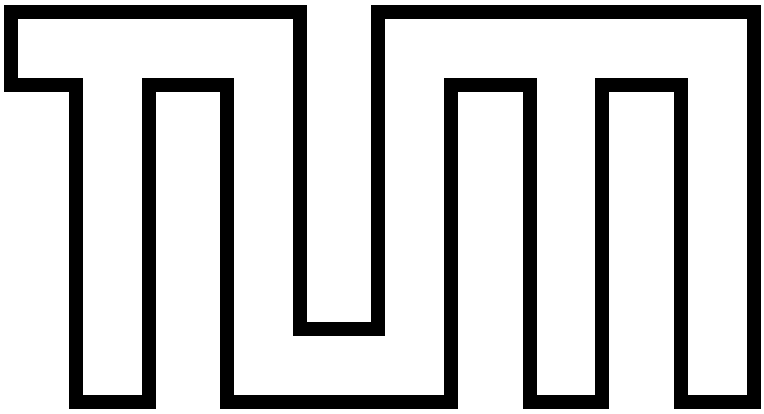


TECHNISCHE UNIVERSITÄT MÜNCHEN

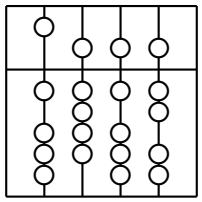
Feature Space Transformation using Directed Equation Discovery

Diploma Thesis in Computer Science

Mark Pflüger

DEPARTMENT OF INFORMATICS

TECHNISCHE
UNIVERSITÄT MÜNCHEN



Computer Science IX
Image Understanding & Knowledge-Based Systems

Feature Space Transformation using Directed Equation Discovery

Diploma Thesis in Computer Science

Mark Pflüger[1]

Advisor	:	Prof. Michael Beetz, PhD
Supervisor	:	Freek Stulp
Submission Date	:	July 15, 2006

[1]email: pflueger@cs.tum.de

I hereby confirm that I have composed this thesis autonomously, based exclusively on the sources and resources stated explicitly.

Munich, the 15th of July 2006

(*Mark Pflüger*)

Abstract

In my thesis an algorithm for improving the representation of problems for supervised learning is developed. First, I show the need for feature transformation, the process of adapting the features used for learning to the task at hand. Then, I present the algorithm, which is based on *Equation Discovery*. The hypothesis space of all terms that can be generated from the initial features and a set of given operators is searched using a hill-climbing approach with single-step lookahead. Construction and selection of new features are iteratively repeated. The fitness of new features is determined according to the linear correlation with the target concept. Due to the complexity of the search space an emphasis of my work lies on the reduction thereof. Another central aspect are means to represent and employ user-supplied domain knowledge. I analyze the properties of the system and evaluate it with artificial and real-world examples. Finally, I point to related work in the field and future directions of research for my project.

Contents

Chapter 1

Introduction

"Crude classifications and false generalizations are the curse of organized life."

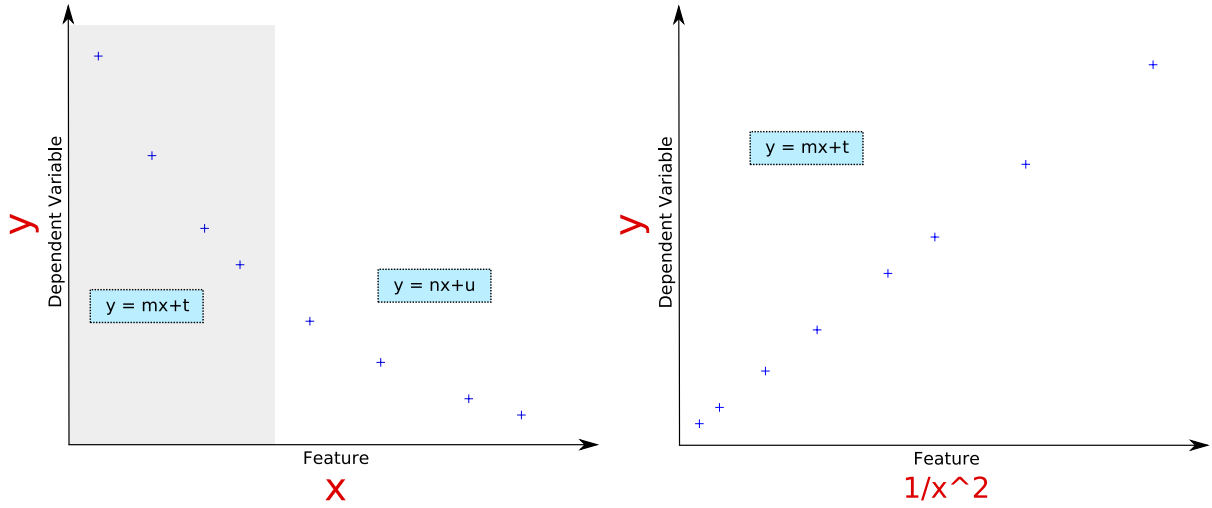
— *George Bernard Shaw (1856-1950)*

1.1 Motivation

The field of Artificial Intelligence aims at constructing machines, or algorithms for that matter, that exhibit intelligence. General apprehension gives that learning plays a substantial role in the formation of intelligence. The discipline in computer science that deals with learning machines is called, rather evidently, Machine Learning. Research is directed towards inducing and deducing concepts from empirical data. Deducing concepts or recognizing patterns can happen with the help of a teacher, who classifies the given examples during the learning phase. This kind of learning is called supervised learning and it is the setting of my inquiry.

The goal of supervised learning is to automatically find a classifier $[f_1, \ldots, f_n] \rightarrow c$ that maps an unseen instance of the feature vector $[f_1, \ldots, f_n]$ to a class c. The input for learning is a vector $[[f_1, \ldots, f_n, c]_1, \ldots, [f_1, \ldots, f_n, c]_m]$ consisting of m observations and their respective class. The domains of the features can be continuous or nominal. The task corresponds with the one of function approximation, which can be tackled with statistical methods like regression, discriminant analysis, or k-nearest-neighbor. Furthermore, multilayer perceptrons can be employed and finally trees that split the data according to some criterion on every node like decision, regression, or model trees.

The success and performance of the learning process using any of these systems often critically depends on finding suitable encodings of the feature space $f_1 \times \ldots \times f_n$ [1] [40]. Feature spaces can be considered as a representation language, and I will use these terms interchangeably. Constructing new feature languages from the initial feature space involves discarding features that are irrelevant to classification, as well as combining features. While



(a) One split on initial feature language resulting in a big error.

(b) No splitting with transformed feature language yields better results.

Figure 1.1: Learning a bivariate problem with a model tree

there is a variety of solutions for feature selection, there are few for feature induction, and thus manual feature design is prevalent. It is guided by human intuition and domain knowledge and, often enough, is a difficult and time consuming process [51].

My work was initially motivated by Freek Stulp, as he searched for a good representation of a problem in the field of robotic navigation. His work [62] focuses on the seamless transition between actions for plan-based robot controllers. There are certain typical problems that arise during manual feature design, which also posed themselves in this case. Aside from the time factor, that was mentioned above, the process is also error-prone. Relevant information might be lost during transformation and irrelevant information needlessly preserved. This problem will be picked up again in section 6.2.

As an example of how feature transformation can improve learning performance, let us consider the simple physical setting described in [22]. The measurements compare the volume of an enclosed quantity of air and the frequency at which it resonates. In figure 1.1 the volume is depicted on the x-axis and the frequency on the y-axis. When learning a decision rule for the frequency with a model tree, the data is split into two regions (see figure 1.1(a)), and for each, the best linear regression is returned. The result is a root relative squared error of 46.48% on the given dataset with 3-fold-cross-validation. On unseen cases the result is even worse if the volume is outside the range of the training set.

The source suggests the reference relation $y = k/x^2$, and using this as a transformation for our new feature language yields much better results. Now the model tree will not make any splits and just return a linear relation between the volume and the frequency as shown in figure 1.1(b). Assuming that the reference relation correctly models the behavior of the underlying process, the error is in the order of magnitude of the measurement error. In

fact, since there is only one input variable, this practically eliminates the need for using a model tree.

This thesis describes an algorithm named *SAFI (System for Automated Feature Induction)*, that discovers feature languages, which enable learning programs to yield a better performance. The system transforms the initial feature space by repeated feature construction and selection. Equation Discovery is used to induce new features. The adequacy of a feature is assessed based on how strongly it is correlated with the concept to be learned. An emphasis lies on the reduction of search space complexity and the possibility to explicitly represent domain knowledge.

1.2 An Example: Iris Classification

To give the reader a better feeling how *SAFI* works and where it can be applied we will walk through a hypothetical session step by step. We will use the often cited Iris Plants Database [49]. There are 150 instances, each classifying the three different kinds of Irises, Setosa, Versicolor and Virginica with the four attributes petal and sepal length and width. Using the C4.5 [54] algorithm in Weka [72] or any other implementation returns a decision tree, which in this case incorrectly classifies 11 instances, see 1.1.

Listing 1.1: Decision Tree for Iris dataset

```
PetalWidth <= 0.6: Setosa (50.0)
PetalWidth > 0.6
|   PetalWidth <= 1.7
|   |   PetalLength <= 4.9: Versicolor (48.0/1.0)
|   |   PetalLength > 4.9: Virginica (6.0/2.0)
|   PetalWidth > 1.7: Virginica (46.0/1.0)
```

Since *SAFI* only works for numeric values we have to convert the three nominal classes to integers with the simple mapping [Setosa, Versicolor, Virginica] $\rightarrow$ [1, 2, 3]. Because we want to discover equations that fit the underlying process well, not only the input dataset, we select a small subset of 10 examples with a uniform class distribution. In general, we would be cautious not to use examples with numerically instable values (e.g. values near zero), but in this case there are none. Now the data are prepared and we can leave them in the standard Weka "arff" format, as *SAFI* uses this same format for input and saving files.

There are several parameters that control the inner workings of *SAFI*. Here is a short explanation of the two most important ones:

- *Operators* – List of operators to be used, e.g. (+ / sin),
- *Depth* – Maximum search depth.

For every level smaller than *Depth*, one generation of features is constructed from the initial features, the ones already found and all *Operators*. The system supports all built-in operators provided by the underlying language plus any user-defined functions. Thus, *SAFI* can effortlessly be applied to new domains by extending the set of available operators. These two parameters influence the complexity of the search space and consequently the runtime of the system.

For this dataset we run the program with *Operators*=$(+ - \cdot /)$ and *Depth*=3. The resulting output is shown in listing 1.4. All features are very similar to one another. The first term in fact just squares $PetalLength$ and $PetalWidth$ in comparison to the second, while the last one squares $SepalLength$. The squaring seems to be an adaption/regression to the given data, not a general rule. This also implies that further searching will not have a positive effect on the learning algorithms and thus we choose the feature from line 2.

Listing 1.2: Output of *SAFI* for the Iris dataset

```
1 (PetalLength^2*PetalWidth)/(SepalWidth*SepalLength^2):0.008
2 (PetalLength*PetalWidth)/(SepalLength*SepalWidth):0.011
3 (PetalWidth*PetalLength)/(SepalLength^2*SepalWidth):0.019
```

As a general rule that has developed during working with my program, it is advisable to use all parts of the final equation also separately. We can write the term we chose as $(PetalWidth \cdot PetalLength) / (SepalWidth \cdot SepalLength)$ or as $(PetalWidth/SepalWidth) \cdot (PetalLength/SepalLength)$ and should include all the sub-terms in parentheses in our feature language. For the final feature evaluation we use a feature subset selection algorithm, preferably one employing the wrapper method. It should return $PetalLength$ and $PetalWidth \cdot PetalLength$ as the best features. Using only these two terms in our final feature language misclassifies only 7 instances, see listing 1.3.

Listing 1.3: Decision Tree for Iris dataset with transformed feature language

```
PetalLength <= 1.9: Setosa (50.0)
PetalLength > 1.9
|   PetalArea <= 7.35: Versicolor (46.0)
|   PetalArea > 7.35: Virginica (54.0/4.0)
```

1.3 Solution Idea

We have seen how *SAFI* works for the user. To accomplish its task in the manner seen above, the system has to meet the following two requirements. First, the new features have to be easily understandable to a human reader, and second, the new feature space has to improve the learning performance. To fulfill the first demand, I employ *Equation Discovery*. For the second, I stop discovery at a certain level and choose a subset that allows supervised learning algorithms to find a better classifier with less examples. This has been visualized in figure 1.2.

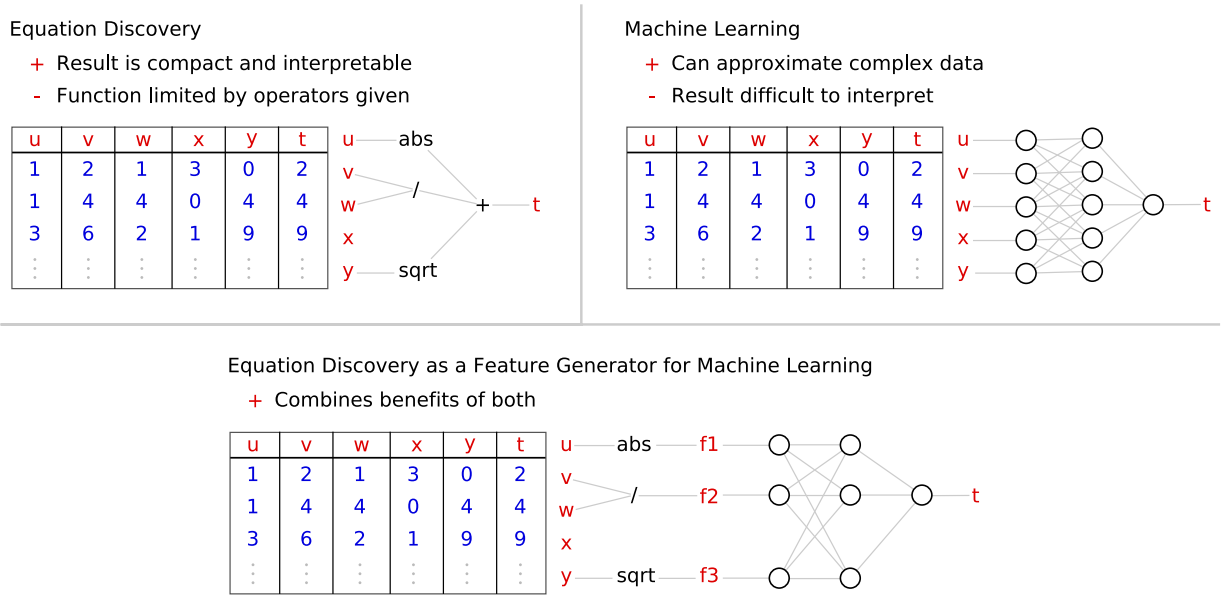


Figure 1.2: Combining Equation Discovery and Machine Learning.

The system searches the hypothesis space of possible equations given certain operators. This process can reduce the dimensionality of the feature space, remove redundancies by exploiting invariances and thus enhance the performance of the post-processing classifiers. However, it can not entirely automate manual feature design and replace human intuition. After all, an important selection rule to the human intellect is the beauty of an equation. This is a concept that remains to be grasped by machines and I leave the task as an exercise to Strong Artificial Intelligence. On a whole, *SAFI* can help guiding and partially automating the process of feature design.

1.4 Contributions

The contributions of my work are threefold:

- Introduction of a novel *Equation Discovery* system for *Feature Transformation*.
- Reduction of search space complexity through user-supplied domain knowledge and term reduction.
- Lastly, the application of *Equation Rediscovery* during the evaluation of the algorithm.

1.5 Outline

To give a short overview over the path that lies ahead, let us first take a look at the main routine of *SAFI*. A dense version of the program in natural language syntax that is loosely similar to Pascal is shown in listing 1.4. The most important parameters were already introduced in section 1.2, here we only add:

- *PreSelect* – Percentage of equations to use for lookahead,
- *PostSelect* – Number of equations to include in feature language after every new generation of features,
- *FinalSelect* – Number of equations to keep in the end.

Listing 1.4: *SAFI* Algorithm

```
procedure generate
   1. Introduce new features from Operators and features
end procedure

program main
repeat i <= Depth
   1. generate() features for Level i   { see Chapters 2 and 3 }

   2. Select the best PreSelect features   { see Chapter 4 }

   3. generate() lookahead from these features

   4. Select the best PostSelect features from level i
end repeat
5. Select the best FinalSelect features
end program
```

The arrangement of the following chapters follows the order of appearance of the relevant lines of code. The upcoming chapter explains how new features are generated with the use of *Equation Discovery*. Due to the complexity of the equation tree measures to leave out irrelevant features have to be taken. The methods employed in this endeavor are discussed in the two following chapters. First, in chapter 3, I show how the generation of irrelevant features is avoided. Then, in chapter 4, intermediate selection of features is the topic.

The two following chapters, 5 and 6, examine the theoretical properties of the algorithm and the empirical results. The two final chapters briefly point to related work and contemplate on the implications and the future direction of my research.

Chapter 2

Feature Generation

> "Children are born true scientists. They spontaneously experiment and experience and reexperience again. They select, combine, and test, seeking to find order in their experiences - "which is the mostest? which is the leastest?""
>
> — *R. Buckminster Fuller (1895-1983)*

2.1 Equation Discovery

The aim of *SAFI* is finding a compact representation of the feature space from given data that enables classic machine learning algorithms to produce more accurate classifiers with less examples. However, the discovered feature language must not only be suitable for further automated processing, but also be easily understandable to the human reader. To this end, the system uses *Equation Discovery* (ED) to generate new feature languages.

"These [ED] systems explore the hypothesis space of all equations that can be constructed given a set of arithmetical operators, functions and variables, searching for an equation that fits the input data best." [64] Generally, ED systems try to find equations with arbitrary left- and right-hand sides. Since *SAFI* prepares the data for use by machine learning algorithms the right-hand side always consists of solemnly the class. Another difference is that regression is avoided, in contrast to other ED programs, because this is handled by the post-processing machine learning system.

The hypothesis space is of course infinite and thus can not be extensively searched in acceptable time. All ED systems use different approaches to constrain the search space to a feasible size. Aside from this, all algorithms have different measures for evaluating the fitness of an equation. Examples for both these steps in other systems will be shortly discussed in chapter 7. The respective means employed in *SAFI* will be explained in the course of the following chapters.

2.2 Equation Tree

For our needs the hypothesis space can be represented as a rooted tree, with the roots being all the initial features. This is depicted in figure 2.2. Expressed formally we define a set of operators $O = o_1, \ldots, o_k$ and a set of initial features $F_0 = f_1, \ldots, f_n$. The operators $\forall i \leq k : o_i$ have arity a_i.

F_1 is constructed by combining all features F_0 with all operators 0, written as $F_0 \times F_0$. Generally speaking F_1 has $|F_1| = \sum_{i=0}^{k} n^{a_i}$ features. For the case that all operators are binary there are $|F_1| = (kn^2)$ features. Level F_{i+1} is constructed by means of $F_{i+1} = F_i \times F_i \cup F_i \times F_{i-1} \cup \ldots \cup F_i \times F_0$. For the tree grown to depth d the number of features $|F_d|$ is in the order of $\Theta(k^{2^d-1}n^{2^d})$.

Figure 2.2 depicts a detail of an equation tree grown with the operators $(\cdot, -)$ and two initial features, x and y. Construction is cut off after level 2, thus F_0, F_1 and F_2 are shown. It is obvious that there is an abundant number of superfluous features. While the next section will just shortly introduce means to deal with their removal, the next chapters will look deeper into this matter.

2.3 Traversal of the Equation Tree

In the introduction a dense version of the *SAFI* algorithm was sketched. We will now refine this code a little, using the notation we just introduced, to show how the equation tree is built and traversed (see listing 2.1). A more detailed description of the functions called will be given in the following two chapters. If any questions remain after reading this section, please browse ahead.

Generally speaking, my approach is that of forward *hill-climbing* with a *single-step lookahead* and a correlation based fitness measure. As [55] puts it, hill-climbing "resembles trying to find the top of Mount Everest in a thick fog while suffering from amnesia." The analogy comes from the fact that single steps in all directions are tried, taking the one that goes uphill the most without being able to see what lies beyond. The amnesia comes into play because at every position the same procedure is applied without considering any past steps taken.

The *single-step lookahead* translates to a slight thinning out of the fog in our metaphor, which allows us to see wether the step after the next one takes us uphill. This approach allows for possibly worse correlated intermediate steps. The algorithm strives to strike a balance between search speed and success of finding the best representation in terms of correlation with the concept.

The generation of features in the function `generate` is a process that takes place in two phases. In the first phase a list of all terms on the next level is generated. This does not include computing the new values of the data yet. Terms that represent neutral elements,

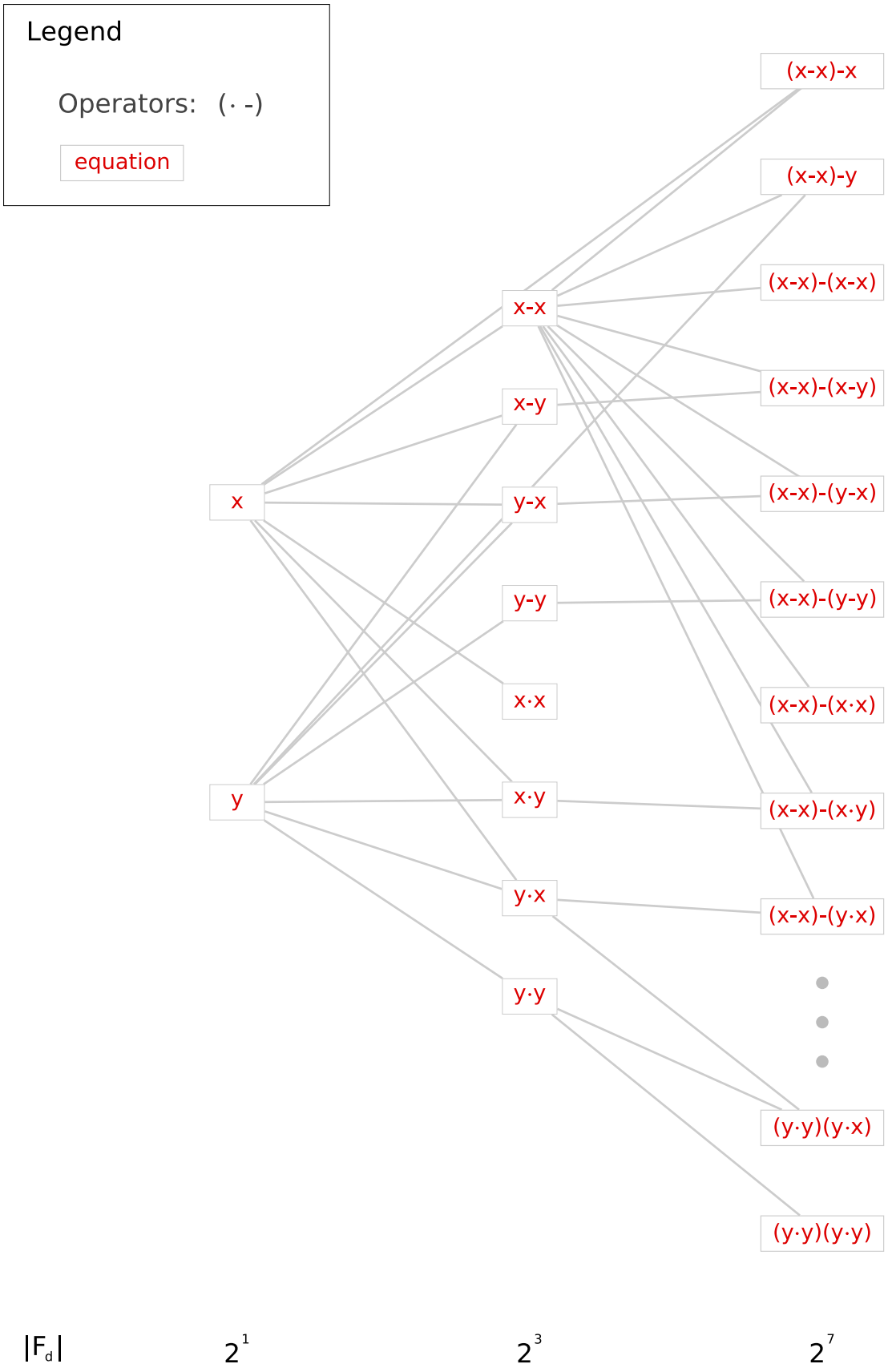


Figure 2.1: Unpruned equation tree grown to depth 2.

like $x - x$, are not generated at all. For commutative operators only one term is generated for any two features. For example, while $x \cdot y$ will be generated, $y \cdot x$ will not.

Listing 2.1: *SAFI* Algorithm

```
function generate(features F_i)
    1. return F_i × F_i ∪ F_i × F_{i-1} ∪ ... ∪ F_i × F_0
end function

program main
for (i = 1; i <= Depth; i++)
    1. F_i = generate(F_{i-1})    { see Chapters 2 and 3 }

    2. F_i = PreSelect(F_i)    { see Chapter 4 }

    3. F_lookahead = generate(F_i)

    4. F_i = PostSelect(F_lookahead, F_i)
end for
5. F_Depth = FinalSelect(F_Depth)
end program
```

All these terms will be checked against the constraints loaded into the system. These constraints can use predicates on the properties of the operators and the features and the data of the respective input features. Afterwards, the terms are subjected to *term reduction*, which rewrites the term by applying mathematical rules repeatedly. If the resulting term has less operators than the terms it was generated from, plus its own operator, it will be discarded. Either it has already been considered on a previous level or is a constant expression.

Finally, the data for the new features are generated by applying the operator to all data examples. Constant features, that have not been eliminated yet, and features that lead to arithmetical errors, like division by zero or floating point overflows, are being eliminated at this point. From the remaining features the best *PreSelect* percent are used to generate the *lookahead* level.

The final step in one iteration is to select the best *PostSelect* features of the level. This parameter should be chosen according to $\frac{1}{2}|f_1, \ldots, f_n| \leq PostSelect \leq 2|f_1, \ldots, f_n|$. For now, we will assume without loss of generality that x and y are features on the current level and $z = x \circ y$ is a feature on the *lookahead* level. The terms x and y will be selected if the feature z has a good fitness value. How the fitness of a feature is determined will be explained in section 4.2. For now let it suffice to say, that the measure is based on the linear correlation coefficient.

The shortcomings of this arrangement, and hill climbing in general, are the suboptimal demeanor when confronted with plateaus, local peaks and ridges. The *lookahead* technique

can alleviate these drawbacks, but only up to the depth of the lookahead. Due to the excessive cost of generating lookahead levels a depth > 1 is prohibitive. The problem is that we want to look in all directions and thus, all features have to be considered at every level.

Chapter 3

Directed Search

> "The ability to simplify means to eliminate the unnecessary so that the necessary may speak."
>
> — *Hans Hofmann (1880-1966)*

3.1 Matters of Size

As we have seen in the last chapter, it is vital for a fast traversal of the equation tree to leave out superfluous features and do intermediate feature selection. This is a problem Eqution Discovery systems in general have to deal with, since "The number of all possible equations is infinite and an infinite number of polynomial curves provide a good fit to any data set. Therefore, some kind of language bias is necessary for restricting the hypothesis space." [64]

In *SAFI* the equation tree has a complexity in the order of $\Theta(k^{2^d-1}n^{2^d})$. To reduce its complexity, the tree is pruned with different methods. Intermediate feature selection was already mentioned in the previous chapter. An analysis of this approach to feature selection and the definition of the fitness function can be found in chapter 4. The other devices employed in this endeavor are discussed right here. They are:

- *Term Reduction*,
- *Mathematical Constraints* and
- *Domain Constraints*.

Aside from these methods the user can influence the complexity enormously by choosing a small set of operators. This implies that the user has some domain knowledge and can thus guess which operators could make sense. In addition to that the initial feature set could

include irrelevant features and/or data examples that can be removed before running the algorithm. This can be done manually or using suitable programs. This is a sub-topic of feature selection and is shortly touched upon in chapter 4.1.

3.2 Term Reduction

Term reduction tries to simplify symbolic mathematical expressions. Two problems immediately arise here. The first is that "simplified" is "probably the most indefinite term used seriously in mathematics." [23] The second is that without a canonical representation the expressions are ambiguous. This topic has long been in the focus of research, starting in the 1960s. There are already plenty of plausible solutions and thus I employ a prepackaged approach, which can be found in [50].

Terms are rewritten according to production rules until they are no longer reducible. Expressions are not represented in canonical form and the system uses a Chomsky-2 grammar. Thus the system will not correctly reduce any expression. However, it works on most cases and is quite fast.

The effect for us is that terms with the same semantics but different syntax (e.g. $x * 1/y = x/y$) and constant terms (e.g. $(x + x + x)/x = 3$) can be removed. The test applied is that a new term has to have one operator more after reduction than both of its parents together. An additional benefit is that the terms returned have a more human readable format.

Listing 3.1 presents an extract of the rules we use. Some of them were added to the original set and some others removed because either, they used operators I do not use or would lead to the elimination of relevant formulas. For example, if $x \cdot x \cdot x$ were reduced to $3 \cdot x$ it would get removed due to the fact, that it does not have more operators than both its parents ($x \cdot x$ and x) combined.

Listing 3.1: Excerpt from the term rewriting rules.

```
(x + 0   = x)
((x + x) - x = x)
(x - 0   = x)
(0 - x   = - x)
(x - x   = 0)
(- - x   = x)
(x * 1   = x)
(x * 0   = 0)
(x / 0   = undefined)
(0 / x   = 0)
(x / 1   = x)
(x / x   = 1)
(x * (y / x) = y)
```

```
((y / x) * x = y)
((x * y) / (x * z) = y / z)
(x / (x * y) = 1 / y)
(x / (y / x) = 1 / y)
((x / y) / x = y)
((x / y) / (z / y) = x / z)
((x / y) * x = (x * x) / y)
((- x) + x = 0)
((x + y) - x = y)
(log 1 = 0)
(log 0 = undefined)
(sin 0 = 0)
(log (e ^ x) = x)
```

3.3 Mathematical Constraints

Mathematical constraints allow us to reduce the feature set according to mathematical definitions. There are two different kinds of mathematical constraints: hard-coded and user-defined. While the hard-coded ones always apply due to the fact that they are part of the code, the user-defined ones can be loaded on demand and easily extended. The user-defined mathematical constraints will be discussed among with other constraints in section 3.4.

Let us first look at the *hard-coded constraints*. Two of them apply during the phase of term generation and disallow the relevant features to be constructed. For one thing, neutral elements will not be constructed. The rule here is that noncommutative operators can not introduce features with both operands being the same. The idea is that the noncommutative operators are at the same time the inverse functions to some function. This holds for $+$ and $*$ and their respective inverse functions $-$ and $/$. Since in group theory, an element a operated with its inverse yields the neutral element e, $a \circ a^{-1} = e$, the same holds for $a \circ^{-1} a = e$.

For another thing, only one term is generated for all possible combinations of the same operands and a commutative operator. For example, while $x * y$ will be constructed, $y * x$ will not. The second one will be ignored because for all commutative operators the operands get sorted alphabetically and only new terms will be used. This includes the following operators: $+$, $\cdot$, *min*, *max*, *eq?*, *and*, *or*, *not* and *xor*.

A third hard-coded constraint takes the form of a user-defined constraint, with the exception that it is always loaded. It is applied to deal with the fact that certain operators are only defined in a certain domain. The restricted operators and their respective domains are depicted in table 3.3. All constraints, including this one, are applied after term construction, but before the operator is applied to the data examples.

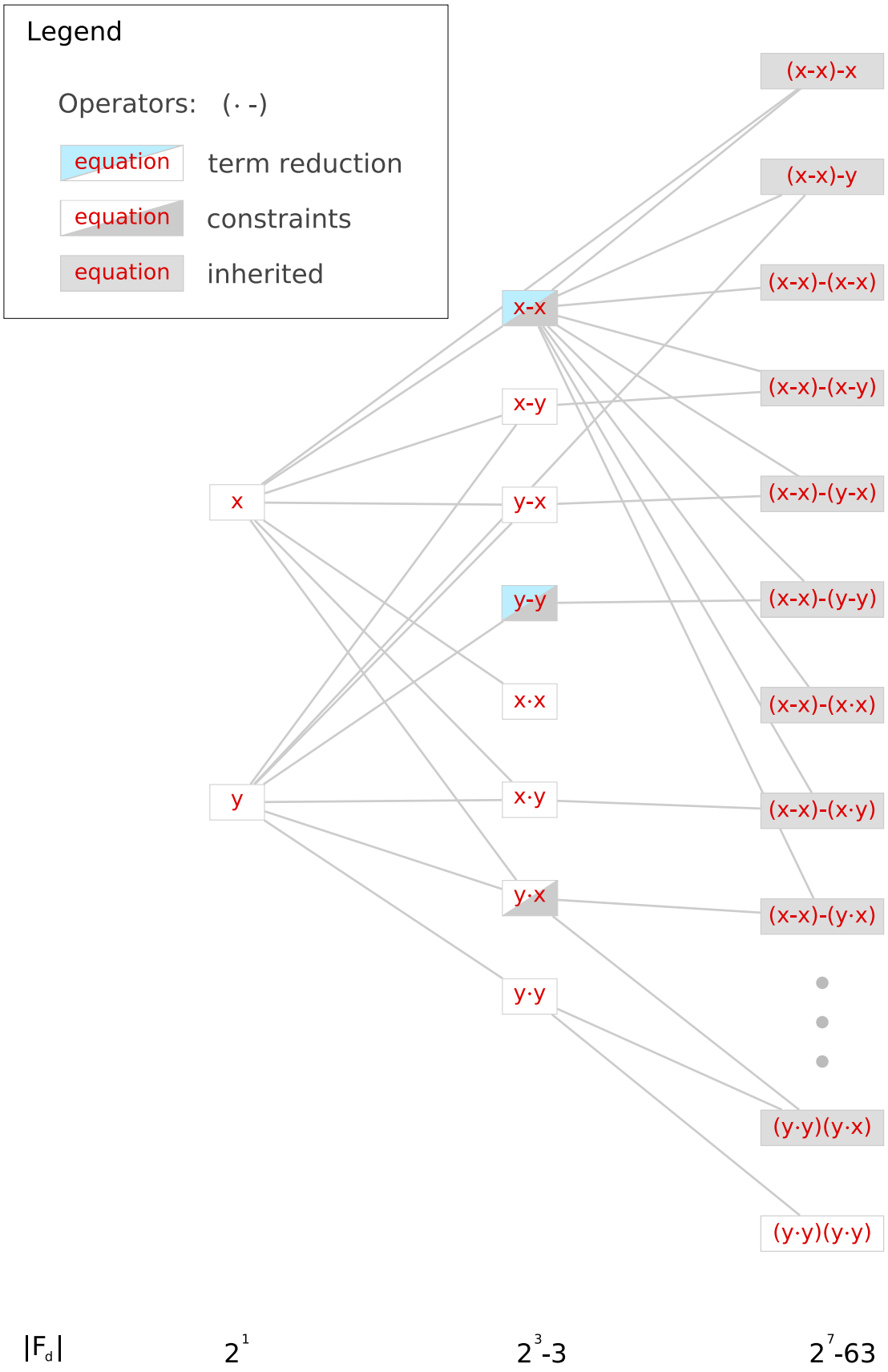


Figure 3.1: Pruned equation tree grown to depth 2.

$\sqrt{x}$, $x!$	$x \in \mathbb{R}_0^+$
x/y, $x \bmod y$	$y \in \mathbb{R} \setminus \{0\}$
$\arcsin(x)$, $\arccos(x)$	$-1 < x < 1$
x and y, x or y, x not y, x xor y	$x, y \in \{0, 1\}$
$ln(x)$	$x \in \mathbb{R}^+$

Table 3.1: Operators and their respective domains.

3.4 Domain Constraints

User-defined constraints consist of the following building blocks: properties, inheritance and predicates. The initial features can have zero or more properties. These properties consist of a tuple (N, P), with N being a unique identifier and P data of any kind. This is possible due to the dynamic type checking facilities of the implementation language.

Every constraint can define a function to handle the inheritance of these properties. All inheritance functions loaded into the system are called after feature construction, given the operator and the features the new feature was constructed from as parameters. Based on the properties that can be retrieved from the features, and the operator, the function decides what properties the new will feature will have.

The crucial part is the predicate that can be defined as a function for every constraint. These functions also get the operator and the parent features as parameters, just like the inheritance functions. They get access to the properties and the data of the features. Depending on this information, they return `true` if and only if the feature should be constructed. Otherwise, if `false` is returned, the equation will be discarded.

These constraints are checked after term generation, but before the actual data is generated. This is crucial for the mathematical constraints that are always loaded (see section 3.3), since otherwise errors, like division-by-zero, can occur during data computation. Aside from that, this behavior saves some CPU time, since the data are not computed for features that get discarded anyways.

This facility is designed for general purpose applicability. Almost any form of domain knowledge can be represented in this way. A typical example in a geometrical domain would be to include the respective axis for every feature as a property. The constraint would be that the operators + and − are only allowed for data on the same axis. Inheritance is straight forward: only if all operands are on the same axis, the result will be to.

Unit constraints can be added in the same manner, and arbitrary other ones. Another, more sophisticated example, can be found in section 6.2. A shortcoming of this approach is that it can only be used to remove terms, not to control the generation of terms per se. Since the hypothesis space is constructed extensively for the next level, this is no general limitation, it just has a negative effect on the performance if constraint systems are very restrictive.

Chapter 4

Feature Evaluation

> "It is of the highest importance in the art of detection to be able to recognize, out of a number of facts, which are incidental and which vital. Otherwise your energy and attention must be dissipated instead of being concentrated."
>
> — *Sherlock Holmes in "The Reigate Puzzle" by Sir Arthur Conan Doyle*

4.1 Feature Selection

In the previous chapter we have seen, how quite a number of terms are pruned from the equation tree by *term reduction* and *constraints*. However, this is not sufficient to guarantee a reasonable runtime in domains with many features, or where either a large number of operators, or a search depth $d > 3$, or both have to be used. As was already mentioned before, the solution to this problem is intermediate feature selection. Only some features are elected on every level to be used for further induction.

Feature selection, in contrast to the other pruning methods, has to remove features that are relevant to the target concept. The ideal solution would elect a subset that contains all the information available for classification with the least possible amount of redundancy. The closer to this goal we get, the better the overall search performance will be.

A similar problem arises for other induction and regression techniques. When learning a classifier $[f_1, \ldots, f_n] \rightarrow c$ from features $[f_1, \ldots, f_n]$ the performance of the algorithm both in the sense of runtime and goodness of the classifier can be improved by removing features that are irrelevant to classification [2]. This has long been acknowledged, and there has been much research in this direction. Therefore, I will discuss related work on this topic right here, and do not defer this to chapter 7.

Formally defined, the problem of feature selection is thus to find a subset $\{f_1, \ldots, f_i\} \subseteq \{f_1, \ldots, f_n\}$ of the initial feature set without loss of predicability. Extensive subset selection

is quite expensive since there exist 2^n subsets of n features and thus, more sophisticated search methods have to be adopted. The one common quality all subset selection algorithms share is a heuristic search approach. The solutions range from beam search [59] over genetic programming [66] [75] to the application of scatter search [28] in [39].

There is another aspect of removing irrelevant information from the input data we have not yet talked about. Aside from irrelevant features there can also be irrelevant, or, even worse, counter-productive, data examples. This includes, but is not limited to, many similar examples or examples afflicted with a big measurement error. Both these methods generally improve the performance of induction algorithms and this does not exclude *SAFI*. An example of this approach is the windowing technique optionally available in *C4.5* [54].

The authors of [7] have termed the different evaluation strategies possible "embedded approach", "wrapper model" and "filter model". Embedded in this case refers to selection techniques used during induction. For example, this is the case with partitioning strategies like the ones used in *ID3* [52], *C4.5* or *CART* [9]. Since this approach is not relevant to us, I will only discuss the other two models in the following two subsections.

4.1.1 The Wrapper Model

The term *wrapper model* was coined by John, Kohavi and Pfleger in [24]. It refers to the fact that the post-processing induction algorithm is also used to determine the fitness of a subset. The classifier is learned for every subset the selection program wants to evaluate. The estimated accuracy for the model is used as a metric to compare subsets. This approach finds a subset that increases the induction algorithms error measure by a user-defined maximal value.

Using this metric there is no need to further define relevance. In this case, it is simply to be understood as "relevant to the induction algorithm". This method has the advantage of inducing "generally smaller" [24] trees with partition algorithms over systems that define relevance in terms of intrinsic properties of the sample data and the class. In some cases, especially hard problems with many irrelevant features, the authors of [26] point out that the prediction error will also be reduced.

An example of this method is presented in [45]. Race search is used to compare subsets, thus avoiding the computational cost of fully evaluating all examples. Another specimen of this family is *Oblivion* [30]. The algorithm uses n-fold cross validation to test the accuracy of the current feature set minus one feature. The attribute that diminishes accuracy the least is pruned from the tree and the step is repeated until removal of any feature increases the prediction error.

The overhead of calling the classifier for every subset of course greatly harms the runtime [7]. This restricts the field of application to small feature languages. One interesting method in the quest to overcome this drawback was introduced in [11]. Decision trees are cached and thus partially reused, which greatly speeds up execution time. The method

seems to be restricted to *ID3* however.

4.1.2 The Filter Model

In the filter model, subset selection is done independently of the post-processing learning approach that it is deployed for. Therefore, it can be used as a pre-processor to any induction algorithm. But since, in contrast to the wrapper model, the post-processing classifier is not used to determine the fitness of a subset, the question of defining relevance arises.

The algorithms have to rely on intrinsic properties of the input data to derive a fitness measure. Finding an unambiguous, formal definition of the term "relevant" is not as straightforward as one would hope and varying definitions can be found in the literature. [47] gives an overview over the available statistical methods. A more recent paper to refer to is [13]. It adds the techniques used in machine learning to the list of known definitions.

Quite a number of algorithms use *correlation* based approaches. This technique has its roots in statistics. A typical example is *RELIEF* [25], which elects features based on the linear correlation coefficient. Many others use variations of this scheme, like [44], which suggests a variation, named "maximum information compression index".

Other authors have employed entropy related measures as a metric. The concept of entropy was introduced by Shannon in [58] as a measure of "uncertainty" or "information". Examples of its use are the "information-theoretic measure of cross-entropy" [27] or the "symmetrical uncertainty" [76]. Then, there are also multiple correlation measures, which are better at discovering non-linear relationships; in [12] a genetic search was combined with the multiple correlation coefficient as the fitness measure.

As a final example of a filter approach let us consider Cardie's work [10]. He indirectly used the embedded approach that comes with *C4.5* by including only the features that *C4.5* uses in the final subset and leaving out all the other features. In the case described in his paper, it is used for post-processing with the nearest-neighbor algorithm.

4.2 Feature Evaluation in *SAFI*

Due to the enormous amount of features constructed by *SAFI* during execution, the main criterion for the subset selection paradigm is speed. Since it is infeasible to construct all features up to a certain depth, we have to result to intermediate selection. Therefore the process of feature selection has to be embedded within the algorithm and can not be handled separately, in the sense of a post-processor.

In spite of the better prediction performance the wrapper model exhibits, I chose the filter model as a template for its superior runtime performance. In fact, we have to deal with so many features, we even have to resort to the most simple method thinkable [35]. Every

feature is evaluated individually and the best *PreSelect / PostSelect* correlated percent are selected. The advantage of this is minimal runtime, at the cost of below-average performance.

This choice brings in its wake the need for another decision. We have to evaluate which notion of relevance is the most appropriate for our purposes. However, we are confronted with the classical problem of the filter model: the fitness measure chosen does not reflect the bias of the classifier and is therefore not always appropriate. The next section will shed some light on the reasons for my final choice.

Now, that we have discussed all the elements that control the generation and removal of features, let us have a look at an equation tree grown with all these methods in place. Figure 4.2 depicts an equation tree grown to *Depth* 2 with the *Operators* $\cdot$ and $-$. There are two initial features, x and y, and the reference relation to be discovered is $x - y^2$. For the sake of clarity an example from *Equation Rediscovery* was chosen, see section 6.1 for more details.

The features removed due to term reduction and mathematical constraints are not shown. The one exception is $x - x$, which is crossed out, to indicate the presence of pruning techniques. The features highlighted in grey on the first level are the ones selected for lookahead. Every feature is depicted with the respective name, the distribution of the 25 data samples and the correlation and error compared to the reference relation.

4.3 Correlation Measures

Since *SAFI* was initially designed to deal with continuous values, defining *relevance* of a given feature towards the dependent variable in terms of the linear correlation coefficient seemed the natural choice. The linear correlation coefficient r between an independent variable x and the dependent variable y is defined as:

$$r = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2}\sqrt{\sum_i (y_i - \overline{y})^2}}$$

where $\overline{x}$ and $\overline{y}$ are the mean values and x_i and y_i elements of the x and y vector respectively.

This definition works well for linear relations and this is indirectly used as a bias in the system. Since we want *SAFI* to find terms that are linearly correlated with the dependent variable linear relations should be preferred. This is exactly the effect of using a measure that works better in cases with a linear correlation between an attribute and the target concept.

Other measures were not extensively tested. Preliminary checks indicated that error measures like the *root mean squared error* or the *mean absolute error* are inferior fitness measures. However, their use spawned the idea of using a second-level selection method. For

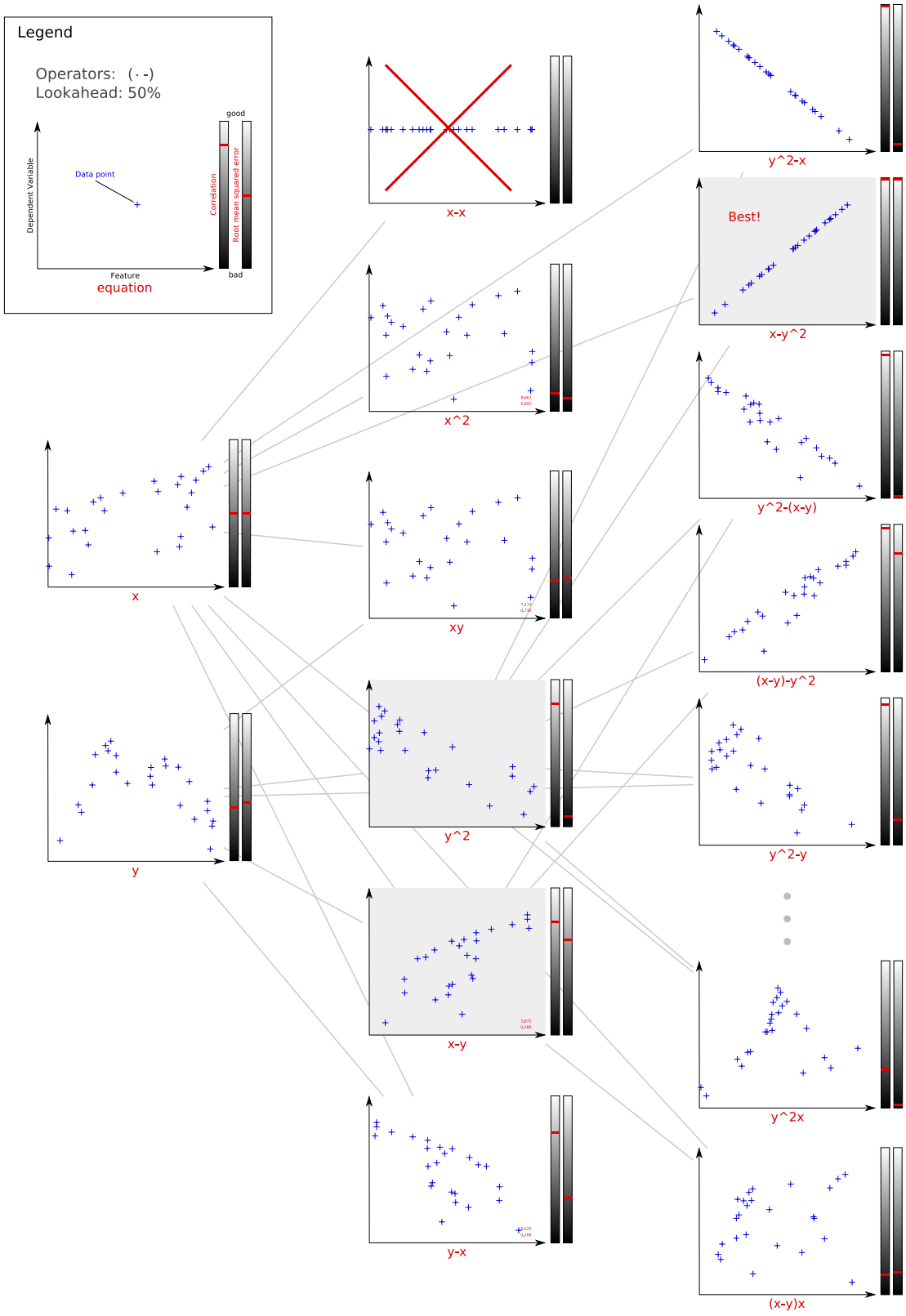


Figure 4.1: Sample equation tree with path to optimal feature.

example, two features that are similarly well correlated can be discriminated by using an error measure. This was only implemented for *Equation Rediscovery* and is explained in section 6.1.

Two variations on the linear correlation coefficient also failed to improve performance. The first does not actually change the formula used, but orders features by the relative fitness increase towards their parents. This measure neither harmed performance in the cases tested, nor improved it. The second splits the data vector into several sub-vectors of the same length and uses only the best correlated as the fitness value.

A last attempt to attain better results was made with an approach based on the ideas formulated in [10]. Instead of a decision tree, a model tree (*M5*) was chosen to do subset evaluation. Only the features included in the tree were elected for further term generation. However, aside from the runtime detriment, the outcome was not favorable on the few tests conducted.

The only modification to the correlation coefficient that would actually improve prediction accuracy in a few cases, simply adds the variance of all data points in a certain class to r. This behavior makes sense mostly for time series, with the dependent variable being the time. This is a severe restriction, but the improvement, in the few cases where it applies, is significant (see section 6.2). The rationale behind this is the fact that, at a certain point in time the system described reaches a final state. Generally, in this final state the attribute values should have the same or similar values.

One general drawback that the utilization of the linear correlation coefficient implies is the restriction to numerical domains. This can sometimes be circumvented by using a mapping $[c_1, \ldots, c_n] \rightarrow [1, \ldots, n]$ from nominal values $[c_1, \ldots, c_n]$ to integers. A better solution would be to use a two-fold measure, computing the correlation differently depending on whether the class and/or feature are nominal or real-valued. An example of this is the "heterogeneous Euclidean overlap metric", defined in [70].

Chapter 5

Algorithm Analysis

"It requires a very unusual mind to undertake the analysis of the obvious."

— *Alfred North Whitehead (1861-1947)*

5.1 Implementation Details

SAFI is implemented in *Lisp* [42] [61], one of the first higher level programming languages devised, but, at the same time, probably the most powerful one. Automatic memory management, run-time typing, macros and closures [19] [71] [56] not only ease the task of programming, but also give the language an expressiveness that makes it suitable for our problem. Aside from that, it is a classical language in the field of Artificial Intelligence and it has been applied to many typical problems. Norvig, for example, discusses a wide range of paradigms in his book [50].

Thanks to the symbolic nature of Lisp, the implementation of term reduction or feature construction relies on the parser included with Lisp and does not have to deal with this explicitly. Since Lisp itself is represented in data-structures that Lisp can deal with, it can change code at runtime and allows a seamless transition between data and code. Thus, the terms generated, like `'((* (+ x y) z))`, can be directly used to compute the data, while at the same time serving as the internal name of the feature.

Taking all the single capabilities of Lisp together paves the way for a new way of thinking about programs: bottom-up programming [18]. This paradigm implies building up the code in layers, each layer further abstracting from the prior one. I followed this programming style and thus, the system is extensible and reusable. Adding new operators comes down to adding a single line naming the operator, respectively function, to be used and its arity. Of course this name can also point to user-defined functions. New constraints are added almost as effortlessly, by simply adding new functions that adhere to a certain calling scheme.

Of course, there is also shadow, where there is so much light. Runtime typing and automatic memory management impair the execution speed of Lisp programs. The problem of loosely typed data can be alleviated by statically defining types for data in critical regions of the code. Besides, using lists exclusively for data representation also harms memory usage. This problem can also be circumvented, since Lisp offers specialized data types like arrays, which in turn can be specialized on the types they accept.

5.2 Complexity

We will now turn to the analysis of the complexity of *SAFI*. At every point in time during execution, there is at least the current set of features, including the data, in memory. The data is held in specialized arrays, while additional information is stored in a structure, which is shown in listing 5.1. In addition to that, at most the features on the next level plus the lookahead level have to be held in memory. The features on the lookahead level can be reduced by setting the value of $PreSelect \leq 1$. Thus space complexity lies in the order of $\Theta(kn^2 + PreSelect \cdot k^3n^4) = \Theta(k^3n^4)$.

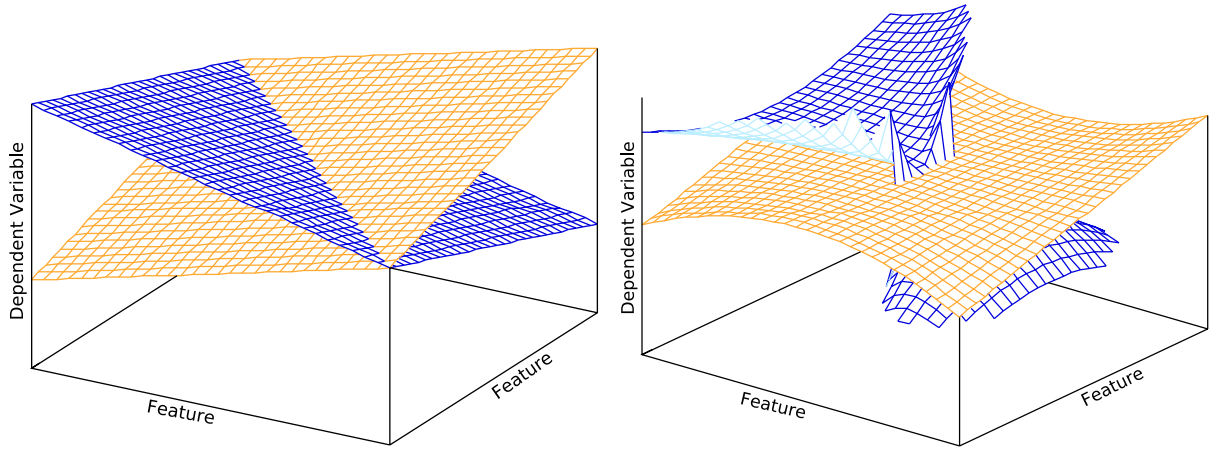
Listing 5.1: The data-structure for features

```
(defstruct (feature (:print-object feature-print))
  name
  parents
  properties
  datavec
  correlation)
```

Time complexity, on the other hand, also lies in the order of the tree complexity, but it depends on the search depth and the value of $PreSelect$ and $PostSelect$. If $PostSelect$ is set to m and $Depth$ to d, then the last run before the full depth is reached will have to generate $\Theta(k(n+dm)^2)$ features for the next level, and $\Theta(PreSelect \cdot k^3(n+dm)^4)$ for the lookahead level. In addition to term and feature generation, constraints are checked and terms are reduced. The runtime of these helpers is small compared to the rest of the algorithm.

5.3 Comparative Evaluation

Theoretically speaking the difference between induction algorithms and *SAFI* is that, while the former find a model that best fits the data given a certain representation, the latter tries to transform the representation itself. This leads to a different behavior in two fields. Due to the fact that *SAFI* only looks for a good representation, additional training examples will not increase the performance. Only as much data is needed, as is sufficient to interpolate the



(a) Partition with a model tree.

(b) Partition with *SAFI*.

Figure 5.1: Non-linear splittings in a 3-dimensional feature space using *SAFI*.

hyperplane constructed from the formula discovered. If the data are afflicted with a large measurement error, extra data may improve performance by eliminating inconsistencies.

An alteration towards the representation yielded by induction trees is the way data are split. The models generated by these systems can be regarded as partitions of the feature space. Decision trees split the data by orthogonal, linear hyper-planes and model/regression trees by linear hyper-planes, as depicted in figure 5.1(a). *SAFI* introduces non-linear hyper-planes which are used for splitting by the induction algorithms. An example of this can be seen in figure 5.1(b).

Not all induction algorithms are restricted to linear partitions of the feature space. *Neural networks*, for example, can generate arbitrary splittings of the data. However, these models can not easily be understood by humans. One major selling point of tree based algorithms is the understandable output format they produce. *SAFI* provides a transformed feature set, that reduces the problem of simple partitions without harming readability.

Chapter 6

Empirical Evaluation

"The path of precept is long, that of example short and effectual."

— *Seneca (5 BC - 65 AD)*

6.1 Equation Rediscovery

6.1.1 Idea

One way to evaluate the performance of the *SAFI* system is by means of *Equation Rediscovery*. Generally speaking, we try to rediscover a given equation from some data samples of the operands and the resulting relation. Equations are generated similar to feature generation during the search of the equation tree. However, instead of exhaustively generating all features possible, one operator is randomly chosen from a list of given operators and random operands from the list of operands, according to the arity of the operator and the maximum number of operands. This process is repeated iteratively until a given number of operators is reached.

In listing 6.1 a sample Weka "arff" file generated with the parameters set to Arguments=4, Operators=5 and Examples=10 is depicted. The reference relation to be rediscovered is the last attribute. After generating one of these files, *SAFI* is run to see if it rediscovers this equation. The only change made in the algorithm for this is that a stopping criterion was added. When a feature has a root mean squared error smaller than a certain threshold, generation is stopped and success is signaled. The threshold is needed because of numeric inaccuracies that arise during computation.

Of course this approach can not replace other tests, but it gives us the possibility to run tests in a sheltered environment. As we can control every aspect of the generation of the formulae, the system can be checked for how well it deals with, for example, complex relations or noisy data. Besides from that, we can generate an infinite number of tests

without having to go looking for new datasets in the wild. I also use it to tell how changes in the parameters or the system affect performance.

Listing 6.1: Sample "arff" file automatically generated for *Equation Rediscovery*

```
@relation 'Generated by SAFI for EquationRediscovery'

@attribute f0 numeric
@attribute f1 numeric
@attribute f2 numeric
@attribute f3 numeric
@attribute (f2/((expf0)-(cos(f1-f3)))) numeric

@data

2.8056,2.8391,2.9415,-0.4502,0.1678
2.8929,2.3277,-2.6568,1.2700,-0.1513
1.0438,1.1527,-2.7636,1.9683,-1.2827
-1.6333,0.4704,1.7430,1.8243,-87.7478
1.9929,1.5897,-1.3283,1.9445,-0.2076
2.4126,-2.2615,-0.7843,0.6520,-0.0646
2.9000,1.8223,-1.9512,2.8607,-0.1104
-0.8905,-0.5175,-2.1196,-2.2030,-4.0374
2.9110,1.5081,-2.7545,2.9032,-0.1514
0.4250,-1.9556,2.9000,-0.8238,2.6253
```

6.1.2 Results

I used *Equation Rediscovery* to evaluate the performance in the face of equations with varying complexity. Thirty random equations were generated with the operators $+$, $-$, $\cdot$ and $/$, since those are the most frequently used operators. Of course tests with other operators could easily be conducted. The maximum number of operators was set to 6, the maximum number of arguments to 7 and the number of examples to 10.

SAFI was run with *PostSelect*=4 in all cases and the same Operators used for generating the equations. The *Depth* was set according to the number of operators and the *PreSelect* was lowered with the number of operators for acceptable runtimes. Altogether rediscovery was attempted $\sum_{i=1}^{6} 30 \cdot (i+1) = 30 \cdot \left(\sum_{i=1}^{7} i - 1\right) = 30 \cdot \left(\frac{7 \cdot 8}{2} - 1\right) = 810$ times. The overall process took about three days on an AT system with 2400 MHz processor clock speed, which results in a mean runtime of five minutes per call. The results considering runtime and percentage of rediscovered equations are depicted in figure 6.1.2.

Equations with up to two operators were always discovered, equations with three operators in most cases and everything above depends a lot on the structure of the term. An example

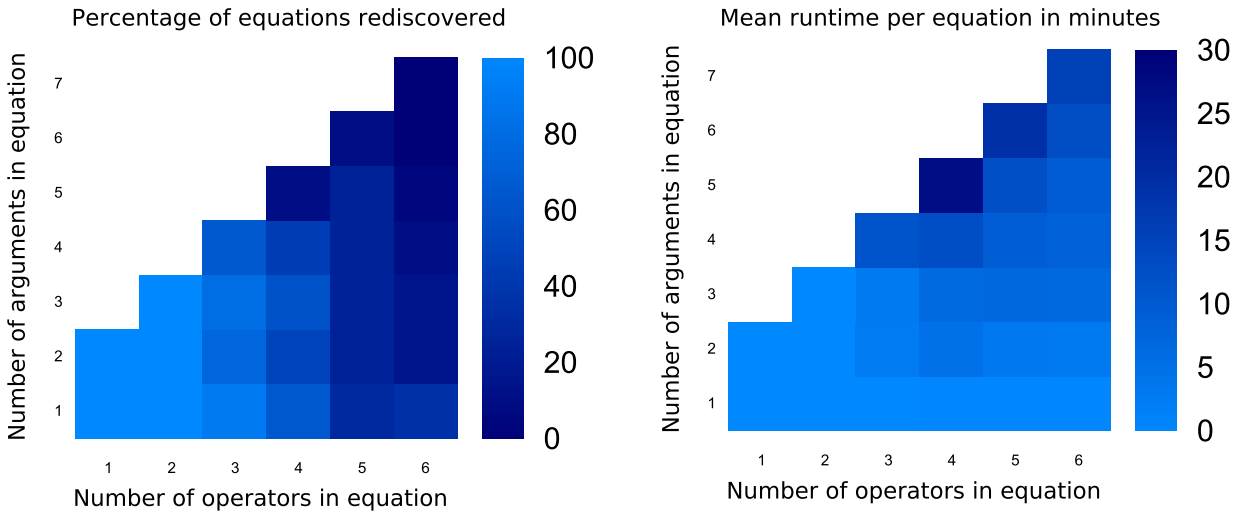


Figure 6.1: Results from Equation Rediscovery

Preselect	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Found %	45%	40%	60%	70%	60%	60%	90%	75%	80%	100%
Runtime (min)	1.0	5.9	13.3	22.9	55.2	159.2	64.9	191.5	302.9	33.8

Table 6.1: Equation Rediscovery with 3 operators, 4 arguments and varying *Preselect*

of a short equation that was not discovered is $(f0/(f1 \cdot f2)) - f3$, while a hypothetical equation that is still found is $f0 \cdot f1 + f2 \cdot f3 + f4 \cdot f5 + f6 \cdot f7 + f8 \cdot f9$. The runtime in both cases is less than a minute on an AT computer with 1500 MHz processor clock speed.

To see the effects of how the system parameters change the performance I evaluated it with varying settings for *PreSelect*. This was done with twenty random equations, all having 3 operators and 4 arguments. The results are shown in table 6.1. Since the *PostSelect* in all tests conducted is always set to 4 and other settings have never improved the performance, no further tests with this parameter were conducted.

6.2 Robot Navigation

6.2.1 Setting

In this section, we will consider a typical example from a robotic navigation task, which is depicted in figure 6.2. It was this the task that initially spawned the idea to employ automated feature design, as I already mentioned in the introduction. The context of this

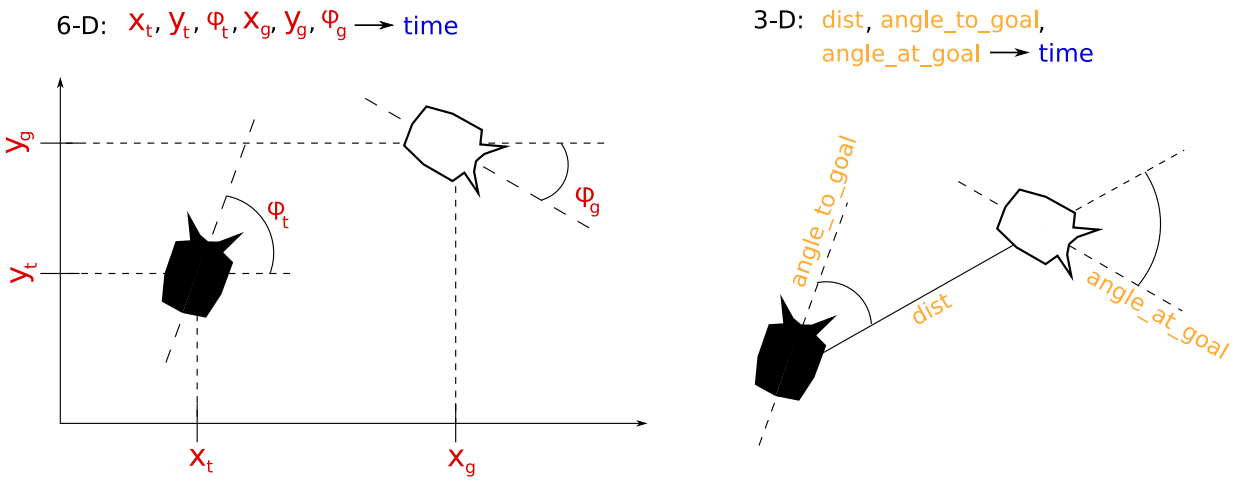


Figure 6.2: Features for a robotic navigation task

problem is further discussed in [62]. After long and tedious manual feature design the final features used for prediction of the navigation duration were found.

The six input variables describing the system depicted in figure 6.2 were reduced to three by exploiting translational and rotational invariances. The benefit is that the learning algorithm, in this case *M5* [53] [17] [68], now needs less examples to learn a more accurate model. The task of *SAFI* was to find the same, or a similar, feature language.

6.2.2 Outcome

The first of the three features from the manual set, the euclidean distance, was discovered in less than five minutes with the parameters set to:

- *Operators* $(+ \; - \; \cdot \; / \; \sqrt{})$
- *Class* 0
- *PreSelect* 0.2
- *PostSelect* 4
- *FinalSelect* 7
- *Depth* 4

Since the sample data takes the form of a time series, and there is a final state, in this case, when the goal position has been reached, it makes sense to use the modified correlation measure, as explained in 4.3. The final state is reached, when the dependent variable, the

time, is zero. Thus, the variance of every attribute in class 0 will be added to the linear correlation coefficient, the standard fitness measure. The output of such a run is shown in listing 6.2.

Listing 6.2: Output of *SAFI* for the robot navigation task

```
1 sqrt((x-xg)^2+(y-yg)^2):0.052
2 (x-xg)^2+(y-yg)^2+sqrt((x-xg)^2):0.060
3 (x-xg)^2+(y-yg)^2)-sqrt((y-yg)^2):0.081
4 (x-xg)^2+(y-yg)^2:0.061
5 (y-yg)^2+(x-xg)/phig:0.098
6 (x-xg)^2:0.220
7 (y-yg)^2:0.238
```

However, the rotational measurements were not found. Therefore geometrical constraints and operators were added to guide the search. The constraints are rather restrictive, but still *domain*, not *problem* specific. They include, for example, the fact that the *atan* function is computed from two distances, one on the x-axis, and one on the y-axis. This is still general enough to be used for other geometrical domains, but quite a restriction to the search space.

Unfortunately, this means that for complex problems domain knowledge can not be dispensed with. Having to find, define and add these constraints is a rather tedious process. The only positive side effect is that, as more and more domain constraint systems are added, they can be reused for learning in the same domain. This extends the scope, performance and speed of *SAFI*. The two rotational attributes discovered with the full constraints are shown in listing 6.3.

Listing 6.3: Output of *SAFI* for the robot navigation task with constraints

```
1 min(abs(atan(y-yg,x-xg)+phi),abs(atan(y-yg,x-xg)-phi)):0.952
2 min(abs(atan(y-yg,x-xg)+phig),abs(atan(y-yg,x-xg)-phig)):0.633
```

6.3 Additional Datasets

6.3.1 Two Spirals

In the subsequent test we use the dataset from [29], which can be generated with the short snippet of c code from listing 6.4. The program outputs points lying on two distinct spirals in the x-y-plane, each representing one of two classes. The initial dataset is plotted in figure 6.3.1. Trying to find a decision tree with *C4.5* on this initial dataset fails entirely. We simply get a tree consisting of one leaf, classifying every point as the same class and thus misclassifying every point on the other spiral.

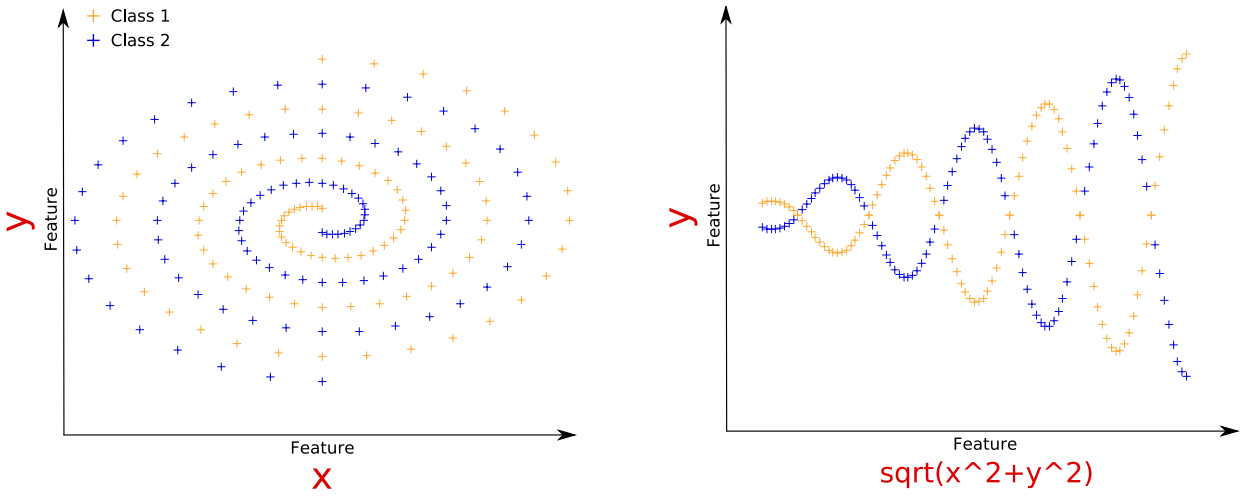


Figure 6.3: Telling two spirals apart with the use of *SAFI*

Listing 6.4: C code to generate dataset for Two Spirals example

```
void main() {
  int i;
  double x, y, angle, radius;

  for (i=0; i<=96; i++) {
    angle = i * M_PI / 16.0;
    radius = 6.5 * (104 - i) / 104.0;
    x = radius * sin(angle);
    y = radius * cos(angle);
    printf("%.5f,%.5f,%3.1f\n", x, y, 1.0);
    printf("%.5f,%.5f,%3.1f\n", -x, -y, 0.0);
  }
}
```

Trying to find a more suitable feature language with *SAFI* also fails, when directly learning better features from the given data. However, an indirect approach will lead to a better end. We first try to find an equation that yields the radius and angle of a point on one spiral from its x and y position. Indeed, this idea originated while experimenting with the code from listing 6.4. Since it was possible to find a good representation, when including the angle and radius in the set of initial features, all that lacked was a way to derive these two from the position.

Running with the set of operators ($+ - * / sqrt\ sin\ cos\ atan$), *PreSelect* 1, *PostSelect* 4, *FinalSelect* 3 and *Depth* 3 returns the output shown in listing 6.5. The first equation is a perfect fit, now we only have to find a relation for the the angle. The program returns

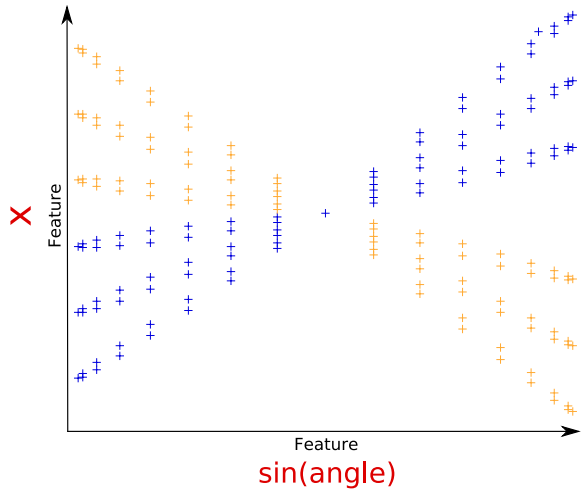


Figure 6.4: Predictive feature for the two spirals example

just the same equation again for the angle. The reason is that the angle and the radius are linearly dependent:

$$angle = -\pi \cdot radius + 20.42$$

Listing 6.5: Output of *SAFI* for the Two Spirals dataset, Part I

```
1 sqrt(x^2+y^2):0.000
2 2*(x^2+y^2):0.024
3 -(x^2+y^2):0.024
```

Utilizing this information, we can compose a new dataset that includes the respective values for radius and angle for every data point. We use this new file as the input for *SAFI* and the operators (*positive*? *xor not and or sin cos tan*). The other settings are left as they were for the prior run. The output this time is shown in listing 6.6. The first equation almost perfectly predicts the two classes, just that all points with angle equal to zero fall into one point and can not be separated anymore. To clarify the equation, see figure 6.3.1, which is a plot of the features x and $sin(angle)$.

Listing 6.6: Output of *SAFI* for the Two Spirals dataset, Part II

```
1 ((positive? x)and(positive? x))xor(positive? (sin angle)):0.072
2 (not(positive? x))xor(positive? (sin angle)):0.072
3 x:0.886
```

6.3.2 Balance Scale

As a final test, let us consider the classic "Balance Scale" dataset from [60], which was used to model psychological experimental results. There are four whole-numbered attributes: LeftWeight, LeftDistance, RightWeight and RightDistance. The scale can tip to the left, right or be balanced, depending on whether (LeftDistance · LeftWeight) is greater than, less than or equal to (RightDistance · RightWeight). The dataset has no missing values and 625 instances.

Running *C4.5* on the initial feature set generates a huge tree, consisting of 52 leaves. However, 23.36% of all cases are still misclassified. On unseen cases outside the range of the attributes in the given dataset this classifier will even perform worse. The initial feature language is simply not well suited for learning a good classifier.

For this experiment I mapped the three nominal classes to integers [Balanced, Right, Left] $\rightarrow$ $[1, 2, 3]$. The parameters of *SAFI* were set as follows:

- *Operators* $(+ - \cdot /)$
- *Class* 1
- *PreSelect* 1
- *PostSelect* 4
- *FinalSelect* 3
- *Depth* 2

Using only the term from the first line of listing 6.7 gives much better results when using *C4.5* as a classificator. The generated tree has only 3 leaves, each correctly classifying one of the three classes and thus misclassifying no cases. Obviously, this is the case because the discovered term neatly models the underlying phenomenon.

Listing 6.7: Output of *SAFI* for the Balance Scale dataset

```
1 (LeftWeight*LeftDistance)-(RightWeight*RightDistance):0.380
2 LeftDistance:2.694
3 LeftWeight:2.694
```

Instead of using class 1, which in this case seems to be a rather random choice, we can also set *PostSelect* to 8. However, this severely increases the hypothesis space and thus also runtime. Running *SAFI* with *Depth* set to 4, reveals another possible representation of the problem: (LeftWeight $*$ LeftDistance)/(LeftWeight $*$ LeftDistance + RightWeight $*$ RightDistance). Since the feature from the prior run also discriminated the classes perfectly this is no improved, just an equivalent, if harder to understand, language.

Chapter 7

Related Work

"I like to remember that "Quoting ... out of context ... is an art.""

— *Quote from Usenet*

7.1 Equation Discovery

We will begin our overview over related work with *Equation Discovery (ED)* systems. The basic building blocks for this method are the introduction of symbolic mathematical computation by McCarthy in his 1958 paper [41] and the first logic programming language, *PLANNER* [20], which was the first language to use procedural plans based on goals and assertions, and sported a search method with backtracking.

A very similar method to ED is *Inductive Logic Programming* (ILP) [48]. It combines the supervised learning approach of machine learning with logic programming. From given positive and negative examples, these systems try to find a logic program that proves all the positive and none of the negative examples. Similar to ED, ILP makes use of classical AI search paradigms and has to deal with a huge, or possibly even infinite search space. The last similarity with ED is the fact that domain knowledge, in the form of logic programs, is one approach to reduce search space complexity.

ED systems deal with the automated discovery of quantitive laws from empirical data [31]. As the name suggests these laws come in the form of equations. The first ED system devised was BACON [32]. It searches the hypothesis space using heuristics to detect trends and constancies in the data. Instead of generating all possible terms and then pruning the resulting tree, only certain terms are introduced based on heuristic rules.

The principle followed here is to build a model with heuristics and then do regression on the resulting equations. Algorithms like *ABACUS* [16] or *ARC* [46] also fall under this category. Another general principle is using a number of fixed models on which regression

is tried. Two examples of this approach are *KEPLER* [74] and *E** [57]. *E** makes use of statistical methods to guarantee good results in the limited domain of bivariate equations.

A third principle applicable is that of generating arbitrary models, possibly based on user-supplied operators. *EF* [77] is a representative of this class, with freely definable operators. But, like *E**, it is limited to bivariate equations. Another exponent of this class is *LAGRANGE* [14]. It can discover relations for dynamic systems, which on the one hand restricts it to the domain of time series, but on the other hand extends the field of machine discovery to non-linear dynamic systems. The hypothesis space is limited by a set of fixed operators, including differentiation.

The main problem using the paradigm of arbitrary models is the huge search space it implies. Different methods have been devised to cope with this. One of them is considering the units of variables, which is already implemented in *ABACUS*. A more aggressive approach heading in the same direction is used in *SDS* [69]. It restricts the search space by user-supplied scale-types. It also includes techniques for learning in the face of noisy data.

Probably the most flexible way to restrict the search space is implemented in the *LAGRAMGE* system [64]. It uses a context-free grammar to represent the search space. Operators can be freely used and rules to combine them. In fact, instead of pruning the search space, it defines the search space. This feature also allows to represent domain knowledge and is similar to my constraint system (see section 3.4).

It is also the closest relative to *SAFI* in other aspects. While most other *ED* systems are limited to finding a single equation and many ask to perform additional experiments with all but one variable fixed, these two systems do not have these restrictions. Differences are for example that *LAGRAMGE* tests dependencies between all terms and does regression on these equations. Since *SAFI* is designed as a pre-processor to machine learning algorithms, dependencies are only checked against the dependent variable, and regression is omitted due to the fact that the induction algorithms take care of this.

7.2 Constructive Induction

Probably the first one to come up with the idea of constructing and selecting features to find a better representation for induction was Michalski. As early as 1983 he coined the term *Constructive Induction (CI)* [43]. *AQ-17* [73], the system he uses for CI combines methods for inducing decision trees and constructing new features. There are modules for hypothesis-driven induction, data-driven induction [4] and knowledge-driven induction. The system combines all these methods to "Multistrategy Constructive Induction" [6]. All together is seems to be the closest living relative of *SAFI*, however feature construction is limited to a single level.

Volume 13, Issue 2 of IEEE Intelligent Systems is devoted to subset selection and feature transformation. The introduction [36] gives an overview over previous work and the follow-

ing articles. We will briefly review the ones that are relevant to my work. Another more depth introduction from the same authors is the book [37]. An article [5] by Michalski is also included, which covers the material mentioned before.

In [78] a system for feature transformation is introduced, but it is restricted to a single level of construction only, has no subset selection facilities and only works for nominal attributes. Function decomposition, a method from circuit design, is employed for constructing features. This has the disadvantage, that the features returned are not easily understandable to a human reader.

The system described in [33] first constructs and then prunes features. The approach is also limited to domains with nominal features. In fact, it is even limited to binary attributes, however a rewrite rule for general nominal features is proposed. The last article in the issue that is dedicated to feature transformation employs a genetic programming approach [65]. There is no discussion of how and if new operators can be added, and domain knowledge can not be represented at all. There are two modules, one for feature selection and one for feature construction, which are applied depending on heuristic rules.

Finally, there is some other work in this area worth mentioning. Another genetic programming algorithm that is closely related to my work, since it is also used as a pre-processor, in this case specifically for *C4.5* is [15]. In [38] an embedded feature transforming system with neural networks for decision trees is presented. A last genetic programming approach I present is discussed in [3]. *EMMA* does not only search the space of possible terms, but also the one of finite state automata to induce classification rules with transformed features.

GALA [21] is a system designed for use with any induction algorithm, unlike most other CI systems, but similar to *SAFI*. It is restricted to a fixed set of boolean operators. However, it can be applied to nominal and real-valued attributes with restrictions. No attempt to restrain the search space was made, and there is no possibility to include domain knowledge.

SAFI [63] combines a modern approach to *ED*, including freely definable operators, techniques to represent domain knowledge and advanced pruning, sporting term reduction and mathematical constraints, with the ideas of *CI*. Another unique characteristic among *CI* systems is the capability to deal with numerical data. In contrast to most other systems it can be used as a pre-processor to any induction algorithm and outputs a human readable feature language.

Chapter 8

Conclusion

"Learning without thought is labor lost; thought without learning is perilous."

— *Confucius (551 BC - 479 BC), from The Analects*

The need for automated feature design was our point of origin and the path that followed, explained my approach to satisfy it. We have seen how this method works and some examples of its application to hypothetical and real-world examples. It need not be mentioned that much more experimentation, in the sense of testing as well as expanding and changing the system, could be invested. However, these first results indicate that my approach can guide and sustain the design of feature languages. Nonetheless, human provided domain knowledge can not be replaced when constructing sensible features for complex real-world tasks.

Domains where future work could lead to interesting results are, among others, further experimentation with different fitness measures, subset selection techniques, integration with other learning tools and testing and improvement of the methods for representing domain knowledge. Aside from that, a comparative review of existing systems could clarify differences and advantages. This might lead to new ideas and possibly a system that combines the best of all the existing approaches.

Starting with the fitness measures, I will give a short hint as to what directions of future investigation might be fruitful. There is a wide variety of fitness measures available that have not been tested, and possibly there is room for improvement here. *Equation Rediscovery* could be used for evaluating different techniques. Furthermore, features could be evaluated by first selecting the best according to one correlation measure and then doing external selection with the wrapper approach to subset selection. In addition to that, similarly well correlated features could be discriminated by "Occam's razor" [8], thus selecting the simplest ones - simple in this case would probably mean a small number of operators.

Such a limited wrapper model could also be used to derive a stopping criterion. When new features do not improve the overall induction performance any more, discovery is halted.

However, this raises the classical hill-climbing problems, as discussed in section 2.3. New correlation measures could also extend the scope of the program to nominal attributes, as was mentioned before (section 4.3).

The integration with other learning tools would include the possibility to do automated feature subset selection on the initial input set and the final set. These measures could improve the performance and usability of the system. Finally, the current method used for representing domain knowledge could be further examined concerning its properties and fields of application. If need be, it can then be improved or replaced. One extension that certainly makes sense in domains with very restrictive generation rules is the possibility to control the generation of features directly, not just prune the fully expanded terms.

The first sentence of this thesis defined *Artificial Intelligence* as intelligence exhibited by a machine, or, for that matter, an algorithm. In absence of another reference, intelligence itself is of course defined in respect to the human intellect. In analogy to this natural form of intelligence, we can see how pre-processing would be an important task. The human approach to solving problems involves encoding the evidence presented to us in a way that is meaningful to us and refining this knowledge.

Another aspect of our intelligence is that we learn as we go. In machine learning this is termed *online learning* and it is an aspect that could be further investigated for *SAFI*. There are already promising results in the field of feature subset selection. The authors of [34] state, that "with sufficient training data the on-line process reaches the same performance as a scheme that has a complete access to the entire training data."

Other aspects of human intelligence and learning are harder to grip and little agreement has been reached on the nature of these concepts. Only as the semantics of these terms are formally defined, they can also be modeled. To this end it is lastly advances in the fields of biology, philosophy and psychology that lay the foundations for advances in *Artificial Intelligence*. However, not everyone agrees with this: "Of course it is very interesting to know how humans can learn. However, this is not necessarily the best way for creating an artificial learning machine. It has been noted that the study of birds flying was not very useful for constructing the airplane." [67]

Bibliography

[1] D. W. Aha. Incremental constructive induction: An instance-based approach. In *Proceedings of the 8th International Conference on Machine Learning*, pages 117–121, 1991.

[2] D. W. Aha and R. L. Bankert. Feature selection for case-based classification of cloud types: An empirical comparison. In *Proceedings of the AAAI Workshop on Case-Based Reasoning*, pages 106–112, 1994.

[3] K. Benson. Evolving automatic target detection algorithms. In *Graduate Student Workshop*, pages 249–252, 2000.

[4] E. Bloedorn and R. S. Michalski. The aq17-dci system for data-driven constructive induction and its application to the analysis of world economics. In *Proceedings of the 9th International Symposium on Methodologies for Intelligent Systems*, pages 108–117, 1996.

[5] E. Bloedorn and R. S. Michalski. Data-driven constructive induction. *IEEE Intelligent Systems*, 13(2):30–37, 1998.

[6] E. Bloedorn, J. Wnek, and R. Michalski. Multistrategy constructive induction. In *Proceedings of the 2nd International Workshop on Machine Learning*, pages 188–203, 1993.

[7] A. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997.

[8] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's razor. *Information Processing Letters*, 24(6):377–380, 1987.

[9] B. Breiman and L. Breiman. *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.

[10] C. Cardie. Using decision trees to improve case–based learning. In *Proceedings of the 10th International Conference on Machine Learning*, pages 25–32, 1993.

[11] R. Caruana and D. Freitag. Greedy attribute selection. In *Proceedings of the 11th International Conference on Machine Learning*, pages 28–36, 1994.

[12] N. Chaikla and Y. Qi. Genetic algorithms in feature selection. In *Proceedings of IEEE Systems, Man, and Cybernetics*, pages 538–540, 1999.

[13] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(3):131–156, 1997.

[14] S. Dzeroski and L. Todorovski. Discovering dynamics. In *Proceedings of the 19th International Conference on Machine Learning*, pages 97–103, 1993.

[15] A. Ekárt and A. Márkus. Using genetic programming and decision trees for generating structural descriptions of four bar mechanisms. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 17(3):205–220, 2003.

[16] B. Falkenhainer and R. S. Michalski. Integrating quantitative and qualitative discovery: The abacus system. *Machine Learning*, 1(4):367–40, 1986.

[17] E. Frank, Y. Wang, S. Inglis, G. Holmes, and I. H. Witten. Using model trees for classification. *Machine Learning*, 32(1):63–76, 1998.

[18] P. Graham. *On LISP: Advanced Techniques for Common LISP*. Prentice Hall, 1993.

[19] P. Graham. *ANSI Common Lisp*. Prentice Hall, 1996.

[20] C. Hewitt. Planner: A language for proving theorems in robots. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pages 295–302, 1969.

[21] Y.-J. Hu and D. Kibler. A wrapper approach for constructive induction. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 806–811, 1996.

[22] D. J. Hustings. *Comprehension and Data Analysis Questions in Advanced Physics*, page 84. John Murray Press, 1975.

[23] G. James and R. C. James. *Mathematics Dictionary*. Van Nostrand, 1949.

[24] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the 11th International Conference on Machine Learning*, pages 121–129, 1994.

[25] K. Kira and L. A. Rendell. A practical approach to feature selection. In *Proceedings of the 9th International Workshop on Machine Learning*, pages 249–256, 1992.

[26] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[27] D. Koller and M. Sahami. Toward optimal feature selection. In *Proceedings of the 13th International Conference on Machine Learning*, pages 284–292, 1996.

[28] M. Laguna and R.Marti. *Scatter Search: Methodology and Implementations in C*. Kluwer Academic Publishers, 2003.

[29] K. J. Lang and M. J. Witbrock. Learning to tell two spirals apart. In *Connectionist Models Summer School*, 1988.

[30] P. Langley and S. Sage. Oblivious decision trees and abstract cases. In *Working Notes of the AAAI Workshop on Case-Based Reasoning*, 1994.

[31] P. Langley, H. Simon, G. Bradshaw, and J. Zytkow. *Scientific Discovery: Computational Explorations of the Creative Processes*. MIT Press, 1987.

[32] P. Langley, H. A. Simon, and G. L. Bradshaw. Heuristics for empirical discovery. Technical Report CMU-RI-TR-84-14, Robotics Institute, Carnegie Mellon University, June 1984.

[33] N. Lavrac, D. Gamberger, and P. D. Turney. A relevancy filter for constructive induction. *IEEE Intelligent Systems*, 13(2):50–56, 1998.

[34] D. Levi and S. Ullman. Learning to classify by ongoing feature selection. In *Proceedings of the 3rd Canadian Conference on Computer and Robot Vision*, page 1. IEEE Computer Society, 2006.

[35] D. D. Lewis. Feature selection and feature extraction for text categorization. In *Proceedings of the workshop on Speech and Natural Language*, pages 212–217. Association for Computational Linguistics, 1992.

[36] H. Liu and H. Motoda. Introduction: Feature transformation and subset selection. *IEEE Intelligent Systems*, 13(2):26–28.

[37] H. Liu and H. Motoda. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publisher, 1998.

[38] H. Liu and R. Setiono. Feature transformation and multivariate decision tree induction. In *Proceedings of The 1st International Conference on Discovery Science*, pages 279–290, 1998.

[39] F. López, M. Torres, B. Batista, J. Pérez, and J. Moreno-Vega. Solving feature subset selection problem by a parallel scatter search. *European Journal of Operational Research*, 169:477–489, 2006.

[40] C. J. Matheus. The need for constructive induction. In *Proceedings of the 8th International Conference on Machine Learning*, pages 173–177, 1991.

[41] J. L. McCarthy. Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91. Her Majesty's Stationary Office, 1959.

[42] J. L. McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Communications of the ACM*, 3(4):184–195, 1960.

[43] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20(2):111–161, 1983.

[44] P. Mitra, C. A. Murthy, and S. K. Pal. Unsupervised feature selection using feature similarity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):301–312, 2002.

[45] A. Moore and M. Lee. Efficient algorithms for minimizing cross validation error. In *Proceedings of the 11th International Conference on Machine Learning*, pages 190–198, 1994.

[46] M. Moulet. Iterative model construction with regression. In *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 448–452, 1994.

[47] A. N. Mucciardi and E. E. Gose. A comparison of seven techniques for choosing subsets of pattern recognition properties. *IEEE Transaction on Computers*, 20(9):1023–1031, 1971.

[48] S. Muggleton. Inductive logic programming. In *Proceedings of the 1st International Workshop on Algorithmic Learning Theory*, pages 42–62, 1990.

[49] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases, 1998.

[50] P. Norvig. *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kaufmann, 1992.

[51] J. R. Quinlan. Learning efficient classification procedures and their application to chess end games. In *Machine Learning: An Artificial Intelligence Approach*, pages 463–482. Morgan Kaufmann, 1983.

[52] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[53] J. R. Quinlan. Learning with Continuous Classes. In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.

[54] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[55] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[56] R. Sangal. *Programming paradigms in LISP*. McGraw-Hill, 1991.

[57] C. Schaffer. A proven domain-independent scientific function-finding algorithm. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 828–833, 1990.

[58] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 2:379–423, 623–656, 1948.

[59] W. Siedlecki and J. Sklansky. On automatic feature selection. In *Internatiol Joint Conference on Pattern Recognition in Artificial Intelligence*, pages 197–200, 1988.

[60] R. S. Siegler. Three aspects of cognitive development. *Cognitive Psychology*, 8:481–520, 1976.

[61] G. L. Steele. *Common Lisp: The Language*. Digital Pres, 1990.

[62] F. Stulp and M. Beetz. Optimized execution of action chains using learned performance models of abstract actions. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 2005.

[63] F. Stulp, M. Pflüger, and M. Beetz. Feature space generation using equation discovery. In *Proceedings of the 29th German Conference on Artificial Intelligence*, 2006.

[64] L. Todorovski. Declarative bias in equation discovery. Master's thesis, University of Ljubljana, 1998.

[65] H. Vafaie and K. DeJong. Feature space transformation using genetic algorithms. *IEEE Intelligent Systems*, 13(2):57–65, 1998.

[66] H. Vafaie and K. D. Jong. Genetic algorithms as a tool for feature selection in machine learning. In *Proceeding of the 4th International Conference on Tools with Artificial Intelligence*, pages 200–204, 1992.

[67] V. N. Vapnik. *The Nature of Statistical Learning Theory*, page 13. Springer, 2000.

[68] Y. Wang and I. Witten. Inducing model trees for continuous classes. In *Poster Papers of the 9th European Conference on Machine Learning*, pages 128–137, 1997.

[69] T. Washio, H. Motoda, and N. Yuji. Discovering admissible model equations from observed data based on scale-types and identity constraints. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 772–779, 1999.

[70] D. R. Wilson and T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[71] P. H. Winston and B. K. P. Horn. *LISP*. Addison-Wesley, 1989.

[72] I. H. Witten and E. Frank. *Data Mining*. Morgan Kaufmann, 2005.

[73] J. Wnek and R. S. Michalski. Hypothesis-driven constructive induction in aq17-hci: A method and experiments. *Machine Learning*, 14(1):139–168, 1994.

[74] Y.-H. Wu and S. Wang. Discovering functional relationships from observational data. In *Knowledge Discovery in Databases*, pages 55–70. 1991.

[75] J. Yang and V. G. Honavar. Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems*, 13(2):44–49, 1998.

[76] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th International Conference on Machine Learning*, pages 856–863, 2003.

[77] R. Zembowicz and J. M. Zytkow. Discovery of equations: Experimental evaluation of convergence. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 70–75, 1992.

[78] B. Zupan, M. Bohanec, J. Demsar, and I. Bratko. Feature transformation by function decomposition. *IEEE Intelligent Systems*, 13(2):38–43, 1998.