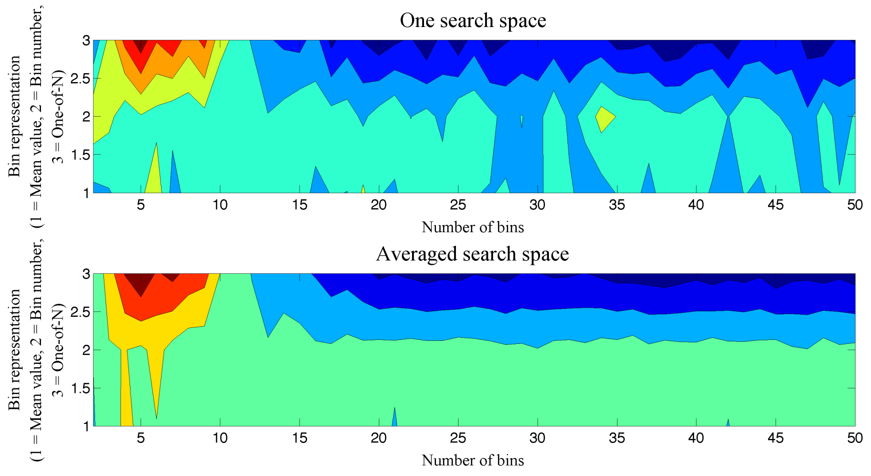


Figure B.25: Search space example, Non Linear dataset

B.4 Outlier dataset

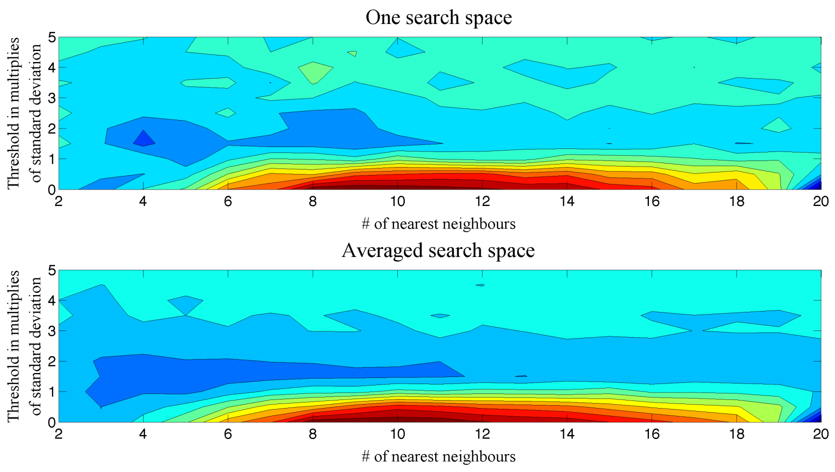


Figure B.26: Search space example, Outliers dataset

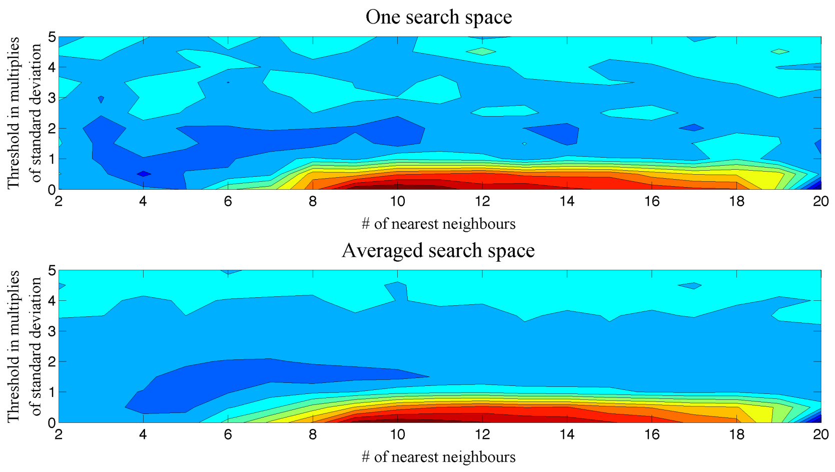


Figure B.27: Search space example, Outliers dataset

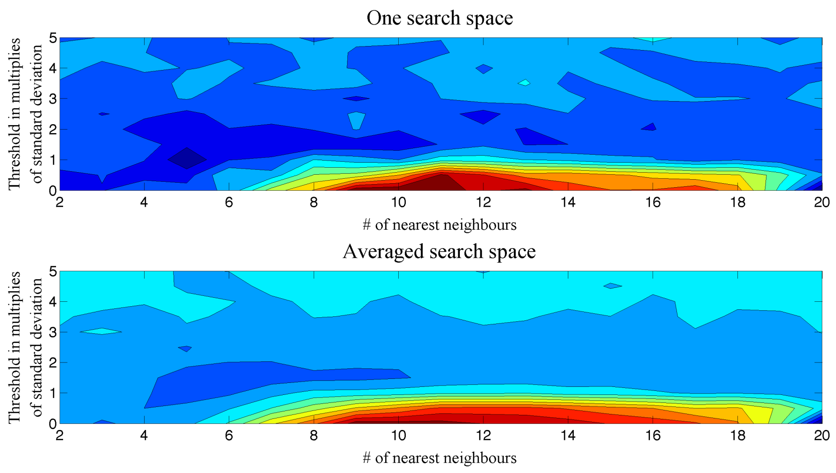


Figure B.28: Search space example, Outliers dataset

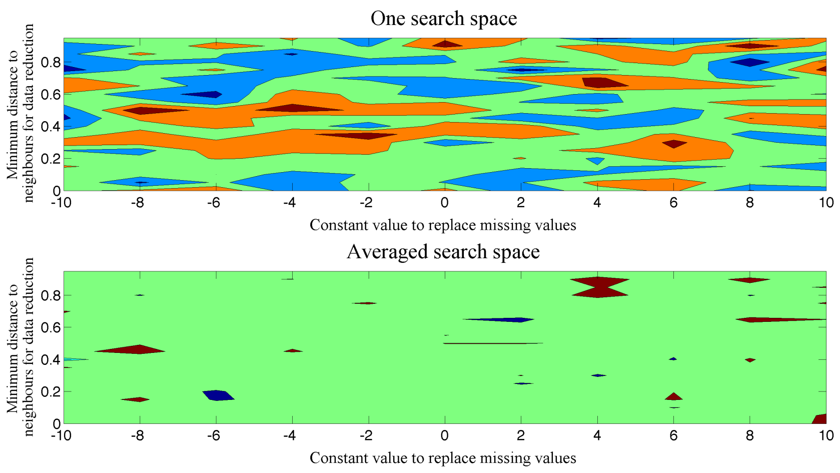


Figure B.29: Search space example, Outliers dataset

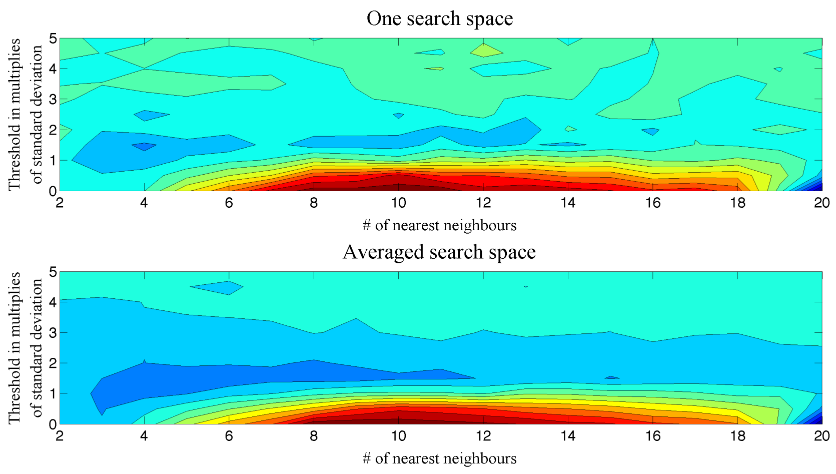


Figure B.30: Search space example, Outliers dataset

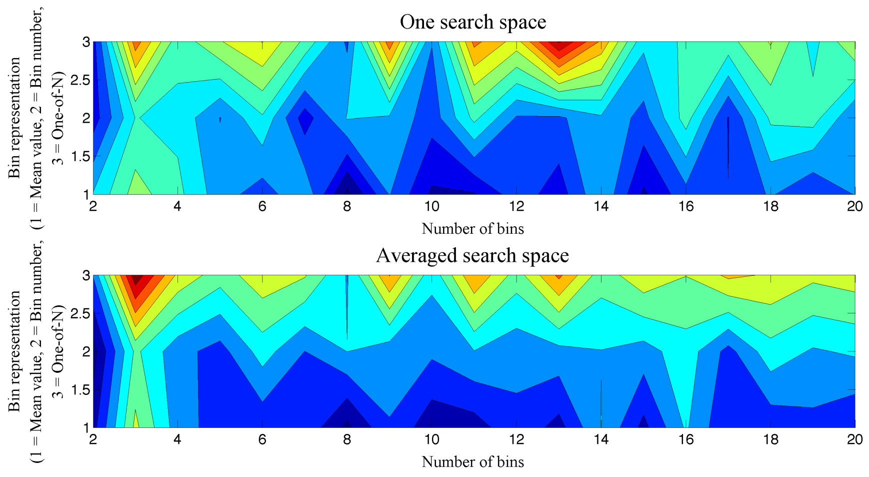


Figure B.31: Search space example, Outliers dataset

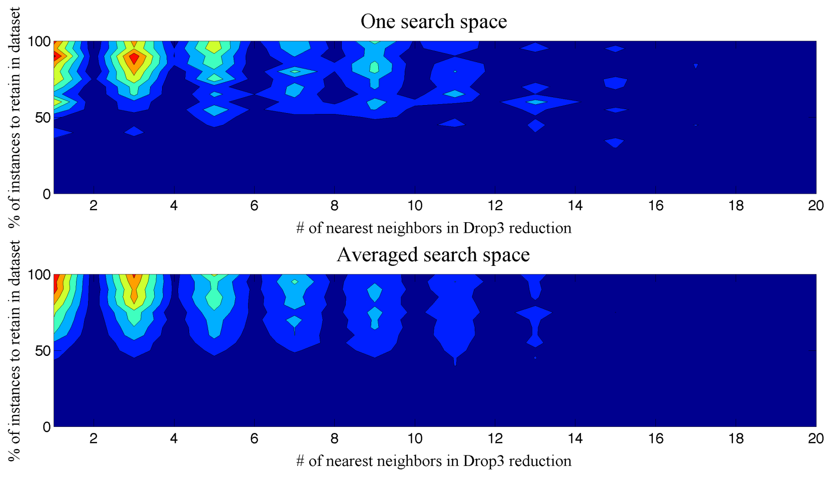


Figure B.32: Search space example, Outliers dataset

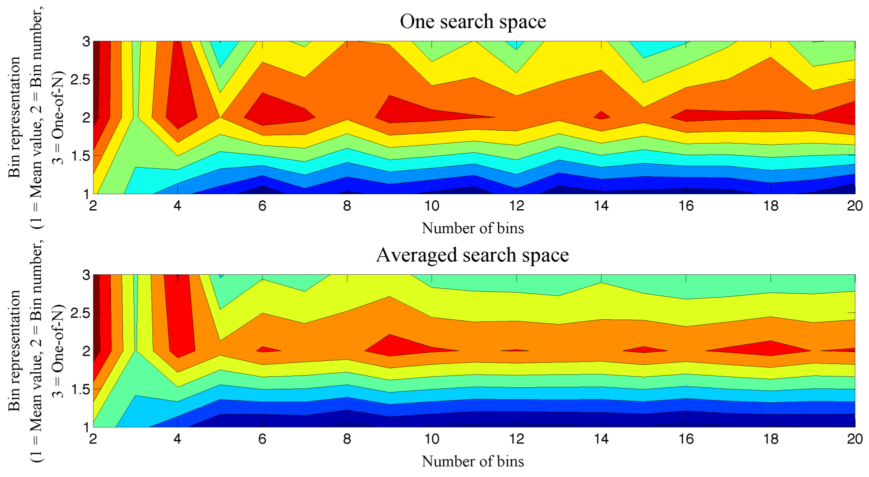


Figure B.33: Search space example, Outliers dataset

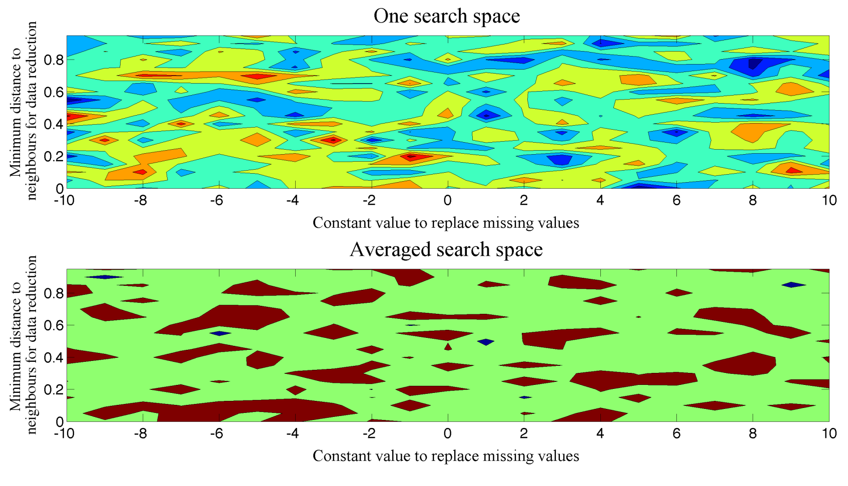


Figure B.34: Search space example, Outliers dataset

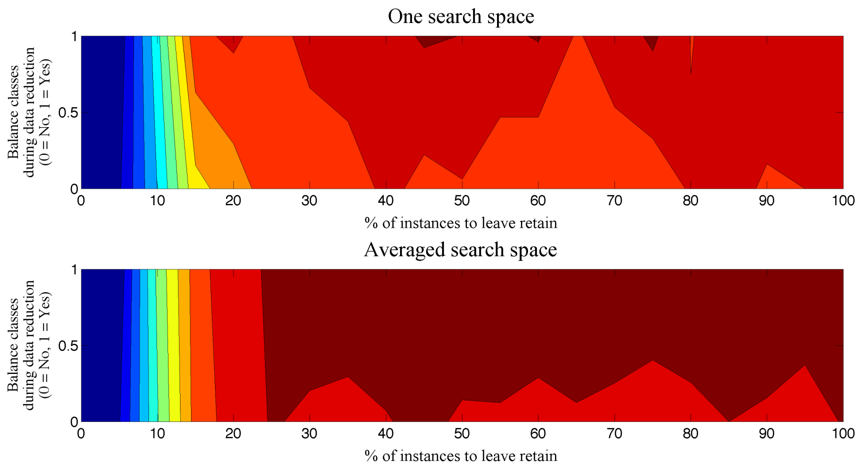


Figure B.35: Search space example, Outliers dataset

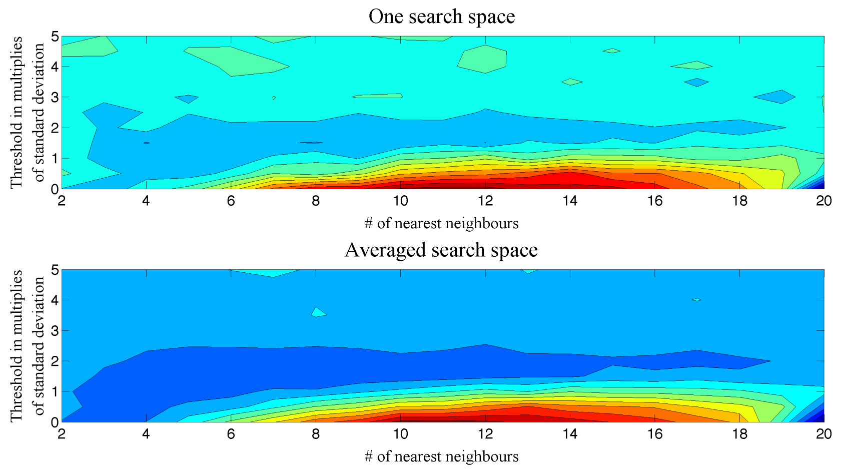


Figure B.36: Search space example, Outliers dataset

C Performance of Shorter Parameter Optimisation

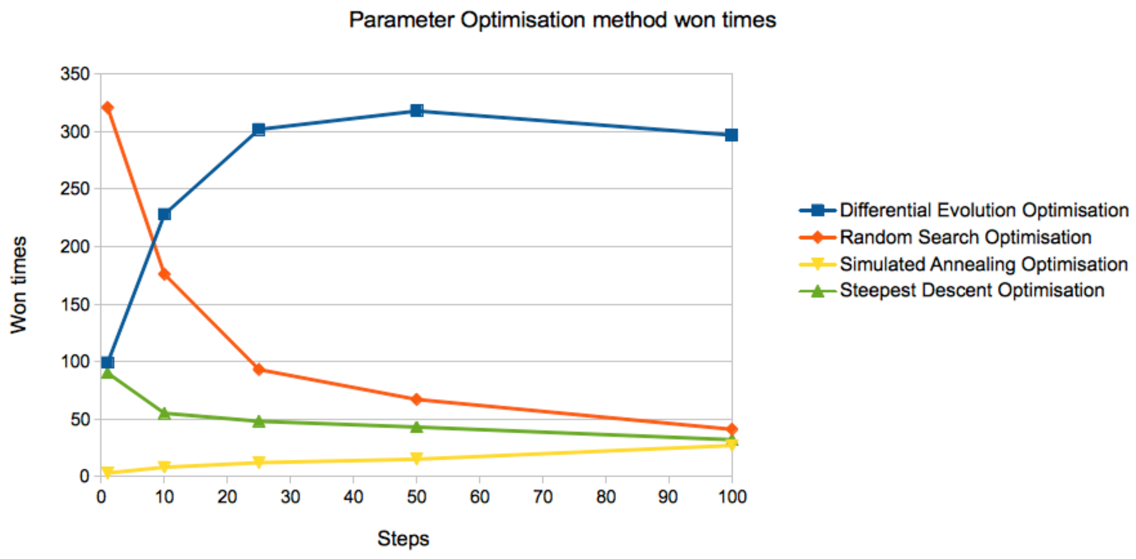


Figure C.1: Illustrates number of cases in which the given parameter optimisation method is able to find the best setup for the *Missing Data* dataset.

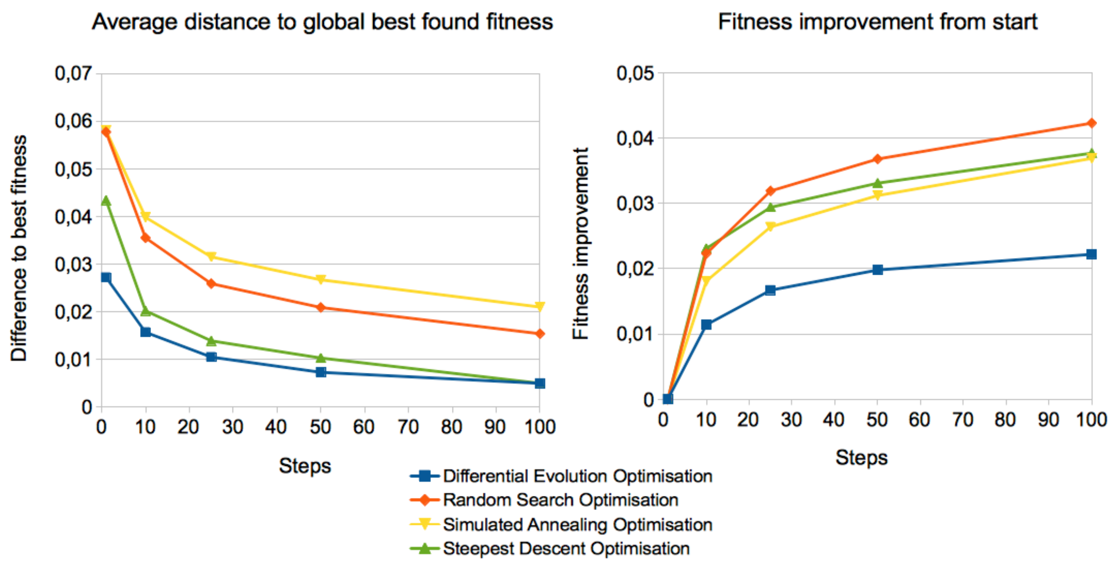


Figure C.2: Fitness improvement for the *Missing Data* dataset. On the left is the difference between average fitness in given optimisation step and the best fitness ever found (Lower values are better). On the right is the improvement of fitness from the initial value in the first step (Higher value is better).

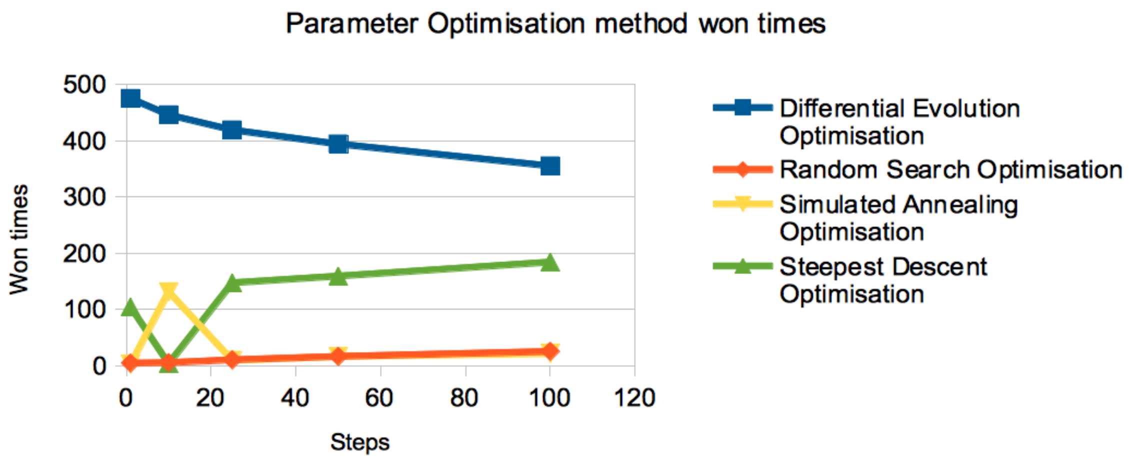


Figure C.3: Illustrates number of cases in which the given parameter optimisation method is able to find the best setup for the *Imbalanced* dataset.

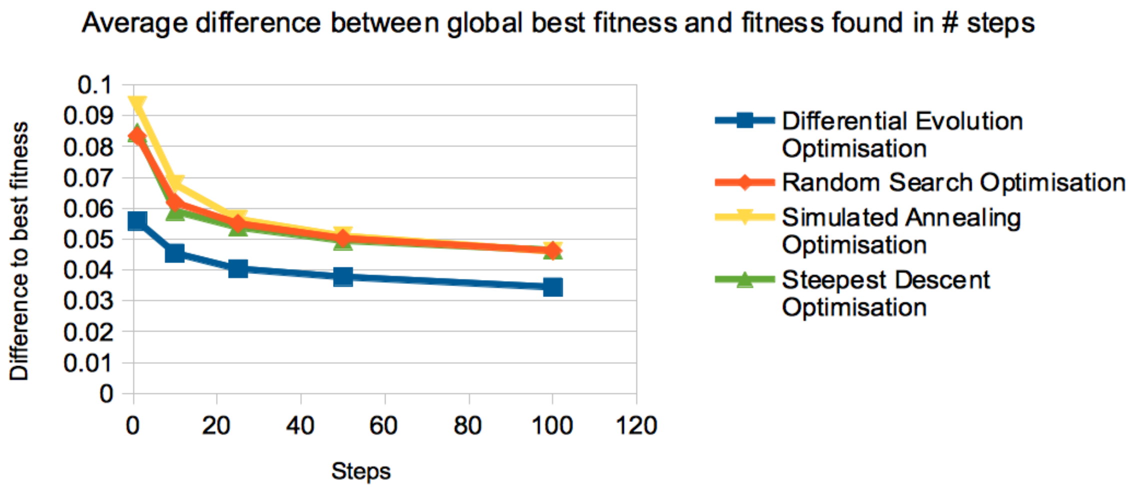


Figure C.4: Difference between average fitness in given optimisation step and the best fitness ever found (Lower values are better) for the *Imbalanced*.

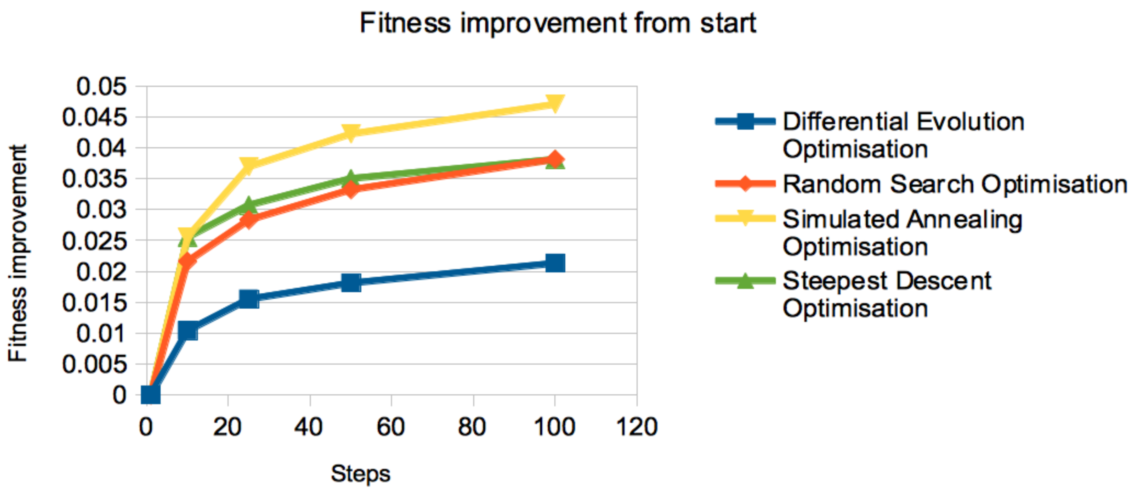


Figure C.5: Improvement of the fitness from the initial value in the first step (Higher value is better) for the *Imbalanced*.

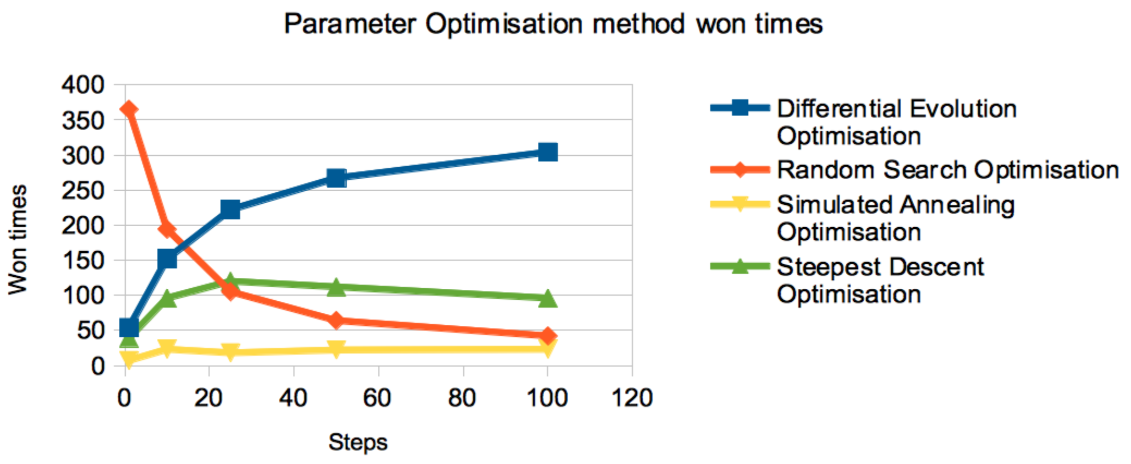


Figure C.6: Illustrates number of cases in which the given parameter optimisation method is able to find the best setup for the *Non Linear* dataset.

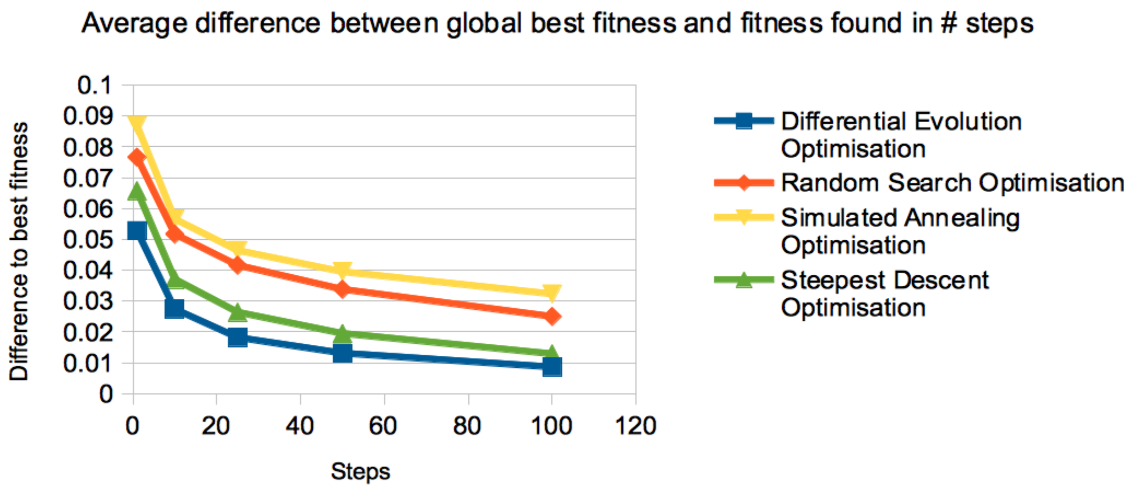


Figure C.7: Difference between average fitness in given optimisation step and the best fitness ever found (Lower values are better) for the *Non Linear*.

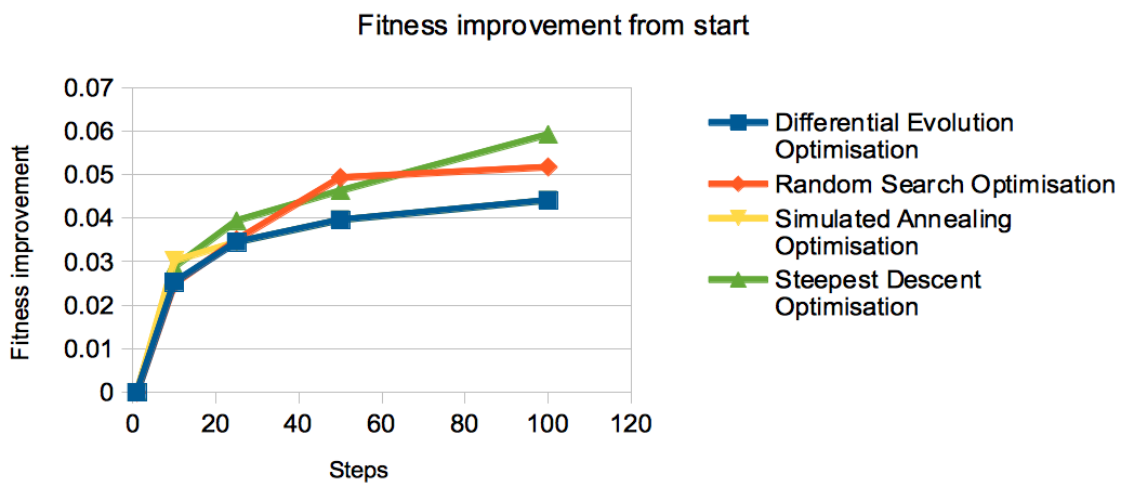


Figure C.8: Improvement of the fitness from the initial value in the first step (Higher value is better) for the *Non Linear*.

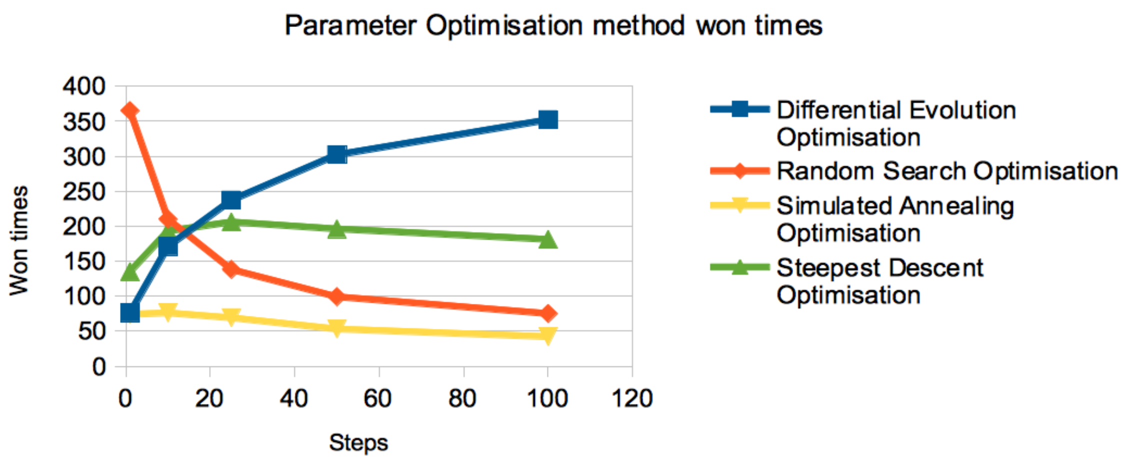


Figure C.9: Illustrates number of cases in which the given parameter optimisation method is able to find the best setup for the *Outlier* dataset.

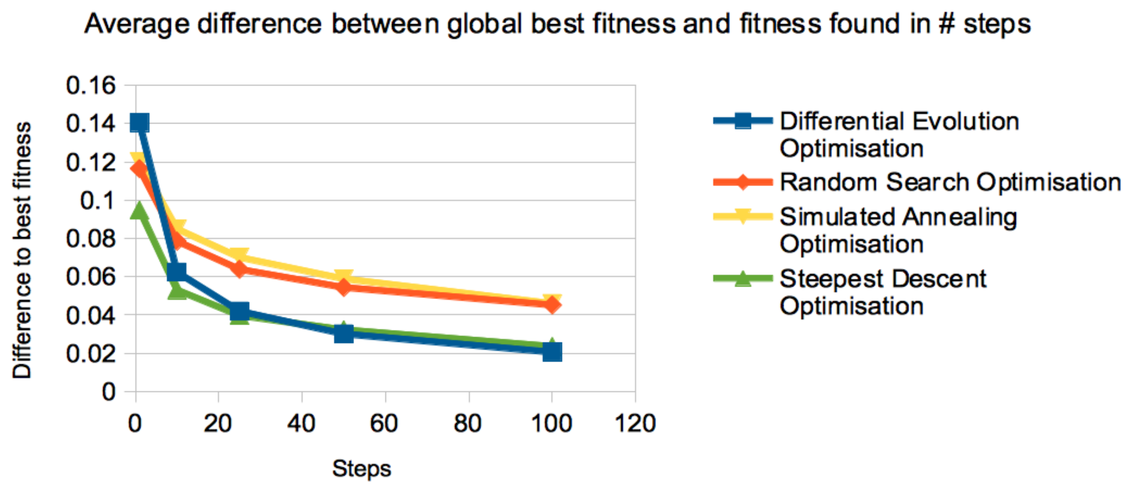


Figure C.10: Difference between average fitness in given optimisation step and the best fitness ever found (Lower values are better) for the *Outlier*.

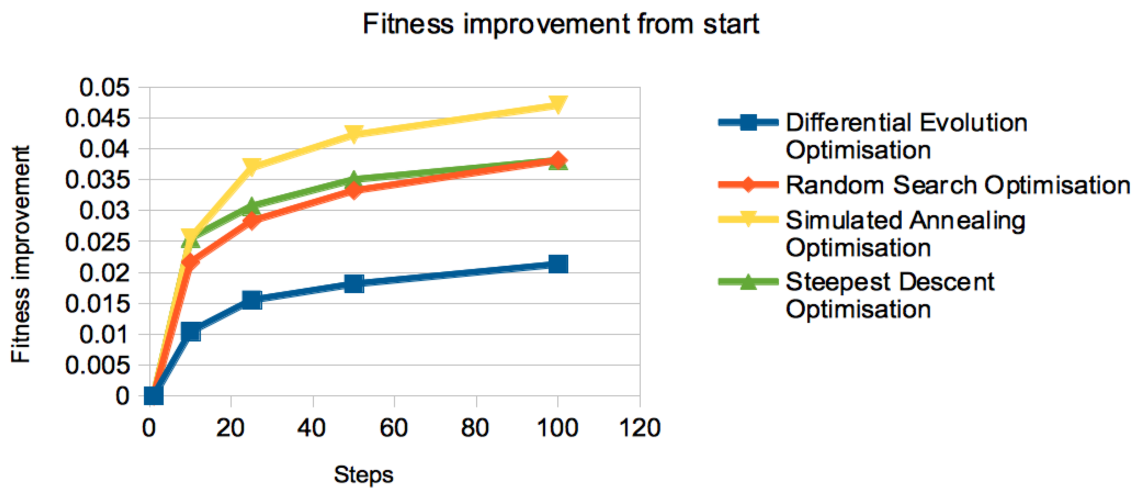


Figure C.11: Improvement of the fitness from the initial value in the first step (Higher value is better) for the *Outlier*.

D Search for the most Accurate Sequences

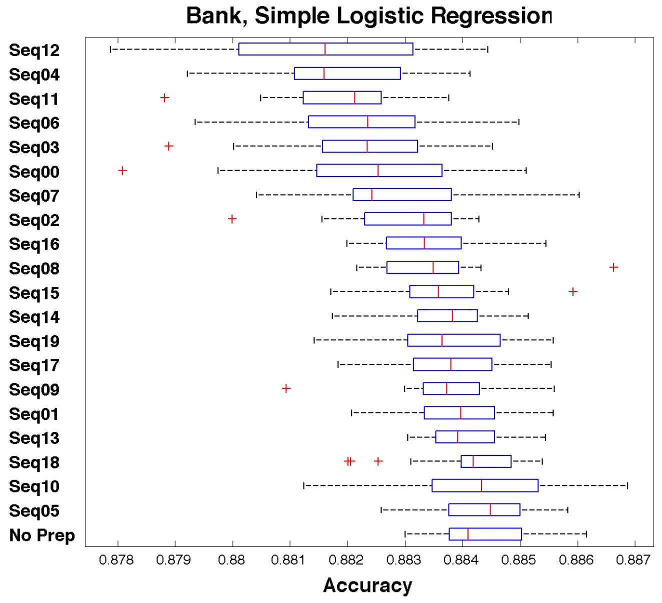


Figure D.1: Fitness of the best sequences found by IPT for the Bank dataset and the logistic regression classifier in repetitive runs and their comparison to the not preprocessed dataset.

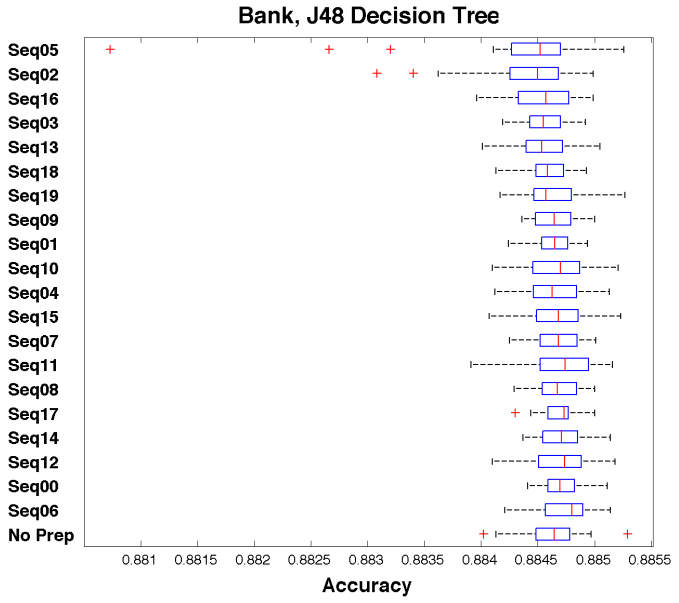


Figure D.2: Fitness of the best sequences found by IPT for the Bank dataset and the J48 Decision Tree in repetitive runs and their comparison to the not preprocessed dataset.

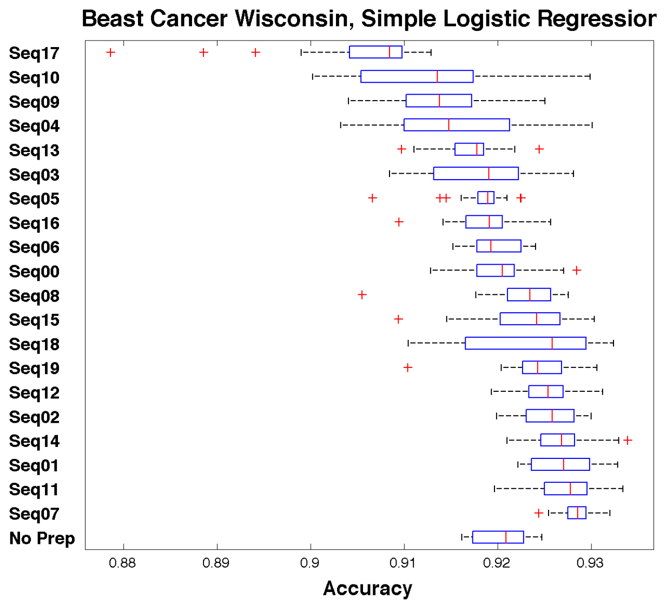


Figure D.3: Fitness of the best sequences found by IPT for the Breast Cancer Wiskonsin dataset and the logistic regression classifier in repetitive runs and their comparison to the not preprocessed dataset.

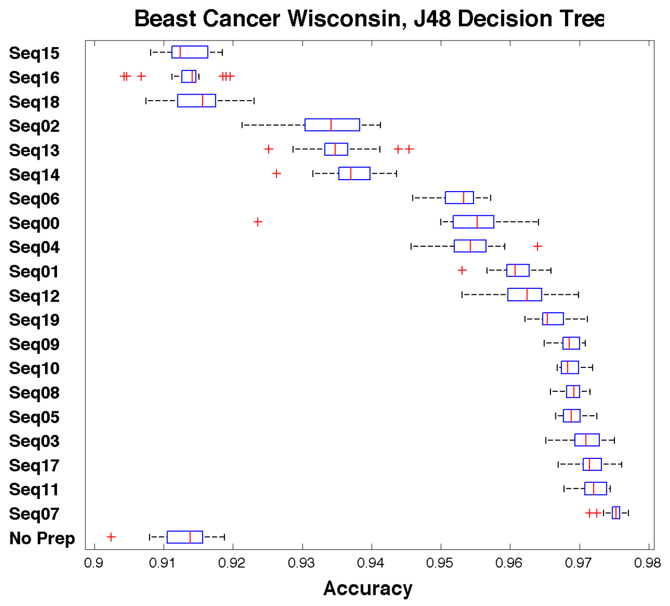


Figure D.4: Fitness of the best sequences found by IPT for the Breast Cancer Wiskonsin dataset and the J48 Decision Tree in repetitive runs and their comparison to the not preprocessed dataset.

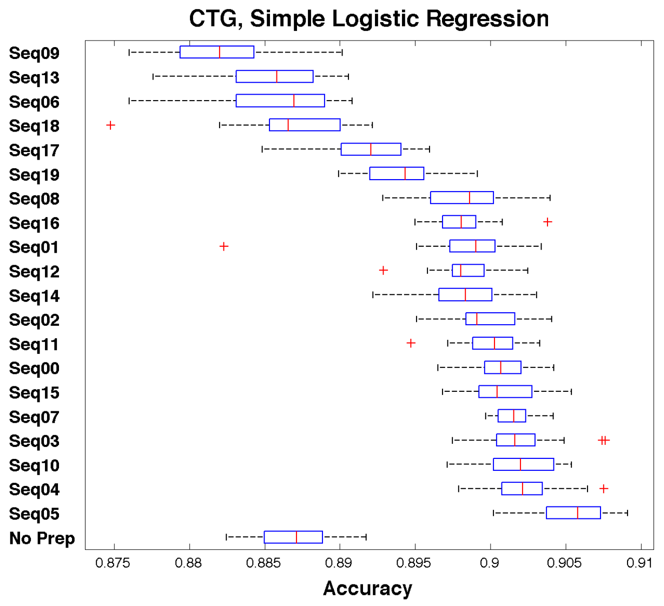


Figure D.5: Fitness of the best sequences found by IPT for the CTG dataset and the logistic regression classifier in repetitive runs and their comparison to the not preprocessed dataset.

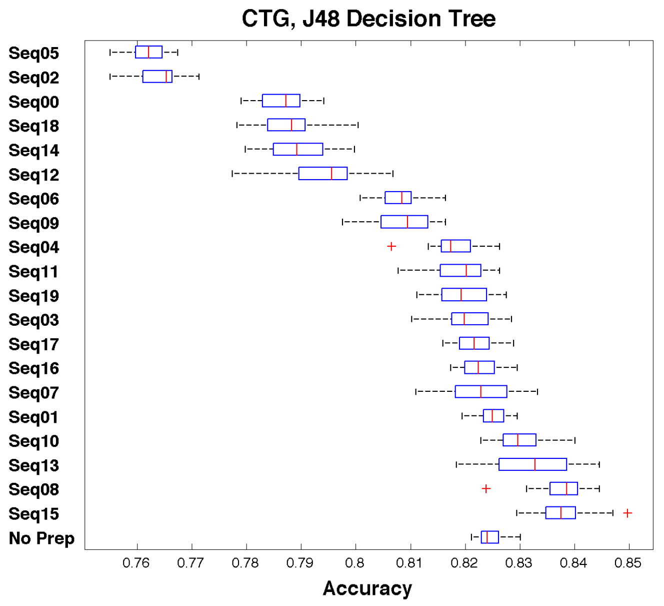


Figure D.6: Fitness of the best sequences found by IPT for the CTG dataset and the J48 Decision Tree in repetitive runs and their comparison to the not preprocessed dataset.

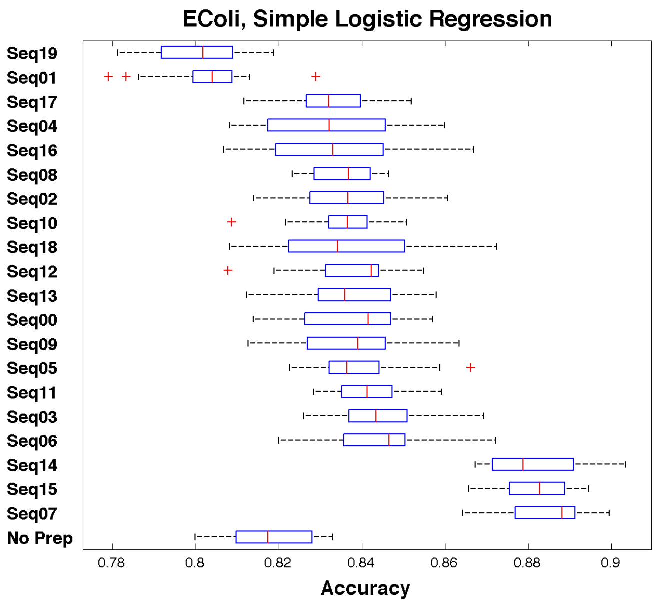


Figure D.7: Fitness of the best sequences found by IPT for the Ecoli dataset and the logistic regression classifier in repetitive runs and their comparison to the not preprocessed dataset.

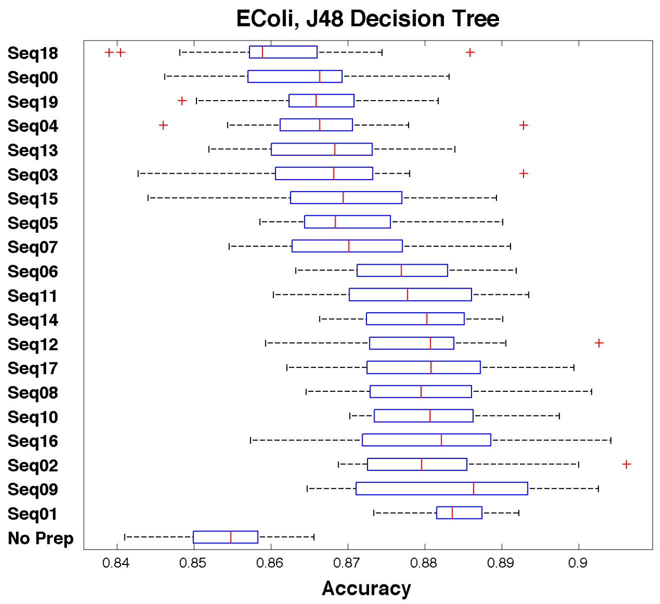


Figure D.8: Fitness of the best sequences found by IPT for the Ecoli dataset and the J48 Decision Tree in repetitive runs and their comparison to the not preprocessed dataset.

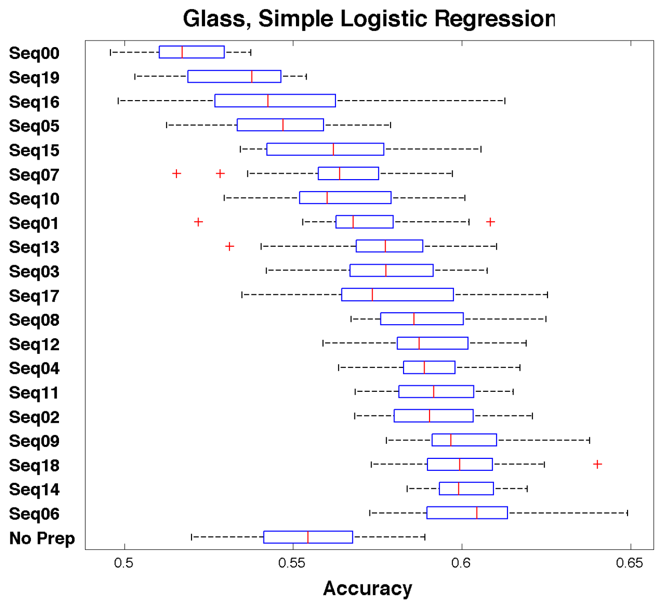


Figure D.9: Fitness of the best sequences found by IPT for the Glass dataset and the logistic regression classifier in repetitive runs and their comparison to the not preprocessed dataset.

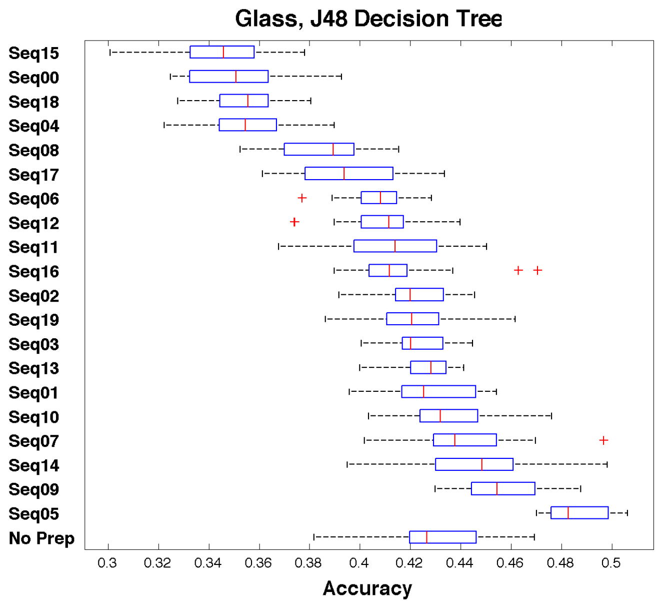


Figure D.10: Fitness of the best sequences found by IPT for the Glass dataset and the J48 Decision Tree in repetitive runs and their comparison to the not preprocessed dataset.

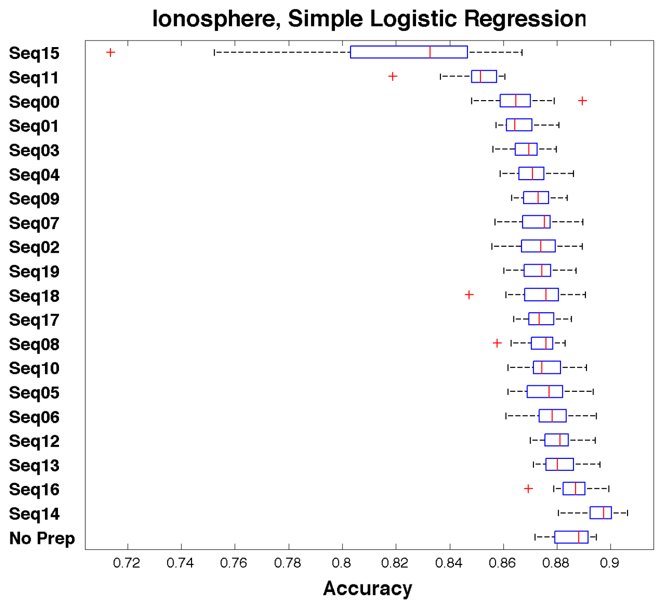


Figure D.11: Fitness of the best sequences found by IPT for the Ionosphere dataset and the logistic regression classifier in repetitive runs and their comparison to the not preprocessed dataset.

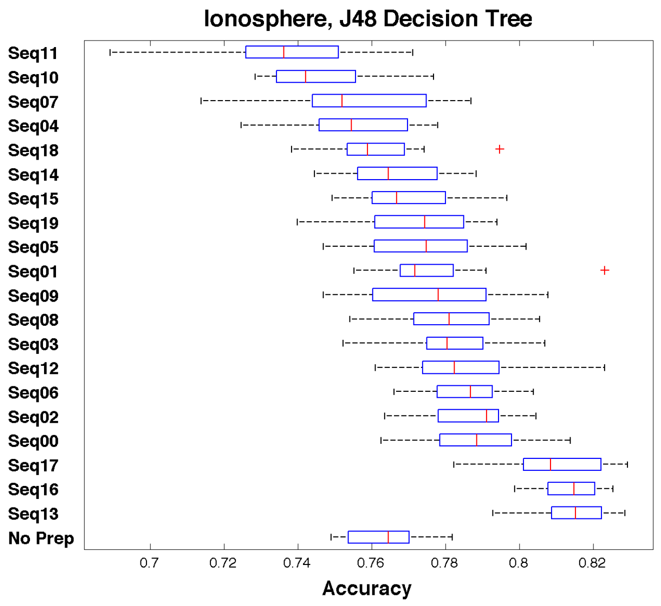


Figure D.12: Fitness of the best sequences found by IPT for the Ionosphere dataset and the J48 Decision Tree in repetitive runs and their comparison to the not preprocessed dataset.

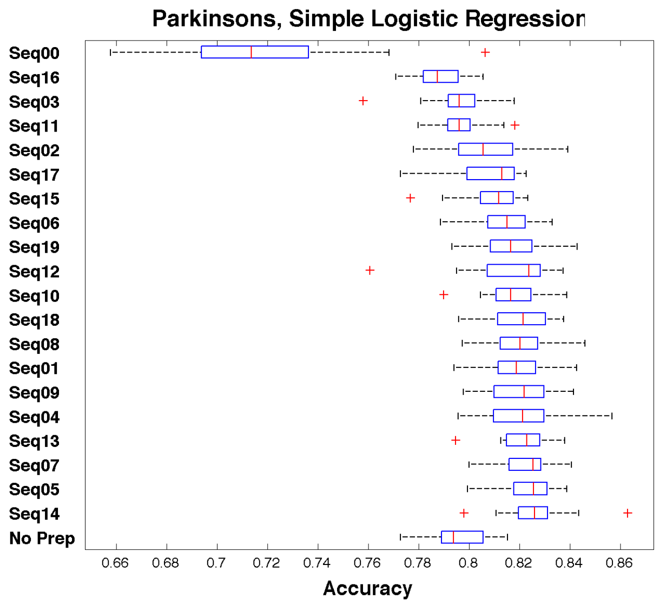


Figure D.13: Fitness of the best sequences found by IPT for the Parkinsons dataset and the logistic regression classifier in repetitive runs and their comparison to the not preprocessed dataset.

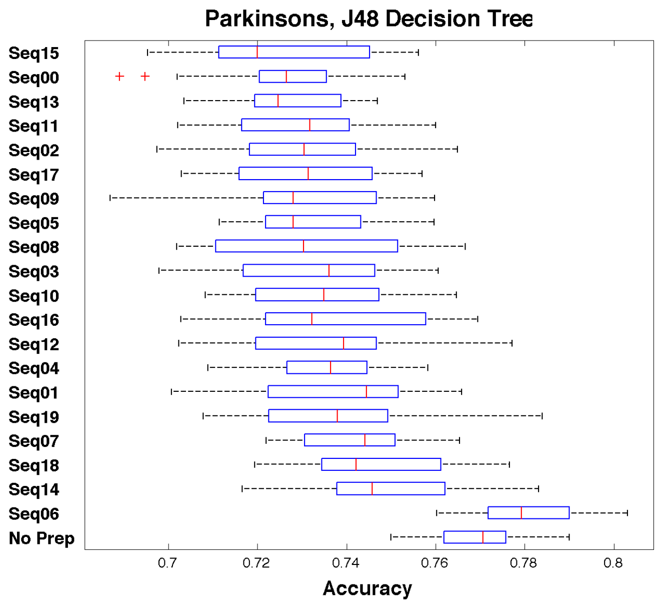


Figure D.14: Fitness of the best sequences found by IPT for the Parkinsons dataset and the J48 Decision Tree in repetitive runs and their comparison to the not preprocessed dataset.

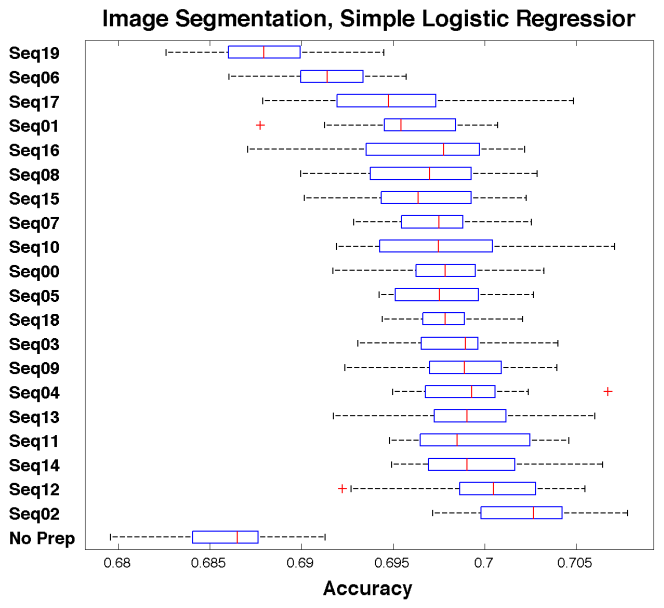


Figure D.15: Fitness of the best sequences found by IPT for the Segment dataset and the logistic regression classifier in repetitive runs and their comparison to the not preprocessed dataset.

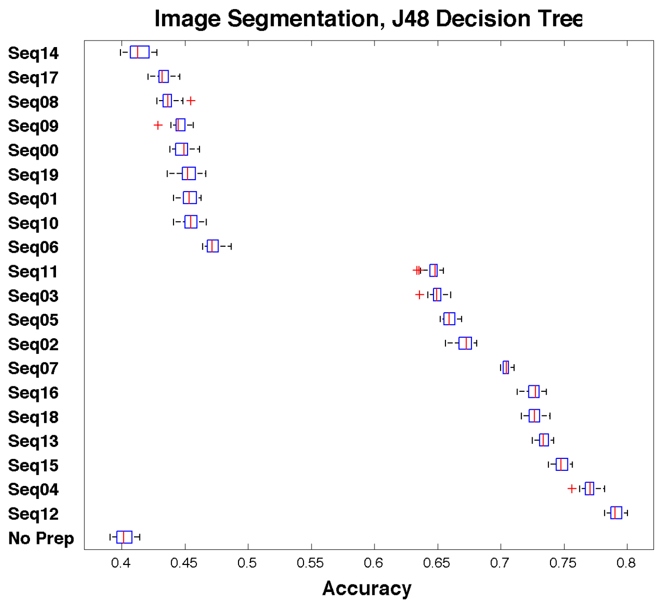


Figure D.16: Fitness of the best sequences found by IPT for the Segment dataset and the J48 Decision Tree in repetitive runs and their comparison to the not preprocessed dataset.

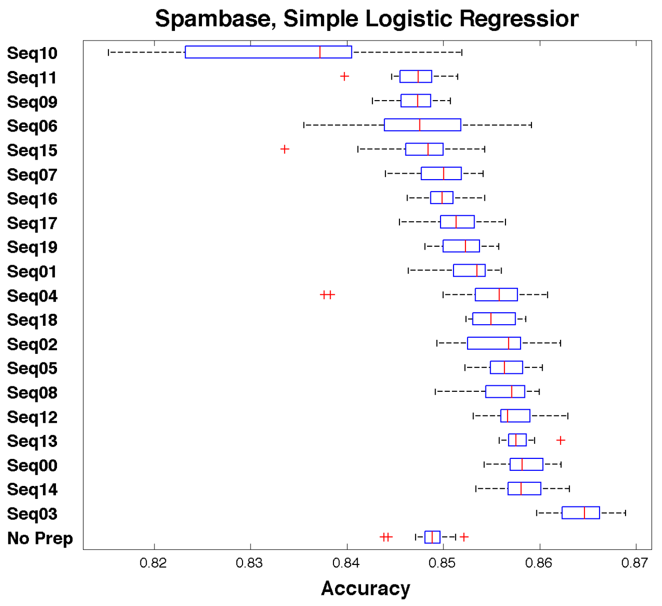


Figure D.17: Fitness of the best sequences found by IPT for the Spambase dataset and the logistic regression classifier in repetitive runs and their comparison to the not preprocessed dataset.

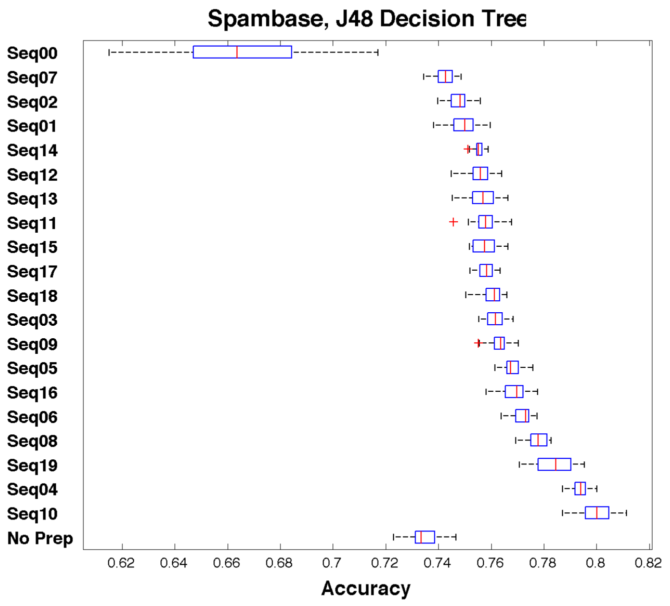


Figure D.18: Fitness of the best sequences found by IPT for the Spambase dataset and the J48 Decision Tree in repetitive runs and their comparison to the not preprocessed dataset.

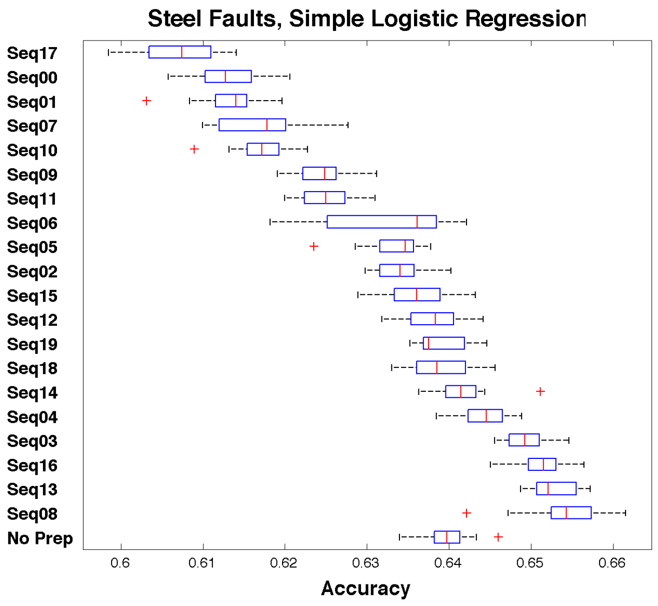


Figure D.19: Fitness of the best sequences found by IPT for the Steel Faults dataset and the logistic regression classifier in repetitive runs and their comparison to the not preprocessed dataset.

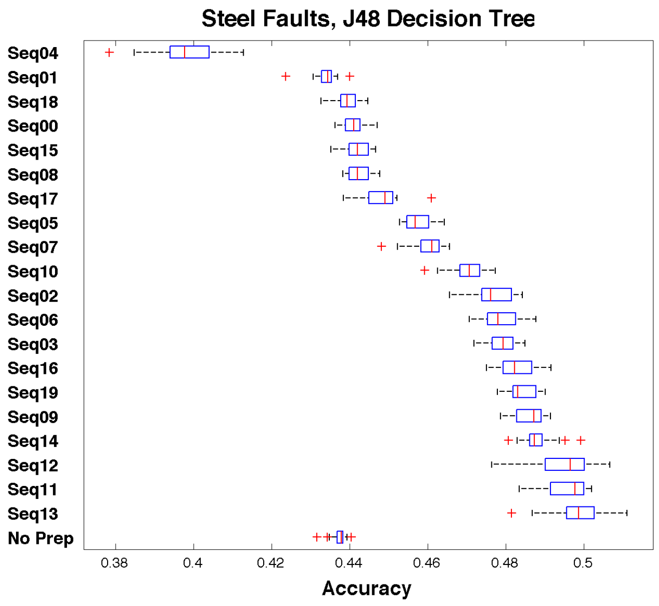


Figure D.20: Fitness of the best sequences found by IPT for the Steel Faults dataset and the J48 Decision Tree in repetitive runs and their comparison to the not preprocessed dataset.

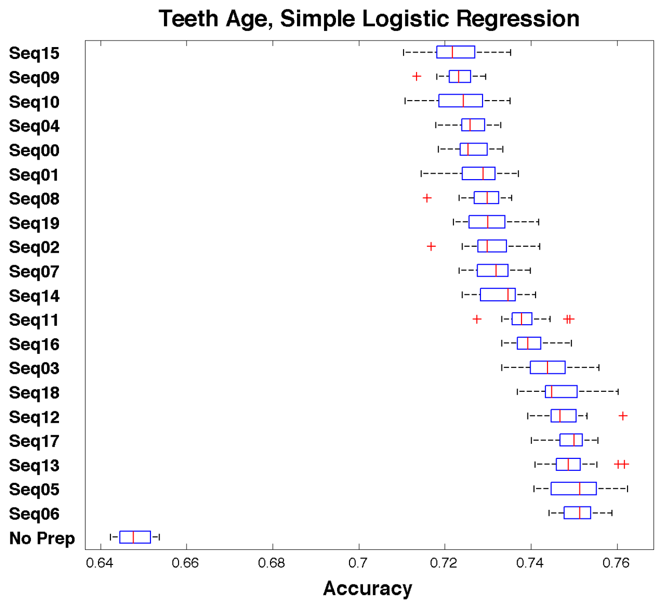


Figure D.21: Fitness of the best sequences found by IPT for the Teeth Age dataset and the logistic regression classifier in repetitive runs and their comparison to the not preprocessed dataset.

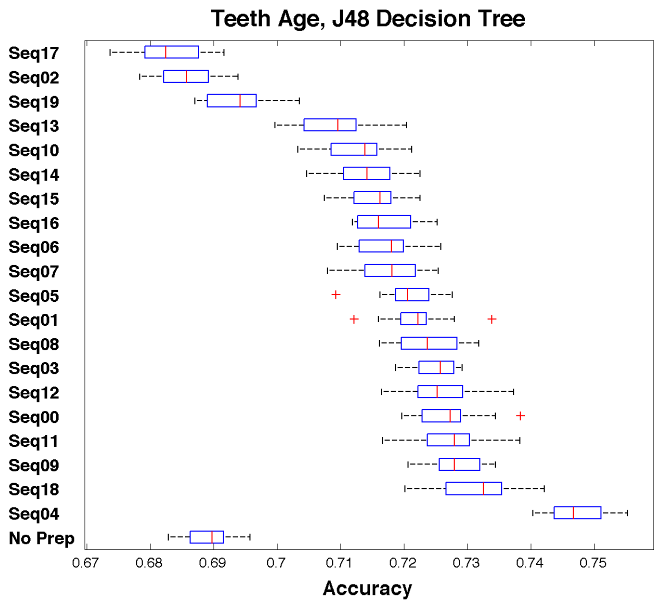


Figure D.22: Fitness of the best sequences found by IPT for the Teeth Age dataset and the J48 Decision Tree in repetitive runs and their comparison to the not preprocessed dataset.

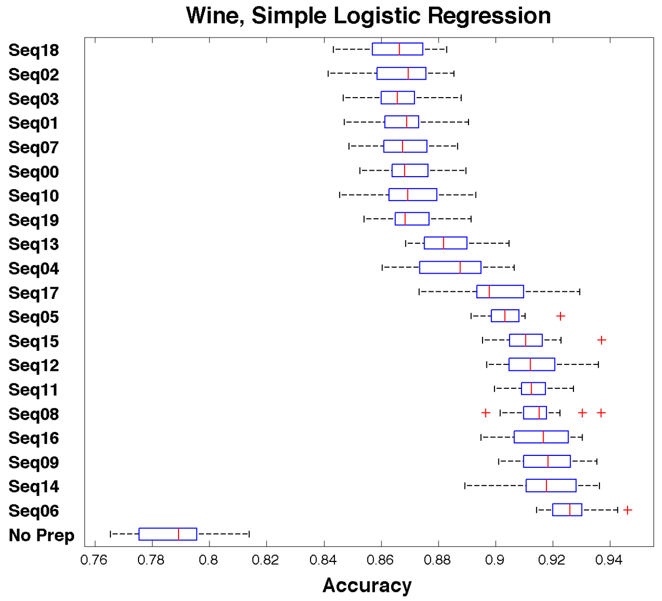


Figure D.23: Fitness of the best sequences found by IPT for the Wine dataset and the logistic regression classifier in repetitive runs and their comparison to the not preprocessed dataset.

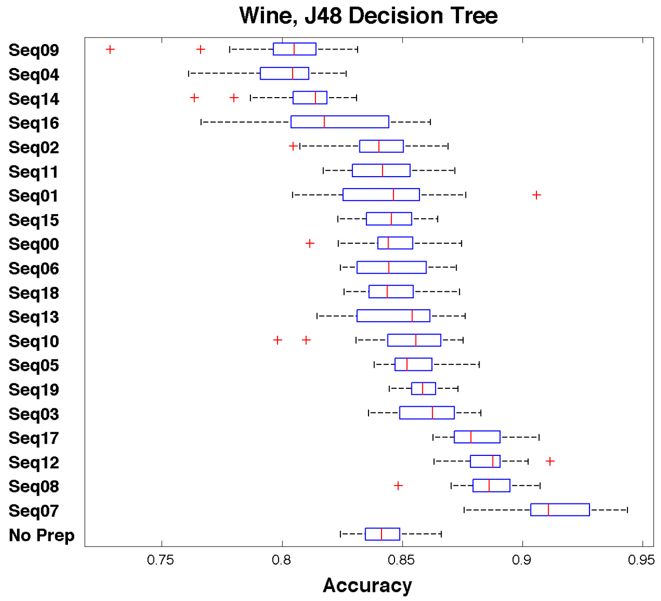


Figure D.24: Fitness of the best sequences found by IPT for the Wine dataset and the J48 Decision Tree in repetitive runs and their comparison to the not preprocessed dataset.

E Performance of the Sequences Generated from the Logistic Regression Meta Database

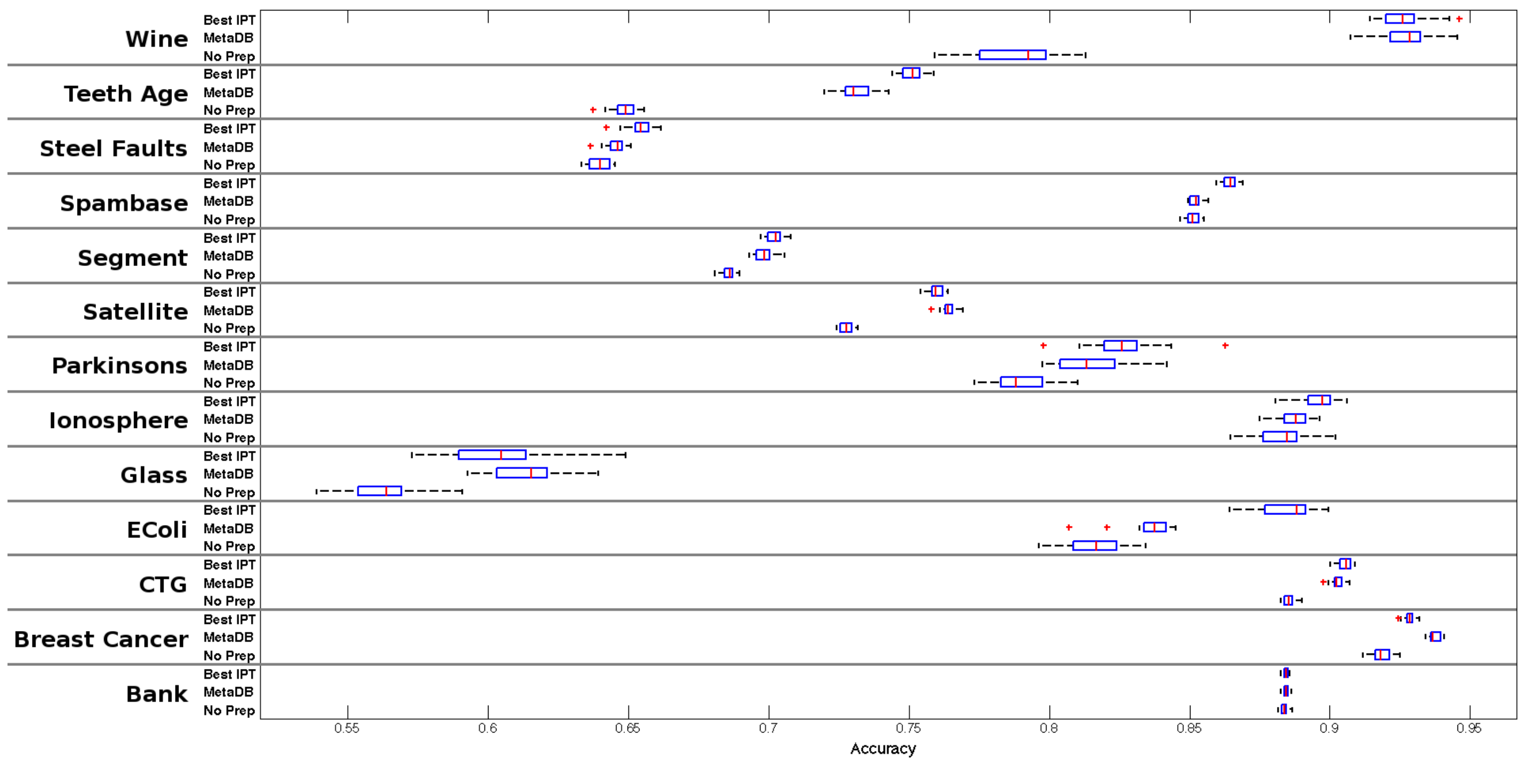


Figure E.1: Comparison of the fitness (accuracy) of the Simple Logistic Regression Classifier on original datasets, sequences generated from the Simple Logistic meta database and the sequences generated by IPT.

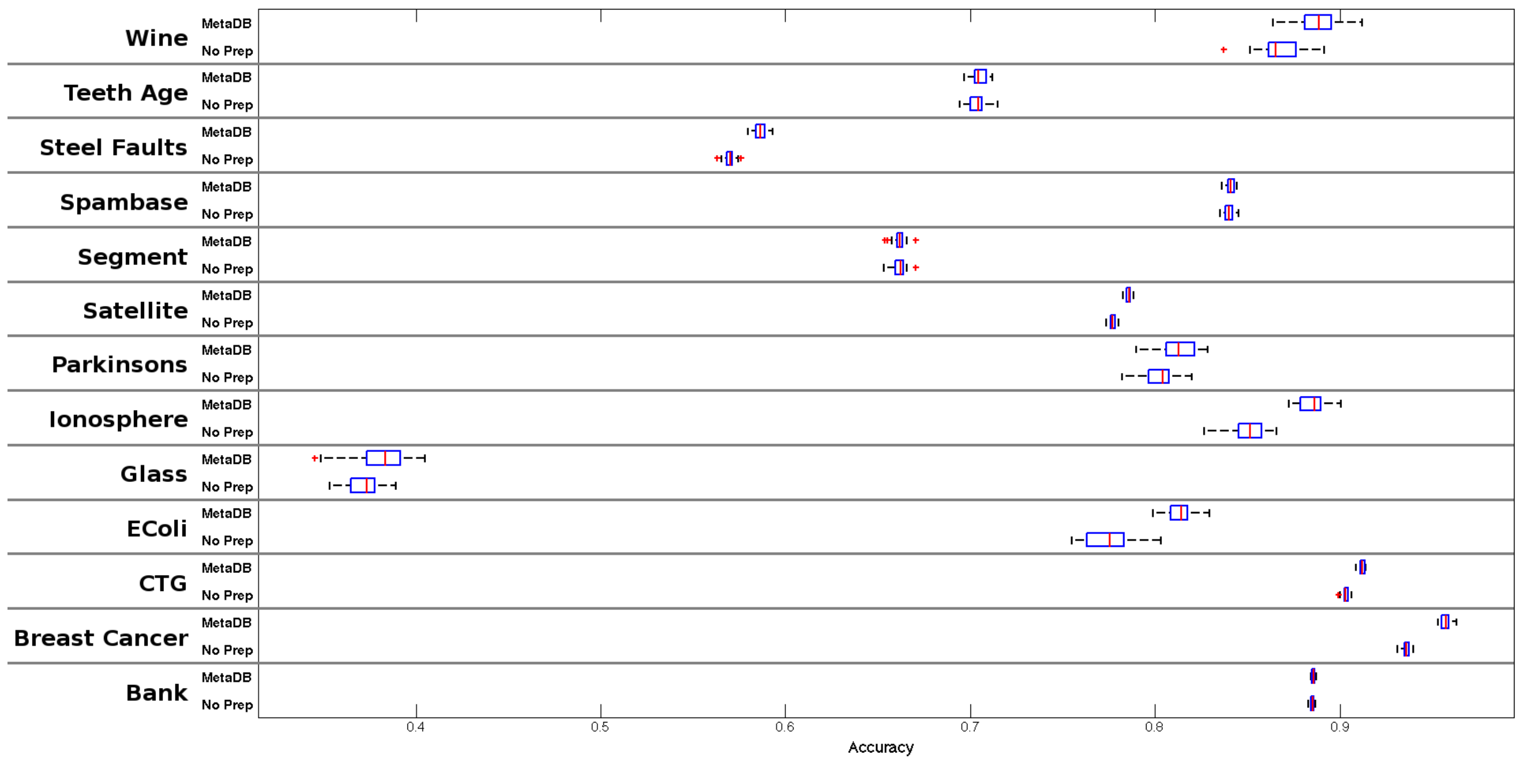


Figure E.2: Comparison of the fitness (accuracy) of the Simple Logistic Regression Classifier on original datasets and datasets preprocessed by sequences generated from the meta database.