Algorithmic Learning: Formal models and prototypical applications

Steffen Lange
Fachhochschule Darmstadt, Fachbereich Informatik
Haardtring 100, D–64295 Darmstadt
s.lange@fbi.fh-darmstadt.de

Sandra Zilles
DFKI GmbH, Forschungsbereich Wissensmanagement
Erwin-Schrödinger-Str. 57, D–67663 Kaiserslautern
zilles@dfki.de

Abstract: The vision of assistance systems is to use machines not merely as tools but as intelligent assistants with which humans can solve their tasks cooperatively. The approach considered below is to enhance the required machine-human interaction by machine learning strategies. In order to enable an analytical study of the relevant general learning tasks and techniques, different formal learning models are compared and related to one another. The capabilities of the corresponding learning algorithms are illustrated and discussed within the framework of prototypical applications.

1 Introduction

The vision of machine tools used by humans for solving hard or time-consuming tasks has enhanced research in computer science from the first steps on. Yet, for many problems in several application domains, computer tools are far from being sufficient. There are tasks which, on the one hand, require expert knowledge in specific situations, but, on the other hand, involve subtasks which are either too complex or too time-consuming even for a human domain expert to solve, thus entailing the demand for something more than a machine tool, namely rather an intelligent assistant. Such an assistant could *interact* with the human expert in a bilateral way: the expert provides the assistant system with the required domain knowledge, while the assistant solves all relevant subtasks and proposes solutions for the general problem to the expert user. The idea behind that is to maintain the autonomy of the machine in proposing solutions, but also to allow the user to incorporate specific constraints on the solutions, resulting from particular needs and objectives. Thus interactivity constitutes an approach to adaptivity. The final objective is that humans and machines interactively solve tasks together – either side utilizing its characteristic potential.

Obviously, in such scenarios the ability of the machine to learn from the expert is essential, similarly to scenarios in which humans solve problems together. This is maybe most purely reflected in typical didactic principles: successful learning very often requires an interaction between a teacher and a student, thus provoking both solving a problem together rather than the student solving the problem completely on her own. Hence machine learning approaches are inherent in the fundamental idea of machine assistants. Consequently, the ability to learn has to be modeled formally in order to enable its implementation.

Unfortunately, there is no formal model that spans all phenomena crucial for human learning. Different models analyzed in algorithmic learning theory focus different aspects of learning, determined by a specification of several parameters, see [Gol67, Ang88, Val84]. Firstly, the interaction between the teacher and the learner (i. e., the student) must be formalized. Here we differentiate between passive and active learners, between teachers providing only examples and teachers providing answers to questions posed by the learner. Secondly, constraints on the learner itself must be specified. This concerns the kind of representations the learner may use to formulate (questions and) hypotheses as well as the algorithmic methods that may be used to implement the learner. Thirdly, the criterion of success in learning has to be defined. From a practical point of view, this reflects the user's quality requirements, e. g., the minimum level of accuracy the user would be content with.

However, no matter how these parameters are chosen, there is no way to define a universal learner which can cope with any conceivable learning problem in any conceivable learning domain. That means in particular, that the parameters described above have to be specified in dependence of the individual learning problem. Often, given such an individual learning task, a first approach might be to check whether this learning task is solvable at all, and – in case this can be answered to the positive – whether it can be solved efficiently.

Although below we consider formal methods for determining whether a certain learning problem can be solved at all, these methods in general do not allow for any conclusions concerning the efficiency of the respective learners. However, the formal models considered are fundamental for generally approaching a learning domain in an analytical way. Combining this with additional specific knowledge about the learning domain and the structure of the concepts subject to learning may result in practically oriented solutions to learning problems which may be applied prototypically for special tasks.

To illustrate these ideas, we start by proposing two prototypical applications in which machine learning problems can be identified and mapped to the problem of learning some special kind of formal concepts, namely the so-called pattern languages [Ang80a]. In order to formalize learning and typical interaction scenarios, we differentiate between two approaches: learning from examples and learning from queries. Here the pattern languages serve for illustrating the basic ideas and limitations of learnability. In addition, the two abstract approaches are formally compared showing quite astonishing interrelations. Basing on this formal analysis, we refine our initial methods for addressing the prototypical applications proposed, thus improving the capabilities of the learners considered so far. Finally, we resume and discuss alternative approaches for improvements of learning capability.

2 Prototypical applications – Basics

In this section, we present two prototypical application scenarios which illustrate how machine learning approaches can enhance the development of intelligent assistants. A first rather simple example, see Subsection 2.1, demonstrates how such approaches may help in situations a human expert might handle without any machine assistance, yet only at the expense of very much time. Here machine intelligence is deployed to perform exactly those steps that are rather tedious and susceptible to errors. The second example, see Subsection 2.2, points to a problem that, most probably, even an expert in the corresponding application area cannot adequately solve without any appropriate machine assistance. Here machine intelligence is used to assist the human expert in finding appropriate models for the application area. Both scenarios have in common that the solutions proposed by the machine are hypothetical in nature. In typical human-machine interactions, these hypothetical solutions are therefore evaluated and – depending on the outcome of the evaluation – modified in further steps of interaction.

2.1 Data migration

A rather typical problem which frequently occurs in different application scenarios is the so-called data migration problem. This problem arises, when different applications use different data formats to represent their data. Before a particular application A can process user data originating from another application B, the format of the data has to be transformed into a format the application A can handle.

For illustration, consider the following simple but prototypical example. Assume that there is a collection of data entries providing information about the results of the German soccer league "Regionalliga Nord". Each data entry provides information about a particular soccer game. The following two data entries should illustrate the data format used so far.

Year:	2004/05	Year:	2004/05
Round:	2	Round:	2
Home:	VfB Lübeck	Home:	1. FC Union Berlin
Guest:	Preußen Münster	Guest:	Wuppertaler SV
Result:	3 : 0	Result:	4 : 2

To make the content of this data collection available via the World Wide Web, every data entry has to be transformed into a data format that can be visualized by standard web browsers. Assume for simplicity that every data entry may be considered as one row in an HTML-table. Consequently, the given data entries have to be rewritten in the following new data format.

```
<tr>                             <tr>
<td>2004/05</ td>                <td>2004/05</ td>
<td>2 </td>                      <td>2 </td>
<td>VfB Lübeck </td>             <td>1. FC Union Berlin </td>
<td>Preußen Münster </td>        <td>Wuppertaler SV </td>
<td>3 : 0</td>                   <td>4 : 2</td>
</tr>                            </tr>
```

The data migration problem in our simple example consists of transforming data entries of the old data format into the corresponding entries of the new data format.

A rather conventional approach to solve such problems is to write a transformation program that takes as input data entries of the old data format and outputs data entries of the new data format. Clearly, this approach requires some programming experience and is, in general, rather tedious and susceptible to errors. Moreover, each particular data migration problem requires a specific transformation program.

Another more adequate solution – which incorporates concepts and algorithmic ideas common in the area of algorithmic learning – is based on the following approach.

In a first step, the syntactic structure of both the data entries in the old data format and the corresponding entries in the new data format is analyzed. In our example, the following patterns p and q form appropriate representations for all data entries in the given collection.

$$p = \textit{Year} : x_0\ \textit{Round} : x_1\ \textit{Home} : x_2\ \textit{Guest} : x_3\ \textit{Result} : x_4$$
$$q = \textit{<tr><td>}\ x_0\ \textit{</td><td>}\ x_1\ \textit{</td><td>}\ x_2\ \textit{</td><td>}\ x_3\ \textit{</td><td>}\ x_4\ \textit{</td></tr>}$$

The concrete data entries can be obtained by substituting appropriate strings for the variables in the corresponding patterns. For instance, replacing x_0 by *2004/05*, x_1 by *2*, x_2 by *1. FC Union Berlin*, x_3 by *Wuppertaler SV*, and x_4 by *4 : 2* in the patterns p and q yields the second data entry in the old and the new data format, respectively.

In a second step, a universal transformation program P can be applied. Given a data entry s in the old format as well as patterns p and q representing the common syntactic structure of all data entries in the old and the new data format, respectively, the transformation program may proceed as follows. First, the given data entry s has to be parsed with respect to the pattern p. As a result, a substitution is found that can be employed to generate s from p. Second, the required data entry in the new data format can be created by applying the substitution found to replace the variables in the pattern q accordingly.

By means of the universal transformation program P, every data migration problem can be solved provided the required patterns are known.[1] Again, doing the relevant analysis by hand is rather tedious. This is the point at which machine intelligence may help. There are a couple of powerful learning algorithms available that can be used to synthesize the required patterns automatically. These learning algorithms take as input just a couple of data entries in the old as well in the new data format, see, e. g., [Nix83]. A detailed experimental evaluation shows that, in real world applications, three data entries in the old data format as well as one data entry in the new data format will, in average, provide enough information to synthesize the required patterns successfully, see [Nix83].

2.2 Protein classification

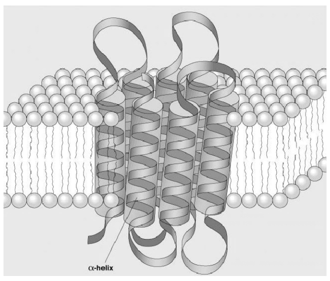
Next we discuss another application scenario which again illustrates the potential benefit from incorporating ideas of algorithmic learning theory. The problem addressed belongs to the most challenging tasks in bio-chemistry, namely the analysis of amino acid sequences.

[1]Obviously, this approach outperforms the conventional one, since, in the latter, each data migration problem requires a specific transformation program.

Amino acid sequences form the primary structure of proteins – the building blocks of all known kinds of life. Different proteins have different primary structures. An empirically approved fact is that there exists a strong relation between the primary structure of proteins and their functionality. Consequently, similarities in the primary structure of different proteins may result in similar functionalities.

Membrane proteins belong to a class of proteins that are of particular interest and importance: they control the transport of molecules through the cell membrane. Various hereditary, virus, and tumor diseases are caused and accompanied by serious irregularities in the transport of molecules through the cell membrane. Thus, for diagnostic and pharmaceutical purposes, it would be very helpful to find a rule for distinguishing membrane from non-membrane proteins, given just their primary structure.

Experiments have shown that the amino acid sequences of membrane proteins contain characteristic regions, the so-called transmembrane domains, which are rather unlikely to appear in non-membrane proteins. The following picture illustrates how the transmembrane domains of a membrane protein are incorporated into the cell membrane.



For the classification of proteins into membrane and non-membrane proteins, it seems to be advantageous to analyze the syntactic structure of the transmembrane domains. In [SA95], this approach has been applied successfully, showing that patterns are suitable for describing the syntactic relations in the primary structure of transmembrane domains.

There are exactly 20 different amino acids that may appear in any protein. Usually, each amino acid is encoded by one capital Latin letter. Thus, the primary structure of a protein can be uniquely encoded as a finite string over an alphabet of 20 terminal symbols.

The following figure displays the primary structure of a particular membrane protein, see [SA95], with the known transmembrane domains printed boldly.

M D V V N Q L V A G G Q F R V V K E **P L G F V K V L Q W V F A I F A F A T C G S** Y
T G E L R L S V E C A N K T E S A L N I E V E F E Y P F R L H Q V Y F D A P S C V K G
G T T K I F L V G D Y S S S A E **F F V T V A V F A F L Y S M G A L A T Y I F L** Q N K
Y R E N N K **G P M M D F L A T A V F A F M W L V S S S A W A** K G L S D V K M A T
D P E N I I K E M P M C R Q T G N T C K E L R D P V T S **G L N T S V V F G F L N L V L**
W V G N L W F V F K E T G W A A P F M R A P P G A P E K Q P A P G D A Y G D A G
Y G Q G P G G Y G P Q D S Y G P Q G G Y Q P D Y G Q P A S G G G G Y G P Q G D Y G
Q Q G Y G Q Q G A P T S F S N Q M

There are a couple of protein databases available – even via the world wide web, see, e. g., `http://pir.georgetown.edu` – which contain a rich pool of examples for membrane and non-membrane proteins. Since the transmembrane domains of the known membrane proteins are indicated, these data can be used to generate positive and negative training examples for learning, both being substrings of the primary structures. Positive examples are substrings that contain known transmembrane domains, while negative examples are substrings that do not contain domains of that kind.

In [SA95], a particular learning algorithm has been applied in order to find a collection of patterns that properly describes the regularities in the positive examples and that allows for discriminating the positive examples from the negative ones. For this purpose, a small percentage of the positive and negative examples from the overall set of examples (less than 10 %) have been selected as training examples for the learning algorithm. The remaining examples have been used as a test set for a subsequent evaluation. In the learning phase, the following collection of five patterns has been generated.

$$p_1 = x_0 I x_1 A L x_2 \quad p_2 = x_0 I x_1 L G x_2 \quad p_3 = x_0 I F x_1 L x_2$$
$$p_4 = x_0 I L x_1 V x_2 \quad p_5 = x_0 L L x_1 L x_2$$

This collection of patterns can be used as follows to predict whether or not a given protein serves as a membrane protein. If the primary structure of the protein can be generated by at least one of these patterns, the protein will be classified as a membrane protein, while, otherwise, it will be classified as a non-membrane protein. As the experiments in [SA95] document, this rather simple rule has a quite high accuracy: the classifier defined by the patterns p_1 to p_5 predicts about 70.6 % of the known membrane proteins and about 67.4 % of the known non-membrane proteins in the test set correctly.

3 Pattern languages

The learning algorithms deployed in the prototypical applications discussed above return patterns and collections thereof, respectively, as their hypotheses. Each pattern describes a particular formal language, such that considering a pattern as a certain kind of formal grammar, see [Cho56], suggests itself. The elements of the corresponding language are, roughly speaking, all strings which can be obtained if the variables in the pattern are substituted by strings. Surprisingly, it makes a remarkable difference whether or not variables can be replaced by the empty string.

In the following, we provide the necessary conceptual background to formally define patterns and their corresponding languages. We discuss characteristic features of the pattern languages, thereby addressing questions and problems that, at first glance, seem to be rather abstract and relevant only in the context of pure formal language theory, see, e. g., [HU69]. Nevertheless, the insights obtained turn out to be of particular importance for the subsequent learning theoretical investigations, in which patterns and the corresponding languages form the subject of learning.

3.1 Non-erasing pattern languages

Patterns and non-erasing pattern languages are defined as follows, see [Ang80a]. Let $\Sigma = \{a, b, \dots\}$ be any non-empty finite alphabet containing at least one terminal symbol.[2] Furthermore, let $X = \{x_0, x_1, x_2, \dots\}$ be an infinite set of variables such that $\Sigma \cap X = \emptyset$. *Patterns* are non-empty strings from $(\Sigma \cup X)^+$. For instance, aa, ax_1bbb, $x_1bx_2ax_1$ are patterns. If p is a pattern, then $L_{\mathrm{ne}}(p)$, the *non-erasing pattern language* generated by the pattern p, is the set of all strings that can be obtained by replacing the variables in p by non-empty strings. Thereby, every occurrence of one variable x_j in p has to be replaced by the same string s_j in Σ^+. For instance, the string $aabbaaa$ belongs to $L_{\mathrm{ne}}(x_1bx_2ax_1)$, while the string $aabbaab$ does not.

Next, we summarize some useful properties of the non-erasing pattern languages. For each pattern p, let $L_{\mathrm{ne}}^{\min}(p)$ be the set of strings in $L_{\mathrm{ne}}(p)$ which are exactly as long as p. Note that $L_{\mathrm{ne}}^{\min}(p)$ is a finite set and can be computed effectively, if p is given.

Proposition 1 ([Ang80a]) *Let Σ and X be fixed. Moreover, let p and q be patterns and let s be a string in Σ^+.*

(1) If $s \in L_{\mathrm{ne}}(p)$, then s is at least as long as p.[3]

(2) If $L_{\mathrm{ne}}^{\min}(p) \subseteq L_{\mathrm{ne}}(q)$, then $L_{\mathrm{ne}}(q) \not\subset L_{\mathrm{ne}}(p)$.

Hence, every pattern language $L_{\mathrm{ne}}(p)$ is uniquely characterized by its shortest strings. As we shall see later, this feature is particularly helpful, when non-erasing pattern languages form the subject of learning.

Next, we focus our attention on the following decision problems for the class of all non-erasing pattern languages.

The membership problem, i. e., the problem to decide – given any pattern p and any string s as input – whether or not s belongs to $L_{\mathrm{ne}}(p)$.

The equivalence problem, i. e., the problem to decide – given any two patterns p and q as input – whether or not the non-erasing pattern languages $L_{\mathrm{ne}}(p)$ and $L_{\mathrm{ne}}(q)$ are equivalent.

The inclusion problem, i. e., the problem to decide – given any two patterns p and q as input – whether or not the non-erasing pattern language $L_{\mathrm{ne}}(p)$ is a subset of $L_{\mathrm{ne}}(q)$.

We consider such a problem solvable if there is an effective algorithm solving the problem for any relevant input. The basic knowledge concerning the solvability/non-solvability of these decision problems can be summarized as follows.

[2]In what follows, Σ^+ denotes the set of all finite non-empty strings that can be built from terminal symbols in Σ, while $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$ is the set of all strings, including the empty string ε.

[3]In particular, $L_{\mathrm{ne}}^{\min}(p)$ is the set of shortest strings in $L_{\mathrm{ne}}(p)$.

Proposition 2 ([Ang80a, JSSY93]) *Let Σ and X be fixed.*

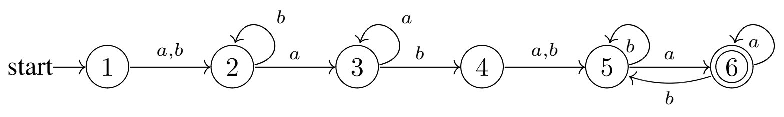
(1) The membership problem for the class of all non-erasing pattern languages is solvable. If $\mathcal{NP} \neq \mathcal{P}$, there is no efficient algorithm for this problem, since it is $\mathcal{NP}$-complete.

(2) The equivalence problem for the class of all non-erasing pattern languages is solvable.

(3) The inclusion problem for the class of all non-erasing pattern languages is not solvable.

By restricting the class of admissible patterns one can define a particular subclass of the class of non-erasing pattern languages that is computationally much easier to handle (cf. [Shi82]). Moreover, several non-trivial applications demonstrate that the expressiveness of this particular subclass is often still sufficient for practical purposes.

A pattern p is said to be a *regular* pattern, if every variable in p occurs exactly ones. For instance, x_1abx_2a constitutes a regular pattern, while $x_1bx_2ax_1$ is not a regular pattern. One easily observes, that – given a regular pattern p – the non-erasing pattern language $L_{\mathrm{ne}}(p)$ forms a regular language, i. e., there is a finite deterministic automaton that accepts the language $L_{\mathrm{ne}}(p)$. For instance, the following automaton A accepts the non-erasing pattern language $L_{\mathrm{ne}}(x_1abx_2a)$, where, for simplicity, it is assumed that $\Sigma = \{a, b\}$.



The language $L_{\mathrm{ne}}(p)$ generated by a regular pattern is said to be a *non-erasing regular pattern language*. Since the class of all non-erasing regular pattern languages forms even a proper subclass of the class of all regular languages, Proposition 2 rewrites as follows:

Proposition 3 ([Shi82]) *Let Σ and X be fixed.*

(1) The membership problem for the class of all non-erasing regular pattern languages is solvable. Moreover, there is a linear-time algorithm for this problem.

(2) The equivalence problem for the class of all non-erasing regular pattern languages is solvable.

(3) The inclusion problem for the class of all non-erasing regular pattern languages is solvable.

3.2 Erasing pattern languages

Patterns can also be used to define a larger class of languages, the so-called erasing pattern languages. In contrast to the non-erasing case, it is now allowed to replace the variables in a pattern by the empty string also, cf. [Shi82]. This seemingly minor modification has several surprising consequences. The following example may help to illustrate the advantages of this approach. Here, $\Sigma = \{A, B, \dots, Z\}$ forms the underlying alphabet.

positive examples	*negative examples*
AKEBONO MUSASHIMARU	WAKANOHANA TAKANOHANA
CONTRIBUTIONS OF AI	CONTRIBUTIONS OF THE UN
BEYOND MESSY LEARNING	TRADITIONAL APPROACHES
BASED ON LOCAL SEARCH ALGORITHMS	GENETIC ALGORITHMS
BOOLEAN CLASSIFICATION	PROBABILISTIC RULE
SYMBOLIC TRANSFORMATION	NUMERIC TRANSFORMATION
BACON SANDWICH	PLAIN OMELETTE
PUBLICATION OF DISSERTATION	TOY EXAMPLES

Though not obvious, the string $BONSAI$ is contained in every positive, but in none of the negative examples. Hence, the pattern $p = x_0Bx_1Ox_2Nx_3Sx_4Ax_5Ix_6$ can generate all positive examples provided replacing variables by the empty string is allowed, while it cannot generate any of the negative examples.

Formally, erasing pattern languages are defined as follows. Let $\Sigma = \{a, b, \dots\}$ and $X = \{x_0, x_1, x_2, \dots\}$ be defined as usual, and let p be a pattern in $(\Sigma \cup X)^+$. Then $L_{\mathrm{e}}(p)$, the *erasing pattern language* generated by the pattern p, see [Shi82], is the set of all strings that can be obtained by replacing the variables in p by strings from Σ^*. Again, every occurrence of one variable x_j in p has to be replaced by the same string s_j in Σ^*. For instance, both the string $aabbaaa$ and the string bba belong to $L_{\mathrm{e}}(x_1bx_2ax_1)$, while the string $aabbaab$ does not. Erasing and non-erasing pattern languages are closely related:

Proposition 4 *Let Σ and X be fixed. Moreover, let p be a pattern from $(\Sigma \cup X)^+$.*

(1) $L_{\mathrm{ne}}(p) \subseteq L_{\mathrm{e}}(p)$.

(2) There are patterns $p_1, \dots, p_n \in (\Sigma \cup X)^+$ such that $L_{\mathrm{ne}}(p_1) \cup \dots \cup L_{\mathrm{ne}}(p_n) = L_{\mathrm{e}}(p)$.

So every erasing pattern language equals a finite union of non-erasing pattern languages. For illustration, consider the pattern $p = x_1bx_2ax_1$. It is not hard to see that the erasing pattern language $L_{\mathrm{e}}(p)$ equals the union of the non-erasing pattern languages $L_{\mathrm{ne}}(ba)$, $L_{\mathrm{ne}}(bx_2a)$, $L_{\mathrm{ne}}(x_1bax_1)$, and $L_{\mathrm{ne}}(x_1bx_2ax_1)$.

In contrast to the non-erasing case, a statement analogous to Proposition 1 does not hold. An erasing pattern language $L_{\mathrm{e}}(p)$ may obviously contain strings that are shorter than the pattern p. Moreover, it is obvious that set of shortest strings in $L_{\mathrm{e}}(p)$ (in fact, this is just a singleton set) cannot uniquely characterize this erasing pattern language. But surprisingly, the situation is much more subtle.

Let $\Sigma = \{a, b\}$, let $p = x_1x_1x_2x_2x_3x_3$, and let S be any finite subset of $L_{\mathrm{e}}(p)$. Then there is a pattern q such that $S \subseteq L_{\mathrm{e}}(q)$, but $L_{\mathrm{e}}(q) \subset L_{\mathrm{e}}(p)$ (cf. [Rei02]). Hence, it may happen that, for a given erasing pattern languages, there is no finite subset at all that uniquely characterizes this particular language. As it turns out, this feature forms the main obstacle when erasing pattern languages form the subject of learning.

In the erasing case, Proposition 2 rewrites as follows:

Proposition 5 ([Ang80a, JSSY93]) *Let Σ and X be fixed.*

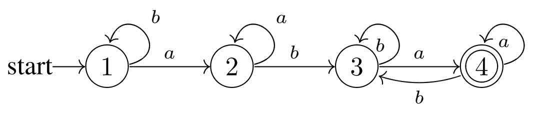
(1) The membership problem for the class of all erasing pattern languages is solvable. Again, if $\mathcal{NP} \neq \mathcal{P}$, there is no efficient algorithm for this problem.

(2) The inclusion problem for the class of all erasing pattern languages is not solvable.

While there is an efficient algorithm solving the equivalence problem for the class of non-erasing pattern languages, it is one of the most difficult open questions whether or not the equivalence problem for the class of erasing pattern languages is solvable at all, cf. [Rei04a] for most recent studies on this subject.

However, by restricting the set of admissible patterns to the regular ones, the differences between the erasing case and the non-erasing case no longer persist.

Let p be a regular pattern. As in the non-erasing case, the erasing pattern language $L_{\mathrm{e}}(p)$ forms a regular language. For illustration, the following automaton A accepts the language $L_{\mathrm{ne}}(x_1abx_2a)$, where again, for simplicity, it is assumed that $\Sigma = \{a, b\}$.



If p is a regular pattern, $L_{\mathrm{e}}(p)$ is called an *erasing regular pattern language*. Since the class of all erasing regular pattern languages forms a proper subclass of the class of all regular languages, too, a statement analogous to Proposition 3 holds.

Proposition 6 ([Shi82]) *Let Σ and X be fixed.*

(1) The membership problem for the class of all erasing regular pattern languages is solvable. Moreover, there is a linear-time algorithm for this problem.

(2) The equivalence problem for the class of all erasing regular pattern languages is solvable.

(3) The inclusion problem for the class of all erasing regular pattern languages is solvable.

4 Formal learning models

The pattern languages may serve as formal representations of target concepts in a machine learning domain. Additionally, such formal representations of target concepts allow for formalizing also different learning models.

Formalizing learning models is an essential part of simulating human learning with machines. Unfortunately, there is no formal approach covering all aspects and phenomena of human learning. However, particular aspects of learning have been reflected in particular learning models, thus enhancing a differentiated view and analysis of learning. Roughly speaking, a formal model requires the specification of the following parameters:

- the form of the (classes of) target concepts subject to learning,
- the form of the hypothesis spaces used and the underlying interpretation (that means, the way a learner represents its hypotheses on the current target concepts),
- the form of information the learner is given about a target concept,
- the form of the learners themselves,
- the constraints defining a successful learning behavior (that means, the criteria defining when a learner has actually learned a target concept).

For instance, in the data migration scenario, concepts may be represented as pairs of erasing pattern languages, the hypothesis space may be the class of all pairs of patterns, the information the learner is presented consists of examples, and the learner itself is a machine taking examples as input and returning a hypothesis after each example seen. Finally, one might define the behavior of a learner successful, if a correct pair of patterns (yielding the correct data transformation rule) is returned after, say, 20 examples.

In what follows, we assume that our target concepts in learning are recursive concepts, that means, there is an effective algorithm which, given any element of the underlying 'learning domain', decides whether or not this element belongs to the target concept. Note that any program for such an algorithm would serve as a representation of the target concept.

In our formal background, the set Σ^* of all strings over our alphabet Σ may form the underlying learning domain. Then a recursive target concept can be understood just as a recursive language over our finite alphabet Σ, i. e. any subset of Σ^*. So, in what follows we consider recursive subsets of Σ^* (called languages) as target concepts and our classes $\mathcal{C}$ of target concepts are classes of recursive languages. But we focus our attention on target classes of a specific structure: the so-called *indexable classes*.

Definition 1 *Let $\mathcal{C}$ be a class of recursive languages over Σ^*. $\mathcal{C}$ is an indexable class (of recursive languages), if there is a family $(L_i)_{i\in\mathbb{N}}$ of all and only the languages in $\mathcal{C}$, for which the membership problem is decidable, i. e., there is an effective algorithm which, given any $i \in \mathbb{N}$ and any string $s \in \Sigma^*$, decides whether or not s belongs to L_i.*

The family $(L_i)_{i\in\mathbb{N}}$ in this definition will subsequently be called an *indexing* of $\mathcal{C}$.

Why do we restrict our analysis on such indexable classes? On the one hand, the analysis of formal learning models as defined below would otherwise become very complex. On the other hand, our restriction is not too severe in the sense that most target classes considered in application domains can be represented as indexable language classes. Note that many classes of formal languages which are of interest in algorithmics and in formal language theory indeed are indexable, such as for instance the class of all regular languages, the class of all non-erasing pattern languages, the class of all erasing pattern languages, etc.

Definition 2 *Let $\mathcal{C}$ be any indexable class of recursive languages. A hypothesis space for $\mathcal{C}$ is a family $\mathcal{H} = (L_i)_{i\in\mathbb{N}}$ which fulfills the following two properties:*

(1) $\mathcal{H}$ comprises $\mathcal{C}$, i. e., $\mathcal{C} \subseteq \{L_i \mid i \in \mathbb{N}\}$;

(2) there is an effective algorithm which, given any $i \in \mathbb{N}$, lists all elements in L_i.

We consider two types of hypothesis spaces: (i) any kind of indexed family, (ii) any Gödel numbering, i. e., any family $(L_i)_{i\in\mathbb{N}}$ of languages fulfilling the following properties:

- $(L_i)_{i\in\mathbb{N}}$ is a hypothesis space,
- each recursively enumerable language (i. e., each language that can be listed by an effective algorithm) belongs to $(L_i)_{i\in\mathbb{N}}$,
- for each hypothesis space $\mathcal{H} = (L'_i)_{i\in\mathbb{N}}$ there exists a computable compiler function c such that $L'_i = L_{c(i)}$ for all $i \in \mathbb{N}$.

Intuitively, Gödel numberings correspond to complete programming languages: any index i encodes a program in the programming language and the language L_i encodes the behavior of that program. Gödel numberings constitute the most comprehensive hypothesis spaces, but do not allow for effectively solving the membership problem. In contrast to that, indexed families are less expressive, but their membership problem is solvable.

These notions are now used for formalizing two different learning models, the first one basing on the approach to simply offer positive examples to the learner, the second one implementing an interaction in which the teacher answers questions posed by the learner.

4.1 Learning from examples

Now let $\mathcal{C}$ be an indexable class, $\mathcal{H} = (L_i)_{i\in\mathbb{N}}$ any hypothesis space for $\mathcal{C}$, and $L \in \mathcal{C}$.

Definition 3 *Any total function $t : \mathbb{N} \to \Sigma^*$ with $\{t(i) \mid i \in \mathbb{N}\} = L$ is called a text for L. A text t is often identified with an infinite sequence $(t(i))_{i\in\mathbb{N}}$ of strings.*

Since we consider learning from text, i. e., from positive examples only, we will assume in the sequel that all languages to be learned are *non-empty*.

Definition 4 *An inductive inference machine (IIM) M is an algorithmic device that reads longer and longer initial segments σ of a text and outputs numbers $M(\sigma)$ as its hypotheses.*

Note that an IIM M returning some i is construed to hypothesize the language L_i. The following definition of learning in the limit is based on [Gol67].

Definition 5 *Given a text t for L, M identifies L in the limit from t with respect to $\mathcal{H}$, if*

(1) the sequence of hypotheses output by M, when fed t, stabilizes on a number i (i. e., past some point M always outputs the hypothesis i) and

(2) this number i fulfills $L_i = L$.

Moreover, M identifies L from positive examples in the limit with respect to $\mathcal{H}$, if M identifies L in the limit from every text with respect to $\mathcal{H}$. Finally, M identifies $\mathcal{C}$ in the limit from positive examples with respect to $\mathcal{H}$, if it identifies every $L' \in \mathcal{C}$ in the limit from positive examples with respect to $\mathcal{H}$.

It has been proven that the expressiveness of Gödel numberings as hypothesis spaces does not have any effect on their suitability for learning $\mathcal{C}$ in the limit from positive examples. In other words, $\mathcal{C}$ can be identified with respect to a Gödel numbering if and only if $\mathcal{C}$ can be identified with respect to an indexed family. In the sequel, we use the notion $\mathit{Lim Txt}$ for the collection of all indexable classes $\mathcal{C}'$ for which there are an IIM M' and a hypothesis space $\mathcal{H}'$ such that M' identifies $\mathcal{C}'$ in the limit from positive examples with respect to $\mathcal{H}'$.

A quite natural and often studied modification of $\mathit{Lim Txt}$ is defined by the model of *conservative inference*, see [Ang80b, LZ93].

Definition 6 *M is a conservative IIM for $\mathcal{C}$ with respect to $\mathcal{H}$, if M performs only justified mind changes, i. e., if M, on some text t for some $L \in \mathcal{C}$, outputs hypotheses i and later j, then M must have seen some string $s \notin L_i$ before returning j.*

Again, to analyze whether or not $\mathcal{C}$ is identifiable by a conservative IIM, it doesn't make a difference whether Gödel numberings or indexed families are considered as hypothesis spaces, see [LZ04a], as communicated by Sanjay Jain. The collection of all indexable classes identifiable in the limit from positive examples by a conservative IIM is denoted by $\mathit{Consv Txt}$. Note that $\mathit{Consv Txt} \subset \mathit{Lim Txt}$, by [LZ93]. That means, the requirement for a conservative learning behavior strictly reduces the learning capabilities of IIMs.

One main aspect of human learning is modeled within learning in the limit: the ability to change one's mind during learning. Thus learning is considered as a process in which the learner may change its hypothesis arbitrarily often until reaching its final correct guess. In particular, it is in general impossible to find out whether or not the final hypothesis has been reached, i. e., whether or not a success in learning has already eventuated.

To illustrate these learning models, we may again consider the pattern languages. For that purpose we will use sufficient or necessary criteria for a class being learnable in the limit from positive examples. A sufficient but not necessary condition is *finite thickness*.

Definition 7 *$\mathcal{C}$ is of finite thickness, if for each string $s \in \Sigma^*$ there are at most finitely many languages in $\mathcal{C}$ containing s, i. e., the class $\{L \in \mathcal{C} \mid s \in L\}$ is finite.*

Proposition 7 ([Ang80b]) *Let $\mathcal{C}$ be an indexable class of languages. If $\mathcal{C}$ is of finite thickness, then $\mathcal{C} \in \mathit{Lim Txt}$.*

An example in which this criterion applies is the class of all non-erasing pattern languages: since by Proposition 1 each string $s \in \Sigma^*$ can only be generated by patterns at most as long as s, there are at most finitely many non-erasing pattern languages containing s. So the class of all non-erasing pattern languages is learnable in the limit from positive examples (no matter how large the alphabet Σ is).

The converse is in general not true, i. e., there are classes in $\mathit{Lim\,Txt}$ which are not of finite thickness. Consider for instance the class of all unions of at most k non-erasing pattern languages, for some $k \in \mathbb{N}$. It is not hard to verify that this class does not have finite thickness. Still, it is learnable in the limit from positive examples! The reason is that the class of all non-erasing pattern languages has *recursive finite thickness*, i. e., it is of finite thickness and there is an effective algorithm, which, for any string s, determines a finite set of patterns representing all non-erasing pattern languages containing s. A result in [Lan00] states that, for any $k \in \mathbb{N}$, the class of all unions of at most k languages from an indexable class $\mathcal{C}$ belongs to $\mathit{Lim\,Txt}$ if $\mathcal{C}$ has recursive finite thickness. Therefore the class of all unions of up to k non-erasing pattern languages belongs to $\mathit{Lim\,Txt}$.

Moreover, it can be shown that every class of recursive finite thickness is in $\mathit{Consv\,Txt}$, whereas finite thickness alone is not sufficient for conservative learnability, see [Lan00].[4]

A criterion necessary for (conservative) learnability in the limit from positive examples is based on families of *telltales*, see [Ang80b].

Definition 8 *Let $(L_i)_{i\in\mathbb{N}}$ be any indexing and $(T_i)_{i\in\mathbb{N}}$ a family of finite non-empty sets. $(T_i)_{i\in\mathbb{N}}$ is called a family of telltales for $(L_i)_{i\in\mathbb{N}}$ iff for all $i, j \in \mathbb{N}$:*

(1) $T_i \subseteq L_i$.

(2) If $T_i \subseteq L_j \subseteq L_i$, then $L_j = L_i$.

T_i is then called a telltale for L_i with respect to $(L_i)_{i\in\mathbb{N}}$.

The concept of telltale families is the best known notion to illustrate the specific differences between indexable classes in $\mathit{Lim\,Txt}$ and $\mathit{Consv\,Txt}$. Telltale families and their algorithmic structure have turned out to be crucial for (conservative) learning in the limit:

Proposition 8 ([Ang80b]) *Let $\mathcal{C}$ be an indexable class of languages. If $\mathcal{C} \in \mathit{Lim\,Txt}$, then each indexing of $\mathcal{C}$ possesses a family of telltales.*

For example, if p is a pattern, the set $L_{\mathrm{ne}}^{\min}(p)$ of all shortest strings in $L_{\mathrm{ne}}(p)$ is a telltale for $L_{\mathrm{ne}}(p)$ with respect to the class of all non-erasing pattern languages, cf. [Ang80a, LW91].

Although the definitions of erasing and non-erasing pattern languages seem to differ only marginally, the telltale criterion helps to demonstrate how much the differences in their structures affect learnability: the class of all non-erasing pattern languages belongs to

[4]However, each class of non-erasing or erasing pattern languages we consider here either belongs to $\mathit{Consv\,Txt}$ or is not even contained in $\mathit{Lim\,Txt}$, although, in general, learnability in the limit does not imply conservative learnability.

$Consv\,Txt$, whereas, under certain constraints on the size of the alphabet Σ, the class of all erasing pattern languages does not even belong to $Lim\,Txt$. The latter has been verified for an alphabet of exactly two elements in [Rei02] by showing that the erasing language generated by the pattern $p = x_1x_1x_2x_2x_3x_3$ does not possess a telltale with respect to the class of all erasing pattern languages, see Section 3.2. Similarly, the class of all erasing pattern languages is not learnable for alphabets of size 3 or 4, see [Rei04b].

In contrast to that, the class of all erasing pattern languages is learnable in the limit from positive examples, if the underlying alphabet contains only one element or infinitely many elements, as has been shown in [Mit98]. Whether or not the class of all erasing pattern languages is learnable for finite alphabets of size greater than 4 remains an open question.

Since, in many practical applications, erasing pattern languages seem to be suitable for modeling target concepts for learning, it is still interesting to identify at least practically relevant subclasses of erasing pattern languages which *are* learnable in the limit. Some prominent subclasses are the class of all regular erasing pattern languages and the class $\mathcal{C}_{\mathrm{pat,k}}$ of all erasing pattern languages generated by patterns with at most k variables, where k is any fixed non-negative integer, see [Mit98]. Note that, in this case, one of the results stated above applies: since each erasing pattern language generated by at most k variables forms the union of up to 2^k non-erasing pattern languages (cf. Proposition 4) and the class of such unions belongs to $Lim\,Txt$, we obtain $\mathcal{C}_{\mathrm{pat,k}} \in Lim\,Txt$.

4.2 Learning from queries

In the query learning model, a learner has access to a teacher that truthfully answers queries of a specified kind. A *query learner* M is an algorithmic device that, depending on the reply on the previous queries, either computes a new query or returns a hypothesis and halts, see [Ang88]. Its queries and hypotheses are interpreted with respect to an underlying hypothesis space according to Definition 2.[5] Thus, when learning $\mathcal{C}$, M is only allowed to query languages belonging to $\mathcal{H}$. More formally:

Definition 9 *Let $\mathcal{C}$ be an indexable class, let $L \in \mathcal{C}$, let $\mathcal{H} = (L_i)_{i\in\mathbb{N}}$ be a hypothesis space for $\mathcal{C}$, and let M be a query learner. M learns L with respect to $\mathcal{H}$ using some type of queries if it eventually halts and its only hypothesis, say i, represents L, i. e., $L_i = L$.*

So M returns its unique and correct guess i after only finitely many queries.[6] Moreover, M learns $\mathcal{C}$ with respect to $\mathcal{H}$ using some type of queries, if it learns every $L' \in \mathcal{C}$ with respect to $\mathcal{H}$ using queries of the specified type. In order to learn a target language L, a query learner M may ask:

Membership queries. Query: a string s. The answer is ‘yes’ if $s \in L$ and ‘no’ if $s \notin L$.

[5]The use of hypothesis spaces comprising proper supersets of $\mathcal{C}$ had not been allowed in the original definition of query learning, see [Ang88], but has meanwhile been studied extensively, e. g., in [LZ04b, LZ04c, LZ04a].

[6]Originally, the main focus in the study of query learning had been the efficiency of query learners in terms of the number of queries they pose. However, as in [LZ04b, LZ04c, LZ04a], we neglect efficiency issues here.

Superset queries. Query: an index i. The answer is 'yes' if $L_i \supseteq L$ and 'no' if $L_i \not\supseteq L$.

Disjointness queries. Query: an index i. The answer is 'yes' if $L_i \cap L = \emptyset$ and 'no' if $L_i \cap L \neq \emptyset$.

Note that this model in general requires a teacher answering undecidable questions. Now, in contrast to the models of learning from examples, the learnability of a target class $\mathcal{C}$ may depend on the type of hypothesis spaces used. In other words, there are classes learnable with respect to any Gödel numbering, but not learnable using an indexed family as a hypothesis space. Therefore, we have to use different notions, depending on which type of hypothesis space is assumed:

$MemQ_{\text{göd}}$, $SupQ_{\text{göd}}$, and $DisQ_{\text{göd}}$ denote the collections of all indexable classes $\mathcal{C}'$ for which there are a query learner M' and a Gödel numbering $\mathcal{H}'$ such that M' learns $\mathcal{C}'$ with respect to $\mathcal{H}'$ using membership, superset, and disjointness queries, respectively. Analogously, we use the subscript ind instead of göd to denote the case in which indexable classes are used as hypothesis spaces.[7] However, for membership queries this differentiation does not have any effect, so $MemQ_{\text{göd}} = MemQ_{\text{ind}}$. Obviously, queries are in general not decidable, i. e., the teacher may be non-computable.

Note that, in contrast to the models of language learning from examples introduced above, learning via queries focuses the aspect of one-shot learning, i. e., it is concerned with learning scenarios in which learning may eventuate without mind changes. Additionally, the role of the learner in this model is an active one, whereas in learning from examples, as defined above, the learner is rather passive; that means, the two models implement quite different scenarios of interaction between a teacher and a learner.

The learnability of different classes of both non-erasing and erasing pattern languages with different types of queries has also been analyzed systematically, e. g., in [Ang88, LW91, NL00, LZ03]. For instance, the class of all non-erasing pattern languages is identifiable in each of the query learning models defined above, whereas the class of all erasing pattern languages is not identifiable in any of them.

5 Relations between learning models

One main difference between learning from examples and query learning lies in the question whether or not a current hypothesis of a learner is already correct. An IIM is allowed to change its mind arbitrarily often (thus in general this question can not be answered), whereas a query learner has to find a correct representation of the target object already in the first guess, i. e., within 'one shot' (and thus the question can always be answered in the affirmative). Another difference is certainly the kind of information provided during the learning process. So, at first glance, these models seem to focus on very different aspects of human learning and do not seem to have much in common.

[7]In the literature, see [Ang88, Ang04], more types of queries such as subset queries and equivalence queries have been analyzed, but in what follows we concentrate on the three types explained above.

Thus the question arises, whether there are any similarities in these models at all and whether there are aspects of learning both models capture. Answering this question requires a comparison of both models concerning the capabilities of the corresponding learners. In particular, one central question in this context is whether IIMs, i. e., limit learners can be replaced by at least equally powerful (one-shot) query learners.

Since, in general, membership queries provide the learner with less information than superset or disjointness queries, it is not astonishing that there are classes in $\mathit{Consv\,Txt}$ which cannot be identified with membership queries, such as, for instance, the class of all unions of up to 2 non-erasing pattern languages. What is more interesting is that, in contrast, each class in $\mathit{MemQ}_{\text{göd}}$ is conservatively learnable in the limit from positive examples. Though this is not hard to prove, it is one of the first results giving evidence of a relationship between our two models of teacher-student interaction:

Proposition 9 $\mathit{MemQ}_{\text{ind}} = \mathit{MemQ}_{\text{göd}} \subset \mathit{Consv\,Txt}$.

However, the more challenging question is whether there are interaction scenarios adequate for replacing limit learners with equally capable one-shot learners. Indeed, the query learning scenarios defined above exactly yield this property: astonishingly, in [LZ03, LZ04b] it has been shown that the collection of indexable classes learnable with superset queries coincides with $\mathit{Consv\,Txt}$. And, moreover, this also holds for the collection of indexable classes learnable with disjointness queries.

Theorem 10 ([LZ03, LZ04b, LZ04c]) $\mathit{Consv\,Txt} = \mathit{SupQ}_{\text{ind}} = \mathit{DisQ}_{\text{ind}}$.

This result concerns only indexed families as hypothesis spaces for query learners. As has already been mentioned, it is conceivable to permit more general hypothesis spaces in the query model, i. e., to demand an even more capable teacher. Interestingly, this relaxation helps to characterize learning in the limit in terms of query learning.

Theorem 11 ([LZ04c]) $\mathit{Lim\,Txt} = \mathit{DisQ}_{\text{göd}}$.

Comparing this result to Theorem 10, it is evident that a characterization of $\mathit{Lim\,Txt}$ in terms of learning with superset queries is missing. Thus there remains the question whether or not $\mathit{DisQ}_{\text{göd}}$ equals $\mathit{SupQ}_{\text{göd}}$. The following result answers this question to the negative.

Theorem 12 ([LZ04a]) $\mathit{DisQ}_{\text{göd}} \subset \mathit{SupQ}_{\text{göd}}$.

Corollary 13 ([LZ04a]) $\mathit{Lim\,Txt} \subset \mathit{SupQ}_{\text{göd}}$.

All in all, this illustrates a trade-off concerning learning capabilities, given by a balance between the type of information given to the learner and the requirements on the number of guesses a learner may propose.

6 Prototypical applications – Refinements

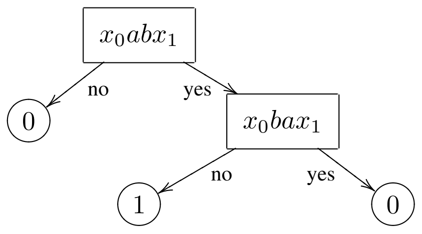
In this section, we return to one of our prototypical application scenario, the protein classification problem. In Subsection 2.2, it has been demonstrated that finite collections of patterns can be used to classify proteins into membrane and non-membrane proteins. Naturally, the question arises whether even more accurate classifiers can be learned.

One rather obvious approach is based on the idea to apply learning algorithms more powerful in the sense that they may generate hypotheses being strictly more expressive than the hypotheses allowed before. Under these circumstances, there may be a chance to learn classifiers of higher accuracy. Another perhaps more sophisticated approach is to exploit the available domain knowledge to simplify the learning task itself. Instead of looking for more powerful learning algorithms, one may, for instance, try to preprocess the training data such that even less complicated hypotheses generalize the regularities in the training data sufficiently well. In what follows, we present a case study for the protein classification problem that covers instantiations of both approaches.

6.1 Decision trees over regular patterns

A *decision tree over regular patterns* (*decision tree*, for short) is a binary tree. As usual, every node in a decision tree is either a *leaf* (a node without a child) or an *inner node* (a node with two children). The labels of the inner nodes of a decision tree are regular patterns, while the labels of the leaves of the tree are elements from the set $\{1, 0\}$. By convention, every decision tree contains at least one node.

A decision tree defines a quite natural classifier. First, it is checked whether or not a given string s belongs to the erasing regular pattern language generated by the pattern at the root of the tree. Depending on the outcome of this test, it is next checked whether or not s belongs to the erasing regular pattern language which is generated by the pattern either at the root of its left subtree or at the root of its right subtree. If a leaf is reached, its label determines whether or not s belongs to the language accepted by the decision tree on hand. Since all the patterns used are regular, decision trees constitute quite fast classifiers. For illustration, consider the following decision tree T.



In case that the underlying alphabet Σ equals the set $\{a, b\}$, the decision tree T accepts

(i. e., classifies with 1) all strings of the form $a^i b^j$ with $i, j \in \mathbb{N}$ and $i, j \geq 1$.[8]

The language accepted by a decision tree is formally defined as follows.

Definition 10 *Let Σ and X be fixed. Let T be a decision tree. The language $L_{\mathrm{t}}(T)$ accepted by T is inductively defined as follows:*

(i) Assume T contains exactly one node. In this case let $L_{\mathrm{t}}(T) = \Sigma^*$, if its label is '1', and $L_{\mathrm{t}}(T) = \emptyset$, otherwise.

(ii) Assume T contains more than one node. Let p be the root label of T, let T_l be the left subtree of T, and let T_r be the right subtree of T. In this case let $L_{\mathrm{t}}(T) = (\overline{L_{\mathrm{e}}(p)} \cap L_{\mathrm{t}}(T_l)) \cup (L_{\mathrm{e}}(p) \cap L_{\mathrm{t}}(T_r))$.[9]

Although the class of all languages acceptable by decision trees forms a proper superclass of the class of all erasing regular pattern languages, it constitutes a proper subclass of the class of all regular languages, see [LN03]. Consequently, a statement analogous to Proposition 6 holds.

Proposition 14 ([LN03]) *Let Σ and X be fixed.*

(1) The membership problem for the class of all languages acceptable by decision trees is solvable. Moreover, there is an efficient algorithm for this problem.

(2) The equivalence problem for the class of all languages acceptable by decision trees is solvable.

(3) The inclusion problem for the class of all languages acceptable by decision trees is solvable.

The class of all languages acceptable by decision trees is closed with respect to the operations union, intersection, and complement, see [LN03], while the class of all erasing regular pattern languages does not share any of these features. However, the observed add-on in expressiveness has a negative side-effect also, as the following non-learnability result illustrates.

Proposition 15 ([LN03]) *Let Σ and X be fixed. The class of all languages acceptable by by decision trees is not in $Lim\,Txt$.*

On the other hand, if both positive and negative examples are available, the class of all languages acceptable by decision trees is trivially learnable in the limit, see [Gol67].

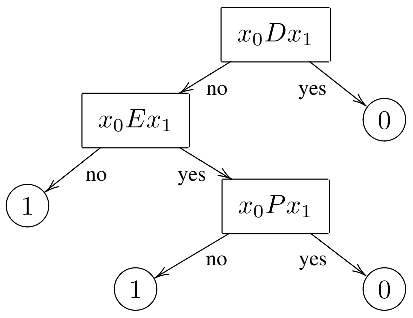
[8] Note that the language $L = \{a^i b^j \mid i, j \geq 1\}$ cannot be represented as the union of finitely many erasing pattern languages.

[9] As usual, for a language $L \subseteq \Sigma^*$, $\overline{L}$ denotes the complement of L, i. e., $\overline{L} = \Sigma^* \setminus L$.

6.2 Protein Classification (continued)

In [AMS$^+$93] a different approach has been applied to the problem of predicting membrane proteins. The underlying learning scenario is almost the same as the one defined in Subsection 2.2. The only difference is that now decision trees over regular patterns are used to represent the required classifier.

For that purpose, a version of one of the standard algorithms for the top-down-induction of decision trees, namely the ID-3 algorithm, see [Qui86], has been used to synthesize an appropriate decision tree. As a result, the following decision tree has been obtained.



As the experiments in [AMS$^+$93] document, this rather simple classifier has an acceptable accuracy. It correctly predicts about 84.7 % of the known membrane proteins and about 89.5 % of the known non-membrane proteins in the test set. Hence, it outperforms the classifier displayed in Subsection 2.2, which is based on the idea to use a collection of patterns to predict whether or not a given protein may serve as membrane protein.

However, this is only half of the story. Until now, only the regularities in the primary structure of proteins have been exploited to induce the required classifier. Additional relevant knowledge from the area of bio-chemistry has been neglected so far.

Empirical results in bio-chemistry have given evidence of the importance of the so-called hydropathic character of amino acids in the context of separating membrane proteins from non-membrane proteins, see [KD82]. According to the hydropathic character – measured in terms of the so-called hydropathic index of amino acids – three groups of amino acids are distinguished:

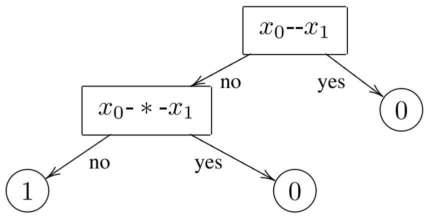
group	hydropathic index	amino acids	character
(i)	between 1.8 and 4.5	A, M, C, F, L, V, I	*
(ii)	between -1.6 and -0.4	P, Y, W, S, T, G	+
(iii)	between -4.5 and -3.2	R, K, D, E, N, Q, H	-

This kind of background knowledge can be directly encoded into the plot of the primary structure of a protein. For illustration, the plot of the primary structure of the membrane protein from Subsection 2.2 rewrites as follows. Again, the known transmembrane domains are printed boldly.

```
* - * * - - * * * + + - * - * * - - + * + * * - * * - + * * * * * * * * + * + +  + + + - * - * + * - * * - -
+ - + * * - * - * - * - + + * - * - - * + * - * + + * * - + + + + - * * * * + - + + + + * - * * * + * * *
* * * * + + * + * * * + + * * *  - - - + - - - - - + + * * - * * * + * * * * * * + * * + + + * + *  - + *
+ - * - * * + - + - - * * - - * + * * - - + + - + * - - * - - + * + + + * - + + * * * + * * - * * * + * + -
* + * * *  - - + + + * * + * * - * + + + * + - - - + * + + - * + + - * + + + - + + + + + + + - - + + + +
- + + + - + - + + - + * + + + + + + + + - + - + + - - + + + - - + * + + + * + - - *
```

In [AMS+93] is has been shown that this kind of data preprocessing has a significant effect on the accuracy of the learned classifiers. In the new learning scenario, the same learning algorithm is used. Instead of strings over an alphabet of 20 characters, the learning algorithm now receives strings over the three letter alphabet $\Sigma = \{*, +, \text{-}\}$. In turn, this results in a considerable simplification of the hypothesis space of the learning algorithm as well as in a significant speed-up of its running time. This is caused by the fact that the regular patterns in the inner nodes of an admissible decision tree can only contain terminal symbols belonging to the smaller alphabet Σ and thus less pattern candidates have to be tested for each inner node.

An application of the learning algorithm to the preprocessed training data yields the following decision tree over regular patterns.



This very simple classifier has a higher accuracy than the decision tree from above. It correctly predicts about 91.4 % of the known membrane proteins and about 94.8 % of the known non-membrane proteins in the test set, see [AMS+93].

7 Discussion

We have considered prototypical applications of machine intelligence based on formal models of language learning and with the vision of intelligent assistant systems in mind. The essential aspect of our analysis has been the approach of incorporating different types of learning scenarios based on different types of interaction between two 'partners' – a teacher and a learner. The formalization of the corresponding models has helped to describe important parameters of such interaction scenarios, the modification of which can enhance the success in learning.

One such parameter is for instance the hypothesis space used, which may be seen as a kind of language or representation scheme the teacher and the learner use for communication. Its relevance has been approved theoretically, e. g., in the query learning model or in the differences between non-erasing and erasing pattern languages, but also empirically, e. g., in the accuracy improvements using decision trees instead of unions of erasing pattern languages for classifying proteins into membrane and non-membrane proteins.

Another relevant parameter is the form of information presented, as expressed in the trade-off between the model of learning from examples and the model of learning from queries. The encoding of information of some specified kind also plays an important role: for instance, in our protein classification problem, hydropathic characters have shown more appropriate for representing the training data than amino acid characters.

Further important parameters have been the mode of interaction itself (e. g., limiting versus one-shot processes) and the additional requirements concerning the behavior of the learner (e. g., conservativeness).

A modification of any of these parameters may have an impact on the capabilities of the corresponding learners. However, there are also alternative methods for enhancing the accuracy performance of learning algorithms obeying certain formal requirements. For instance, in the model of PAC learning (an alternative formal model of learning from examples, see [Val84]), the use of so-called ensemble methods has shown to have a significant positive effect on the accuracy of hypotheses generated by the corresponding learners.

The approach of ensemble learning is to use a team of learners instead of a single learner and then combine the hypotheses generated by the team members to a somehow aggregated hypothesis. Typical variants of this approach are Bagging, see [Bre96] (where actually only a single learner but different sets of training data are employed), and Boosting (where the team members result from invoking a single learner with different parameters like weights and thresholds), cf. [Sch99]. The effect impressively shows in a theoretically approved result, cf. [Sch99]: even if the team members emerge from a rather 'weak' learner (informally speaking, a learner guaranteeing only a result slightly better than 'guessing at random'), Boosting algorithms can aggregate this team to learners producing hypotheses of arbitrarily high accuracy.

An important fact implicit in our survey is that there does not exist any universal learning algorithm suitable for any kind of learning task considered above. Knowledge about the target concepts, the learning domain, and the structure of the admissible hypotheses must be exploited to design an adequate learning algorithm. Therefore different approaches have been used to model some kind of meta-learning where machines solve learning problems uniformly in the sense that the relevant knowledge is presented as additional information in the learning process, cf. [BCJ99, Jan79, Zil04]. As has been verified, even in this meta-model no universal algorithm can be constructed. However, corresponding studies have given evidence of collections of learning problems involving some common structures sufficient for the existence of uniform solutions.

All in all, one additional conclusion of this discussion is that formal models may be not only useful but even essential

- in analytically approaching and categorizing learning problems,
- in identifying aspects relevant for their solution, and, finally,

- in making statements concerning the existence of adequate learners, i. e., statements on whether or not successful learning is possible at all.

References

[AMS$^+$93] S. Arikawa, S. Miyano, A. Shinohara, S. Kuhara, Y. Mukouchi, and T. Shinohara. A machine discovery from amino acid sequences by decision trees over regular patterns. *New Generation Computing*, 11:361–375, 1993.

[Ang80a] D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

[Ang80b] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[Ang04] D. Angluin. Queries revisited. *Theoretical Computer Science*, 313:175–194, 2004.

[BCJ99] G. Baliga, J. Case, and S. Jain. The synthesis of language learners. *Information and Computation*, 152:16–43, 1999.

[Bre96] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[Cho56] N. Chomsky. Three models for the description of languages. *IRE Transaction on Information Theory*, 2, 1956.

[Gol67] E.M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[HU69] J.E. Hopcroft and J.D. Ullman. *Formal Languages and their Relation to Automata*. Addison-Wesley, 1969.

[Jan79] K.P. Jantke. Natural properties of strategies identifying recursive functions. *Journal of Information Processing and Cybernetics (EIK)*, 15:487–496, 1979.

[JSSY93] T. Jiang, A. Salomaa, K. Salomaa, and S. Yu. Inclusion is undecidable for pattern languages. In *Proc. 20th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 700, pages 301–312, Springer, 1993.

[KD82] J. Kyte and R. Doolittle. A simple method for displaying the hydropathic character of a protein. *Journal of Molecular Biology*, 157:105–132, 1982.

[Lan00] S. Lange. *Algorithmic Learning of Recursive Languages*. Mensch & Buch Verlag, 2000.

[LN03] S. Lange and J. Nessel. Decision lists over regular patterns. *Theoretical Computer Science*, 298:71–87, 2003.

[LW91] S. Lange and R. Wiehagen. Polynomial-time inference of arbitrary pattern languages. *New Generation Computing*, 8:361–370, 1991.

[LZ93] S. Lange and T. Zeugmann. Language learning in dependence on the space of hypotheses. In *Proc. 6th Annual ACM Conference on Computational Learning Theory*, pages 127–136, ACM Press, 1993.

[LZ03] S. Lange and S. Zilles. On the learnability of erasing pattern languages in the query model. In *Proc. 14th Int. Conference on Algorithmic Learning Theory*, volume 2842 of *Lecture Notes in Artificial Intelligence*, pages 129–143. Springer, 2003.

[LZ04a] S. Lange and S. Zilles. Comparison of query learning and Gold-style learning in dependence on the hypothesis space. In *Proc. 15th Int. Conference on Algorithmic Learning Theory*, Lecture Notes in Artificial Intelligence 3244, pages 99–113, Springer, 2004.

[LZ04b] S. Lange and S. Zilles. Formal language identification: Query learning vs. Gold-style learning. *Information Processing Letters*, 91:285–292, 2004.

[LZ04c] S. Lange and S. Zilles. Replacing limit learners with equally powerful one-shot query learners. In *Proc. 17th Conference on Learning Theory*, Lecture Notes in Artificial Intelligence 3120, pages 155–169, Springer, 2004.

[Mit98] A.R. Mitchell. Learnability of a subclass of extended pattern languages. In *Proc. 11th Annual ACM Conference on Computational Learning Theory*, pages 64–71, ACM Press, 1998.

[Nix83] R.P. Nix. Editing by examples. Technical Report 280, Yale University, Dept. Computer Science, 1983.

[NL00] J. Nessel and S. Lange. Learning erasing pattern languages with queries. In *Proc. 11th Int. Conference on Algorithmic Learning Theory*, volume 1968 of *Lecture Notes in Artificial Intelligence*, pages 86–100, Springer, 2000.

[Qui86] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Rei02] D. Reidenbach. A negative result on inductive inference of extended pattern languages. In *Proc. 13th Int. Conference on Algorithmic Learning Theory*, Lecture Notes in Artificial Intelligence 2533, pages 308–320, Springer, 2002.

[Rei04a] D. Reidenbach. On the equivalence problem for e-pattern languages over small alphabets. In *Proc. 8th International Conference on Developments in Language Theory*, Lecture Notes in Computer Science 3340, pages 368–380, Springer, 2004.

[Rei04b] D. Reidenbach. On the learnability of e-pattern languages over small alphabets. In *Proc. 17th Conference on Learning Theory*, Lecture Notes in Artificial Intelligence 3120, pages 140–154, Springer, 2004.

[SA95] T. Shinohara and S. Arikawa. Pattern inference. In K.P. Jantke and S. Lange, editors, *Algorithmic Learning for Knowledge-Based Systems*, Lecture Notes in Artificial Intelligence 961, pages 259–291. Springer, 1995.

[Sch99] R.E. Schapire. Theoretical views of boosting and applications. In *Proc. 10th Int. Conference on Algorithmic Learning Theory*, Lecture Notes in Artificial Intelligence 1720, pages 13–25, Springer, 1999.

[Shi82] T. Shinohara. Polynomial time inference of extended regular pattern languages. In *RIMS Symposia on Software Science and Engineering*, Lecture Notes in Computer Science 147, pages 115–127, Springer, 1982.

[Val84] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[Zil04] S. Zilles. Separation of uniform learning classes. *Theoretical Computer Science*, 313:229–265, 2004.