$\mathbb{TCS}$-$\mathbb{TR}$-$\mathbb{B}$-**05-01**

$\mathbb{TCS}$ Technical Report

Course Notes on Theory and Practice of Algorithms – Part I: Algorithmic Learning

by

THOMAS ZEUGMANN

Division of Computer Science

Report Series B

June 7, 2005

Hokkaido University
Graduate School of
Information Science and Technology

Email: thomas@ist.hokudai.ac.jp

Phone: +81-011-706-7684
Fax: +81-011-706-7684

Abstract

This report contains the course notes of Part I of *Theory and Practice of Algorithms*. Within this part we deal with *algorithmic learning*. Some learning algorithms for fundamental learning problems are studied. Furthermore, we focus our attention on the complexity theoretical issues involved.

In addition to the usual wort-case analysis, we also deal with the average-case behavior. Therefore, one lecture is devoted to introduce the subject of average-case analysis of algorithms.

Finally, after having gained a deeper understanding of what algorithmic learning really is, we exemplify the application of algorithmic learning to the field of data mining.

Contents

"Everything should be made as simple as possible, but not simpler."

Albert Einstein

LECTURE 1: ALGORITHMIC LEARNING

This course is about algorithms. The history of algorithms goes back, approximately, to the origins of mathematics at all. For thousands of years, in most cases, the solution of a mathematical problem had been equivalent to the construction of an algorithm that *solved* it. The ancient development of algorithms culminated in Euclid's famous "Elements". Unfortunately, after Euclid the design of algorithms faced roughly 2000 years of decline. It got increasingly popular to prove the existence of mathematical objects by using the technique of *reductio ad absurdum*.

Modern computation theory starts with the question:

Which problems can be solved algorithmically ?

It was posed due to the fact that despite enormous efforts of numerous mathematicians several problems had remained unsolved yet, e.g., the construction of an algorithm deciding whether a given Diophantine equation has an integral solution (Hilbert's 10th problem). In order to answer it, first of all, the *intuitive notion of an algorithm has to be formalized mathematically*. Starting from different points of view, Turing [16], Church [3], and Gödel [4] have given their formalizations (i.e., the Turing machine, the λ–calculus, the recursive functions). However, all these notions are equivalent, i.e., each computation in formalism-1 can be simulated in formalism-2 and vice versa. Since then, many other proposals had been made to fix the notion of an algorithm (e.g., by Post [10] and by Markov [7]). This led to the well-known "Church Thesis," i.e., "*the intuitive computable functions are exactly the Turing machine computable ones*.

After the theory explaining which problems can and cannot be solved algorithmically had been well-developed, the attention has turned to the qualitative side, i.e., *How "good" a problem can be solved?* First influential papers in this direction were written by Rabin [11] and [12], as well as by Hartmanis and Stearns [5]; the latter paper gave the field its title: ***computational complexity***. The central topic studied here was to clarify what does it mean to say that a function f is more difficult to compute than a function g. The distinction between problems solvable within a time bounded by a polynomial in the length of the input (that is, problems having a feasible solution) and those not having this property, first was made by J. von Neumann (1953). Since then, the class of polynomial time-bounded algorithms has been an object of intensive and continuing research. On the other hand, it is not known whether many of the problems that are very important for numerous applications can be solved in polynomial time.

Since this is a very large area, we have to select certain problems we wish to study. Recently, the field of knowledge discovery has attracted enormous interest. Therefore, we would like to focus our attention in the first part of this course to several aspects related to this area. In particular, we shall deal with *algorithmic learning*. But also this area is much too large to be covered within such a short amount of time. Throughout this course, we shall have a closer look at the developments within the field of algorithmic learning. We shall study some learning algorithms for fundamental learning problems. Additionally, we focus our attention on the complexity theoretical issues involved.

After having gained a deeper understanding of what algorithmic learning really is, we shall turn our attention to the application of algorithmic learning to the field of data mining.

The second part of this course is on *efficient data structures for manipulating Boolean functions*. This part will be covered in a separate report.

Next, we would like to motivate the subject of algorithmic learning. So, let us ask

Why do we dream about algorithmic learning?

After all, humans are excellent learners. As Bertrand Russell [14] already noted

> "How comes it that human beings, whose contacts with the world are brief and personal and limited, are nevertheless able to know as much as they do know?"

But nowadays, it becomes more and more difficult to know what is already known. There are many learning tasks that are far too complex for humans, or solving them by humans is quite expensive, and/or takes a huge amount of time, thus, the knowledge gained may be out of data before it is even ready. So, it would be better, if computers could learn, for instance, knowledge bases from examples. It would also be desirable to have computers that can discover knowledge by their own.

This idea, however, is not as new as it may seem. Shortly after the first computers have been emerged, pioneering researchers formulated the goal to construct a machine which is capable to learn. Allan Turing [17] considered the capability to learn as a major part of intelligence.

Nevertheless, the problem reached a new dimension during the last decade. Nowadays, across a wide variety of fields, data are being collected and accumulated at a dramatic pace. For example, within the human genome project there are now billions of data available. But researchers in this field are not mainly interested in the data themselves. Instead, they want to find out, for example, which genes may cause a particular kind of cancer. So, there is an urgent need for a new generation of computational techniques and tools to assist humans in extracting useful information (knowledge) from the rapidly growing volumes of data.

As Pieter Adriaans and Dolf Zantinge [1] put it:

> "We are confronted with the paradox of the growth of data, that more data means less information. In the future, the ability to read and to interpret alone will not be enough to sur vive as a professional, a scientist or a commercial organization. The mechanical production and reproduction of data force us to adapt our strategies and develop mechanical methods of filtering, selecting, and interpreting data."

A similar viewpoint has been expressed by Richter *et al.* [13]

> "Moreover, nowadays the data collected in in various fields such as biology, finance, retail, astronomy, medicine are extremely rapidly growing, but our ability to discover useful knowledge from such huge data sets is still too limited. Clearly, powerful learning systems would be an enormous help in automatically extracting new interrelations, knowledge, patterns and the like from those and other *huge* collections of data."

Within the process of discovering knowledge from data bases, very often algorithmic learning plays an important role. Therefore, we shortly introduce this subject here. Before doing this, it is very useful to recall a point made by Donald Knuth [6] in his Turing Award lecture.

> "Science is knowledge that we understand so well that we can teach it to a computer; and if don't fully understand something, it is an art to deal with it. . . . we should continually be striving to transform every art into science: in the process, we advance the art."

Thus it is only natural to ask whether learning is an *art* or a *science*.

1.1. What is Learning?

Since humans are excellent learners, at first glance it should be easy to define what learning is. So, let us continue by looking at some of the definitions proposed.

Simon [15] gave the following well-known definition;

> "Learning denotes changes in the system that . . . enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time."

It is not hard to see that this definition also covers phenomena that are usually not considered as learning. For example, if we run the same program on a faster computer then execution will be faster. Please think about more examples along this line. On the other hand, this definition does not cover all aspects of learning. For example you may study a map before driving to destination unknown to you. Then just by knowing your way (because you learned it before starting your drive) you behave probably much better than someone who has not studied the map in advance. But you can do better the *first* time and not only when driving the second time.

So, let us see what else has been proposed.

> "Learning is making useful changes in our minds."

This definition has been given by Marvin Minsky [9]. However, it may be require substantial effort to make it applicable to the construction of a learning computer.

Another famous definition is due to Michalski [8].

> "Learning is constructing or modifying representations of what is being experienced."

And sometimes we even find (cf. Adriaans and Zantinge [1])

> "We will not state what learning is."

So, we see it is by no means easy to arrive at a useful definition. The state of the art can be characterized as follows (cf. Richter *et al.* [13]).

> Numerous mathematical models of learning have been proposed during the last three decades. Nevertheless, different models give vastly different results concerning the learnability and non-learnability of objects one wants to learn. Hence, finding an appropriate definition of learning which covers most aspects of learning is also part of the goals aimed at in algorithmic learning theory.

For getting a first understanding of what we have to talk about, let us continue by looking at some examples.

1.2. Examples

We restrict ourselves here to inductive learning, i.e., to scenarios in which the learner, at every stage, has access to a finite set of data, though it has possibly to generalize its findings to an infinite set of data. Our first example is drawn from the area of function learning. A teacher is providing input-output examples of the target function. So, let us assume that we have received

$$f(0) = 0 \quad \text{and} \quad f(1) = 1 \; .$$

What function could it be? Since there is no need to introduce anything sophisticated at this point, most people would conjecture that the target function can be expressed as $f(x) = x$.

But the next example is destroying this conjecture, our teacher just supplied

$$f(3) = 720! \; ,$$

where the *factorial* function $n!$ is inductively defined as $0! = 1$ and $(n + 1)! = (n + 1)n!$ for all $n \geq 0$.

At this point we make the following observation.

$$f(3) = 720! = (6!)! = ((3!)!)! = 3!!!$$

This suggests the following solution.

$$\begin{aligned} g(x,0) &= x \\ g(x,n+1) &= g(x,n)! \\ f(x) &= g(x,x) \; . \end{aligned}$$

A quick check shows that we are correct, since

$$\begin{aligned} f(0) &= g(0,0) = 0 \\ f(1) &= g(1,1) = g(1,0+1) = g(1,0)! = 1! = 1 \\ f(3) &= g(3,3) = g(3,2+1) = g(3,2)! = g(3,1+1)! = (g(3,1)!)! \\ &= (g(3,0+1)!)! = ((g(3,0)!)!)! = ((3!)!)! \end{aligned}$$

As a matter of fact, for finding this solution we have applied a famous principle from the philosophy of science usually referred to as ***Occam's Razor*** which was formulated around 1320 by William of Occam as follows.

"Entities should not be multiplied unnecessarily."

The best reformulation of this principle for scientists is: *When you have two competing theories which make exactly the same predictions, the one that is simpler is the better.*

Therefore, if we look at the examples as expressions of the form $y = f(x)$ we had to express y in termini of x.

Furthermore, we made a couple of implicit assumptions which we would like to make explicitly right now. First, we have assumed that we have to learn a function. This target function is chosen by the teacher from a set of functions. Additionally, we assumed that the function is defined over the set $\mathbb{N}$ of natural numbers. So, the *domain* is a class of computable functions over the natural numbers.

Note that the assumption to have a computable function is essential in this context, since otherwise we cannot find a *finite* description for the target function. Moreover, we can interpret our solution as a *program* computing f. Thus, our learner produced a sequence of hypotheses and each hypothesis has been chosen from the set of all programs computing functions over $\mathbb{N}$. From now on, we shall refer to set of all allowed hypotheses as to the *hypothesis space*.

Assuming that we indeed learned the target function, we see that the learning success occurred only after a certain number of examples. So, in general we always have to define a *criterion of success*.

Last but not least, since we want to clarify what algorithmic learning is, we have assumed the learner to be an algorithm. This learner may have searched for the simplest program explaining the data.

Putting it all together, we can summarize our approach as follows.

domain:	class of computable functions
Information source:	examples: $(x, f(x))$
hypothesis space:	all programs
Learning method:	search
Semantics of hypotheses:	program
Success criterion:	correct after finitely many examples

Next, we look a problem that may serve as a typical example for many data mining tasks. So, let us look at the following tiny data set called the weather data. Of course, this data set is fictitious, but it serves its purpose to explain some typical features. The data concern the conditions for playing some unspecified game.

outlook	temperature	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Figure 1.1: ***The Weather Data***

In the table displayed above, we have four *attributes*, i.e., `outlook`, `temperature`, `humidity`, and `windy`. These attributes can take symbolic values rather than numerical values. The rightmost column shows the recommendation, that is whether or not one should play.

outlook	temperature	humidity	windy	play
sunny	cool	high	true	?

Figure 1.2: The data for tomorrow.

Next, we ask if there are any rules behind this table. This is the typical question asked in *data mining*. The typical problem is then to make predictions. For example, we are given the following data for tomorrow (cf. Figure1.2).

So, what should we do, play or not play? Of course, this prediction should be be consistent with the table given above. Now, the idea is first to learn a decision tree from the table above. Then, we apply it to the data for tomorrow. A decision tree explaining the data above could look as follows.

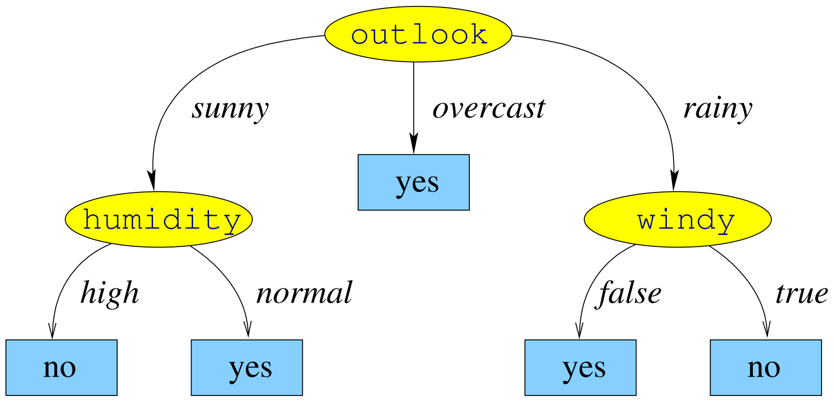

Figure 1.3: Decision tree for the weather data.

In order to see that this decision tree explains the table, let us look at the following example (the first data row of the table shown in Figure 1.1).

outlook	temperature	humidity	windy	play
sunny	hot	high	false	no

Figure 1.4: First entry of the weather data.

Using the decision tree above, we obtain the classification shown in Figure 1.4 by following the path determined by the values given for the attributes (cf. Figure 1.5).

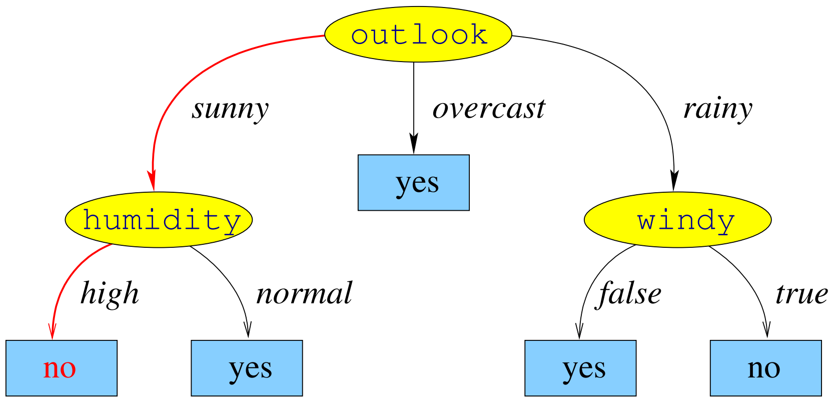

Figure 1.5: Application of the decision tree to the data in Figure

The rest of the table is verified analogously.

Now, we can apply the decision tree from Figure 1.3 to make the desired prediction. Since `outlook` is *sunny* and `humidity` is *high* we have to follow the same path as in Figure 1.5 and obtain *no*. Thus, we obtain:

`outlook`	`temperature`	`humidity`	`windy`	`play`
sunny	cool	high	true	no

Figure 1.6: The data for tomorrow completed.

Of course, you may wonder why the attribute `temperature` does not appear in our decision trees. Answering this question is closely related to the problem of how to construct decision trees for tables such as the weather data. We shall deal with these questions in Lecture 7.

Finally, we again summarize the approach undertaken in an analogous way as we did for the function learning problem. Thus, we obtain the following.

domain:	discrete functions
Information source:	positive and negative examples
hypothesis space:	all decision trees
Learning method:	decision tree learner
Semantics of hypotheses:	classifier
Success criterion:	correct after finitely many examples

Next, we take a look at a slightly different example. We wish to model learning tasks such as "Learning of Data Structures" from examples. Data structures are used in many applications. In order to keep the example simple, let us look at something familiar, i.e., a list of references.

GOLD, M.E. (1967), Language identification in the limit, *Information and Control* **10**, 447 – 474.

LANGE, S. AND WIEHAGEN, R. (1991), Polynomial-time inference of arbitrary pattern languages, *New Generation Computing* **8**, 361 – 370.

Of course, this list is a very short one. Usually, one has large or even huge bibliographies. These bibliographies may come from different sources and thus one may wish to unify them. So, one could choose the liked best, learn its underlying data structure and apply the result to the remaining ones.

Looking at the two examples given, the underlying data structure might have the following *pattern*:

AUTHORS, INITIALS. (Year), Title, *Journal* **Volume**, number – number.

or more formally: $x_1, x_2.(x_3), x_4, x_5x_6, x_7 - x_8$.

Any pattern is a non-empty string over $\mathcal{A} \cup X$, where $\mathcal{A}$ is an alphabet, X is a countably infinite set of variables and $\mathcal{A} \cap X = \emptyset$. Pattern define languages in a natural way. The language generated by a pattern π is the set of all strings that can be obtained by substituting all occurrences of variables by (non-empty) strings over $\mathcal{A}$, where the same variables have to be substituted by the same string. For example, the pattern xx generates the language $L = \{ww|\ w \in \mathcal{A}^+\}$. We shall formally define patterns and pattern languages in Lecture 3.

So, in our example above, $\mathcal{A}$ consists of the Latin alphabet, the punctuation symbols, the digits $\{0, 1, \ldots, 9\}$, and the parentheses ( ).

Note, however, that the references given above could be already generated by the following pattern

$$x_1, x_2.(19x_3), x_4, x_5x_6, x_7 - x_87x_9\ .$$

Summarizing in an analogous way as above, now we obtain:

domain:	pattern languages
Information source:	positive examples
hypothesis space:	all patterns
Learning method:	string algorithms
Semantics of hypotheses:	interpretation of patterns
Success criterion:	correct after finitely many examples

1.3. Specifying a Learning Model

After having seen three examples, we continue by clarifying the subject of Algorithmic Learning Theory. We focus our attention in partially answering of what *machine learning* is supposed to mean. Answering this question includes both *developing mathematical models* of machine learning and *deriving results within the models* developed. Both parts deserve special attention. In general, a model should capture at least significant applications. However, as already mentioned, the state of the art in modeling learning is still much less satisfactory than in other areas of theoretical computer science.

For example, around 60 years ago *computability theory* emerged. Initially, many different models have been introduced, e.g., Turing machines, partial recursive functions, Markov algorithms, Church' λ-calculus. Nevertheless, later on, all those models have been proved to be equivalent. Subsequently, researchers focused their attention to obtaining results within the model.

Another example is the rapidly emerging field of parallel computation. Unlike sequential algorithms, for which the above standard models exist, there are many contending models on which parallel algorithms can be based. Different parallel computer models differ from each other in various ways, e.g., whether or not different instructions on different data can be performed at one time, with respect to the geometrical arrangement of the processors and their interconnection, etc. However, there is a huge amount of literature relating these models to one another, and there is at least a universal interconnection pattern; i.e., the so called *parallel computation thesis* (cf., e.g., Zeugmann [18] and the references therein).

The situation in algorithmic learning theory is however, quite different. Numerous mathematical models of learning have been proposed during the last three decades. Nevertheless, different models give vastly different results concerning the learnability and non-learnability of objects one wants to learn. Hence, finding an appropriate definition of learning which covers most aspects of learning is also part of the goals aimed at in algorithmic learning theory.

As we have seen above, every learning model has to specify several aspects. In the following we provide a more detailed explanation of the main aspects to be modeled.

1. The Learner: Specifying the learner means answering the question "Who" is doing the learning? The most general answer one may give in algorithmic learning is that the learner is supposed to be an algorithm, i.e., a computer program. For example, the subfield of inductive inference generally specifies the learner in this way (cf., e.g., Angluin and Smith [2]). However, the learner may be restricted in one way or another. One may require the learner to be time efficient, i.e., to use an amount of time that is uniformly bounded by a polynomial in the length of its inputs. The learner may be also restricted with respect to its available amount of space. Further restrictions are possible, and we shall discuss them throughout this course.

2. The Domain: "What" is being learned? The answer to this question specifies the objects the learner has to deal with. For example, we may request the learner to learn an unknown concept, such as a table, a chair, a sofa. Then, the *learning goal* consists in synthesizing a "rule" to separate positive examples from negative ones. That means, after having finished its learning task, the learner must be able to correctly classify an object presented to him either as being a chair (a table, a sofa) or as not being a chair (a table, a sofa). This type of learning is referred to as concept learning, and has been studied extensively.

However, there are many other objects the learnability of which has been extensively studied, too. For example, the object to be learned may be an unknown function, an unknown language, an unknown device, an unknown technique (e.g., how to drive a car), an unknown environment (e.g., a new building), an unknown family of similar phenomena (e.g., voice recognition, face recognition, character recognition (hand written or printed), an unknown graph, and many, many more.

3. The Information Source: This part of the specification deals with the question from "what" information the learner should perform its task. That means, one has to clarify how the learner is informed about the target object. There is a huge variety of ways how this can happen. In the following we provide several examples that focus on widely studied models.

(i) **Examples:** In this scenario, the learner is fed with examples of the object to be learned. For example, when learning the concept of a chair, the learner is provided particular instances that constitute or do not constitute a chair. When learning an unknown function f, the learner is provided input–output examples, i.e.,$(x_0, f(x_0))$, $(x_1, f(x_1))$, $(x_2, f(x_2))$, ... When learning a language, the learner may be fed arbitrary strings over the underlying alphabet that are classified with respect to their containment in the target language. Alternatively, the learner may be also requested to learn from *positive* examples, only. Moreover, examples can be chosen in very different ways. They might be chosen with respect to some underlying but unknown (or known) probability distribution. Furthermore, they might be chosen arbitrarily, they can be chosen systematically, they can be chosen maliciously (this is of special interest if one is aiming to study the worst-case behavior of a learning algorithm). On the other hand, the examples can be chosen carefully, too, e.g., by a teacher who wants to facilitate the learning process.

There is another important aspect that will be described later, i.e., the problem how to *describe* the examples.

(ii) **Queries:** In this scenario, the learner is enabled to ask questions about the target object to a teacher. For example, when learning the concept of a car, the learner may ask "Is an Audi an example of a car?" Or, when learning a language, it may ask "Is w a word of the target language?" This type of question is usually referred to as *membership query* . Alternatively, it may ask "Is G a grammar for the target language?" When learning an unknown environment, it may ask "Is this a correct floor plan of the ground floor?" The latter type of question is referred to as *equivalence queries*, since the learner provides a rule and asks if that rule is correct. There are further types of possible questions, e.g., *subset queries, superset queries*, and *disjointness queries* that have been studied intensively.

(iii) **Experimentation:** In this scenario, the learner may get information concerning the object to be learned by actively experimenting with it. For example, the learner may learn a new environment by walking through it. Again, different walking strategies are imaginable, e.g., a random walk, a nondeterministic walk, a systematic walk, a.s.o.

When talking about the information source, another very important aspect has to be considered. This issue is whether or not the model can handle *noisy* or *erroneous* information sources. As an example, consider a learner that aims to recognize zip codes on letters. The learner is given examples provided by a post office employee. However, sometimes the employee may misread the handwritten codes. This results in erroneous information.

4. Hypothesis Space: Generally, a learner is supposed to map evidence (e.g., examples) on the object to be learned into a hypotheses about it. Therefore, one has to choose a set of possible descriptions. Clearly, each rule contained in the learning domain has to possess at least one description in the hypothesis space. However, the hypothesis space may additionally contain descriptions not describing any object in the learning domain. Furthermore, the descriptions provided by the hypothesis space may be slightly different from those ones used in defining the objects to be learned in the learning domain.

5. Prior Knowledge: Here, one has to specify what does the learner know about the domain initially? This generally restricts the learner's uncertainty and/or biases and expectations about the objects to be learned. Obviously, specifying the hypothesis space already provides some prior knowledge. In particular, the learner knows that the target object is representable in a certain way, e.g., as a graph having at most 1000 nodes, as a language acceptable by a finite automaton, or as function computable by a program having at most 100 instructions. However, such type of assumptions can be very unrealistic in practice. Furthermore, prior knowledge my also be provided by "telling" the learner that "simple" answers are preferable to more "complex" hypotheses. Finally, looking at important applications one has to take into account that prior knowledge may be "incorrect." Thus, when developing advance learning techniques one has to deal with the problem how to combine or trade-off prior versus new information.

6. Success Criteria: Finally, one has to specify the criteria for successful learning. This part of the specification must cover at least some aspects of our intuitive understanding of learning. For example, an automatic camera may record everything around it, but it is intuitively obvious that it does not learn. On the other hand, if we have an algorithm which, after having been provided a set of examples for the concept sofa, can label objects as either being a sofa or not being a sofa, we may agree that it has learned something, i.e., the concept sofa. In particular, we have to deal with questions like: "How do we know whether, or how well, the learner has learned?" "How does the learner demonstrate that it has learned something?"

Each specification of the six items described above leads to a model of learning. The interested reader is referred to the references listed below for further information. In the following lecture, we shall continue by exemplifying the general outline given above.

References

[1] P. ADRIAANS AND D. ZANTINGE (1997), *Data Mining*, Addison-Wesley Longman Publishing Co, Boston, MA.

[2] D. ANGLUIN AND C.H. SMITH (1983), Inductive inference: theory and methods, *Computing Surveys* **15**, 237 - 269.

[3] A. CHURCH (1936), An unresolvable problem of elementary number theory, *Am. J. Math.* **58**, 345 – 365.

[4] K. GÖDEL (1931), Über formal unentscheidbare Sätze der Principia Mathematica und Verwandter Systeme, *Monatshefte Mathematik Physik* **38**, 173 – 198.

[5] J. HARTMANIS AND R.E. STEARNS (1965), On the computational complexity of algorithms, *Trans. Am. Math. Soc.* **117**, 285 – 306.

[6] D.E. KNUTH (1974), Computer Programming as an Art, *Commun. ACM* bf 17 No. 12, 667-673.

[7] A.A. MARKOV (1954), Theoria Algorithmov, *Akad. Nauk SSSR, Math. Inst. Trudy* **42**.

[8] R. MICHALSKI (1986), Understanding the nature of learning: issues and research directions, *in* (R.S. Michalski, J.G. Carbonell and T.M. Mitchell, eds.), "Machine Learning: An Artificial Intelligence Approach, Vol. 2," pp. 3 – 25, Morgan Kaufman, Los Altos, Calif.

[9] M. MINSKY, *The Society of Mind* (1985), Simon & Schuster Inc., New York.

[10] E. POST (1943), Formal Reductions of General Combinatorial Decision Problems, *Am. J. Math.* **65**, 197 – 215.

[11] M.O. RABIN (1959), Speed of computation and classification of recursive sets, *in* Third Convention Sci. Soc., Israel, pp. 1 – 2.

[12] M.O. RABIN (1960), Degrees of difficulty of computing a function and a partial ordering of recursive sets, *in* Technical Report No. 1, University of Jerusalem.

[13] M.M. RICHTER, C.H. SMITH R. WIEHAGEN AND T. ZEUGMANN (1998), Editors' Introduction, *in* (M.M. Richter, C.H. Smith R. Wiehagen and T. Zeugmann, eds.), Algorithmic Learning Theory, 9th International Conference, ALT '98, Otzenhausen, Germany, October 1998, Proceedings, Lecture Notes in Artificial Intelligence 1501, pp. 1 – 10, Springer, Berlin.

[14] B. RUSSELL (1948), *Human Knowledge: Its Scope and Limits*, Simon and Shuster, New York.

[15] H. SIMON (1983), Why should machines learn?, *in* (R.S. Michalski, J.G. Carbonell and T.M. Mitchell, eds.), "Machine Learning: An Artificial Intelligence Approach," pp. 25–38, Tioga Publishing, Palo Alto, Calif.

[16] A.M. TURING (1936/1937), On computable numbers with an application to the Entscheidungsproblem, *Proc. London Math. Soc.* **42**, 230 – 265.

[17] A.M. TURING (1950), Computing Machinery and Intelligence, Mind **59**, 433-460.

[18] T. ZEUGMANN (1990), Parallel Algorithms, *Encyclopedia of Computer Science and Technology* Vol. 21, Supplement 6, Allen Kent and James G. Williams (eds.), pp. 223 – 244, Marcel Dekker Inc. New York and Basel.

LECTURE 2: LEARNING MODELS

After having seen what one has to specify in order to arrive at a learning model, we aim at defining some learning models.

2.1. Modeling a Learning Task

Humans are able to distinguish between different “things,” e.g., chair, table, sofa, book, newspaper, car, airplane, a.s.o. Also, there is no doubt that humans have to learn how to distinguish “things.” Therefore, we ask whether this particular learning problem allows an algorithmic solution, too. That is, we specify the learner to be an *algorithm*. Furthermore, we may be tempted to specify the learning domain to be the set of all things. However, since we aim to model learning, we have to convert “real things” into *mathematical descriptions of things*. This can be done as follows. We fix some language to express a *finite* list of properties. Afterwards, we decide which of these properties are relevant for the particular things we want to deal with, and which of them have to be fulfilled or not to be fulfilled, respectively. For example, the list of properties may be fixed as follows:

- possesses 4 legs, - possesses a rest, - has brown color, - possesses 4 wheels,

- it needs fuel, - possesses a seat,- possesses wings, . . . , - has more than 100 pages.

Now, we can answer

“What is a chair?” “What is a table?”

by deciding which of the properties are relevant for the concept “chair,” and which of them have to be fulfilled or not to be fulfilled, respectively.

	For chair, we obtain:	and for table:
1.	possesses 4 legs - yes	- yes
2.	possesses a rest - yes	- no
3.	has brown color - irrelevant	- irrelevant
4.	possesses 4 wheels - no	- no
5.	it needs fuel - no	- no
6.	possesses a seat - yes	- no
7.	possesses wings - no	- no
	.	.
	.	.
	.	.
n.	has more than 100 pages - no	- no

As you can see, there are two rows at which the entries differ, i.e., possesses a rest and possesses a seat. Thus, by using our properties, we could distinguish chair and table.

Next, we denote the n properties by the variables $x_1, \ldots, x_n$, where $\mathrm{range}(x_j) \subseteq \{0,1\}$ for $j = 1, \ldots, n$. The semantics is then obviously as follows. $x_j = 1$ means property j is fulfilled, while $x_j = 0$ refers to property j is not fulfilled.

Then $\mathrm{chair}\colon \{0,1\}^n \to \{0,1\}$, where

$$\mathrm{chair}(x_1, \ldots, x_n) = \begin{cases} 1, & \text{if } x_1 = x_2 = x_6 = 1, \text{ and } x_4 = x_5 = x_7 = \ldots = x_n = 0 \\ 0, & \text{otherwise} \end{cases}$$

Now, setting $\mathcal{L}_n = \{x_1, \bar{x}_1, x_2, \bar{x}_2 \ldots, x_n, \bar{x}_n\}$ (set of literals) we can express the concept "chair" by the monomial that contains precisely the relevant literals, i.e., $\mathrm{chair} = x_1 x_2 \bar{x}_4 \bar{x}_5 x_6 \bar{x}_7 \ldots \bar{x}_n$. Note that x_3 does not appear in the monomial "chair, " since it is irrelevant.

Hence, we may redefine the *learning domain* to be the set $\mathcal{X} = \{0,1\}^n$. We regard $\mathcal{X}$ as the instance space consisting of instances described by bit vectors of length n. Then the collection of objects to be learned is the set $\mathcal{C}$ of all *concepts* $c\colon \mathcal{X} \to \{0,1\}$ that are *describable by a monomial* over $\mathcal{X}$. Thus, the concepts c classify each instance b as negative (if $c(b) = 0$) or positive (if $c(b) = 1$). Hence, we may identify the concept c with the set of all positive instances for it. That means, we regard c as a subset of $\mathcal{X}$, and identify it with its characteristic function. There are two constant characteristic functions, i.e., 1 and 0 referring to the concept "TRUE" and "FALSE," respectively. "TRUE" can be represented by the empty monomial, and "FALSE" can be represented by, e.g., $x_1\bar{x}_1$. There is a *conceptional difference* between the class $\mathcal{C}$ of all concepts describable by a monomial and the set of all monomials itself. Namely, different monomials may describe the *same* concept. For example, the concept "FALSE" can be represented by $x_1\bar{x}_1$, $x_2\bar{x}_2$, ..., $x_1\bar{x}_1 \ldots x_n\bar{x}_n$. However, the concept "FALSE" is the only one allowing different descriptions. Therefore, in the following we often identify the set of all monomials over $\mathcal{L}_n$ and the concept class $\mathcal{C}$. Note that n is fixed. Thus, $\mathcal{C}$ is finite.

Exercise 1. Determine $\mathrm{card}(\mathcal{C})$.

We still have to specify the source of information, the hypothesis space, the prior knowledge, and the criterion of success. There are several ways to do this, and we shall have a look at some variants. In all variants we are going to consider, the *prior knowledge* is that the target concept is a monomial.

Variant 1: On-line prediction

In this setting, the *source of information* is specified as follows. The learner will be fed successively more and more labeled examples of the monomial to be learned. That is, the learner is given a sequence $\langle b_1, m(b_1), b_2, m(b_2), \ldots \rangle$, where the $b_j \in \{0,1\}^n$, i.e., Boolean vectors, and $m(b_j) \in \{0,1\}$ denote the value of the target monomial under the obvious assignment of the values provided in b_j to its literals. The examples are picked up arbitrarily, and are noise free.

We do not specify the *hypothesis space* explicitly. It is left to the learner to choose it.

Finally, we have to specify the *criterion of success*. In this example, we consider *on-line prediction*. That is, the learner must predict each $m(b)$, after having fed b. Then, it receives the true value of $m(b)$, and the next Boolean vector. We measure the performs of the learner by the number of prediction mistakes made. The learner has successfully learned if it eventually reaches a point beyond which it always correctly predicts.

Obviously, there is a trivial solution. The learner simply keeps track of all examples and their labels provided. If it is fed a vector b the label of which it has already seen, it predicts the correct label. Otherwise, it predicts 0. After having seen all possible examples, it surely can correctly predict. However, there are 2^n Boolean vectors of length n. As a consequence, this trivial algorithm may make many prediction errors.

Exercise 2. Determine the number of prediction errors made by the trivial algorithm in the worst-case.

This number is terribly large. For any n of practical relevance, it may take centuries before the algorithm has achieved its learning goal. Clearly, this is not acceptable. Hence, we continue by asking whether we can do it better. The affirmative answer is provided by the following theorem.

Theorem 2.1. *On-line prediction of monomials over $\mathcal{L}_n$ can be done with at most $n+1$ prediction errors.*

Proof. For all $i = 1, 2, \ldots$ let $b_i = b_i^1 b_i^2 \ldots b_i^n$ denote the ith Boolean vector. First of all, we describe the prediction algorithm $\mathcal{P}$.

```
Algorithm $\mathcal{P}$: "On successive input $\langle b_1, m(b_1), b_2, m(b_2), \ldots\rangle$ do the following:
    Initialize $h_0 = x_1\bar{x}_1 \ldots x_n\bar{x}_n$.
    For $i = 1, 2, \ldots$ let $h_{i-1}$ denote $\mathcal{P}$'s internal prediction hypothesis produced before
    receiving $b_i$. After having received $b_i$ predict $h_{i-1}(b_i)$. Then, read $m(b_i)$.
    If $h_{i-1}(b_i) = m(b_i)$ then $h_i = h_{i-1}$ else
    for $j := 1$ to $n$ do
    if $b_i^j = 1$ then delete $\bar{x}_j$ in $h_{i-1}$ else delete $x_j$ in $h_{i-1}$.
    Denote the result by $h_i$. By convention, if all literals have been removed, then $h_i = \emptyset$,
    and $h_i(b) = 1$ for all $b \in \mathcal{X}$.
    end."
```

Before analyzing the prediction algorithm $\mathcal{P}$ we illustrate it at the following example. Let $n = 7$, and let $m = x_1\bar{x}_2x_4x_7$ be the target monomial. Suppose the input sequence to start as follows: $\langle 1001111,\ 1,\ 0110110,\ 0,\ 1011101,\ 1,\ 1011001,\ 1,\ \ldots\rangle$.

Then, $\mathcal{P}$ initially predicts $h_0(1001111) = 0$. However, the true value is 1, and thus $\mathcal{P}$ executes the loop described above resulting in $h_1 = x_1\bar{x}_2\bar{x}_3x_4x_5x_6x_7$. After having read $b_2 = 0110110$, $\mathcal{P}$ predicts $h_1(0110110) = 0$ which is true. Hence, $h_2 = h_1 = x_1\bar{x}_2\bar{x}_3x_4x_5x_6x_7$. Next, $\mathcal{P}$ reads 1011101, and predicts $h_2(1011101) = 0$ which is wrong.

Thus, $\mathcal{P}$ executes the loop, and computes $h_3 = x_1\bar{x}_2x_4x_5x_7$. Now, $\mathcal{P}$ reads 1011001 and predicts 0 which is again wrong. Therefore, $\mathcal{P}$ executes the loop again, and computes $h_4 = x_1\bar{x}_2x_4x_7$. Since this internal hypothesis equals the target monomial, $\mathcal{P}$ makes no more prediction errors. Consequently, it does not change its internal hypothesis any further.

We proceed by proving the correctness of $\mathcal{P}$. Since $\mathcal{P}$ computes all its internal hypotheses by possibly removing literals from h_0, it suffices to show that all the literals in the target monomial survive and that every prediction error results in removing at least one literal from $\mathcal{P}$'s actual internal hypothesis. This is done via the following claims.

Claim 1. *No literal in the target concept m is ever removed from h_0.*

Suppose the converse, i.e., there is a literal, say ℓ_i, in the target monomial m which is sometimes removed from h_0. Let ℓ_i be the first such literal. Consequently, before this literal ℓ_i is removed, all literals in m are still contained in $\mathcal{P}$'s internal hypothesis h. Thus, every example b satisfying $\mathcal{P}$'s internal hypothesis satisfies m, too, i.e., $h(b) = 1$ implies $m(b) = 1$. Thus, the removal must happen on an example b such that $m(b) = 1$ but $h(b) = 0$. However, while processing its loop $\mathcal{P}$ removes exclusively those literals ℓ_j for which $\ell_j(b^j) = 0$. But for all literals ℓ_k in m we have $\ell_k(b^k) = 1$, and hence they survive. This contradiction proves Claim 1.

Claim 2. *Prediction errors are exclusively made on positive examples.*

By construction, $\mathcal{P}$ performs all of its predictions by its internal hypotheses which constitute monomials. The value of a monomial is 1 if and only if all its literals evaluate to 1. Hence, if $m(b) = 0$, then at least one of its literals must evaluate to 0. By Claim 1, all literals contained in m are contained in all of $\mathcal{P}$'s internal hypotheses, too. Thus, at least one of the literals in $\mathcal{P}$'s actual internal hypothesis also evaluates to 0 causing $\mathcal{P}$ to predict 0. Hence, no prediction error can occur on a negative example.

Claim 3. *Each prediction error causes $\mathcal{P}$ to remove at least one literal from its actual internal hypothesis.*

Suppose the converse, i.e., after some prediction error no literal is removed. Thus, the example on which the prediction error occurred must satisfy all literals in $\mathcal{P}$'s actual internal hypothesis, since otherwise at least one literal is removed. Consequently, $\mathcal{P}$ has predicted 1, but since we had a prediction error, the true value of the target must be 0. Therefore, the prediction error had to occur on a negative example. This contradicts Claim 2.

Finally, we have to prove that at most $n + 1$ prediction errors occur. Initially, $\mathcal{P}$'s internal hypothesis h_0 contains $2n$ literals. However, for each example exactly n literals evaluate to 1 and exactly n literals evaluate to 0. Hence, the first prediction error causes $\mathcal{P}$ to remove precisely n literals from its actual internal hypothesis $h = h_0$. Now, the worst-case obviously occurs if *all* literals must be eliminated from h_0, i.e., if m is the constant monomial 1 (= TRUE). Now, the assertion directly follows by Claim 3. This proves the theorem. ∎

Exercise 3. Construct a sequence of examples such that the worst-case bound stated in Theorem 2.1 happens.

You may test your sequence by using our implementation of Algorithm $\mathcal{P}$ (there called *Wholist Algorithm*) at:
http://www-alg.ist.hokudai.ac.jp/ thomas/BOOLE/menue1.jhtml

As we have seen, when learning the class of all concepts describable by a monomial over the domain $\{0, 1\}^n$ within the on-line prediction model, then $n + 1$ prediction errors may occur in the *worst-case*.

It is also easy to see that two prediction errors are sufficient in the *best-case*.

Moreover, if we run our algorithm $\mathcal{P}$ in practice, we may just observe that usually much less than $n + 1$ prediction errors occur. So, we may ask how many prediction errors do occur during a "typical" run of algorithm $\mathcal{P}$. Clearly, dealing with the latter question also requires to define what is meant by "typical" run. This will directly lead us to the subject of *average-case* analysis.

Note that it is often much more complicated to perform an average-case analysis than to determine the worst-case and (or) best-case behavior of an algorithm. Therefore, we shall come back to this point later. But in the meantime, you can perform some experiments with the on-line prediction algorithms at:
http://www-alg.ist.hokudai.ac.jp/ thomas/BOOLE/menue2.jhtml

As a matter of fact, it is advantageous to introduce another learning model at this point.

2.2. Learning in the Limit

As we have seen, the class of concepts describable by a monomial is learnable in the on-line prediction model. Of course, the class of concepts describable by a monomial is not too difficult. Thus, it would be only natural to ask what other concept classes are learnable within the prediction model. Instead of answering this question right now, we want to take a broader approach and continue with the fundamental notion of learning in the limit. The corresponding theory, called ***inductive inference*** may be viewed as roughly as computational theory is to complexity and analysis of algorithms. Thus, dealing with this part of algorithmic learning theory will provide us a deeper understanding of what can be or cannot be expected from algorithmic learning in general.

Note that induction constitutes an important feature of learning. Inductive inference may be characterized as the study of systems that map evidence on a target concept into hypotheses about it. Investigating scenarios in which the sequence of hypotheses stabilizes to an accurate and finite description of the target concept is of particular interest. Precise definitions of the notions evidence, stabilization, and accuracy go back to Gold [4] who introduced the model of ***learning in the limit***.

Next, we informally describe this model. The source of information are infinite sequences of examples. We distinguish learning from positive data and learning from positive and negative data, synonymously called learning from ***text*** and ***informant***, respectively. A text for a target concept c is an infinite sequence of elements of c such that every element from c appears eventually. Alternatively, an informant is an infinite sequence of elements exhausting the underlying learning domain that are classified with respect to their containment in the target concept.

An algorithmic learner, henceforth called inductive inference machine (abbr. IIM), takes as input larger and larger initial segments of a text (an informant) and outputs, from time to time, a hypothesis about the target concept. The set of all admissible hypotheses is called hypothesis space. The sequence of all hypotheses output is required to converge to a a hypothesis *correctly* describing the target concept.

By the definition of convergence, only finitely many data about the target concept c have been observed by an IIM M at the (unknown) point of convergence. Hence, some form of learning must take place in order for M to learn the target concept c. For this reason, hereafter the terms *identify*, *learn* and *infer* are used interchangeably.

So far we left it open what kind of convergence we are going to require. We shall distinguish between *syntactical* convergence, where the sequence of hypotheses has to stabilize. That is, after some point the IIM always outputs one and the same hypothesis. The resulting learning model is usually called *learning in the limit*.

Alternatively, we shall also consider *semantical* convergence, where the sequence of hypotheses has to behave as follows. After some point, the IIM always outputs hypotheses that are *correct* for the target concept, but which are not necessarily equal to one another. For example, looking again at our class of concepts describable by a monomial, an IIM which is supposed to infer the concept “FALSE” may output many different hypotheses for it, e.g., $x_1\overline{x}_1$, $x_1x_2\overline{x}_1$, $x_1x_2\cdots x_n\overline{x}_1\cdots\overline{x}_n$. Though the resulting learning model also learns in the limit, one usually refers to it as *behaviorally correct learning* in order to distinguish it from the one where syntactical convergence is required.

We continue with a formal introduction of learning in the limit and behaviorally correct learning.

2.2.1. Basic Notations and Definitions

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of all natural numbers and let $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$.

Any recursively enumerable set $\mathcal{X}$ is called a *learning domain*. By $\wp(\mathcal{X})$ we denote the power set of $\mathcal{X}$. Let $\mathcal{C} \subseteq \wp(\mathcal{X})$ and let $c \in \mathcal{C}$. We refer to $\mathcal{C}$ and c as to a *concept class* and a *concept*, respectively. Sometimes, we will identify a concept c with its characteristic function, i.e., we write $c(x) = 1$, if $x \in c$, and $c(x) = 0$, otherwise.

Within this course, we mainly study the learnability of indexable concept classes (cf. Angluin [1]). A class of non-empty concepts $\mathcal{C}$ is said to be an *indexable concept class* iff there are an effective enumeration $(c_j)_{j\in\mathbb{N}}$ of all and only the concepts in $\mathcal{C}$ and a recursive function f such that, for all $j \in \mathbb{N}$ and all $x \in \mathcal{X}$, $f(j, x) = c_j(x)$ holds. By $\mathcal{IC}$ we denote the collection of all indexable classes.

Next, we describe some well-known examples of indexable classes. Let Σ denote any fixed finite alphabet of symbols, and let Σ^* be the free monoid over Σ. We set $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$, where ε denotes the empty string. Then $\mathcal{X} = \Sigma^*$ serves as the learning domain.

As usual, we refer to subsets $L \subseteq \Sigma^*$ as to languages (instead of concepts). Then, the set of all context sensitive languages, context free languages, regular languages, respectively, form indexable classes.

Next, let $X_n = \{0,1\}^n$ be the set of all n-bit Boolean vectors. We consider $\mathcal{X} = \bigcup_{n\geq 1} X_n$ as learning domain. Then, the set of all concepts expressible as a monomial, a k-CNF, a k-DNF, and a k-decision list form indexable classes.

Angluin [1] started the systematic study of learning indexable concept classes. This setting attracted a lot of attention, since many natural concept classes are indexable (see above). For an overview we refer the interested reader to Zeugmann and Lange [6].

Let $c \subseteq \mathcal{X}$ be any concept, and let $t = s_0, s_1, s_2, \ldots$ be any infinite sequence such that $\mathrm{range}(t) = \{s_k \mid\ k \in \mathbb{N}\} = c$. Then t is said to be a ***text*** for c, or synonymously, a ***positive presentation***. By (c) we denote the set of all positive presentations of c. Furthermore, for every indexable class $\mathcal{C}$ we set $text(\mathcal{C}) = \bigcup_{c\in\mathcal{C}} (c)$.

Furthermore, let $i = (s_0, c(s_0)), (s_1, c(s_1)), (s_2, c(s_2)), \ldots$ be any infinite sequence such that $\mathrm{range}(i) = \{s_k \mid\ k \in \mathbb{N}\} = \mathcal{X}$; then we refer to i as an ***informant***, or synonymously as to ***positive and negative data***. By $info(c)$ we denote the set of all informants of c. Furthermore, for every indexable class $\mathcal{C}$ we set $Info(\mathcal{C}) = \bigcup_{c\in\mathcal{C}} info(c)$.

Moreover, let t, i be a text and an informant, respectively, and let $x \in \mathbb{N}$. Then t_x, i_x denote the initial segment of t and of i of length $x + 1$, respectively. Furthermore, we use t_x^+ to denote the range of t_x, and i_x^+ and i_x^- to denote the set of all positive and negative instances in i_x, respectively.

Furthermore, the learner is an algorithmic device which works as follows: It takes as its input larger and larger initial segments of a text t (an informant i) and it either requests the next input string (labeled input string), or first outputs a hypothesis, i.e., a number encoding a certain computer program, and then it requests the next input string (the next labeled example). Note that an IIM, when learning a target class $\mathcal{C}$, is required to produce an output on every initial segment of all texts in $text(\mathcal{C})$.

It is common to refer to the learner as to an ***inductive inference machine*** (abbr. IIM) in this setting. The hypotheses output by an IIM are interpreted with respect to a suitably chosen hypothesis space $\mathcal{H} = (h_j)_{j\in\mathbb{N}}$. When an IIM M outputs some number j, we interpret it to mean that M hypothesizes h_j.

When dealing with indexable concept classes, we usually require the hypothesis space $\mathcal{H}$ to be an indexing of some possibly larger indexable concept class. Hence, in this case, membership is uniformly decidable in $\mathcal{H}$, too. In general, however, we allow any mapping $h\colon \mathcal{X} \to \{0,1\}$ such that the partial characteristic function of it is computable as hypothesis and assume furthermore the set of all hypotheses to be recursively enumerable. So, as a most general hypothesis space one could choose any fixed Gödel numbering φ (sometimes also called *acceptable programming system*) of the recursively enumerable sets over the underlying learning domain.

Furthermore, we shall distinguish the following types of hypothesis spaces. Let $\mathcal{C}$ be any concept class. A hypothesis space $\mathcal{H}$ is said to be *class preserving* for $\mathcal{C}$ iff each concept $c \in \mathcal{C}$ possesses a description h in the hypothesis space, and each description available in the hypothesis space corresponds to a concept in the target concept class. More formally, we can express the requirement for a hypothesis space to be class preserving by $\mathrm{range}(\mathcal{C}) = \mathrm{range}(\mathcal{H})$. Next, we weaken this requirement to $\mathrm{range}(\mathcal{C}) \subseteq \mathrm{range}(\mathcal{H})$, i.e., we allow the hypothesis space to contain descriptions of sets not necessarily corresponding to target concepts. We refer to such hypothesis spaces as to *class comprising* hypothesis spaces. At first glance, it might seem useless to enlarge the hypothesis space, since any hypothesis not describing a target concept is definitely wrong. However, as we shall see later, sometimes it is unavoidable to allow *class comprising* hypothesis spaces in order to achieve superior learners.

Next, we specify the *criterion of success*. Let σ be a text or informant, respectively, and $x \in \mathbb{N}$. Then we use $M(\sigma_x)$ to denote the last hypothesis produced by M when successively fed σ_x. The sequence $(M(\sigma_x))_{x\in\mathbb{N}}$ is said to ***converge in the limit*** to the number j if and only if either $(M(\sigma_x))_{x\in\mathbb{N}}$ is infinite and all but finitely many terms of it are equal to j, or $(M(\sigma_x))_{x\in\mathbb{N}}$ is non-empty and finite, and its last term is j. Now we are ready to define learning in the limit.

Definition 1 (Gold [4]). *Let $\mathcal{C}$ be a concept class, let c be a concept, and let $\mathcal{H} = (h_j)_{j\in\mathbb{N}}$ be a hypothesis space.* ***An IIM M CLimTxt– [CLimInf]–identifies c from text [informant] with respect to $\mathcal{H}$*** *iff for every text t [informant i] for c, there exists a $j \in \mathbb{N}$ such that the sequence $(M(t_x))_{x\in\mathbb{N}}$ $[(M(i_x))_{x\in\mathbb{N}}]$ converges in the limit to j and $c = h_j$.*

Furthermore, M $CLimTxt$– $[CLimInf]$–identifies $\mathcal{C}$ with respect to $\mathcal{H}$ iff, for each $c \in \mathcal{C}$, M $CLimTxt$– $[CLimInf]$–identifies c with respect to $\mathcal{H}$.

Finally, let $CLimTxt$ $[CLimInf]$ denote the collection of all concept classes $\mathcal{C}$ for which there are an IIM M and a hypothesis space $\mathcal{H}$ such that M $CLimTxt$– $[CLimInf]$–identifies $\mathcal{C}$ with respect to $\mathcal{H}$.

In the above definition, Lim stands for limit and the C preceding it points to the fact that we allow class comprising hypothesis spaces. If we just write $LimTxt$ then we additionally require $\mathcal{H}$ to be a class preserving hypothesis space. Moreover, we sometimes write $CLimTxt_{\mathcal{H}}$ to denote the fact that learning has to be performed with respect to the hypothesis space $\mathcal{H}$. We use these conventions for all learning types defined within this course, unless otherwise stated.

Note that, in general, it is not decidable whether or not an IIM M has already inferred a target concept c. Hence, when dealing with learning in the limit, we have a limiting effective criterion of learning instead of an effective one. However, this scenario also reflects some typical aspects of human learning. For example, when humans are learning their mother tongue, usually they are gradually improving their ability to speak and to understand it. These intermediate stages may be viewed as the generation of a sequence of hypotheses that eventually converges to a grammar that is (sufficiently) correct.

Figure 2.1 displays the basic features of the learning in the limit model.

Description at time t:

$$\begin{tikzcd} h_t & \boxed{M} \arrow[l, "\text{Out}"'] & d_t, \ldots, d_0 \arrow[l, "\text{In}"'] \end{tikzcd}$$

M **learns successfully in this model** iff there is a point T such that h_T is **correct** for c and $h_T = h_{T+1} = h_{T+2} = h_{T+3} = \ldots$.

Figure 2.1: Learning in the Limit

The following exercise is intended to help you gaining a first understanding of learning in the limit.

Exercise 4. Prove or disprove.

(a) Let $n \in \mathbb{N}$ be arbitrarily fixed. The set $\mathcal{C}_n$ of all monomials over $\mathcal{L}_n$ can be $CLimInf$–identified with respect to the set of all monomials as hypothesis space. Does your answer remain valid for $CLimTxt$?

(b) Let $\mathcal{X} = \bigcup_{n\in\mathbb{N}}\{0,1\}^n$, let $\mathcal{C} = \bigcup_{n\in\mathbb{N}} \mathcal{C}_n$ be the set of all monomials over $\bigcup_{n\in\mathbb{N}} \mathcal{L}_n$. Then $\mathcal{C}$ is $CLimInf$–identifiable with respect to $\mathcal{C}$. Does your answer remain valid for $CLimTxt$?

(c) Let $\mathcal{X}$ be any learning domain, let $\mathcal{C}$ be any concept class over $\mathcal{X}$, and let $\mathcal{H}$ be any hypothesis space for $\mathcal{C}$ such that each $h \in \mathcal{H}$ is recursive. Then,

(α) $\mathcal{C} \in CLimInf$ implies that $\mathcal{C}$ is on-line predictable.

(β) If $\mathcal{C}$ is on-line predictable then $\mathcal{C} \in CLimInf$.

Next, we formally define *behaviorally correct learning*. For the sake of presentation, this is only done for learning from positive data.

Definition 2 (Bārzdiņš [2], Case and Lynes [3]). *Let $\mathcal{C}$ be a concept class, let c be a concept, and let $\mathcal{H} = (h_j)_{j\in\mathbb{N}}$ be a hypothesis space. An IIM M **$CBcTxt$–identifies** c with respect to $\boldsymbol{\mathcal{H}}$ iff for every text $t \in (c)$ and for all but finitely many $y \in \mathbb{N}$, $h_{M(t_y)} = c$.*

Furthermore, M $CBcTxt$–identifies $\mathcal{C}$ with respect to $\mathcal{H}$ iff M $CBcTxt$–identifies each $c \in \mathcal{C}$ with respect to $\mathcal{H}$.

Finally, let $CBcTxt$ denote the collection of all concept classes $\mathcal{C}$ for which there are an IIM M and a hypothesis space $\mathcal{H}$ such that M $CBcTxt$–identifies $\mathcal{C}$ with respect to $\mathcal{H}$.

If you compare Definitions 1 and 2 then you see that the only difference is the kind of convergence required. That is, Definition 1 requires syntactical concergence while Definition 2 weakens this requirement to *semantical* convergence.

Exercise 5. *Formally define behaviorally correct learning from informant.*

Finally, we refer the interested reader to Osherson, Stob and Weinstein [5] for a comprehensive treatment of learning recursively enumerable languages in the limit. The learnability of language classes with uniformly decidable membership is surveyed in Zeugmann and Lange [6].

References

[1] D. ANGLUIN (1980), Inductive inference of formal languages from positive data, *Information and Control* **45**, No. 2, 117–135.

[2] J. BĀRZDIŅŠ (1974), Two theorems on the limiting synthesis of functions, In *Theory of Algorithms and Programs Vol. 1*, pages 82–88, Latvian State University, (in Russian).

[3] J. CASE AND C. LYNES (1982), Machine inductive inference and language identification, *in* Automata, Languages and Programming, 9th Colloquium, Proceedings, volume 140 of *Lecture Notes in Computer Science*, pages 107–115. Springer-Verlag.

[4] E.M. GOLD (1967), Language identification in the limit, *Information and Control* **10**, 447 – 474.

[5] D. OSHERSON, M. STOB, AND S. WEINSTEIN (1986), *Systems that Learn, An Introduction to Learning Theory for Cognitive and Computer Scientists*, The MIT-Press, Cambridge, MA.

[6] T. ZEUGMANN AND S. LANGE (1995), A guided tour across the boundaries of learning recursive languages, *in* “Algorithmic Learning for Knowledge-Based Systems” (K.P. Jantke and S. Lange, eds.), Lecture Notes in Artificial Intelligence 961, pp. 190 – 258, Springer-Verlag, Berlin.

LECTURE 3: MORE ABOUT LEARNING IN THE LIMIT

We finished the last lecture by introducing learning in the limit. As far as learning of indexed families from informant is concerned, a very general result can be obtained which we present here.

Theorem 3.1 (Gold [5]). *$\mathcal{C} \in LimInf$ for all $\mathcal{C} \in \mathcal{IC}$.*

Proof. Let $\mathcal{X}$ be the underlying learning domain. Since $\mathcal{C} \in \mathcal{IC}$, there are an indexing $(c_j)_{j\in\mathbb{N}}$ of all and only the concepts in C and a recursive function f such that $f(j,x) = c_j(x)$ for all $j \in \mathbb{N}$ and all $x \in \mathcal{X}$. Now, let $c \in \mathcal{C}$ be any concept and let $i = ((s_k, c(s_k)))_{k\in\mathbb{N}} \in info(c)$ be any informant for c.

The desired IIM M learns by the so-called *identification by enumeration* principle which is formally defined as follows.

Let $y \in \mathbb{N}$. On input i_y we set:

$M(i_y) =$ "Search the least $k \in \mathbb{N}$ such that $(s_j, c(s_j)) = f(k, s_j)$ for all $j = 0, \dots, y$."

Since $c \in \mathcal{C}$ and $(c_j)_{j\in\mathbb{N}}$ is an indexing for $\mathcal{C}$ with the above properties for f, we see that the test $(s_j, c(s_j)) = f(k, s_j)$ is recursive for all $k \in \mathbb{N}$ and all $j \in \mathbb{N}$. Furthermore, let k^* be the least $k \in \mathbb{N}$ such that $c = c_k$. Consequently, $(s_j, c(s_j)) = f(k^*, s_j)$ for all $j \in \mathbb{N}$ and hence, the search performed by M will always terminate. That is, all hypotheses k output by M satisfy $k \le k^*$. Moreover, $c_\ell \neq c$ for all $\ell < k^*$. Therefore, for every $\ell < k^*$ there must be a y_ℓ such that $(s_{y_\ell}, c(s_{y_\ell})) \neq f(\ell, s_{y_\ell})$. Hence, for $y \ge \max\{y_\ell |\ \ell < k^*\}$ the IIM M cannot output any $\ell < k^*$. Thus, for all $y \ge \max\{y_\ell |\ \ell < k^*\}$ it outputs k^* and thus converges to k^*. Since $c = c_{k^*}$, M learns c as required.

Consequently, M $LimInf$–identifies $\mathcal{C}$ with respect to $(c_j)_{j\in\mathbb{N}}$ and we are done. ▮

The learning method used by the IIM defined in the proof of Theorem 3.1, i.e., identification by enumeration, has some interesting and important properties which we want to discuss shortly.

First, it is consistent. Here by ***consistency*** we mean that $i_x^+ \subseteq c_k$ and $i_x^- \cap c_k = \emptyset$ whenever k is output by M. We shall discuss the consistency phenomenon in some detail below.

Second, any *mind change* performed by M is justified by a "provable misclassification" of its previous guess. Here, by mind change we mean that $M(i_y) \neq M(i_{y+1})$. Therefore, M will never reject a guess that is correct for the concept to be learned. Hence, M is *semantical finite*.

Third, M is *set-driven*, too, i.e., its output exclusively depends on the range of its input (cf. Definition 5 below).

Finally, identification by enumeration is the most efficient learning method with respect to learning time, i.e., the first time such that M outputs a correct guess that will be repeated in every subsequent learning step (cf. Gold 5). More precisely, Gold 5 proved that there is no IIM $\hat{M}$ inferring $\mathcal{L}$ which is *uniformly faster* than the IIM M described above with respect to learning time. Hence, in the setting of learning indexed families identification by enumeration is particularly tailored for learning from informant.

3.1. The Weakness of Text

First, we present a very simple example for a concept class that is not in $LimTxt$.

Example 1. *The following language class $\mathcal{L}$ cannot be learned in the limit from positive data with respect to the hypothesis space $\mathcal{L}$.*

We define the language class $\mathcal{L}$ by the following enumeration of languages over the alphabet $\{a\}$. Let $\mathcal{L} = (L_j)_{j\in\mathbb{N}}$ where $L_0 = \{a\}^+$ and $L_j = \{a^m \mid \ 1 \le m \le j\}$ for all $j \in \mathbb{N}^+$. We show that $\mathcal{L}$ is not learnable in the limit from positive data. Suppose the converse, i.e., there is an IIM M which witnesses $\mathcal{L} \in CLimTxt$ with respect to $\mathcal{L}$. Then, M in particular has to identify the language L_1 on its uniquely defined text t. Hence, there exists an $x \in \mathbb{N}$ such that $M(t_x) = 1$. Obviously, it is possible to extend t_x in order to obtain a text for the infinite language L_0. Namely, we may choose the text $t_x \cdot \hat{t}$ where $\hat{t}$ is the lexicographically ordered text of L_0. That is, $\hat{t} = a,\ aa,\ aaa,\ aaaa,\ \ldots$. Since M has to infer L_0 from this text, too, M is forced to change its mind to the hypothesis 0. Therefore, there is a $y \in \mathbb{N}^+$ such that $M(t_x \cdot \hat{t}_y) = 0$. But now we may conclude that $t_x \cdot \hat{t}_y$ is an initial segment of a text $\tilde{t}$ for the finite language L_y. Consequently, M has to perform one more mind change when successively fed $\tilde{t}$. By iterating this idea one may effectively construct a text for the infinite language L_0 on which M has to change its mind infinitely often, a contradiction. Hence $\mathcal{L} \notin CLimTxt$ with respect to $\mathcal{L}$. ▮

Exercise 6. *Prove or disprove: The language class $\mathcal{C} = range(\mathcal{L})$, where $\mathcal{L}$ is defined as above, cannot be learned in the limit with respect to every hypothesis space $\mathcal{H}$ for it.*

This negative result is mainly caused by the problem that both finite and infinite languages have to be simultaneously handled.

We continue with Gold's [5] famous result.

Theorem 3.2 (Gold [5]). *Let $\mathcal{L}$ be any class of languages containing all finite languages and at least one infinite language L. Then $\mathcal{L} \notin CLimTxt$.*

Proof. Assume any hypothesis space $\mathcal{H}$ for $\mathcal{L}$, and suppose there is an IIM M learning $\mathcal{L}$ in the limit with respect to $\mathcal{H}$. We are going to construct a text t for the infinite language L on which M does not converge in the limit. First, consider any text $\hat{t} = s_0,\ s_1,\ s_2,\ \ldots$ for L. Since we have M supposed to learn L, there must be an x such that $j_0 = M(t_x)$ fulfills $L = L(h_{j_0})$. Set $t_x = \hat{t}_x$. Clearly, $\hat{t}_x$ is finite, and thus $\hat{t}_x$ is an initial segment of the finite language $L_1 = \hat{t}_x^+$. Since M has to learn the finite language L_1, too, the initial segment $\hat{t}_x$ can be extended by $s_0,\ s_0\, s_0,\ \ldots$ resulting in a text $\tilde{t}$ for L_1. Thus, there must be a y such that $j_1 = M(\tilde{t}_{x+y})$ satisfies $L_1 = L(h_{j_1})$. Note that $\tilde{t}_{x+y}$ is an initial segment extending $\hat{t}_x$ such that $\tilde{t}_{x+y}^+ \subseteq L$. Thus, we may continue the definition of t by setting $t_{x+y} = t_x\ \underbrace{s_0, \ldots, s_0}_{y \text{ times}}$.

Now, we continue by extending $\tilde{t}_{x+y}$ by $\hat{t}$. Again, there must be $z > x + y$ such that M, when fed $\tilde{t}_{x+y} \cdot \hat{t}_z$ outputs a hypothesis j_2 such that $L = L(h_{j_2})$. Thus, we continue the definition of t as follows: $t_{x+y+z} = \hat{t}_x \underbrace{s_0, \ldots, s_0}_{y \text{ times}} \hat{t}_z$. Noting that t^+_{x+y+z} is again a finite language, it is obvious how to iterate this procedure. The requirement $z > x + y$ ensures that t will be indeed a text for L, and by construction, M changes its hypotheses infinitely often when successively fed t, a contradiction. ∎

The latter theorem allows the following corollary.

Corollary 3.3. *Neither the class of all regular languages nor any superset thereof can be learned in the limit from positive data.*

Taking the latter corollary into account, many researchers thought that there is no interesting class of languages at all that can be learned from positive data. As a result, the study of learning from positive data faced more than a decade of decline after Gold's [5] pioneering paper. In order to gain a better understanding why this has been the case, you should solve the following exercise.

Exercise 7. *Prove or disprove: Let $\mathcal{L}$ be any class of languages containing all finite languages and at least one infinite language L. Then $\mathcal{L} \notin CBcTxt$.*

But the situation considerably changed when Angluin [1] showed that the pattern languages are learnable in the limit from text. Therefore, we continue with the pattern languages which we have already met in Lecture 1.

3.2. The Pattern Languages

Patterns are a very natural way to define formal languages. Suppose you are interested in the language of all strings over the alphabet $\mathcal{A} = \{0, 1\}$ starting with 11, ending with 010, and containing the substring 01011 somewhere, but may be otherwise arbitrary. Thus, all strings in your language follow the pattern $\pi_1 = 11x_0 01011 x_1 010$, provided you are willing to allow the variables x_0, x_1 to be substituted by any string over $\{0, 1\}$ including the empty one. As for another example, consider the set of all strings having even length $2n$ such that the prefix of length n is identical to the suffix starting at position $n + 1$. In that case, the wanted language follows the pattern $\pi_2 = x_0 x_0$.

Though patterns have already been considered since the beginning of this century (cf., e.g., Thue [15], and Bean *et al.* [3]), the formal introduction of patterns and pattern languages goes back to Angluin [1]. Since then, pattern languages and variations thereof have been widely investigated (cf., e.g., Salomaa [12, 13], and Shinohara and Arikawa [14] for an overview).

Following Angluin [1] we define patterns and pattern languages as follows. Let $\mathcal{A} = \{0, 1, \ldots\}$ be any non-empty finite alphabet containing at least two elements. By $\mathcal{A}^*$ we denote the free monoid over $\mathcal{A}$ (cf. Hopcroft and Ullman [6]). The set of all finite non-null strings of symbols from $\mathcal{A}$ is denoted by $\mathcal{A}^+$, i.e., $\mathcal{A}^+ = \mathcal{A}^* \setminus \{\epsilon\}$, where ϵ denotes the empty string. By $|\mathcal{A}|$ we denote the cardinality of $\mathcal{A}$. Furthermore, let $X = \{x_i \mid i \in \mathbb{N}\}$

be an infinite set of variables such that $\mathcal{A} \cap X = \emptyset$. *Patterns* are non-empty strings from $\mathcal{A} \cup X$, e.g., $01,\ 0x_0111,\ 1x_0x_00x_1x_2$ are patterns. The length of a string w and of a pattern π is denoted by $|w|$ and $|\pi|$, respectively. A pattern π is in *canonical form* provided that if k is the number of different variables in π then the variables occurring in π are precisely $x_0, \dots, x_{k-1}$. Moreover, for every j with $0 \leq j < k-1$, the leftmost occurrence of x_j in π is left to the leftmost occurrence of x_{j+1} in π. The examples given above are patterns in canonical form. In the sequel we assume, without loss of generality, that all patterns are in canonical form. By Pat we denote the set of all patterns in canonical form. Let $\pi \in Pat$, $1 \leq i \leq |\pi|$; we use $\pi(i)$ to denote the i-th symbol in π. If $\pi(i) \in \mathcal{A}$, then we refer to $\pi(i)$ as to a constant; otherwise $\pi(i) \in X$, and we refer to $\pi(i)$ as to a variable. By $\#\text{var}(\pi)$ we denote the number of different variables occurring in π, and by $\#_{x_i}(\pi)$ we denote the number of occurrences of variable x_i in π. If $\#\text{var}(\pi) = k$, then we refer to π as a *k-variable pattern*. Let $k \in \mathbb{N}$, by Pat_k we denote the set of all *k-variable patterns*. Furthermore, let $\pi \in Pat_k$, and let $u_0, \dots, u_{k-1} \in \mathcal{A}^+$; then we denote by $\pi[x_0 : u_0, \dots, x_{k-1} : u_{k-1}]$ the string $w \in \mathcal{A}^+$ obtained by substituting u_j for each occurrence of x_j, $j = 0, \dots, k-1$, in the pattern π. The tuple $(u_0, \dots, u_{k-1})$ is called *substitution*. Furthermore, if $|u_0| = \cdots = |u_{k-1}| = 1$, then we refer to $(u_0, \dots, u_{k-1})$ as to a *shortest substitution*. Now, let $\pi \in Pat_k$, and let $S = \{(u_0, \dots, u_{k-1}) |\ u_j \in \mathcal{A}^+, j = 0, \dots, k-1\}$ be any finite set of substitutions. Then we set $S(\pi) = \{\pi[x_0 : u_0, \dots, x_{k-1} : u_{k-1}] |\ (u_0, \dots, u_{k-1}) \in S\}$, i.e., $S(\pi)$ is the set of all strings obtained from pattern π by applying all the substitutions from S to it. For every $\pi \in Pat_k$ we define the *language generated by pattern* π by $L(\pi) = \{\pi[x_0 : u_0, \dots, x_{k-1} : u_{k-1}] |\ u_0, \dots, u_{k-1} \in \mathcal{A}^+\}$.

By PAT_k we denote the set of all *k-variable pattern languages*. Finally, $PAT = \bigcup_{k \in \mathbb{N}} PAT_k$ denotes the set of all pattern languages over $\mathcal{A}$. Note that for every $L \in PAT$ there is precisely one pattern $\pi \in Pat$ such that $L = L(\pi)$ (cf. Angluin [1]).

If we take a look back to the patterns considered in Lecture 1, we that they have a peculiarity, i.e., each variable occurs at most once. Such pattern are called ***regular patterns***. the following exercise sheds some more light on this naming.

Exercise 8. *Prove or disprove:*

(1) *If $\pi \in Pat$ is regular, then $L(\pi)$ is a regular language.*

(2) *If π is not regular, then $L(\pi)$ is necessarily a non-regular language.*

The pattern languages constitute an indexable class. Clearly, one can recursively enumerate all canonical patterns, say by $\pi_0,\ \pi_1,\ \pi_2,\ \dots$. Moreover, given any pattern π and a string s it suffices to check all substitutions having a length not exceeding $|s|$ until either s is generated by π or all substitutions have been tested. Thus, $(L(\pi_j))_{j \in \mathbb{N}}$ is indeed an indexable class.

Angluin [1] showed that PAT is learnable from text with respect to the hypothesis space Pat. The key idea of her proof was to observe that it suffices to find descriptive patterns which we define next.

Definition 3. *Let $\mathcal{A}$ be any finite alphabet.*

(1) *A finite set* $\{s_0,\, s_1,\, \ldots, s_r\} \subseteq \mathcal{A}^+$ *of strings is called a* ***sample***.

(2) *A pattern* π *is* ***consistent*** *with a sample* S *if* $S \subseteq L(\pi)$.

(3) *A pattern* π *is said to be* ***descriptive*** *for* S *if it is consistent and there is no other consistent pattern* τ *such that* $L(\tau) \subset L(\pi)$.

The following structural property of indexable classes is important with respect to their learnability from text. This property has been first observed for the class of all pattern languages, but its importance justifies the following definition.

Definition 4. *Let* $\mathcal{X}$ *be any learning domain and let* $\mathcal{C}$ *be any indexable class.* $\mathcal{C}$ *is said to have* ***finite thickness*** *provided for each* $S \subseteq \mathcal{X}$ *the set* $\{c \mid\ S \subseteq c \ \text{ and } \ c \in \mathcal{C}\}$ *is of finite cardinality.*

Now, let us check whether or not PAT indeed has finite thickness.

Lemma 3.4. *PAT has finite thickness.*

Proof. Let $S \subseteq \mathcal{A}^*$ be any set, and let $m = \min\{|s| \mid\ s \in S\}$. Since we restricted ourselves to nonempty substitutions, we can immediately conclude that for every pattern $\pi \in Pat$, $|\pi| > m$ implies $S \not\subseteq L(\pi)$. But $\{\pi \mid\ \pi \in Pat,\ |\pi| < m\}$ is finite, and thus, $\{\pi \mid\ \pi \in Pat,\ S \subseteq L(\pi)\}$ must be finite, too. ▮

Corollary 3.5. *For every set* $S \subseteq \mathcal{A}^*$ *there is always at least one descriptive pattern.*

Proof. For every set S we have $S \subseteq L(x_0)$. Thus, the set of all consistent patterns for S is always nonempty. Since *PAT* has finite thickness, the set of all consistent patterns for S is finite, too. Consequently, there must be a minimal element (with respect to set-containment) in the set of all consistent patterns for S. ▮

Next, we establish the learnability of the set of all pattern languages from text.

3.2.1. Learning the Pattern Languages via Descriptive Patterns

We continue with the following important theorem.

Theorem 3.6. *Assume any recursive subprocedure* ***des*** *computing descriptive patterns for every set* $S \subseteq \mathcal{A}^*$. *Then,* $PAT \in CLimTxt$.

Proof. Let $L \in PAT$, let $t = (s_j)_{j\in\mathbb{N}} \in text(L)$, and $x \in \mathbb{N}$. We define the wanted IIM as follows:

```
$M(t_x) =$ “If $x = 0$ compute $\pi = \boldsymbol{des}(\{s_0\})$, and output it.
    If $x > 0$, let $\tau = M(t_{x-1})$. If $t_x^+ \subseteq L(\tau)$, output $\tau$.
    Otherwise, compute $\pi = \boldsymbol{des}(t_x^+)$, and output it.”
```

Since the subprocedure ***des*** is recursive, M is indeed an IIM. It remains to show that M infers L. Let $\pi \in Pat$ be the unique pattern satisfying $L = L(\pi)$. Clearly, π is consistent with t_x^+ for every $x \in \mathbb{N}$. Now, consider any τ output by M. If $\tau = \pi$ we are done, since π will be then output on t_{x+r} for every $r \geq 0$. Suppose, $\tau \neq \pi$. Since τ is descriptive, we already know that $L(\pi) \not\subset L(\tau)$. Hence, $L(\pi) \setminus L(\tau) \neq \emptyset$. But that means, there must be at least one string $s \in L(\pi) \setminus L(\tau)$ which has to appear sometimes in t, say at position x_0. Thus, $M(t_{x_0}) \neq \tau$. Now, taking into account that there are only finitely many patterns that are consistent with s_0, we see that every mind change does shrink the set of descriptive patterns by at least one element. Since every descriptive pattern but π has to be abandoned as shown above, M has to converge to π. ▮

Thus, for establishing the learnability of all pattern languages from text, we finally have to prove that there is a procedure ***des*** computing descriptive patterns. This is done next. The main problem we have to handle is how to figure out whether or not $L(\pi) \subset L(\tau)$ for any two patterns π and τ that are consistent with some given sample S. For that purpose, we need the following definitions and notations.

Let Hom be the set of all non-erasing homomorphism with respect to concatenation of *Pat* to itself. An element of Hom that is the identity when restricted to $\mathcal{A}$ is called ***substitution***. Next, we define a binary relation $\preceq$ over *Pat* by $\pi \preceq \tau$ iff $\pi = f(\tau)$ for some substitution f. The intuitive meaning of $\preceq$ may be taken to be "is less general than."

Example. *Let $\mathcal{A} = \{0, 1\}$, and let $\pi = x_0 0 x_0 01$ and $\tau = x_1 x_1 x_0$. Then $\pi \preceq \tau$.*

For seeing this, we define a substitution f by $f(x_0) = 1$ and $f(x_1) = x_0 0$ (remember, $f(0) = 0$ as well as $f(1) = 1$). Thus $f(\tau) = x_0 0 x_0 01 = \pi$.

Exercise 9. *Prove that for all π, $\tau \in Pat$,*

(1) $\preceq$ *is transitive,*

(2) $\pi \preceq \tau$ *implies* $L(\pi) \subseteq L(\tau)$,

(3) $L(\pi) = L(\tau)$ *iff* $\pi \preceq \tau$ *and* $\tau \preceq \pi$,

(4) $\preceq$ *is computable.*

Now, we are ready to establish the following theorem.

Theorem 3.7. *There is a recursive procedure* ***des*** *which, for every sample S given as input, outputs a descriptive pattern π for S.*

Proof. The procedure ***des*** is defined as follows. Let S be any sample, and let $\ell = \min\{|s| \mid s \in S\}$. Enumerate the finitely many canonical patterns π satisfying $|\pi| \leq \ell$. For every pattern enumerated check whether or not $S \subseteq L(\pi)$. Let C be the set of all enumerated patterns successfully passing this test. Let $m = \max\{|\pi| \mid \pi \in C\}$, and let $\hat{C} = \{\pi \mid \pi \in C,\ |\pi| = m\}$. Find and output any element π in $\hat{C}$ that is minimal with respect to $\preceq$.

We have to show that π is descriptive for S. Let τ be any consistent pattern for S. Then $|\tau| \leq \ell$, and if $|\tau| < |\pi|$, then $L(\tau) \not\subset L(\pi)$, too. Now, assume $|\tau| = |\pi|$. Consequently, $\tau \in \hat{C}$. By construction, either $\tau = \pi$ or $\tau \not\preceq \pi$. If $\tau = \pi$, we are done. The proof is finished by showing the following claim actually stating that the second case cannot happen.

Claim. Let $\pi,\ \tau \in Pat$ *such that* $|\pi| = |\tau|$. *Then* $L(\tau) \subseteq L(\pi)$ *implies* $\tau \preceq \pi$.

Let $0,\ 1 \in \mathcal{A}$ be distinct. Consider the substitutions $f_0(x_i) = 0$, and $f_1(x_i) = 1$ for all $i \in \mathbb{N}$ as well as the substitutions g_j defined for all $i\,, j \in \mathbb{N}$ as follows:

$$g_j(x_i) = \begin{cases} 0, & \text{if} \quad i = j, \\ 1, & \text{otherwise.} \end{cases}$$

Let $S(\pi)$ consist of the set of all strings $\{f_0(\pi),\ f_1(\pi),\ g_0(\pi),\ g_1(\pi),\ g_2(\pi),\ \ldots\}$, and define $S(\tau)$ analogously. If $\#\text{var}(\pi) = 0$, then $S(\pi) = L(\pi) = \{\pi\}$, and we are already done. If $\#\text{var}(\pi) = 1$, then $S(\pi) = \{f_0(\pi),\ f_1(\pi)\}$, and in the general case, i.e., if $\#\text{var}(\pi) = k \geq 2$, then $|S(\pi)| = k + 2$.

Furthermore, by construction we obtain: For all $1 \leq m,\ n \leq |\pi|$:

(1) if $\pi(m) = c \in \mathcal{A}$, then $s(m) = c$, for all $s \in S(\pi)$, and the same holds for τ,

(2) if $\pi(m),\ \pi(n) \in X$ such that $\pi(m) \neq \pi(n)$, then there is a string $s \in S(\pi)$ such that $s(m) \neq s(n)$, and again, the same holds for τ.

Now, taking into account that $L(\tau) \subseteq L(\pi)$, for every $s \in S(\tau)$ there exists a substitution h such that $s = h(\pi)$. But $|s| = |\tau| = |\pi|$; thus h must map each variable in π into a string of length one. Let $1 \leq m,\ n \leq |\pi|$. If $\pi(m) = c \in \mathcal{A}$, then $s = h(\pi)$ must fulfill $s(m) = c$, and since this holds for all $s \in S(\tau)$, we already know $\tau(m) = c$. If $\pi(m) = \pi(n) = x_i \in X$, then $s = h(\pi)$ must satisfy $s(m) = s(n) \in \mathcal{A}$, and since this holds for all $s \in S(\tau)$, we conclude $\tau(m) = \tau(n)$. Thus, the set of positions of x_i are all occupied by the same symbol, say $g(x_i)$, in τ. Hence, we may extend g to be a substitution such that $\tau = g(\pi)$, and therefore, $\tau \preceq \pi$. ▮

Now the principal learnability of all pattern languages in the limit from positive data has been established.

Note that Angluin's [1] learner M as described above has two additional properties. First, it is ***consistent***. Here consistency means that for all texts t of pattern languages, and $x \in \mathbb{N}$ we have, if $\pi = M(t_x)$, then $t_x^+ \subseteq L(\pi)$. Moreover, it is ***set-driven***, where set-drivenness is formally defined as follows:

Definition 5. *Let $\mathcal{L}$ be any indexable class. An IIM is said to be **set-driven** with respect to $\mathcal{L}$ iff its output depends only on the range of its input; that is, iff $M(t_x) = M(\hat{t}_y)$ for all $x, y \in \mathbb{N}$ and all texts $t,\ \hat{t} \in \bigcup_{L\in\mathcal{L}} Text(L)$ provided $t_x^+ = \hat{t}_y^+$.*

Note that in general one cannot expect to learn set-drivenly. For more information concerning this subject the reader is referred to Lange and Zeugmann [8].

Actually, our proof for the learnability of the pattern languages from text was a bit more complicated than necessary. The solution to the following exercise will establish an easier proof.

Exercise 10. *Let $\mathcal{C} \in \mathcal{IC}$ such that $\mathcal{C}$ has finite thickness. Then $\mathcal{C} \in CLimTxt$.*

Of course, the next issue one should study is the complexity of learning the pattern languages. Unfortunately, due to the lack of time, we have to skip this issue. Instead, we take a short look at another branch of learning in the limit, i.e., function learning.

3.3. Learning Recursive Functions

Next, we want to adapt our model to *function learning*. Here, the objects to be learned are classes U of recursive functions. Following a general convention in recursion theory, by *recursive functions* we usually mean the total recursive functions synonymously called *general recursive functions*.

The source of information are then growing sequences of the graph of the target function, i.e.,

$$\langle (x_0, f(x_0)),\ (x_1, f(x_1)),\ (x_2, f(x_2)),\ \ldots \rangle$$

where we require $\{x_n \mid n \in \mathbb{N}\} = \mathbb{N}$.

As hypothesis space we can choose all programs in a universal programming language (e.g. JAVA), or more formally natural numbers which are then interpreted as *encodings* of such programs (recall a universal Turing machine, a universal RAM, ...). Alternatively, we can also use any restricted programming language as long as we can we can write at least one program for every function to be learned. Below, we shall formalize this idea by the notion of *numbering*.

The sequence of all computed hypotheses has then to *converge* to a *correct* program for the target function f.

Figure 3.1 displays the basic features of the function learning in the limit model.

Description at time t:

$$h_t \xleftarrow{\text{Out}} \boxed{M} \xleftarrow{\text{In}} \langle (x_t, f(x_t)), \ldots, (x_0, f(x_0)) \rangle$$

M **learns successfully in this model** iff there is a point T such that h_T is **correct** for f and $h_T = h_{T+1} = h_{T+2} = h_{T+3} = \ldots$.

Figure 3.1: Function Learning in the Limit

In order to arrive at a formal definition, some more notations are needed. We denote the set of all finite sequences of natural numbers by $\mathbb{N}^*$. The classes of all partial recursive and recursive functions of one, and two arguments over $\mathbb{N}$ are denoted by $\mathcal{P},\ \mathcal{P}^2,\ \mathcal{R}$, and $\mathcal{R}^2$, respectively. $\mathcal{R}_{0,1}$ denotes the set of all $0-1$ valued recursive functions. Sometimes it will be suitable to identify a recursive function with the sequence of its values, e.g., let $\alpha = (a_0, ..., a_k) \in \mathbb{N}^*,\ j \in \mathbb{N},\ \text{and}\ p \in \mathcal{R}_{0,1}$; then we write αjp to denote the function f for which $f(x) = a_x$, if $x \leq k$, $f(k+1) = j$, and $f(x) = p(x-k-2)$, if $x \geq k+2$.

Any function $\psi \in \mathcal{P}^2$ is called a numbering. Moreover, let $\psi \in \mathcal{P}^2$, then we write ψ_i instead of $\lambda x\psi(i,x)$ and set $\mathcal{P}_\psi = \{\psi_i \mid\ i \in \mathbb{N}\}$ as well as $\mathcal{R}_\psi = \mathcal{P}_\psi \cap \mathcal{R}$. Consequently, if $f \in \mathcal{P}_\psi$, then there is a number i such that $f = \psi_i$. If $f \in \mathcal{P}$ and $i \in \mathbb{N}$ are such that $\psi_i = f$, then i is called a ψ–program for f. A numbering $\varphi \in \mathcal{P}^2$ is called a Gödel numbering (cf. Rogers [11]) iff $\mathcal{P}_\varphi = \mathcal{P}$, and for any numbering $\psi \in \mathcal{P}^2$, there is a $c \in \mathcal{R}$ such that $\psi_i = \varphi_{c(i)}$ for all $i \in \mathbb{N}$. *Göd* denotes the set of all Gödel numberings.

Using a fixed encoding $\langle \ldots \rangle$ of $\mathbb{N}^*$ onto $\mathbb{N}$ we write f^n instead of $\langle (f(0), \ldots, f(n)) \rangle$, for any $n \in \mathbb{N},\ f \in \mathcal{R}$. Furthermore, the set of all permutations of $\mathbb{N}$ is denoted by $\Pi(\mathbb{N})$. Any element $X \in \Pi(\mathbb{N})$ can be represented by a unique sequence $(x_n)_{n\in\mathbb{N}}$ that contains each natural number precisely ones. Let $X \in \Pi(\mathbb{N}), f \in \mathcal{P}$ and $n \in \mathbb{N}$. Then we write $f^{X,n}$ instead of $\langle (x_0, f(x_0), \ldots, x_n, f(x_n)) \rangle$ provided $f(x_k)$ is defined for all $k \leq n$. Finally, a sequence $(j_n)_{j\in\mathbb{N}}$ of natural numbers is said to ***converge*** to the number j iff all but finitely many numbers of it are equal to j.

Now we are ready to define learning in the limit for functions.

Definition 6 (Gold [5]). *Let $U \subseteq \mathcal{R}$ and let $\psi \in \mathcal{P}^2$. The class U is said to be **learnable in the limit** with respect to ψ iff there is an IIM $M \in \mathcal{P}$ such that for each function $f \in U$ and every $X \in \Pi(\mathbb{N})$*

(1) *for all $n \in \mathbb{N}$, $M(f^{X,n})$ is defined,*

(2) *there is a $j \in \mathbb{N}$ such that $\psi_j = f$ and the sequence $(M(f^{X,n}))_{n\in\mathbb{N}}$ converges to j.*

If U is learnable in the limit with respect to ψ by an IIM M, we write $U \in LIM_\psi^{arb}(M)$. Let $LIM_\psi^{arb} = \{U \mid\ U \text{ is learnable in the limit w.r.t. } \psi\}$, and let $LIM^{arb} = \bigcup_{\psi\in\mathcal{P}^2} LIM_\psi^{arb}$.

Some remarks are mandatory here. Let us start with the semantics of the hypotheses produced by an IIM M. If M is defined on input $f^{X,n}$, then we always interpret the number $M(f^{X,n})$ as a ψ–number. This convention is adopted to all the definitions below.

Furthermore, note that $LIM_\varphi^{arb} = LIM^{arb}$ for every Gödel numbering φ. In the above definition LIM stands for “limit.” Moreover, in accordance with the definition of convergence, only finitely many data of the graph of a function f were available to the IIM M up to the unknown point of convergence. Therefore, some form of learning must have taken place. Thus, the use of the term “learn” in the above definition is indeed justified.

Moreover, the arb in LIM^{arb} points to the requirement to learn from arbitrary input. That is, within Definition 6 we make no assumptions concerning the *order* in which input data should be presented.

For the sake of illustration, let us look at the following example. Consider the class U of all polynomials of one variable with coefficients from $\mathbb{N}$. For constructing an appropriate hypothesis space ψ we can proceed by canonically enumerating for $d = 0, 1, 2, \ldots$ the finite number of $(d + 1)$ tuples of natural numbers in the range $[0, d]$ and using them as the coefficients of $1,\, x,\, x^2,\, \ldots,\, x^d$. One such enumeration begins

$$\begin{gathered}0,\ 1 + x,\ x,\ 0,\ 1\\ 2 + 2x + 2x^2,\ 2 + 2x + x^2,\ \ldots\end{gathered}$$

Then, one could learn the class U of all polynomials of one variable with coefficients from $\mathbb{N}$ by using again the *identification by enumeration principle*. But there is another method which is much more efficient (though not universally faster).

On input

$$\langle (x_0, f(x_0)),\ (x_1, f(x_1)), \ldots, (x_t, f(x_t))\rangle$$

one simply outputs the canonical index of the Lagrangian interpolation polynomial, i.e.,

$$p_t(x) = \sum_{i=0}^{t} f(x_i) \prod_{\substack{k=0\\ k\neq i}}^{t} \frac{x - x_k}{x_i - x_k}$$

The fundamental theorem of algebra implies convergence.

Note that in the literature you will quite often find the following definition for learning in the limit for functions.

Definition 7 (Gold [5]). *Let $U \subseteq \mathcal{R}$ and let $\psi \in \mathcal{P}^2$. The class U is said to be* ***learnable in the limit*** *with respect to ψ iff there is an IIM $M \in \mathcal{P}$ such that for each function $f \in U$,*

(1) *for all $n \in \mathbb{N}$, $M(f^n)$ is defined,*

(2) *there is a $j \in \mathbb{N}$ such that $\psi_j = f$ and the sequence $(M(f^n))_{n\in\mathbb{N}}$ converges to j.*

If U is learnable in the limit with respect to ψ by an IIM M, we write $U \in LIM_\psi(M)$. Let $LIM_\psi = \{U \mid \ U \text{ is learnable in the limit w.r.t. } \psi\}$, and let $LIM = \bigcup_{\psi\in\mathcal{P}^2} LIM_\psi$.

The difference between Definitions 6 and 7 is that Definition 7 assumes the graph of the function to be presented in natural order while Definition 6 does not make any assumption concerning the order in which the graph is presented. However, it is quite easy to show that $LIM = LIM^{arb}$ (cf., e.g., Jantke and Beick [7]).

3.4. Points of Concern

Next, we address some fundamental remarks concerning the definition of learning in the limit. These remarks apply to both learning languages and functions.

Our learning model is not satisfactory with respect to the following points of concern.

– The limit learner has access to the whole initial segment of the data sequence provided.

– The limit learner is only supposed to converge but one *never knows* whether or not it already did so.

– We have not incorporated any complexity requirement. So, what is the right measure of complexity to be used here?

We shall address all these points in this and later lectures. Let us start with the problem to define an appropriate measure of complexity for learning in the limit.

The first complexity measure we will consider is the ***mind change*** complexity. A mind change occurs if

$$\begin{aligned} M(d_j) &\neq M(d_{j+1}) \text{ , where } d \text{ is a text or an informant ,} \\ M(f^n) &\neq M(f^{n+1}) \text{ , where } f \text{ is a recursive function .} \end{aligned}$$

Clearly, this measure is closely related to the number of prediction errors. Both complexity measures say little about the total amount of data and time needed until a concept is guessed correctly.

Therefore, one has also proposed to study the ***update-time***. By update-time we mean the time needed by a learner to compute its **new** hypothesis from the actual input. We measure this time as a function of the length of the input. But this measure also has a serious drawback. That is, one can always achieve linear update-time if no extra requirements are made to the hypotheses. We leave it as an exercise to prove this statement formally.

3.5. Consistency

In order to deal with this problem, the actual hypotheses are often required to be consistent. Intuitively speaking, a hypothesis is consistent if and only if all information obtained so far about the unknown object is completely and correctly encoded in this hypothesis. Otherwise, a hypothesis is said to be *inconsistent*. Consistency seems to be a very natural requirement. If we look at function learning then the function g computed by an inconsistent hypothesis produced on input $f^{X,n}$ has the following property. There must be an x_i, $i \leq n$ such that $g(x_i) \neq f(x_i)$. Note that there are two possible reasons for g to differ from f on argument x_i; namely, $g(x_i)$ may be not defined, or the value $g(x_i)$ is defined and does not equal $f(x_i)$. Hence, if a hypothesis is inconsistent then it is not only wrong but it is wrong on an argument for which the IIM does already know the correct value. At first glance we are tempted to totally exclude IIMs producing inconsistent hypotheses from our considerations. It might seem that *consistent IIMs*, i.e., IIMs that produce always consistent hypotheses, are the only reasonable learning devices.

Surprisingly enough this is a misleading impression. As it turns out, in a sense learning seems to be *the art of knowing what to overlook*. Since this phenomenon is of fundamental importance, we shall deal with it in some more detail here. Interestingly enough, there are also different ways to define consistency and these different definitions will also provide some surprises. We start our investigations of consistent learning within the setting of function learning.

Next we formally define different models of consistent learning.

Definition 8 (Barzdin [2]). *Let $U \subseteq \mathcal{R}$ and let $\psi \in \mathcal{P}^2$. The class U is called* ***consistently learnable*** *in the limit with respect to ψ iff there is an IIM $M \in \mathcal{P}$ such that*

(1) $U \in LIM_\psi(M)$,

(2) $\psi_{S(f^n)}(x) = f(x)$ *for all* $f \in U$, $n \in \mathbb{N}$ *and* $x \leq n$.

$CONS_\psi(M),\ CONS_\psi$ *and* $CONS$ *are defined analogously as above.*

Intuitively, a consistent IIM does correctly reflect all the data it has already seen. If an IIM does not always work consistently, we call it inconsistent.

Next, we add a requirement to the definition of the learning type $CONS_\psi$ that is often implicitly assumed in applications, namely, that the strategy is defined on every input, cf. Michalski *et al.* [9,10].

Definition 9 (Jantke and Beick [7]). *Let $U \subseteq \mathcal{R}$ and let $\psi \in \mathcal{P}^2$. The class U is called* ***$\mathcal{R}$−consistently learnable*** *in the limit with respect to ψ iff there is an IIM $M \in \mathcal{R}$ such that $U \in CONS_\psi(M)$.*

$\mathcal{R}$-$CONS_\psi(M)$, $\mathcal{R}$-$CONS_\psi$ *and* $\mathcal{R}$-$CONS$ *are defined analogously as above.*

The latter definition has a peculiarity that should be mentioned. Although the strategy is required to be recursive, consistency is only demanded for inputs that correspond to some function f from the class to be learned. With the next definition we model the scenario in which consistency is required on all inputs. In order to distinguish the resulting learning type from the latter defined one, we use the prefix T. Informally, T points to *total* consistency.

Definition 10 (Wiehagen and Liepe [16]). *Let $U \subseteq \mathcal{R}$ and let $\psi \in \mathcal{P}^2$. The class U is called* ***T−consistently learnable*** *in the limit with respect to ψ iff there is an IIM $M \in \mathcal{R}$ such that*

(1) $U \in CONS_\psi(M)$,

(2) $\psi_{S(f^n)}(x) = f(x)$ *for all* $f \in \mathcal{R}$, $n \in \mathbb{N}$ *and* $x \leq n$.

T-$CONS_\psi(S)$, T-$CONS_\psi$ *and* T-$CONS$ *are defined in the same way as above.*

Finally, looking at potential applications it is often highly desirable to make no assumptions concerning the *order* in which input data should be presented. Therefore, we sharpen Definitions 8 through 10 by additionally demanding an IIM to behave consistently independently of the order of the input.

Definition 11 (Blum and Blum [4]). *Let $U \subseteq \mathcal{R}$ and let $\psi \in \mathcal{P}^2$.* $\boldsymbol{U \in T\text{-}CONS_\psi^{arb}}$ *iff there is an IIM $S \in \mathcal{R}$ such that*

(1) *for all $f \in U$ and every $X \in \Pi(\mathbb{N})$, there is a $j \in \mathbb{N}$ such that $\psi_j = f$, and $(M(f^{X,n}))_{n\in\mathbb{N}}$ converges to j,*

(2) $\psi_{M(f^{X,n})}(x_m) = f(x_m)$ *for every permutation* $X \in \Pi(\mathbb{N}),\ f \in \mathcal{R},\ n \in \mathbb{N}$*, and* $m \leq n$*.*

$T\text{-}CONS_{\psi}^{arb}(M)$ *as well as* $T\text{-}CONS^{arb}$ *are defined in analogy to the above.*

Furthermore, appropriately incorporating the requirement to learn from arbitrary input directly yields the learning types $CONS^{arb}$, and $\mathcal{R}\text{-}CONS^{arb}$. Therefore, the formal definition of these learning models is omitted here. Note that for all learning types $LT \in \{T\text{-}CONS,\ T\text{-}CONS^{arb},\ \mathcal{R}\text{-}CONS,\ \mathcal{R}\text{-}CONS^{arb},\ CONS,\ CONS^{arb}\}$ we have $LT_{\varphi} = LT$ for every Gödel numbering φ.

In the following we aim to compare the learning power of the different models of consistent learning to one another as well as to learning in the limit. Note that in the following $\subseteq$ denotes subset and $\subset$ denotes *proper* subset. Finally, incomparability of sets is denoted by $\#$.

As already mentioned, in machine learning it is often assumed that learning algorithms are defined on all inputs. On the one hand, this requirement is partially justified by a result of Gold [5]. He proved that learning in the limit is insensitive with respect to the requirement to learn exclusively with recursive IIMs, i.e., if $U \in LIM(M)$, then there is an IIM $\hat{M} \in R$ such that $U \in LIM(\hat{M})$. One the other hand, consistency is a common requirement in machine learning. Therefore, it is natural to ask whether or not the power of consistent learning algorithms further decreases if one restricts itself to recursive IIMs. The answer to this question is provided by our next theorem.

Theorem 3.8. $T\text{-}CONS \subset \mathcal{R}\text{-}CONS \subset CONS$.

Proof. By definition, $T\text{-}CONS \subseteq \mathcal{R}\text{-}CONS \subseteq CONS$. In order to show $\mathcal{R}\text{-}CONS \setminus T\text{-}CONS \neq \emptyset$, let $U = \{f \mid\ f \in R,\ \varphi_{f(0)} = f\}$ where $\varphi \in G\ddot{o}d$. Obviously, $U \in \mathcal{R}\text{-}CONS_{\varphi}(M)$ by the IIM $M(f^n) = f(0)$ for all $n \in N$.

Now assume that $U \in T\text{-}CONS_{\varphi}(M)$ for some IIM M. Hence, by Definition 10, $M \in R$ and $\varphi_{M(f^n)}(x) = f(x)$ for every $f \in \mathcal{R},\ n \in \mathbb{N}$ and $x \leq n$. By an implicit use of the Recursion Theorem, let $f = \varphi_i$ be the following function.

$$\begin{aligned} f(0) &= i, \\ f(n+1) &= \begin{cases} 0, & \text{if } M(f^n 0) \neq M(f^n) \\ 1, & \text{if } M(f^n 0) = M(f^n) \text{ and } M(f^n 1) \neq M(f^n). \end{cases} \end{aligned}$$

Clearly, $f \in U$ (note that one of the two cases in the definition of f must happen for all $n \geq 1$). On the other hand, $M(f^n) \neq M(f^{n+1})$ for all $n \in \mathbb{N}$, contradicting $U \in T\text{-}CONS_{\varphi}(M)$. Hence $U \notin T\text{-}CONS$. This completes the proof of $T\text{-}CONS \subset \mathcal{R}\text{-}CONS$.

The proof of $CONS \setminus \mathcal{R}\text{-}CONS \neq \emptyset$ can be done by using a class similar to the class above, namely

$$U = \{f \mid\ f \in \mathcal{R},\ \text{either } \varphi_{f(0)} = f \text{ or } \varphi_{f(1)} = f\}\ .$$

Next, we show that $U \in CONS$. The wanted IIM M is defined as follows. Let $f \in \mathcal{R}$ and $n \in \mathbb{N}$.

$M(f^n) =$ "Compute in parallel $\varphi_{f(0)}(x)$ and $\varphi_{f(1)}(x)$ for all $x \le n$ until (A) or (B) happens.

(A) $\varphi_{f(0)}(x) = f(x)$ for all $x \le n$.

(B) $\varphi_{f(1)}(x) = f(x)$ for all $x \le n$.

If (A) happens first, then output $f(0)$. If (B) happens first, then output $f(1)$. If neither (A) nor (B) happens, then $M(f^n)$ is not defined."

By the definition of U, it is obvious that $M(f^n)$ is defined for all $f \in U$ and all $n \in \mathbb{N}$. Moreover, M is clearly consistent. Hence, it suffices to prove that $(M(f^n))_{n\in\mathbb{N}}$ converges for all $f \in U$. But this is also an immediate consequence of the definition of U, since either $\varphi_{f(0)} \neq f$ or $\varphi_{f(1)} \neq f$. Hence M cannot oscillate infinitely often between $f(0)$ and $f(1)$. Consequently, $U \in CONS_\varphi(M)$.

Now, it is intuitively clear that $U \notin \mathcal{R}\text{-}CONS$. The formal proof is done by using Smullyan's Recursion Theorem. We refer the interested reader to Wiehagen and Zeugmann [18]. ▮

Next, consider

$$\begin{aligned} U_1 &= \{f \in \mathcal{R} \mid \varphi_{f(0)} = f \text{ and } f(x) > 0 \text{ for all } x\}\ , \\ U_2 &= \{f \in \mathcal{R} \mid \varphi_{f(0)} = f\}\ , \text{ and} \\ U_0 &= \{f \in \mathcal{R} \mid f(x) = 0 \text{ for almost all } x\}\ . \end{aligned}$$

Then, it is not hard to verify that $U_1,\ U_2,\ U_0 \in LIM$. Moreover, using similar ideas as above one can easily show the following exercise.

Exercise 11. $U_1 \cup U_0 \in LIM \setminus CONS$.

Moreover, again using similar ideas it is not hard to see that the following theorem holds.

Theorem 3.9 (Barzdin [2]). $U_2 \cup U_0 \notin LIM$.

The latter theorem directly implies that $\mathcal{R} \notin LIM$. Therefore, the property of a function class U to be learnable is *not trivial*. Furthermore, Theorem 3.9 also shows that it is not always possible to combine two learners into a more powerful one. So, it is only natural to ask whether or not the different consistent learning types LT are closed under union. Here, by closure under union we mean that for all classes $U,\ V \in LT$ we also have $U \cup V \in LT$.

The answer is provided by the following theorem.

Theorem 3.10.

(1) *$CONS,\ CONS^{arb},\ \mathcal{R}\text{-}CONS$ and $\mathcal{R}\text{-}CONS^{arb}$ are not closed under finite union.*

(2) *$\mathcal{T}\text{-}CONS$ and $\mathcal{T}\text{-}CONS^{arb}$ are closed under recursively enumerable union.*

Proof. For showing Assertion (1) we can use the classes U_0 and U_2 defined above. After a bit of reflection, one easily sees that $U_0,\ U_2 \in \mathcal{R}\text{-}CONS^{arb}$. Hence, we also have $U_0,\ U_2 \in \mathcal{R}\text{-}CONS$ as well as $U_0,\ U_2 \in CONS^{arb}$ and $U_0,\ U_2 \in CONS$. But by Theorem 3.9 we already know $U_0 \cup U_2 \notin LIM$.

In order to prove (2), we restate this assertion more formally. Let $(M_i)_{i\in\mathbb{N}}$ be a recursive enumeration of T–consistent IIMs. Then there exists an IIM M such that $T\text{-}CONS(M) = \bigcup_{i\in\mathbb{N}} T\text{-}CONS(M_i)$.

Without loss of generality, we may assume that all the IIMs M_i output as hypotheses programs in some fixed Gödel numbering φ. Let $f \in \mathcal{R}$. Then two cases are possible. Either there is an IIM M_i that learns f or all IIMs fail to learn it. However, in the latter case each of the IIMs M_i has to change its mind infinitely often. On the other hand, if f is learned by some M_i then at least one IIM stabilizes its output. The wanted IIM M searches for an enumerated machine that might learn f as follows.

The IIM M dovetails the computation of more and more outputs of the enumerated IIMs. For each IIM M_i that is already included in its dovetailed computations, it counts the number of equal outputs. This number is called weight. As long as an IIM repeats its actual guess on the next input, the weight increments. If an IIM performs a mind change, its weight reduces to zero. After having read the initial segment f^k of the function f the IIM M favors from the first $k+1$ IIMs $M_0,\dots,M_k$ that one which actually has the greatest weight. In case there are two IIMs M_i and M_j taking the greatest weight the IIM M chooses that one having the smallest index.

We formally define M as follows. Let $f \in \mathcal{R}$, and let $k \in \mathbb{N}$.

$M(f^k) =$ "Compute in parallel

$M_0(f^0),\ \dots,\ M_0(f^k),$

$M_1(f^0),\ \dots,\ M_1(f^k),$

–

–

–

$M_k(f^0),\ \dots,\ M_k(f^k),$

and assign to each IIM M_i, $i \le k$, its weight, i.e., the greatest number $m \le k-i$ satisfying the condition that $M_i(f^{k-i-m}) = M_i(f^{k-i-m+1}) = \cdots = M_i(f^{k-i})$. Note that we calculate the weights in a triangular fashion. This is necessary in order to achieve convergence of M. Choose $w(k)$ to be the smallest $i \le k$ such that the IIM M_i has the greatest weight.

In case all considered IIMs have weight zero, output a φ–program of $f(0)\cdots f(k)0^\infty$.

If $w(k) = w(k-1)$, then output $M_{w(k)}(f^k)$. Otherwise, output a φ–program of $f(0)\cdots f(k)0^\infty$ that is different from $M(f^{k-1})$."

It remains to show that $T\text{-}CONS_\varphi(M) = \bigcup_{i\in\mathbb{N}} T\text{-}CONS_\varphi(M_i)$. Obviously, M is consistent on any initial segment it is fed, since all of the IIMs M_i, $i \in \mathbb{N}$, do so. Now, let $f \in \mathcal{R}$ and suppose that f is learned by some IIM M_i. Consequently, there exists numbers $j,\ n_0$ such that $M_i(f^n) = j$ for all $n \geq n_0$, and $\varphi_j = f$. Hence, for any IIM that learns f its weight increases after some point in each step of M's computation. Therefore, for all but finitely many k the IIM M must favor exactly one of the IIMs M_i that learns f and, after some point, M outputs always $M_i(f^k)$. Note that the computation of weights in a triangular fashion really ensures the desired convergence, since any new IIM included in M's computation initially gets weight zero.

On the other hand, if none of the IIMs M_i, $i \in \mathbb{N}$, learns f, then each IIM M_i has to perform infinitely many mind changes. This is ensured by our assumption that each M_i is T–consistent. Hence, the case $w(k) \neq w(k-1)$ occurs infinitely often. But each occurrence of this case forces M to perform a mind change. Consequently, M cannot converge. ∎

So far we have seen that the requirement to learn consistently may have serious consequences including the loss of learnability at all. Next, we outline that the requirement to learn consistently may also seriously affect the complexity of learning. For doing this, we look again at the pattern languages. But this time we investigate their consistent learnability from informant.

Definition 12. *PAT is called **consistently learnable in the limit from informant** with respect to Pat (abbr. $PAT \in CONS\text{-}INF$) iff there is an IIM M such that*

(1) *$PAT \in LimInf$ w.r.t. Pat by M,*

(2) *for all $L \in PAT$, $i \in$ info(L) and $n \in \mathbb{N}$, $i_n^+ \subseteq L(M(i^n))$ and $i_n^- \cap L(M(i^n)) = \emptyset$.*

We add *Poly* if the time to compute $M(i^n) \leq pol(length(i^n))$, where pol is a fixed polynomial.

Then, one can prove the following (cf. Wiehagen and Zeugmann [17]). Note that $\mathcal{P}$ stands here for the set of all languages acceptable in deterministic polynomial time and $\mathcal{NP}$ denotes the set of all languages acceptable in nondeterministic polynomial time.

Theorem 3.11.

(1) $PAT \in CONS\text{-}INF$.

(2) $PAT \notin Poly\text{-}CONS\text{-}INF$, *provided* $\mathcal{P} \neq \mathcal{NP}$.

(3) $PAT \in Poly\text{-}LimInf$.

Summarizing, we may conclude the following interpretation.

If it comes to "reasonable" ideas, it is not a good a idea to trust the common sense.

Now, let us return to the problem of defining an appropriate measure of complexity for learning in the limit.

3.6. Total Learning Time

So, as we have seen, it is not a good idea to introduce “natural” requirements such as consistency. Instead, we shall study the total amount of data and time needed by a learner until convergence.

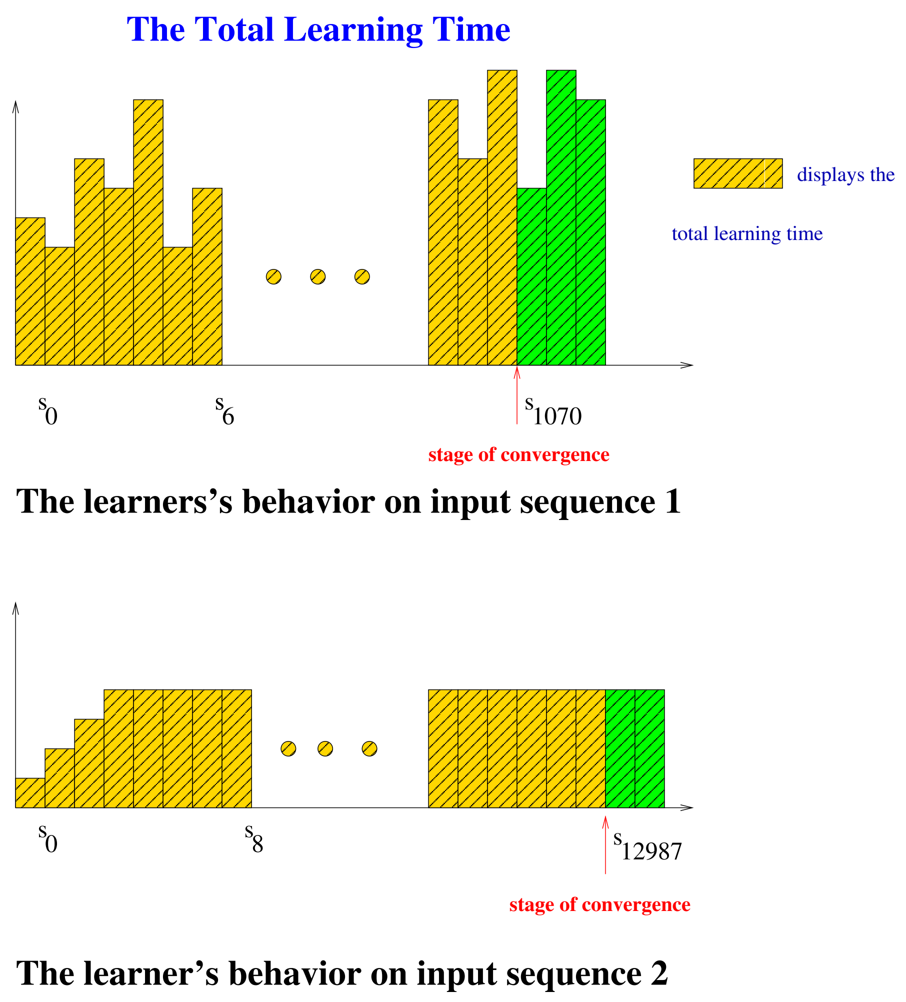

Figure 3.2: The total learning time

We define the total learning time as follows. Let M be any IIM that learns a concept class $\mathcal{C}$ in the limit. Then, for $c \in \mathcal{C}$ and a text or informant d for c, let

$$\boldsymbol{Conv(M,d)} =_{df} \text{the least number } i \in \mathbb{N}^+ \text{ such that for all } j \geq i,\ M(d_j) = M(d_i)$$

denote the ***stage of convergence*** of M on d. Moreover, by $T_M(d_j)$ we denote the number of steps to compute $M(d_j)$. We measure this quantity as a function of the length of the input and refer to it as the *update time*. Finally, the ***total learning time*** taken by the IIM M on a sequence d is defined as

$$\boldsymbol{TT(M,d)} =_{df} \sum_{j=1}^{Conv(M,d)} T_M(d_j)\ .$$

Given a probability distribution D on the data sequences d, we like to evaluate the *expectation* of $TT(M,d)$ with respect to D, the ***average total learning time***.

Figure 3.11 displays the total learning time on different input sequences as the measure under the curve until the stage of convergence happened. Looking at Figure 3.11 we see the following. On different input sequences the amount of data needed until convergence may considerably vary.

Still, there is a problem. What one usually likes to have is a learner that has a polynomial total learning time. This could be easily said, however, we have to specify what is really meant, i.e., polynomial in what? Thus, the crucial point is the right definition of the problem size. If one takes the sum of the length of all elements seen until convergence (and possibly the length of target concept as additional parameter) then a learner could simply delay convergence until sufficiently long examples have been appeared. On the other hand, if we take the length of the shortest hypothesis describing the target concept as problem size then the total learning time is usually unbounded in the *worst-case*. This is caused by the fact that the learner has to learn from all data sequences. Thus, taking a data sequence containing as many repetitions as necessary of elements that do not suffice to learn the target, every bound can be exceeded. See for example the learner's behavior on input sequence 2 below.

On the other hand, such worst-case data sequences may be rare in practice if they occur at all. Consequently, in order to arrive at a complexity measure that has much more practical value than a worst-case analysis, one has to study the average-case behavior of a learner.

So, in the next lecture, we shall explain what an average-case analysis is by using a much simpler example than learning in the limit.

References

[1] D. ANGLUIN (1980), Finding patterns common to a set of strings, *Journal of Computer and System Sciences* **21**, 46 – 62.

[2] J. BARZDIN (1974a), Inductive inference of automata, functions and programs, *in* "Proceedings International Congress of Math.," Vancouver, pp. 455 – 460.

[3] D.R. BEAN, A. EHRENFEUCHT, AND G.F. MCNULTY (1979), Avoidable patterns in strings of symbols, *Pacific J. of Mathematics* **85**, Vol. 2, 261 – 294.

[4] L. BLUM AND M. BLUM (1975), Toward a mathematical theory of inductive inference, *Information and Control* **28**, 122 – 155.

[5] E.M. GOLD (1965), Limiting recursion, *Journal of Symbolic Logic* **30**, 28 – 48.

[6] J.E. HOPCROFT AND J.D. ULLMAN (1969), *Formal Languages and their Relation to Automata*, Addison-Wesley, Reading, Massachusetts.

[7] K.P. JANTKE AND H.R. BEICK (1981), Combining postulates of naturalness in inductive inference, *Journal of Information Processing and Cybernetics (EIK)* **8/9**, 465 – 484.

[8] S. LANGE AND T. ZEUGMANN (1996), Set-driven and rearrangement-independent learning of recursive languages, *Mathematical Systems Theory* **29**, No. 6, 599 – 634.

[9] R.S. MICHALSKI, J.G. CARBONELL, AND T.M. MITCHELL (1984), "Machine Learning, An Artificial Intelligence Approach," Vol. 1, Springer-Verlag, Berlin.

[10] R.S. MICHALSKI, J.G. CARBONELL, AND T.M. MITCHELL (1986), "Machine Learning, An Artificial Intelligence Approach," Vol. 2, Morgan Kaufmann Publishers Inc., San Mateo.

[11] H.JR. ROGERS (1967), "Theory of Recursive Functions and Effective Computability," McGraw–Hill, New York.

[12] A. SALOMAA (1994), Patterns, (The Formal Language Theory Column). EATCS Bulletin **54**, 46 – 62.

[13] A. SALOMAA (1994), Return to patterns (The Formal Language Theory Column), EATCS Bulletin **55**, 144 – 157.

[14] T. SHINOHARA AND S. ARIKAWA (1995), Pattern inference, *in* "Algorithmic Learning for Knowledge-Based Systems" (K.P. Jantke and S. Lange, eds.), Lecture Notes in Artificial Intelligence 961, pp. 259 – 291, Springer-Verlag, Berlin.

[15] A. THUE (1906), Über unendliche Zeichenreihen, Norske Vid. Selsk. Skr. I. Mat. Nat. Kl., Christiana No. 7, 1 – 22.

[16] R. WIEHAGEN AND W. LIEPE (1976), Charakteristische Eigenschaften von erkennbaren Klassen rekursiver Funktionen, *Journal of Information Processing and Cybernetics (EIK)* **12**, 421 – 438.

[17] R. WIEHAGEN AND T. ZEUGMANN (1994), Ignoring data may be the only way to learn efficiently, *Journal of Theoretical and Experimental Artificial Intelligence* **6**, 131 – 144.

[18] R. WIEHAGEN AND T. ZEUGMANN (1995), Learning and Consistency, *in* "Algorithmic Learning for Knowledge-Based Systems" (K.P. Jantke and S. Lange, eds.), Lecture Notes in Artificial Intelligence 961, pp. 1 – 24, Springer-Verlag, Berlin.

LECTURE 4: AVERAGE-CASE COMPLEXITY

Within this lecture we exemplify the task to perform an average-case analysis of the complexity of a particular algorithm. When dealing with data mining, usually huge data sets have to be analyzed. Thus, an excellent average-case behavior is often more important than a good worst-case time complexity. However, performing an average-case analysis of the complexity of a given algorithm is often much more complicated than analyzing its worst-case complexity. Due to the lack of time, therefore we shall often omit such average-case studies throughout this course.

Before we can perform any average-case analysis, we have to understand what average-case analysis is all about. For reaching this goal, we use this lecture to provide some easy examples.

4.1. Introductory Examples

We start with a very simple question. Let us assume all five digit numbers are equally likely including 00000 (that is we also allow leading zeros). Now, we want to add 1 to such a number. The question is

How many digits will be changed?

In the best-case, the last digit is not 9, and we have to change just one digit, i.e., the last one.

In the worst-case, the number is 99999 and we have to change 6 digits, since

$$99999 + 1 = 100000$$

Now, we are interested in the *expected number* of digits that will be changed. Saying that all five digit numbers are equally likely is equivalent to the following. We choose each digit independently of the other digits and the probability to choose a digit from $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ is $1/10$.

So, with probability $9/10$ we do *not* choose 9 to be the last digit. That is, with probability $9/10$ we *only* have to change the last digit.

What can be said about the probability that we have to change exactly two digits? This event will happen if and only if the last digit is 9 and the second digit (from the right) is *not* 9. Furthermore, the probability for the last digit to be 9 is $1/10$ and the probability for the second digit to be not 9 is $9/10$. Thus, the probability to change exactly two digits is

$$\frac{1}{10} \cdot \frac{9}{10} = \frac{9}{100} \ .$$

For seeing the general pattern, we also calculate the probability that we have to change exactly 3 digits. This event will happen if and only if the last digit is 9, the second digit is 9 and the third digit is *not* 9. Thus, the probability to change exactly three digits is

$$\frac{1}{10} \cdot \frac{1}{10} \cdot \frac{9}{10} = \frac{9}{1000} \ .$$

Analogously, the probability to change exactly 4 digits is $9/10^4$ and the probability to change exactly 5 digits is $9/10^5$. Additionally, we need the probability to change exactly 6 digits. As shown above, this event will happen if the number is 99999, i.e., with probability $1/10^5$.

For completing our analysis, let R denote the random variable telling us how many digits we have to change. Thus, R can take the values $1,\ 2,\ 3,\ 4,\ 5,\ 6$. The formula for computing the expection is

$$E[R] = \sum_{x=1}^{6} x \cdot \Pr(R = x)\ .$$

Thus, the expected number of digits to be changed is

$$E[R] = \sum_{x=1}^{5} x \cdot \frac{9}{10^x} + \frac{6}{10^5} = 1,11111\ .$$

Next, we look at the more general case of having an n digit number, where $n \in \mathbb{N}$, $n \geq 1$. Now, we denote the random variable by R_n and obviously the range of R_n is the set $\{1, \ldots, n, n+1\}$. Clearly, then we get

$$E[R] = \sum_{x=1}^{n} x \cdot \frac{9}{10^x} + \frac{n+1}{10^n} \tag{4.1}$$

How can we calculate the sum on the right hand side? Let us start from something we know, i.e., the finite geometrical series. That is, we take into account that

$$f(q) = \sum_{x=0}^{n} q^x = \frac{1-q^{n+1}}{1-q}\ , \qquad \text{for all } q \neq 1\ .$$

Now, taking derivatives on both sides yields

$$f'(q) = \sum_{x=0}^{n} xq^{x-1} = \frac{1-(n+1)q^n + nq^{n+1}}{(1-q)^2}\ , \quad q \neq 1\ .$$

Consequently, we arrive at

$$\sum_{x=0}^{n} xq^x = \frac{q-(n+1)q^{n+1} + nq^{n+2}}{(1-q)^2}\ , \quad q \neq 1\ .$$

This allows us to rewrite the expectation in (4.1) as follows.

$$\begin{aligned} E[R] &= 9 \cdot \sum_{x=1}^{n} x \cdot \left(\frac{1}{10}\right)^x + \frac{n+1}{10^n} \\ &= 9 \cdot \frac{(1/10) - (n+1)(1/10)^{n+1} + n(1/10)^{n+2}}{(1-(1/10))^2} + \frac{n+1}{10^n} \\ &= \frac{10}{9} - \frac{1}{9 \cdot 10^n}\ . \end{aligned}$$

Thus, as n tends to infinity, the expected number of digits to be changed if you add 1 to an n digit number in decimal notation chosen uniformly at random is $10/9$. So, this is very close to the best-case (just one digit) and far from the worst case of $n + 1$ digits.

Exercise 12. *Compute the expected number of bits to be changed if you add* 1 *to an* n *bit number chosen uniformly at random.*

4.2. Analyzing the Common Algorithm for Finding the Maximum

Let us consider the following well-known algorithm for finding the maximum in a list of elements.

Algorithm Max

Input: n elements $X[1],\ X[2],\ \ldots, X[n]$ of a totally ordered set.

Output: Numbers m and j such that $m = X[j] = \max\{X[k] |\ 1 \leq k \leq n\}$.

Method:

```
A1. (Initialize) $j := n$, $k := n$, $m := X[n]$

A2. (All tested ?) Test whether $k = 0$.
    If it is, output $j,\ m$, and stop.
    Otherwise, execute A3.

A3. (Compare) If $X[k] \leq m$, goto A5.
    Otherwise, execute A4.

A4. (Change $m$) $j := k$, $m := X[k]$

A5. (Decrease $k$) $k := k - 1$, goto A2.
```

Algorithm Max requires a fixed amount of storage. So, we shall analyze the time required to perform it. For doing this, we count the number of times each step is executed. This is done Figure 4.1 below.

Instruction	Number of executions
A1	1
A2	n
A3	$n-1$
A4	A
A5	$n-1$

Figure 4.1: Counting the number of executions of A1 through A5

Everything is clear except the quantity A. Therefore, we study the quantity A, only. Before looking at the expectation of A, we look at the minimum and maximum of it.

(1) The minimum value of A is zero; this happens iff $X[n] = \max\{X[k] | \ 1 \le k \le n\}$.

(2) The maximum value of A is $n-1$; this happens iff $X[1] > X[2] > \cdots > X[n]$.

Thus, the best-case and worst-case behavior of ***Algorithm Max*** are clear and trivially to obtain. But what can be said about the average?

To answer this question, we need to define what we mean by average. For doing that, we must make some assumptions about the *expected* characteristics of the input data. These assumptions are:

(1) All $X[k]$ are pairwise distinct.

(2) Without loss of generality we can assume that $\{1, 2, \ldots, n\}$ is the the range of the input data.

(3) We assume all possible $n!$ permutations of the input data $\{1, 2, \ldots, n\}$ to be equally likely.

In particular, we made Assumption (3), since we do not have any reason here to assign different probabilities to the different possible permutations. When dealing with particular input data sets, things may change, however. In such cases one has to figure out the underlying probability distribution of the inputs. But for our goal and in many applications Assumption (3) is reasonable.

Next, we introduce some notations. By p_{nk} we denote the probability that A has value k. Using the classical definition of probability, we can thus write

$$p_{nk} = \frac{\text{number of permutations of } \{1, 2, \ldots, n\} \text{ for which } A = k}{n!} .$$

Then, the *average* (mean) value is defined by

$$E[A] = A_n = \sum_k k p_{nk} .$$

Moreover, the *variance* V_n is defined as

$$V_n = E[(A - A_n)^2] = \sum_k (k - A_n)^2 p_{nk} = \sum_k k^2 p_{nk} - A_n^2 .$$

Furthermore, the *standard deviation* σ_n is defined to be $\sqrt{V_n}$. Thus, all we have to do is to compute A_n, V_n and σ_n. But how can we do this?

Approach 1. Determine p_{nk}.

Claim 1. $p_{nk} = \frac{1}{n} \cdot p_{(n-1)(k-1)} + \frac{n-1}{n} \cdot p_{(n-1)k}$, where $p_{1k} = \delta_{0k}$, and $p_{nk} = 0$ if $k < 0$.

Proof. If $x_1 = n$, then the value of A is one higher than the value of A obtained on $\{x_2, \ldots, x_n\}$. Moreover, if $x_1 \neq n$, then the value of A is the *same* as its value on $\{x_2, \ldots, x_n\}$. ∎

But how can we proceed? There is a very powerful tool, called *generating functions* which seems appropriate to be used here. Therefore, we shortly recall the definition of generating functions and an important theorem from calculus.

Let $(a_n)_{n\in\mathbb{N}}$ be any sequence of real (or complex) numbers. Then

$$g(z) = \sum_{n=0}^{\infty} a_n z^n$$

is called ***generating function*** of $(a_n)_{n\in\mathbb{N}}$. The following theorem is often applied to generating functions.

Theorem 4.1. *Let $(a_n)_{n\in\mathbb{N}}$ and $(b_n)_{n\in\mathbb{N}}$ be any sequences such that their generating functions have a radius $r > 0$ of convergence. Then*

$$\sum_{n=0}^{\infty} a_n z^n = \sum_{n=0}^{\infty} b_n z^n$$

iff $a_n = b_n$ for all $n \in \mathbb{N}$.

Moreover, recall that power series can be differentiated by differentiating their summands. Thus, we also know that

$$g'(z) = \sum_{n=0}^{\infty} n \cdot a_n z^{n-1} .$$

For more information about generating functions the interested reader is referred to Graham, Knuth, and Patashnik [1]. Furthermore, you should consult my *Additional Notes on Counting and Probability* which are available at

http://www-alg.ist.hokudai.ac.jp/ thomas/TCSTRB/tcstr_05_2/tcstr_05_2.ps.gz

So, let us try to apply generating functions for computing the p_{nk}. We consider

$$G_n(z) = \sum_{k\ge 0} p_{nk} z^k .$$

In fact, G_n is even a polynomial but due to the convention made in Claim 1, we can write it as a power series, which is technically more convenient for our purposes. Also, note that $G_1(z) = 1$. Then, we obtain

$$\begin{aligned} G_n(z) = \sum_{k\ge 0} p_{nk} z^k &= \sum_{k\ge 0} \left(\frac{1}{n} \cdot p_{(n-1)(k-1)} + \frac{n-1}{n} \cdot p_{(n-1)k} \right) \cdot z^k \\ &= \frac{z}{n} \sum_{k\ge 0} p_{(n-1)(k-1)} z^{k-1} + \frac{n-1}{n} \sum_{k\ge 0} p_{(n-1)k} z^k \\ &= \frac{z}{n} G_{n-1}(z) + \frac{n-1}{n} G_{n-1}(z) = \frac{z+n-1}{n} G_{n-1}(z) . \end{aligned}$$

Hence, we arrive at

$$\begin{aligned} G_n(z) &= \frac{1}{n!}\prod_{i=1}^{n-1}(z+n-i) = \frac{1}{(z+n)n!}\prod_{i=0}^{n-1}(z+n-i) \\ &= \frac{1}{(z+n)n!}\cdot n!\binom{z+n}{n} = \frac{1}{z+n}\binom{z+n}{n} \end{aligned}$$

Thus, we have found a closed formula for G_n. Using Stirling numbers of the first kind[*], we get

$$\prod_{i=0}^{n}(z+n-i) = \prod_{i=0}^{n-1}(z+i) = \sum_{k=1}^{n}\left[{n \atop k}\right] z^k ,$$

and hence, we can conclude

$$\begin{aligned} \prod_{i=0}^{n-1}(z+n-i) &= \frac{1}{z}\prod_{i=0}^{n}(z+n-i) = \frac{1}{z}\sum_{k=1}^{n}\left[{n \atop k}\right] z^k \\ &= \sum_{k=1}^{n}\left[{n \atop k}\right] z^{k-1} = \sum_{k\geq 0}\left[{n \atop k+1}\right] z^k . \end{aligned}$$

Consequently, we can apply Theorem 4.1 and get

$$G_n(z) = \frac{1}{n!}\prod_{i=0}^{n-1}(z+n-i) = \sum_{k\geq 0}\left(\frac{1}{n!}\left[{n \atop k+1}\right]\right) z^k ,$$

and therefore

$$p_{nk} = \frac{1}{n!}\left[{n \atop k+1}\right] .$$

However, it is not easy to compute A_n by using the formula just obtained. You are strongly advised to try it as an exercise. We shall provide here an easier and even more general method. Consider

$$G(z) = \sum_{k\geq 0} q_k z^k ,$$

where the q_k are probabilities. Then

$$G(1) = \sum_{k\geq 0} q_k = 1 ,$$

and moreover

$$G'(1) = \sum_{k\geq 0} k q_k = E[G] .$$

Analogously, one easily verifies

$$V(G) = G''(1) + G'(1) - (G'(1))^2 .$$

[*]We use here the notation from [1] to denote Stirling numbers of the first kind.

Thus, $A_n = G_n'(1)$ which can be computed as follows. Using the identity

$$G_n(z) = \frac{z+n-1}{n} G_{n-1}(z)$$

derived above, we directly obtain

$$G_n'(z) = \frac{1}{n} G_{n-1}(z) + \frac{z+n-1}{n} G_{n-1}'(z) \ .$$

Consequently,

$$\begin{aligned} G_n'(1) &= \frac{1}{n} + G_{n-1}'(1) \\ &= \frac{1}{n} + \frac{1}{n-1} + \cdots + \underbrace{G_1'(1)}_{=0} \ . \end{aligned}$$

Therefore, we can rewrite $G_n'(1) = \sum_{k=2}^{n} \frac{1}{k}$ and recalling that the nth Harmonic number is defined as $H_n = \sum_{k=1}^{n} \frac{1}{k}$ we finally get

$$A_n = H_n - 1 \ .$$

Furthermore, taking into account that

$$\int_1^n \frac{dx}{x} = \ln n - \ln 1 = \ln n$$

it is easy to see that $\ln n < H_n < \ln n + 1$.

Now, letting $H_n^{(2)} = \sum_{k=1}^{n} \frac{1}{k^2}$ and putting it all together, we finally get

$$V(A_n) = H_n - H_n^{(2)} \ .$$

Thus, the complete picture concerning A can be written as

$$A = \left(\min:\ 0,\ \text{average: } H_n - 1,\ \max:\ n-1, \text{dev: } \sqrt{H_n - H_n^{(2)}} \right) \ .$$

Hopefully, this easy example has provided you a basic idea of what average-case analysis is all about and what basic techniques one might apply.

References

[1] R.L. GRAHAM, D.E. KNUTH AND O. PATASHNIK (1989), *Concrete Mathematics* (Addison-Wesley, Reading, Massachusetts).

Lecture 5: Average-Case Analysis II

Within this lecture, we want to perform an average-case analysis for a small variation of Algorithm $\mathcal{P}$, synonymously called Wholist Algorithm, introduced in Lecture 2.

Recall that our concept class $\mathcal{C}_n$ is the set of all concepts $c \subseteq \mathcal{X}_n = \{0,1\}^n$ describable by a monomial. For the concept class $\mathcal{C}_n$ the hypothesis space $\mathcal{H}_n$ will be chosen as the set of all monomials over $\mathcal{L}_n$ for the learners considered below.

This is a good place to address the issue of our points of concerns that a limit learner has always access to the whole initial segment of the data sequence provided. In contrast to that we next define ***iterative IIMs***.

An iterative IIM is only allowed to use its last guess and the next element in the data presentation of the target concept for computing its actual guess. Conceptionally, an iterative IIM M defines a sequence $(M_n)_{n\in\mathbb{N}}$ of machines each of which takes as its input the output of its predecessor. We use $data(c)$ to denote the set of all data sequences for c. In order to avoid misunderstandings it should be recalled that data sequences are either texts or informants. Furthermore, if $d = (d_j)_{j\in\mathbb{N}} \in data(c)$ is any data sequence, then we use here d_j to denote the jth entry of it.

Definition 13 (Wiehagen [2]). *Let $\mathcal{C}$ be a concept class, let c be a concept, and let $\mathcal{H} = (h_j)_{j\in\mathbb{N}}$ be a hypothesis space.* ***An IIM M IT-infers c from data sequence d with respect to $\mathcal{H}$*** *iff for every $d = (d_j)_{j\in\mathbb{N}} \in data(c)$ the following conditions are satisfied:*

(1) *for all $n \in \mathbb{N}$, $M_n(d)$ is defined, where $M_0(d) =_{df} M(d_0)$ and for all $n \geq 0$: $M_{n+1}(d) =_{df} M(M_n(d), d_{n+1})$,*

(2) *the sequence $(M_n(d))_{n\in\mathbb{N}}$ converges to a number j such that $c = h_j$.*

Finally, ***M IT-infers $\mathcal{C}$*** *with respect to $\mathcal{H}$ iff, for each $c \in \mathcal{C}$, M IT-infers c from data sequence with respect to $\mathcal{H}$.*

In the latter definition $M_n(d)$ denotes the $(n+1)$th hypothesis output by M when successively fed the data sequence d.

When the data sequence are informants and texts, then we also say that M IT-infers $\mathcal{C}$ from informant and text, respectively, with respect to the hypothesis space $\mathcal{H}$ considered.

For the sake of better understandability we first modify Algorithm $\mathcal{P}$ into an iterative learner (cf. Algorithm $\mathcal{IML}$). Again, we identify the target concept c and the monomial m describing it. The iterative learner is defined in stages, where Stage ℓ conceptually describes M_ℓ.

Algorithm $\mathcal{IML}$: "Let $c \in \mathcal{C}_n$, let $i = (b_0, m(b_0)),\ (b_1, m(b_1)),\ \ldots$ be any informant for c. Go to Stage 0.

```
Stage 0. $\mathcal{IML}$ receives as input $(b_0, m(b_0))$.
Initialize $h_0 = x_1\bar{x}_1 \dots x_n\bar{x}_n$.
If $m(b_0) = 0$ then $h_0$ remains unchanged; else
for $j := 1$ to $n$ do
if $b_0^j = 1$ then delete $\bar{x}_j$ in $h_0$ else delete $x_j$ in $h_0$.
Denote the result by $h_0$, output $h_0$ and go to Stage 1.
Stage $\ell$, $\ell \geq 1$. $\mathcal{IML}$ receives as input $h_{\ell-1}$ and the $(\ell+1)$th element $(b_\ell, m(b_\ell))$ of $i$.
If $m(b_\ell) = 0$ then set $h_\ell = h_{\ell-1}$; else
for $j := 1$ to $n$ do
if $b_0^j = 1$ then delete $\bar{x}_j$ in $h_{\ell-1}$ else delete $x_j$ in $h_{\ell-1}$.
Denote the result by $h_\ell$, output $h_\ell$ and go to Stage $\ell+1$.
By convention, if all literals have been removed, then $h_\ell = \emptyset$, and $h_\ell(b) = 1$ for all $b \in \mathcal{X}_n$.
end."
```

Now, we can directly state the following theorem.

Theorem 5.1. *For every $n \geq 1$ we have: $\mathcal{IML}$ IT-learns $\mathcal{C}_n$ from informant with respect to the hypothesis space $\mathcal{H}_n$.*

Proof. Using the same arguments as in the proof of Theorem 2.1 one easily sees that Algorithm $\mathcal{IML}$ IT-learns $\mathcal{C}_n$ from informant with respect to the hypothesis space $\mathcal{H}_n$. We omit the details. ∎

Moreover, Algorithm $\mathcal{IML}$ can be easily adapted to learn from positive data only. We have just to omit the tests whether or not $m(b_\ell) = 0$. We call the resulting algorithm $\mathcal{IMLP}$. Now, the following corollary is obvious.

Corollary 5.2. *For every $n \geq 1$ we have: $\mathcal{IMLP}$ IT-learns $\mathcal{C}_n$ from positive data only with respect to the hypothesis space $\mathcal{H}_n$.*

Also, it should be noted that Algorithm $\mathcal{IMLP}$ is a true limit learner, since its convergence is undecidable. You should prove this as an exercise. On the other hand, for Algorithm $\mathcal{IML}$ convergence is *decidable*. This is due to the fact that $\{0,1\}^n$ is finite for all $n \in \mathbb{N}$. However, deciding whether or not Algorithm $\mathcal{IML}$ has already converged is *practically infeasible* for all n of practical relevance, since $|\{0,1\}^n| = 2^n$.

Let us first take a quick look at the best and worst case complexity with respect to the number of prediction errors (mind changes) made. Algorithm $\mathcal{P}$ does not make any prediction errors and Algorithm $\mathcal{IML}$ does not make any mind change iff the initial hypothesis h_0 equals the target monomial, i.e., if the concept to be learned is "FALSE." We call this concept ***minimal***. The remaining concepts are said to be ***non-minimal***.

To study the general case, let us call the literals appearing in a non-minimal monomial m **relevant**. All the other literals in $\mathcal{L}_n$ will be called **irrelevant** for m. There are $2n - \#(m)$ irrelevant literals, where $\#(m)$ denotes the number of literals in monomial m. One can also consider $\#(m)$ to be the length of monomial m.

We call bit i **relevant** for m if x_i or $\bar{x}_i$ is relevant for m and use

$$\boldsymbol{k} =_{df} \boldsymbol{k(m)} =_{df} n - \#(m)$$

throughout the rest of the lecture to denote the number of irrelevant bits.

Now we can expresses the best-case and worst-case number of prediction errors and mind changes made by Algorithm $\mathcal{P}$ and Algorithm $\mathcal{IML}$ as follows:

Theorem 5.3. *If c is a non-minimal concept of $\mathcal{C}_n$ Algorithm $\mathcal{P}$ (Algorithm $\mathcal{IML}$) makes 2 prediction errors (mind changes) in the best-case and $1 + k(m)$ prediction errors (mind changes) in the worst-case.*

Note that Theorem 5.3 remains valid for Algorithm $\mathcal{IMLP}$. So, the gap between the best-case and the worst-case can become quite large. But still, we have no idea about the average. The situation changes even more, if we consider the total learning time. Since the hypotheses internally computed by Algorithm $\mathcal{P}$ or output by Algorithm $\mathcal{IML}$ can be calculated in linear time, it is easy to see that the best-case requires a total learning time $O(n)$. However, the worst case total learning time is *unbounded*, since every positive presentation and every informant may contain as many repetitions of data that do not possess enough information to achieve the learning goal.

Hence, as far as learning in the limit and the complexity measure *total learning time* are concerned, there is a huge gap between the best-case and the worst-case behavior. Therefore, we continue by studying the average-case behavior of the limit learner $\mathcal{IML}$. We start by restricting ourselves to learning from positive presentations only.

This restriction is also quite natural, since Algorithm $\mathcal{IML}$ does not learn anything from negative examples and so does Algorithm $\mathcal{P}$.

Note that one can also study further questions such as estimating the accuracy of a hypothesis after the Algorithm $\mathcal{IML}$ has seen ℓ many positive (or ℓ many positive and negative) examples. You are encouraged to think about this question.

5.1. Average-Case Analysis for Learning in the Limit from Positive Presentations

For the following average case analysis we assume that the data sequences are generated at random with respect to some probability distribution D taken from some class of admissible distributions $\mathcal{D}$, which will be specified later.

We are interested in the *average number* of positive examples necessary to achieve convergence. Let d be a positive presentation of the concept c to be learned that is generated at random according to D.

If the concept to be learned is "FALSE" no examples are needed (and none exist). Otherwise, if the target concept contains precisely n literals then one positive example suffices (note that this one is unique). Thus, for these two cases everything is clear and the probability distributions D on the set of positive examples for c are trivial.

Thus it remains to analyze the nontrivial cases. Let $c = L(m) \in \mathcal{C}_n$ be a concept with monomial $m = \bigwedge_{j=1}^{\#(m)} \ell_{i_j}$. Let $k := k(m) = n - \#(m) > 0$. Note that there are 2^k positive examples for c.

We shall consider the class of ***binomial distributions,*** where in a random positive example all entries corresponding to irrelevant bits are selected independently to one another. With some probability p this will be a 1, and with probability $q =_{df} 1 - p$ a 0. We consider only nontrivial distributions where $0 < p < 1$. Note that otherwise the data sequence does not contain all positive examples. The resulting data sequences are referred to as to binomially distributed with parameter p.

We would like to compute the expected number of examples taken by $\mathcal{IMLP}$ until convergence.

The first example received forces $\mathcal{IMLP}$ to delete precisely n of the $2n$ literals in h_0. Thus, this example always plays a *special* role.

Note that the resulting hypothesis h_0 depends on b_0, but the number k of literals that remain to be deleted from h_0 until convergence is *independent* of b_0. It is therefore convenient to compute the expectation by considering $\mathcal{IMLP}$'s behavior after having read the first example.

For the following analysis, we will denote by **CON** a random variable that counts the number of examples till the algorithm has converged to a correct hypothesis.

Theorem 5.4. *Let $c = L(m)$ be a non-minimal concept in $\mathcal{C}_n$, and let the positive examples for c be binomially distributed with parameter p. Define $\psi := \min\{\frac{1}{1-p}, \frac{1}{p}\}$ and $\tau = \max\{\frac{p}{1-p}, \frac{1-p}{p}\}$. Then the expected number of positive examples needed by algorithm $\mathcal{IMLP}$ until convergence can be bounded by*

$$E[\mathrm{CON}] \le \lceil \log_\psi k(m) \rceil + \tau + 2 \, .$$

Proof. Let $k := k(m)$. The first positive example contains ν times a 1 and $k - \nu$ many 0 with probability $\binom{k}{\nu} p^\nu q^{k-\nu}$ at the positions not corresponding to a literal in the target monomial m. Now, assuming any such vector, we easily see that h_0 contains ν positive irrelevant literals and $k - \nu$ negative literals. Therefore, in order to achieve convergence, the algorithm $\mathcal{IMLP}$ now needs positive examples that contain at least one 0 for each positive irrelevant literal and at least one 1 for each negative irrelevant literal. Thus, the probability that at least one irrelevant literal survives μ subsequent positive examples is bounded by $\nu p^\mu + (k - \nu) q^\mu$. Therefore,

$$\Pr(\mathrm{CON} - 1 > \mu) \le \sum_{\nu=0}^{k} \binom{k}{\nu} p^\nu q^{k-\nu} \cdot (\nu p^\mu + (k - \nu) q^\mu) \, .$$

Next, we derive a closed formula for the sum given above.

Claim 1. $\sum_{\nu=0}^{k}\binom{k}{\nu}p^\nu q^{k-\nu}\cdot\nu = kp$ and $\sum_{\nu=0}^{k}\binom{k}{\nu}p^\nu q^{k-\nu}\cdot(k-\nu) = kq$

The first equality can be shown as follows.

$$\begin{aligned}\sum_{\nu=0}^{k}\binom{k}{\nu}p^\nu q^{k-\nu}\cdot\nu &= \sum_{\nu=1}^{k}\binom{k}{\nu}p^\nu q^{k-\nu}\cdot\nu\\ &= \sum_{\nu=0}^{k-1}\binom{k}{\nu+1}p^{\nu+1}q^{k-1-\nu}\cdot(\nu+1)\\ &= \sum_{\nu=0}^{k-1}k\cdot\binom{k-1}{\nu}p^{\nu+1}q^{k-(\nu+1)}\\ &= kp\cdot\sum_{\nu=0}^{k-1}\binom{k-1}{\nu}p^\nu q^{(k-1)-\nu}\\ &= kp\cdot(p+q)^{k-1} = kp\ .\end{aligned}$$

The other equality can be proved analogously, which yields Claim 1.

Now, proceeding as above, we obtain

$$\begin{aligned} & E[\mathrm{CON}-1] = \sum_{\mu=0}^{\infty}\Pr(\mathrm{CON}-1 > \mu)\\ \le\ & \lambda + \sum_{\mu=\lambda}^{\infty}\sum_{\nu=0}^{k}\binom{k}{\nu}p^\nu q^{k-\nu}\cdot(\nu p^\mu + (k-\nu)q^\mu)\\ =\ & \lambda + \sum_{\mu=\lambda}^{\infty}\sum_{\nu=0}^{k}\binom{k}{\nu}p^\nu q^{k-\nu}\cdot\nu p^\mu + \sum_{\mu=\lambda}^{\infty}\sum_{\nu=0}^{k}\binom{k}{\nu}p^\nu q^{k-\nu}\cdot(k-\nu)q^\mu\\ =\ & \lambda + \sum_{\mu=\lambda}^{\infty}p^\mu\cdot\underbrace{\sum_{\nu=0}^{k}\binom{k}{\nu}p^\nu q^{k-\nu}\nu}_{=kp \quad \text{by Claim 1}} + \sum_{\mu=\lambda}^{\infty}q^\mu\cdot\underbrace{\sum_{\nu=0}^{k}\binom{k}{\nu}p^\nu q^{k-\nu}\cdot(k-\nu)}_{=kq \quad \text{by Claim 1}}\\ =\ & \lambda + kp\cdot\sum_{\mu=\lambda}^{\infty}p^\mu + kq\cdot\sum_{\mu=\lambda}^{\infty}q^\mu \ = \ \lambda + k\cdot\left(\frac{p}{q}\cdot p^\lambda + \frac{q}{p}\cdot q^\lambda\right)\\ \le\ & \lambda + k\cdot\left(\frac{p}{q}\cdot\psi^{-\lambda} + \frac{q}{p}\cdot\psi^{-\lambda}\right)\\ \le\ & \lambda + k\psi^{-\lambda}\cdot(1+\tau)\ .\end{aligned}$$

Finally, choosing $\lambda = \lceil\log_\psi k\rceil$ gives the statement of the theorem. ∎

As a corollary we get

Corollary 5.5. *For every binomially distributed positive presentation with parameter* $0 < p < 1$ *the average total learning time of algorithm* $\mathcal{IMLP}$ *for concepts in* $\mathcal{C}_n$ *is at most* $O(n \log n)$. *More precisely, a concept* $c = L(m)$ *requires time* $O(n \log(n - \#(m) + 2))$ *on the average.*

The expectation alone does not provide complete information about the average case behavior of an algorithm. It is helpful to know larger moments, too, in particular the variance. Then one can deduce bounds how often the algorithm exceeds the average considerably by applying, for example, Chebyshev's inequality. If the variance is not available, Markov's inequality provides us with worse tail bounds:

If X is any random variable taking only positive real values then

$$\Pr(X \geq t \cdot E[X]) \leq \frac{1}{t} \text{ for all } k \geq 1 .$$

Markov's inequality is quite general but produces only weak bounds.

However, Algorithm $\mathcal{IMLP}$ possesses two favorable properties that simplify the analysis considerably, it is ***set-driven*** and ***conservative***. They allow to establish good bounds for the tail probabilities.

Set-driven means that the output depends only on the *range* of the input sequence. More formally, for all $c \in \mathcal{C}_n$ all $t,\ \hat{t} \in text(c)$ and all $i, j \in \mathbb{N}^+$ the equality $t_i^+ = \hat{t}_j^+$ of the range of the two prefixes implies $\mathcal{IMLP}(t_i) = \mathcal{IMLP}(\hat{t}_j)$.

It is easy to see that Algorithm $\mathcal{IMLP}$ fulfills this property.

Furthermore, a learner is said to be *conservative* if every mind change is caused by an inconsistency with the data seen so far. Algorithm $\mathcal{IMLP}$ satisfies this condition, too, i.e., for all $c \in \mathcal{C}_n$, all $t \in text(c)$ and all $i,\ j \in \mathbb{N}^+$ it holds: if $\mathcal{IMLP}(t_i) \neq \mathcal{IMLP}(t_{i+j})$ then $t_{i+j}^+ \not\subseteq L(\mathcal{IMLP}(t_i))$.

Now, we can apply the following theorem to obtain exponentially shrinking tail bounds for the expected number of examples needed in order to achieve convergence.

Theorem 5.6 (Rossmanith and Zeugmann [1]). *Let* CON *be the sample complexity of a conservative and set-driven learning algorithm. Then for arbitrary* $t \in \mathbb{N}$ *it holds*

$$\Pr\left(\text{CON} > 2\, t \cdot E[\text{CON}]\ \right) \ \leq\ 2^{-t} .$$

Since Theorem 5.6 is of central importance, we also provide a proof. First, recall the definition of *median*. If X is a random variable then μX is a median of X iff

$$\Pr(X \geq \mu X) \geq 1/2 \text{ and } \Pr(X \leq \mu X) \geq 1/2.$$

A nonempty set of medians exists for each random variable and consists either of a single real number or of a closed real interval. We will denote the smallest median of X by μX, since this choice gives the best upper bounds. Now, we can show the following theorem.

Theorem 5.7 (Rossmanith and Zeugmann [1]). *Let X be the sample complexity of a conservative and set-driven learning algorithm. Then* $\Pr(X \geq t \cdot \mu X) \leq 2^{-t}$ *for all* $t \in \mathbb{N}$.

Proof. We divide the data sequence $d_0, d_1, \ldots$ into blocks of length μX. The probability that the algorithm converges after reading any of the blocks is then at least $1/2$. Since the algorithm is set-driven the order of the blocks does not matter and since the algorithm is conservative it does not change its hypothesis after computing once the right hypothesis. ∎

Theorem 5.6 now follows by taking into account that $\mu X \leq 2E[X]$ for every positive random variable X in accordance with the Markov inequality. Therefore,

$$\Pr(X \geq 2t \cdot E[X]) \leq 2^{-t} \quad \text{for every } t \in \mathbb{N} \,.$$

A simple calculation shows that in case of exponentially shrinking tail bounds the variance is bounded by $O(E[\mathrm{CON}]^2)$.

So, the deviation from the average will be quite small due to our exponentially shrinking tail bounds. In other words, Algorithm $\mathcal{IMLP}$ is a quite data efficient learning algorithm on average. This should surprise you a bit, since it can be as inefficient as you like in the worst-case. Thus, for many practical problems, like the one exemplified in the last lectures, analyzing the average-case behavior is of much more practical importance than studying the worst case complexity.

References

[1] P. ROSSMANITH AND T. ZEUGMANN, Stochastic Finite Learning of the Pattern Languages, *Machine Learning* **44**, No. 1-2, 2001, 67–91.

[2] R. WIEHAGEN, Limes-Erkennung rekursiver Funktionen durch spezielle Strategien. *Journal of Information Processing and Cybernetics (EIK)* **12**, 1976, 93–99.

Lecture 6: Stochastic Finite Learning and PAC Learning

6.1. Stochastic Finite Learning

After having analyzed the average-case behavior of Algorithm $\mathcal{IMLP}$ we want to address our remaining point of concern. As pointed out in Lecture 3, a limit learner is only supposed to converge in the limit, but *never knows* whether or not it already did so. Such an uncertainty may be prohibitive in many applications of learning. So, how can we recover?

Of course, we cannot achieve certainty concerning convergence, since otherwise convergence would be decidable. But, as we have already mentioned a couple of times, convergence is undecidable in general. But even if convergence is decidable as in the case of learning monomials from positive and negative data, it is practically infeasible to decide whether or not the learner did already converge.

Therefore, we have to replace certainty concerning convergence by a weaker requirement. Again, recall that there are always optimistic, pessimistic and probabilistic people around. So, if we suppose optimistic people to optimistically assume that the learner has already converged and pessimistic people to pessimistically assume that the learner did not yet converge, then it remains to explain what probabilistic people can do.

Looking at real life examples, we see that we quite often have a confidence in doing something which is based on experience. Thus, we introduce a new parameter δ into our learning model called ***confidence parameter***.

Then, in the following we always assume a class $\mathcal{D}$ of admissible probability distributions over the relevant learning domain. Ideally, this class should be parameterized. Furthermore, the data fed to learner are generated randomly with respect to one of the probability distributions from the class $\mathcal{D}$ of underlying probability distributions.

Additionally, the learner takes the confidence parameter δ as input. But in contrast to learning in the limit, the learner *itself* decides how many examples it wants to read. Then it computes a hypothesis, outputs it and stops. The hypothesis output is correct for the target with probability at least $1-\delta$.

The explanation given so far explains how it works, but not why it does. Intuitively, the stochastic finite learner simulates the limit learner until an upper bound for twice the expected total number of examples needed until convergence has been met. Assuming this to be true, by Markov's inequality the limit learner has now converged with probability $1/2$. All what is left, is to decrease the probability of failure. This is done by using the tail bounds for CON. Applying Theorem 5.6, one easily sees that increasing the sample complexity by a factor of $O(\log \frac{1}{\delta})$ results in a probability of $1-\delta$ for having reached the stage of convergence. If Theorem 5.6 is not applicable, one can still use Markov's inequality but then the sample complexity needed will increase by a factor of $1/\delta$.

It remains to explain how the stochastic finite learner can calculate the upper bound for $E[\mathrm{CON}]$. This is precisely the point where we need the parameterization of the class $\mathcal{D}$ of underlying probability distributions. Since in general, it is not known which distribution from $\mathcal{D}$ has been chosen, one has to assume a bit of *prior knowledge* or *domain knowledge* provided by suitable upper and/or lower bounds for the parameters involved. A more serious difficulty is to incorporate the unknown target concept into this estimate. This step depends on the concrete learning problem on hand, and requires some extra effort. We shall exemplify it below. Figure 6.2 displays the basic features of stochastic finite learning. You should compare it to learning in the limit (cf. Figure 2.1 at Page 23).

Depicted when Learning finishes:

$$\boldsymbol{h} \xleftarrow{\text{Out}} \boxed{M} \xleftarrow{\text{In}} \boldsymbol{\delta}, \underbrace{\boldsymbol{d_t, \ldots, d_0}}_{\textbf{drawn w.r.t.s. } D \in \mathcal{D}}$$

M* learns** $(\mathcal{C}, \mathcal{D})$ **stochastically finite** $\overset{\text{def}}{\Leftrightarrow}$ for all $\delta \in (0,1)$, after having read a finite number of examples, M outputs a ***single hypothesis $h \in \mathcal{H}$, and ***stops***. With probability at least $1 - \delta$ (w.r.t. $\mathcal{D}$) h has to be correct, i.e., $L(h) = c$. (Note that t may depend on δ.)

Figure 6.2: Stochastic Finite Learning

Now we are ready to formally define stochastic finite learning.

Definition 14 ([4, 5, 6]). *Let $\mathcal{D}$ be a set of probability distributions on the learning domain, $\mathcal{C}$ a concept class, $\mathcal{H}$ a hypothesis space for $\mathcal{C}$, and $\delta \in (0,1)$. $(\mathcal{C}, \mathcal{D})$ is said to be* stochastically finitely learnable with δ-confidence *with respect to $\mathcal{H}$ iff there is an IIM M that for every $c \in \mathcal{C}$ and every $D \in \mathcal{D}$ performs as follows. Given any random data sequence θ for c generated according to D, M stops after having seen a finite number of examples and outputs a single hypothesis $h \in \mathcal{H}$. With probability at least $1 - \delta$ (with respect to distribution D) h has to be correct, that is $c = h$.*

If stochastic finite learning can be achieved with δ-confidence for every $\delta > 0$ then we say that $(\mathcal{C}, \mathcal{D})$ can be learned stochastically finite with high confidence.

Next, we turn our attention to the design of a stochastic finite learner for learning monomials from positive data. We study the case that the positive examples are binomially distributed with parameter p. But we do not require precise knowledge about the underlying distribution. Instead, we reasonably assume that *prior knowledge* is provided by parameters p_{low} and p_{up} such that $p_{low} \leq p \leq p_{up}$ for the true parameter p. Binomial distributions fulfilling this requirement are called $\boldsymbol{(p_{low}, p_{up})}$***–admissible distributions.*** Let $\boldsymbol{\mathcal{D}_n[p_{low}, p_{up}]}$ denote the set of such distributions on $\mathcal{X}_n$.

If bounds p_{low} and p_{up} are available, the Algorithm $\boldsymbol{\mathcal{IMLP}}$ can be transformed into a stochastic finite learner inferring all concepts from $\mathcal{C}_n$ with high confidence.

Theorem 6.1 (Reischuk and Zeugmann [4]). *Let $0 < p_{low} \leq p_{up} < 1$ and $\psi := \min\{\frac{1}{1-p_{low}}, \frac{1}{p_{up}}\}$. Then $(\mathcal{C}_n, \mathcal{D}_n[p_{low}, p_{up}])$ is stochastically finitely learnable with high* *confidence from text. To achieve δ-confidence no more than $O\left(\log_2 1/\delta \cdot \log_\psi n\right)$ many examples are necessary.*

Proof. The stochastic finite learner is based on Algorithm $\mathcal{IMLP}$ and a counter for the number of examples already processed. We set

$$\tau_{\max} = \left\lceil \max\left\{ \frac{p_{low}}{1-p_{low}}, \frac{1-p_{low}}{p_{low}}, \frac{p_{up}}{1-p_{up}}, \frac{1-p_{up}}{p_{up}} \right\} \right\rceil .$$

If Algorithm $\mathcal{IMLP}$ is run for $\vartheta := \lceil \log_\psi n \rceil + \tau_{\max} + 2$ many examples, Theorem 5.4 implies that ϑ is at least as large as the expected convergence stage $E[\mathrm{CON}]$.

In order to achieve the desired confidence, the learner sets $\gamma := \lceil \log \frac{1}{\delta} \rceil$ and runs Algorithm $\mathcal{IMLP}$ for a total of $2\ \gamma \cdot \vartheta$ examples. This is the reason why we need a counter for the number of examples processed. The algorithm outputs the last hypothesis $h_{2\ \gamma\cdot\vartheta}$ produced by Algorithm $\mathcal{IMLP}$ and stops thereafter. The reliability follows from the tail bounds established in Theorem 5.6. ▮

So far, our results provide evidence that analyzing the average-case behavior of limit learners with respect to their total learning time may be considered as a promising path towards a new theory of efficient algorithmic learning. But what happens if we have to learn monomials from both positive and negative examples? We shall address this question in the following.

6.2. Learning Monomials from Informant

Next, we ask how the results obtained so far translate to the case of learning from informant. Since Algorithm $\mathcal{IML}$ does not learn anything from negative examples, one may expect that it behaves much poorer in this setting. First, we investigate the uniform distribution over $\mathcal{X}_n$. Again, we have the trivial cases that the target concept is “FALSE” or m is a monomial without irrelevant bits. In the first case, no example is needed at all, while in the latter one, there is only one positive example having probability 2^{-n}. Thus the expected number of examples needed until successful learning is $2^n = 2^{\#(m)}$.

Theorem 6.2. *Let $c = L(m) \in \mathcal{C}_n$ be a nontrivial concept. If a data sequence for c is generated from the uniform distribution on the learning domain by independent draws the expected number of examples needed by Algorithm $\mathcal{IML}$ until convergence is bounded by*

$$E[\mathrm{CON}] \ \leq \ 2^{\#(m)} \left(\lceil \log_2 k(m) \rceil + 3 \right) .$$

Proof. Let $\mathrm{CON}+$ be a random variable for the number of positive examples needed until convergence. Every positive example is preceded by a possibly empty block of negative examples. Thus, we can partition the initial segment of any randomly drawn informant read until convergence into $\mathrm{CON}+$ many blocks B_j containing a certain number of negative examples followed by precisely one positive example. Let Λ_j be a random variable for the length of block B_j. Then $\mathrm{CON} = \Lambda_1 + \Lambda_2 + \cdots + \Lambda_{\mathrm{CON}+}$, where the Λ_j are independently identically distributed. In order to compute the distribution of Λ_j, it suffices to calculate the probabilities to draw a negative and a positive example, respectively. Since the overall number of positive examples for c is 2^k with $k = k(m)$, the probability to generate a positive example is 2^{k-n}. Hence, the probability to draw a negative example is $1 - 2^{k-n}$. Consequently,

$$\Pr[\Lambda_j = \mu + 1] \;=\; \left(1 - 2^{k-n}\right)^{\mu} \cdot 2^{k-n}\;.$$

Therefore,

$$\begin{aligned} E[\mathrm{CON}] &= E[\Lambda_1 + \Lambda_2 + \cdots + \Lambda_{\mathrm{CON}+}] \\ &= \sum_{\zeta=0}^{\infty} E[\Lambda_1 + \Lambda_2 + \cdots + \Lambda_\zeta \mid \mathrm{CON}+ = \zeta] \;\cdot\; \Pr[\mathrm{CON}+ = \zeta] \\ &= \sum_{\zeta=0}^{\infty} \zeta \;\cdot\; E[\Lambda_1] \;\cdot\; \Pr[\mathrm{CON}+ = \zeta] \\ &= E[\mathrm{CON}+] \;\cdot\; E[\Lambda_1] \end{aligned}$$

By Markov's inequality, we have $E[\mathrm{CON}+] \le \lceil \log_2 k \rceil + 3$, and thus it remains to estimate $E[\Lambda_1]$. A simple calculation shows

Lemma 6.3. *For every* $0 < a < 1$ *holds:*

$$\sum_{\mu=0}^{\infty} (\mu + 1) \cdot a^{\mu} \;=\; (1 - a)^{-2}\;.$$

Using this estimation we can conclude

$$\begin{aligned} E[\Lambda_1] &= \sum_{\mu=0}^{\infty} (\mu + 1) \;\cdot\; \Pr[\Lambda_1 = \mu + 1] \\ &= 2^{k-n} \sum_{\mu=0}^{\infty} (\mu + 1) \cdot \left(1 - 2^{k-n}\right)^{\mu} \;=\; 2^{n-k}\;, \end{aligned}$$

and thus the theorem follows. ∎

Hence, as long as the length of m is constant, and therefore $k(m) = n - O(1)$, we still achieve an expected total learning time of order $n \log n$. But if $\#(m)$ grows linearly the expected total learning becomes exponential. On the other hand, if there are many relevant literals then even h_0 may be considered as a not too bad *approximation* for c. Consequently, it is natural at this point to introduce an error parameter $\varepsilon \in (0, 1)$ as in the PAC model (see below), and to ask whether one can achieve an expected sample complexity for computing an ε-approximation that is bounded by a function depending on $\log n$ and $1/\varepsilon$.

To answer this question, let us formally define $error_m(h_j) = D(L(h_j) \triangle L(m))$ to be the error made by hypothesis h_j with respect to monomial m. Here $L(h_j) \triangle L(m)$ stands for the symmetric difference of $L(h_j)$ and $L(m)$ and D for the underlying probability distribution with respect to which the examples are drawn. Note that by construction of Algorithm $\mathcal{IML}$ we can conclude $error_m(h_j) = D(L(m) \setminus L(h_j))$.

We call h_j an *ε–approximation* for m if $error_m(h_j) \leq \varepsilon$. Finally, we redefine the stage of convergence. Let m be any monomial, and let $d = (d_j)_{j\in\mathbb{N}^+}$ be an informant for $L(m)$, then

$$\mathbf{CON}_{\boldsymbol{\varepsilon}}(\boldsymbol{d}) \quad =_{df} \quad \text{the least number } j \text{ such that } error_m(\mathcal{IML}\ (d_i)) \leq \varepsilon \ \text{ for all } i \geq j \ .$$

Note that once the Algorithm $\mathcal{IML}$ has reached an ε-approximate hypothesis all further hypotheses will also be at least that close to the target monomial.

The following theorem gives an affirmative answer to the question posed above.

Theorem 6.4. *Let $c = L(m) \in \mathcal{C}_n$ be a nontrivial concept. Assuming that examples are drawn at random independently from the uniform distribution, the expected number of examples needed by Algorithm $\mathcal{IML}$ until converging to an ε–approximation for c can be bounded by*

$$E[\mathrm{CON}_\varepsilon] \;\leq\; \frac{1}{\varepsilon}\cdot(\lceil\log_2 k(m)\rceil + 3)\ .$$

Proof. It holds $error_m(h_0) = 2^{k(m)-n}$, since h_0 misclassifies exactly the positive examples. Therefore, if $error_m(h_0) \leq \varepsilon$, we are already done. Now suppose $error_m(h_0) > \varepsilon$. Consequently, $1/\varepsilon > 2^{n-k(m)}$, and thus the bound stated in the theorem is larger than $2^{n-k(m)}(\lceil\log_2 k(m)\rceil + 3)$, which, by Theorem 6.2 is the expected number of examples needed until convergence to a correct hypothesis. ∎

Thus, additional knowledge concerning the underlying probability distribution pays off again. Applying Theorem 5.6 and modifying the stochastic finite learner presented above *mutatis mutandis*, we get a learner identifying ε-approximations for all concepts in $\mathcal{C}_n$ stochastically with high confidence using $O(\frac{1}{\varepsilon}\cdot\log\frac{1}{\delta}\cdot\log n)$ many examples. Comparing this bound with the sample complexity given in the PAC model, we see that it is reduced exponentially, i.e., instead of a factor n now we have the factor $\log n$ (cf. Theorem 6.5 below).

So, let us continue with the PAC model. In contrast to our considerations above we do not assume any prior knowledge concerning the underlying probability distribution.

6.3. PAC Learning

PAC stands for *probably approximately correct* and the corresponding learning model goes back to Valiant [6]. Comprehensive treatises of this topic include Anthony and Biggs [1], Kearns and Vazirani [2] as well as Natarajan [3]. We use our example, i.e., the set of all concepts describable by a monomial over $\mathcal{L}_n$ to explain the basic ideas behind the PAC approach. The main difference to the models considered so far starts with the *source of information*. We assume an unknown probability distribution D over the learning domain $\mathcal{X}$. There is a *sampling oracle* $EX(\ )$, which has no input. Whenever $EX(\ )$ is called, it draws an element $x \in \mathcal{X}$ according to the distribution D and returns the element x together with an indication of whether or not x belongs to the target concept c. Thus, every example returned by $EX(\ )$ may be written as $(x, c(x))$, where $c(x) = 1$ if $x \in c$ and $c(x) = 0$ otherwise. If we make s calls to the example $EX(\ )$ then the elements $x_1, \ldots x_s$ are drawn independently from one another. Thus, the resulting probability distribution over all s-tuples of elements from $\mathcal{X}$ is the s-fold product distribution of D, i.e.,

$$\Pr(x_1, \ldots, x_s) = \prod_{i=1}^{s} D(x_i).$$

In the following, we use $\Pr(A)$ to denote the probability of event A, where A is a set of s-tuples over $\mathcal{X}$, $s \geq 1$. The actual s will be always clear from the context.

The *criterion of success*, i.e., *probably approximately correct* learning, is parameterized with respect to two quantities, the *accuracy parameter* ε, and the confidence parameter δ, where $\varepsilon, \delta \in (0,1]$. Next, we define a notion of the difference between two sets $c, c' \subseteq \mathcal{X}$ with respect to the probability distribution D as

$$d(c, c') = \sum_{x \in c \triangle c'} D(x).$$

A learning method $\boldsymbol{\mathcal{A}}$ is said to ***probably approximately correctly*** *identify a target concept c with respect to a hypothesis space $\mathcal{H}$, and with sample complexity $s = s(\varepsilon, \delta)$, if for all $\varepsilon, \delta \in (0,1)$ it makes s calls to the oracle $EX(\ )$, and after having received the answers produced by $EX(\ )$ (with respect to the target c), it always stops and outputs a hypothesis $h \in \mathcal{H}$ such that*

$$\Pr(d(c,h) \leq \varepsilon) \geq 1 - \delta.$$

A learning method $\boldsymbol{\mathcal{A}}$ is said to ***probably approximately correctly*** *identify a target concept class $\mathcal{C}$ with respect to a hypothesis space $\mathcal{H}$ and with sample complexity $s = s(\varepsilon, \delta)$, if it probably approximately correctly identifies every concept $c \in \mathcal{C}$ with respect to $\mathcal{H}$, and with sample complexity s.*

Finally, a learning method is said to be efficient with respect to sample complexity, if there exists a polynomial pol such that $s \leq pol(1/\varepsilon, 1/\delta)$.

O.k, this looks fairly complicated, and hence, some explanation is in order. First of all, the inequality

$$\Pr(d(c,h) \leq \varepsilon) \geq 1 - \delta$$

means that with high probability (quantified by δ) there is not too much difference (quantified by ε) between the conjectured concept (described by h) and the target concept c. Formally, let $\boldsymbol{\mathcal{A}}$ be any fixed learning method. Let $s = s(\varepsilon, \delta)$ for any fixed $\varepsilon, \delta \in (0,1)$ be the actual sample size. Furthermore, let c be any fixed target concept. Now, we have to consider all possible outcomes of $\boldsymbol{\mathcal{A}}$ when run on every labeled s-sample $S(c, \bar{x}) = (x_1, c(x_1), \ldots, x_s, c(x_s))$ returned by $EX(\ )$. Let $h(S(c, \bar{x}))$ denote the hypothesis produced by $\boldsymbol{\mathcal{A}}$ when processing $S(c, \bar{x})$. Then we have to consider the set W of all s-tuples over $\mathcal{X}$ such that $d(c, h(S(c, \bar{x}))) \leq \varepsilon$. The condition $\Pr(d(c,h) \leq \varepsilon) \geq 1 - \delta$ can now be formally rewritten as $\Pr(W) \geq 1 - \delta$. Clearly, one has to require that $\Pr(W)$ is well defined. This is obvious as long as $\mathcal{X}$ is finite.

In order to exemplify this approach, remember that our set of all concepts describable by a monomial over $\mathcal{L}_n$ actually refers to the set of all things. We consider a hypothetical learner (e.g., a student, a robot) that has to learn the concept of a chair. Imagine that the learner is told by some teacher whether or not particular things visible by the learner are instances of a chair. Clearly, what things are visible depends on the environment the learner is in. The formal description of this dependence is provided by the unknown probability distribution. For example, the learner might be led to a kitchen, a sitting room, a book shop, a garage, a beach a.s.o. Clearly, it would be unfair to teach you the concept of a chair in a book shop, and then testing your learning success in a sitting room. Therefore, the learning success is measured with respect to the same probability distribution D with respect to which the sampling oracle has drawn its examples. However, the learner is required to learn with respect to any probability distribution. That is, independently of whether the learner is led to a kitchen, a book shop, a sitting room, a garage, a beach a.s.o., it has to learn with respect to the place it has been led to. The sample complexity refers to the amount of information needed to ensure successful learning. Clearly, the smaller the required distance of the hypothesis produced, and the higher the confidence desired, the more examples are usually needed. However, there might be atypical situations. To have an extreme example, the kitchen the learner is led to turned out to be empty. Since the learner is required to learn with respect to a typical kitchen (described by the probability distribution D) it may well fail under this particular circumstance. Nevertheless, such failure has to be restricted to atypical situations. This requirement is expressed by demanding the learner to be successful with confidence $1 - \delta$.

This corresponds well to real life situations. For example, a student who has attended a course in probability theory might well suppose that he/she is examined in probability theory and not in graph theory. However, a good student, say in computer science, has to pass all examinations successfully, independently of the particular course attended. That is, he must successfully pass examinations in computability theory, complexity theory, cryptology, parallel algorithms, formal languages, recursion theory, learning theory, graph theory, combinatorial algorithms, logic programming, a.s.o. Hence, he/she has to learn a whole concept class. The sample complexity refers to the time of interaction performed by the student and his/her teacher.

Now, we are ready to prove the PAC learnability of our concept class. We use the same notation as above.

Theorem 6.5. *The set of all monomials over $\mathcal{L}_n$ can be probably approximately correctly learned with respect to the hypothesis space $\mathcal{H}$ and with sample complexity $s = O(1/\varepsilon \cdot (n + \ln(1/\delta)))$.*

Proof. As a matter of fact, we can again use a suitable modification of algorithm $\boldsymbol{\mathcal{P}}$ presented in Lecture 2.

***Algorithm $\boldsymbol{\mathcal{PA}}$*:** "For all $\varepsilon, \delta \in (0, 1)$, call the oracle $EX(\ )$ s times, where $s = O(1/\varepsilon \cdot$

```
$(n + \ln(1/\delta)))$. Let $\langle b_1, m(b_1), b_2, m(b_2), \ldots, b_s, m(b_s)\rangle$ be the sequence returned by
$EX(\ )$. Let $b_i = b_i^1 b_i^2 \ldots b_i^n$ denote the $i$th vector $b_i \in \mathcal{X}$ returned.
Initialize $h = x_1\bar{x}_1 \ldots x_n\bar{x}_n$.
For $i = 1, 2, \ldots, s$ do
if $m(b_i) = 1$ then
for $j := 1$ to $n$ do
if $b_i^j = 1$ then delete $\bar{x}_j$ in $h$ else delete $x_j$ in $h$.
Output $h$.
end."
```

Let D be any probability distribution over $\mathcal{X}$. Furthermore, let m be any target monomial. We have to show that the algorithm $\mathcal{PA}$ outputs with confidence at least $1 - \delta$ a hypothesis h such that $d(m, h) \leq \varepsilon$. We easily observe that

$$d(m, h) = \sum_{b \in m \triangle h} D(b) = D(\{b \in \mathcal{X} \mid m(b) \neq h(b)\}).$$

Since the algorithm $\mathcal{PA}$ is essentially the same as Algorithm $\mathcal{P}$ we can exploit the proof of Theorem 2.1. First of all, for every target monomial m, if h is the hypothesis output by $\mathcal{PA}$ then $h(b_i) = m(b_i)$ for all $i = 1, \ldots, s$. Recall that such hypotheses are said to be *consistent*.

Now, suppose any particular hypothesis $h \in \mathcal{H}$ such that $d(m, h) > \varepsilon$. Any such hypothesis will not be consistent with s randomly drawn examples unless all examples are drawn outside the symmetric difference of m and h. Let $b \in \mathcal{X}$ be any randomly chosen vector. Then we have: the probability that $m(b) = h(b)$ is bounded by $1 - \varepsilon$. Hence, if we have s randomly and independently drawn vectors $b_1, \ldots, b_s \in \mathcal{X}$, then the probability that $m(b_i) = h(b_i)$ for all $i = 1, \ldots, s$ is bounded by $(1 - \varepsilon)^s$. Furthermore, $(1 - \varepsilon)^s < e^{-\varepsilon s}$.

Additionally, there are $\mathrm{card}(\mathcal{H})$ many possible choices for h.

Thus, the overall probability that s randomly and independently drawn vectors do not belong to $m \triangle h$, for any $h \in \mathcal{H}$, is bounded by $\mathrm{card}(\mathcal{H})e^{-\varepsilon s}$.

Therefore, if $s > 1/\varepsilon \cdot \ln \frac{\mathrm{card}(\mathcal{H})}{\delta}$ we get:

$$\mathrm{card}(\mathcal{H})e^{-\varepsilon s} < \mathrm{card}(\mathcal{H})e^{-\ln \frac{\mathrm{card}(\mathcal{H})}{\delta}} = \delta.$$

Consequently, since all hypotheses h output by $\mathcal{PA}$ are consistent, now we know that

$$\Pr(d(m, h) > \varepsilon) < \delta, \text{ and thus}$$

$$\Pr(d(m, h) \leq \varepsilon) \geq 1 - \delta.$$

Finally, by Exercise 1 we know that $\mathrm{card}(\mathcal{H}) = 3^n + 1$, hence the theorem follows. ∎

Next, we consider *disjunctions* over $\mathcal{L}_n$, i.e., all expressions $f = \ell_{i_1} \vee \ldots \vee \ell_{i_k}$, where $k \le n$, $i_1 < \ldots < i_k$, and all $\ell_{i_j} \in \mathcal{L}_N$ for $j = 1, \ldots, k$. Hence, disjunctions are the logical duals of monomials. Additionally, we include $f = x_1 \vee \bar{x}_1 \vee \ldots \vee x_n \vee \bar{x}_n$ into the set of all disjunctions over $\mathcal{L}_n$ to represent the concept "TRUE." Furthermore, let $b = b^1 \ldots b^n \in \mathcal{X}$, then $f(b) = \ell_{i_1}(b^{i_1}) \vee \ldots \vee \ell_{i_k}(b^{i_k})$, where $\ell_{i_j}(b^{i_j}) = 1$ iff $\ell_{i_j} = x_i$ for some i, and $b^{i_j} = 1$ or $\ell_{i_j} = \bar{x}_i$ for some i, and $b^{i_j} = 0$. Otherwise, $\ell_{i_j}(b^{i_j}) = 0$. Then, if f is a disjunction over $\mathcal{L}_n$ we set $L(f) = \{b \in \mathcal{X} \mid f(b) = 1\}$. Finally, let $\mathcal{C}$ be the set of all concepts describable by a disjunction over $\mathcal{L}_n$, and let $\hat{\mathcal{H}}$ be the hypothesis space that consists of all disjunctions as described above.

Exercise 13. Prove or disprove.

(a) The set of all disjunctions over $\mathcal{L}_n$ can be probably approximately correctly learned with respect to the hypothesis space $\hat{\mathcal{H}}$ and with sample complexity $s = O(1/\varepsilon \cdot (n + \ln(1/\delta)))$.

Next, we continue with a closer at probably approximately correct learning. So far in our study we have only proved the class of concepts describable by a monomial to be PAC learnable. Therefore, we are interested in gaining a better understanding of what finite concept classes are PAC learnable. Furthermore, we aim to derive general bounds on the sample complexity needed to achieve successful PAC learning provided the concept class under consideration is PAC identifiable at all.

Moreover, concept classes of infinite cardinality make up a large domain of important learning problems. Therefore, it is only natural to ask whether or not there are interesting infinite concepts classes which are PAC learnable, too. The affirmative answer will be provided in the following lectures. For the sake of presentation, we start with a general analysis of PAC learnability for finite concept classes. Subsequently, we investigate the case of infinite concept classes.

6.3.1. PAC Learning - the Finite Case

Let $\mathcal{X}$ be any finite learning domain, let D be any probability distribution over $\mathcal{X}$, and let $\mathcal{C} \subseteq \wp(\mathcal{X})$ be a concept class. Furthermore, we use $\mathcal{H}$ to denote any hypothesis space for $\mathcal{C}$. Note that, in general, we do not require $\mathcal{C} \subseteq \mathcal{H}$. To simplify notation, we use $|M|$ to denote the cardinality of any set M. Let $m \in \mathbb{N}$, $m \ge 1$; then we use $\mathcal{X}^m$ to denote the m-fold Cartesian product of $\mathcal{X}$. Let $\bar{x} \in \mathcal{X}^m$, the we write $\bar{x} = (x_1, \ldots, x_m)$. Now, let $c \in \mathcal{C}$ be any concept. The m-sample of c generated by $\bar{x}$ is denoted by $S(c, \bar{x}) = \langle x_1, c(x_1), \ldots, x_m, c(x_m) \rangle$. A hypothesis $h \in \mathcal{H}$ is said to be ***consistent*** for an m-sample $S(c, \bar{x})$ iff $h(x_i) = c(x_i)$ for all $1 \le i \le m$.

The sample space S_c of a concept c is the set of all m-samples of c, i.e.,

$$S_c = \bigcup_{m \ge 1} \bigcup_{\bar{x} \in \mathcal{X}^m} S(c, \bar{x}).$$

The sample space $S(\mathcal{C})$ of a concept class $\mathcal{C}$ is the union of over all S_c, $c \in \mathcal{C}$. Then, a learner is any computable mapping from $S(\mathcal{C})$ into $\mathcal{H}$. A learner is said to be ***consistent*** iff all its outputs are consistent hypotheses.

The formal definition of PAC learning has already been presented above. Moreover, we showed the class of all monomials to be PAC learnable. The general idea behind the algorithm given there can be described as follows:

(1) Draw a sufficiently large sample from the oracle $EX(\;)$, say m examples.

(2) Find some $h \in \mathcal{H}$ that is consistent with all the examples drawn.

(3) Output h.

Therefore, it is only natural to ask whether or not this strategy may be successful in the general finite case, too. Let us assume that we have a consistent learner. Let $c \in \mathcal{C}$ be any concept, and let h be any hypothesis output by the learner on any m-sample $S(c, \bar{x})$, where $\bar{x}$ has been drawn with respect to the unknown probability distribution D. Assume h to be *bad*, i.e., $d(c, h) > \varepsilon$. Any such hypothesis will not be consistent with m randomly drawn examples unless all examples are drawn outside the symmetric difference of c and h. Hence, the probability that the particular bad hypothesis h survives m examples is at most $(1-\varepsilon)^m$. Consequently, the probability that some bad hypothesis survives m examples is at most $|\mathcal{H}|(1-\varepsilon)^m$. Furthermore, we want $Pr(d(c, h) > \varepsilon) < \delta$. Hence, we must require:

$$|\mathcal{H}|(1-\varepsilon)^m \leq \delta.$$

Now, the latter requirement directly allows to lower bound m. Taking the natural logarithm of both sides, we obtain:

$$\ln|\mathcal{H}| + m\ln(1-\varepsilon) \leq \ln\delta.$$

Therefore, we have:

$$m > \frac{\ln\delta - \ln|\mathcal{H}|}{\ln(1-\varepsilon)}$$

Because of $(1-\frac{1}{z})^z < e^{-1}$ for all $z > 0$, we additionally obtain:

$$(1-\varepsilon) = ((1-\varepsilon)^{1/\varepsilon})^\varepsilon < e^{-\varepsilon},$$

and thus

$$\ln(1-\varepsilon) < -\varepsilon.$$

Putting it all together, we see that

$$m > \frac{1}{\varepsilon}\left(\ln|\mathcal{H}| + \ln\frac{1}{\delta}\right) = \frac{1}{\varepsilon}\ln\frac{|\mathcal{H}|}{\delta}.$$

We summarize the insight obtained by the following theorem.

Theorem 6.6. *Let $\mathcal{X}$ be any finite learning domain, let $\mathcal{C} \subseteq \wp(\mathcal{X})$ be any concept class, and let $\mathcal{H}$ be any hypothesis space for $\mathcal{C}$. Then every consistent learner PAC identifies $\mathcal{C}$ with respect to $\mathcal{H}$ with sample complexity $m = \frac{1}{\varepsilon}\ln\frac{|\mathcal{H}|}{\delta} + 1$.*

The latter theorem delivers a first upper bound on the sample complexity needed to achieve efficient PAC learning. However, it does not say anything concerning the problem to compute consistent hypotheses. Clearly, there is a trivial algorithm to achieve this goal. We may just enumerate all hypotheses. Then we may simply search for the first consistent one in the enumeration fixed. Nevertheless, taking into account that $\mathcal{H}$ might be huge, this method will usually take too much time. Hence, further effort is necessary to arrive at practical learning algorithms.

The latter observation motivates us to strengthen our requirements concerning the efficiency of PAC learning. It might be not enough to bound the number of examples. Additionally, we shall demand the overall running time to be polynomial in the appropriate parameters.

Definition 15. *A concept class $\mathcal{C}$ is said to be* ***efficiently PAC learnable with respect to the hypothesis space $\mathcal{H}$*** *if $\mathcal{C}$ is PAC learnable with respect to $\mathcal{H}$, and there exists a PAC learning algorithm A for $\mathcal{C}$ that runs in time polynomial in $1/\varepsilon, 1/\delta, n$ (the size of an instance in $\mathcal{X}$), and size(c) for all $\varepsilon, \delta \in (0,1)$ and all $c \in \mathcal{C}$.*

Now, we are ready to establish the PAC learnability of a couple of important finite concept classes. A *clause* is a disjunction of literals. By k-CNF we denote the class of all conjunctions such that each clause contains at most k literals. For example,

$$(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$$

is a 3-CNF.

The overall number of clauses containing at most k literals is bounded by

$$2n + (2n)^2 + \dots (2n)^k < O(n^k) \; .$$

Hence, $\ln(|k\text{-CNF}|) = O(n^k)$. Therefore, we get the following general theorem.

Theorem 6.7. *Let $k \in \mathbb{N}^+$ be arbitrarily fixed. The class of all concepts describable by a k-CNF formula is efficiently PAC learnable with respect to k-CNF.*

Proof. On input ε, δ make $m = O\left(\frac{1}{\varepsilon}(n^k + \ln\frac{1}{\delta}\right)$ calls to the oracle $EX(\;)$. Let $S(c, \bar{b}) = \langle b_1, c(b_1), \dots, b_m, c(b_m)\rangle$ be the sample returned.

Initialize h as the conjunction of all clauses containing at most k literals.

For $i = 1, 2, \dots m$ do

if $c(b_i) = 1$ then delete all clauses in h that do not contain a literal evaluating to 1 on the assignment given by b_i.

Output h.

By construction, h is consistent. Hence, by Theorem 6.6 we know that the algorithm described above PAC learns k-CNF with respect to k-CNF. The running time of this algorithm is obviously polynomially bounded in $1/\varepsilon, 1/\delta$, and n^k. ∎

Next, by k-DNF we denote the class of all disjunctions such that each monomial contains contains at most k literals.

Exercise 14. *Prove the following: Let $k \in \mathbb{N}^+$ be arbitrarily fixed. The class of all concepts describable by a k-DNF formula is efficiently PAC learnable with respect to k-DNF.*

References

[1] M. ANTHONY AND N. BIGGS (1992), "Computational Learning Theory," Cambridge University Press, Cambridge.

[2] M.J. KEARNS AND U.V. VAZIRANI (1994), "An Introduction to Computational Learning Theory," MIT-Press.

[3] B.K. NATARAJAN (1991), "Machine Learning," Morgan Kaufmann Publishers Inc.

[4] R. REISCHUK AND T. ZEUGMANN, A Complete and Tight Average-Case Analysis of Learning Monomials, *in* "Proceedings 16th International Symposium on Theoretical Aspects of Computer Science," (C. Meinel and S. Tison, Eds.), Lecture Notes in Computer Science, Vol. 1563, pp. 414–423, Springer-Verlag , Berlin 1999.

[5] R. REISCHUK AND T. ZEUGMANN, An average-case optimal one-variable pattern language learner, *Journal of Computer and System Sciences* **60**, No. 2, 2000, 302–335.

[6] P. ROSSMANITH AND T. ZEUGMANN, Stochastic finite learning of the pattern languages, *Machine Learning* **44**, No. 1-2, 2001, 67–91.

[6] L.G. VALIANT (1984), A theory of the learnable, *Communications of the ACM* **27**, 1134 – 1142

LECTURE 7: DECISION TREE LEARNING

So far, we have mainly studied the learnability of simple Boolean concepts. That is, we looked at concepts describable by a monomial. However, over the learning domain $\{0,1\}^n$ there are only 3^n+1 many concepts that can be described by a monomial. In contrast, there are 2^{2^n} many Boolean functions. Since every Boolean function can be represented by a decision tree, we shall finish our tour on learning by learning at the learnability of decision trees. As a matter of fact, decision tree learning is used quite often in practice. So, you should know at least a bit about it. Also, we shall not restrict ourselves to the Boolean case.

For the sake of presentation, let us have a look at the whether data already used in Lecture 1. Recall that this data set is fictitious, but it serves its purpose to explain some typical features. The data concern the conditions for playing some unspecified game.

The Weather Data

`outlook`	`temperature`	`humidity`	`windy`	`play`
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

In the table displayed above, we have four *attributes*, i.e., outlook, temperature, humidity, and windy. These attributes can take symbolic values rather than numerical values. The rightmost column shows the recommendation, that is whether or not one should play. In other words, the attributes are observable variables, and the recommendation is assumed to be a function of the attributes. Let us assume that `outlook` can take the three values *sunny, overcast, rainy*, that `temperature` can take the three values *hot, mild, cool*, that `humidity` can take the two values *high, normal* and that `windy` can take the two values *true, false*. Then, the complete table should have 36 entries. In contrast, the table given has only 14 entries. This is quite a typical situation in many applications.

Next, we ask if there are any rules behind this table. This is the typical question asked in *data mining*. Here the typical problem is again to make predictions. For example, we are given the following data for tomorrow.

outlook	temperature	humidity	windy	play
sunny	cool	high	true	?

So, what should we do, play or not play?

Of course, there may be many different ways to express such rules and to find them. Decision trees are one possibility.

Thus, let us first look at a decision tree for the weather data set given above.

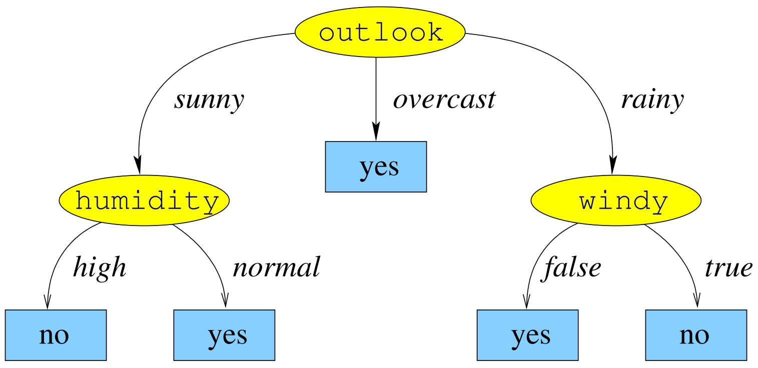

Figure 7.1: Decision tree for the weather data.

Clearly, we have to ask what this decision tree is representing.

7.1. Decision Tree Representation

Decision trees classify instances by sorting them down the tree from the root to some leaf. The label in the leaf provides the classification of the instance. Each node in the tree that is not a leaf specifies a *test* of some *attribute* of the instance by which it is labeled. Each branch descending from that node corresponds to some possible value for this attribute. Any given instance is the classified by starting at the root, testing the attribute specified there and then moving down the tree branch which corresponds to the value given. This process is then recursively repeated for the subtree rooted at the new node. For example, let $(sunny, hot, high, false)$ from the weather data set be the given instance. First, we test `outlook`, which is *sunny*. Thus, we move down the left branch of the tree shown in Figure 7.1. Next, we test the attribute humidity which has value *high*. Thus, the instance $(sunny, hot, high, false)$ is correctly classified by `no`.

In general, decision trees represent a *disjunction* of conjunctions of constraints on the attribute values of instances. For example, using the natural interpretation of `yes` as 1 the decision tree given in Figure 7.1 corresponds to the disjunction

$$\begin{gathered}(\texttt{outlook} = \mathit{sunny} \wedge \texttt{humidity} = \mathit{normal}) \\ \vee (\texttt{outlook} = \mathit{overcast}) \\ \vee (\texttt{outlook} = \mathit{rainy} \wedge \texttt{windy} = \mathit{false})\end{gathered}$$

Before we turn our attention to the problem of learning decision trees from examples we would like to give a short characteristic of decision tree learning.

Generally speaking, decision tree learning is a method for approximating discrete-valued target functions. The function learned is represented by a decision tree. Decision tree learning is robust to noisy data and thus very popular. In fact, decision tree learning is one of the most widely used and practical methods for inductive inference. There are several decision tree learning methods around that differ to a certain extent with respect to their capabilities and requirements. Nevertheless, all these algorithms also share many features. In particular, it can be said that decision tree learning is best suited for problems having the following characteristics.

1. Instances of the problem are represented by *attribute-value pairs*. The instances can be described by a fixed set of attributes and their values. For example, in Lecture 7 we presented the weather data set. The attributes in this problem are `outlook`, `temperature`, `humidity`, and `windy`. Each attribute could take a small fixed set values, e.g., for `outlook` the set of possible is {*sunny, overcast, rainy*} and for `temperature` the set of values is {*hot, mild, cool*}. It should be noted, however, that suitable extensions to the basic algorithms allow handling of real-valued attributes as well.

2. The *target function* has discrete output values. In our weather data set these output values have been {`yes, no`} and in our data set from molecular biology {`P, N`}. Of course, decision tree learning methods can also handle more than two possible discrete output values. There are also decision tree learning algorithms around that can even handle target functions with real-valued outputs, but applications of decision trees in such settings are less common.

3. *Disjunctive descriptions* may be required. As we shall see below, every function described by a decision tree allows also a representation as disjunction.

4. *The training data may contain errors*. As already mentioned, decision tree learning methods are robust to errors. The type of errors decision tree learning methods can handle comprises errors in the classification (e.g., some of the protein sequences in our example from the last lecture may have erroneously been classified as positive by the domain experts) as well as errors in the attribute values describing these examples (e.g., *normal* instead of *high* may have appeared erroneously in our weather data set as value for humidity).

5. *The training data may contain missing attribute values*. For example, some attributes in a true weather data set may be missing, since at some particular day it was impossible to measure the temperature because of a technical problem.

Since many practical problems have been found that fit the characteristics given above, decision tree learning is often used. Examples comprise the classification of medical patients by their diseases, loan applicants by their likelihood of defaulting on payment and many more.

Now, we are ready to deal with the problem how to learn decision trees.

7.2. The Basic Decision Tree Learning Algorithm

The material we are going to represent here is based on Quinlan's (1986) ID3 algorithm and its successor C4.5 (cf. Quinlan(1993)). These algorithms employ a top-down, greedy search through the set of possible decision trees. Our exposition here roughly corresponds to the ID3 algorithm. Intuitively, this algorithm first answers the question "which attribute should be tested at the root of a decision tree?"

For seeing how to answer this question, let us try the different attributes from the weather data set first. Figure 7.2 shows the four possible choices for the root and the resulting tree stumps for the weather data.

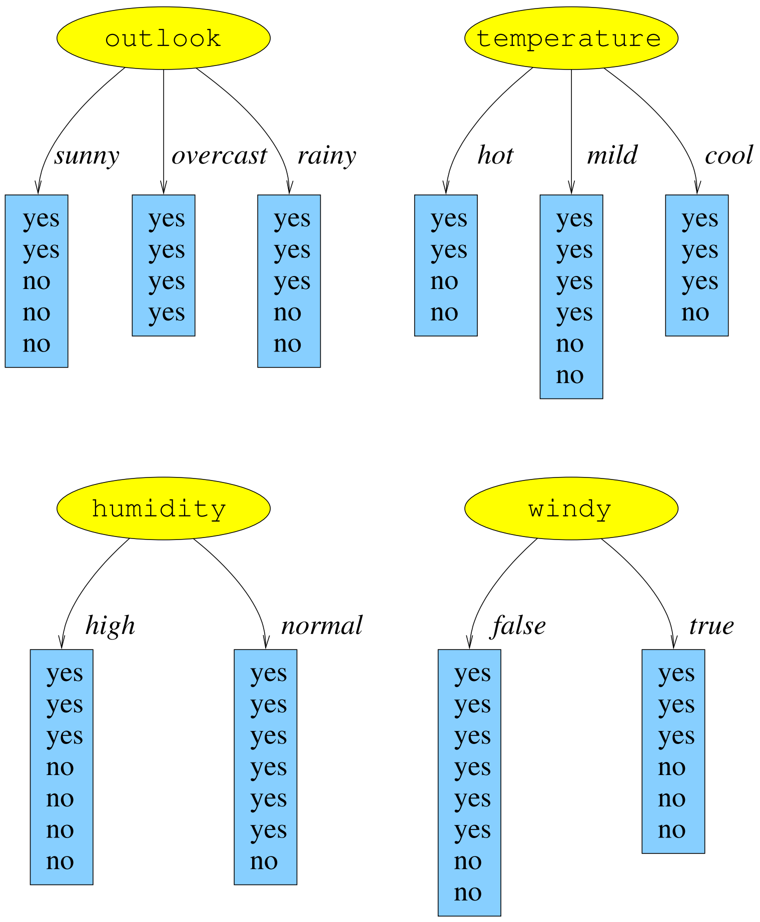

Figure 7.2: Decision tree stumps for the weather data.

So, how did we obtain these decision tree stumps? First, we have selected the attribute to be put into the root, e.g., `outlook`, or `temperature`. There are five entries in the weather data set having the value *sunny* for `outlook`. Two of them classify `yes` and three classify `no`. Moreover, there are four entries having the value *overcast* for `outlook` and all of them classify `yes`. Finally, there are five entries having the value *rainy* for `outlook` and three of them classify `yes` while two classify `no`. Thus, we get the tree stump as drawn in Figure 7.2 when taking `outlook` as root. The remaining three stumps are obtained analogously.

Our goal is to arrive at small decision trees. Any leaf with only one class – `yes` or `no` – will not have to be split further. This should happen as early as possible. But clearly, if a leaf has more than one class, we have to repeat the process recursively. Of course, we can thus generate all possible decision trees and then choose the smallest one from the set of all decision trees obtained. But intuitively it is clear, that this approach is computationally infeasible, since the number of all trees to be generated will be exponential in the number of attributes. So, how can we recover? The idea is to decide which of the decision tree stumps is the best for further processing. Once we have made a choice, all other decision tree stumps are abandoned and we recursively repeat the process only for the stump chosen.

Suppose we have a measure of purity for each node. Then we could choose the attribute that produces the purest daughter nodes. The measure of purity we are going to use is called *information* and measured in bits. But unlike usual bits, our bits here can also be less than one. We shall look at details of how to compute these bits later. These bits are then used to compute another measure that will be actually used to decide which of the stumps is best. This measure Quinlan (1986) has used and it is called *information gain*. The information gain is a statistical property measuring how well a given attribute separates the training examples according to their target classification. Thus, ID3 computes the information gain for each of the tree stumps obtained and chooses one having the highest information gain.

So, let us assume that we can compute the bits mentioned above. Bits are computed for the number of `yes` and `no` classes at the leaf nodes. For the decision tree stump having `outlook` in its root, there are three leaf nodes, i.e., $[2,3]$, $[4,0]$, and $[3,2]$. The information values of these nodes are

$$\begin{aligned}\text{info}([2,3]) &= 0.971\text{bits}\\ \text{info}([4,0]) &= 0.0\text{bits}\\ \text{info}([3,2]) &= 0.971\text{bits}\ .\end{aligned}$$

Next, the average information value of these informations is calculated. Here we take the number of instances into account that go down each branch, i.e., five down the first, four down the second and five down the third one. This gives a total of 14 instances, and thus we obtain

$$\begin{aligned}\text{info}([2,3],[4,0],[3,2]) &= \frac{5}{14}\cdot 0.971 + \frac{4}{14}\cdot 0.0 + \frac{5}{14}\cdot 0.971\\ &= 0.693\text{bits}\ .\end{aligned}$$

This calculation is also applied to the training examples corresponding just to the root. There are 14 examples, nine of which are classified `yes` and five of which are classified `no`. Using the still unknown formula we get

$$\text{info}([9,5]) = 0.940\text{bits}\ .$$

The *information gain* is then just the difference, i.e.,

$$\text{gain}(\texttt{outlook}) = \text{info}([9,5]) - \text{info}([2,3],[4,0],[3,2]) = 0.247\text{bits} .$$

Analogously, we can compute

$$\begin{aligned}\text{gain}(\texttt{temperature}) &= 0.029\text{bits}\\ \text{gain}(\texttt{humidity}) &= 0.152\text{bits}\\ \text{gain}(\texttt{windy}) &= 0.048\text{bits} .\end{aligned}$$

So, `outlook` has the highest information gain and is thus chosen.

Next, we continue recursively. Clearly, a further split on `outlook` will produce nothing new, thus only the remaining three attributes are considered. Looking again at the weather data set, we see that there are the following five examples having the value *sunny*.

`outlook`	`temperature`	`humidity`	`windy`	play
sunny	hot	high	false	no
sunny	hot	high	true	no
sunny	mild	high	false	no
sunny	cool	normal	false	yes
sunny	mild	normal	true	yes

So, we look at the possible roots `temperature`, `humidity` and `windy` for the subtree starting at the *sunny* branch (cf. Figure 7.3).

Then one gets the following values for the information gain

$$\begin{aligned}\text{gain}(\texttt{temperature}) &= 0.571\text{bits}\\ \text{gain}(\texttt{humidity}) &= 0.971\text{bits}\\ \text{gain}(\texttt{windy}) &= 0.020\text{bits} .\end{aligned}$$

Thus, at this point `humidity` is selected as splitting attribute. Then, we see that all remaining classes contain either `yes` or `no`. Thus, there is no need to split these nodes further and the process terminates for this branch. Continued application of this idea finally leads to the decision tree shown in Figure 7.1.

Ideally, the process finishes when all leaf nodes contain only one classification. But it might be impossible to reach this situation if the training data contain two sets of attributes with identical values but different classes. Thus, one stops when no further split is possible.

This explains the basic idea of the ID3 algorithm. We still have to provide the remaining details. So, we continue by revealing the secret of how to compute the information gain.

For defining information gain precisely, we start from recalling the definition of a measure commonly used in *information theory* to measure in a precise way the amount of information in a source. This measure is called ***entropy***. Since this measure is very important

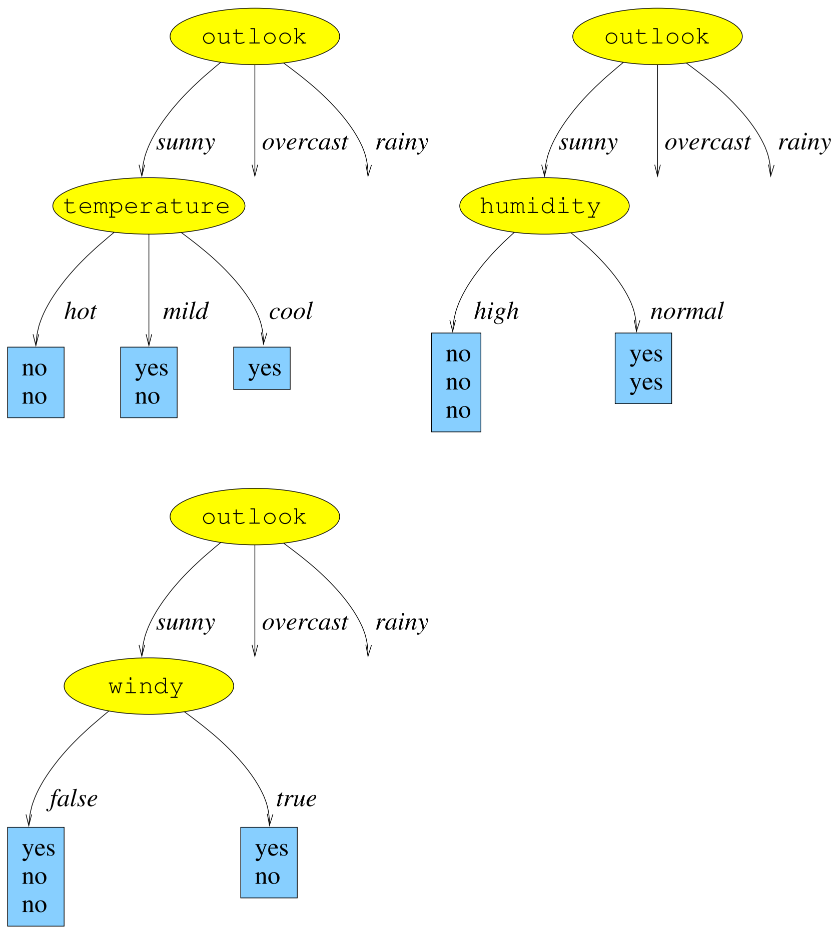

Figure 7.3: Expanded decision tree stumps for the weather data.

in our context of decision tree learning and in other parts of data mining we shall describe it here in some more detail.

For the sake of motivation, let us first assume that a contest is taking place. We have a black box that emits source symbols from a source $\mathcal{S}$ with a source alphabet $S = \{s_1, s_2\}$. The probability to see s_1 is $p_1 = \frac{99}{100}$ and the probability to see s_2 is $p_2 = \frac{1}{100}$. The winner of the contest is the first one who can name both symbols in S. Thus, the winner is the first who has full information about the set S. Of course, we assume that neither contestant has seen the source symbols beforehand. Now, suppose that in the first round the first contestant gets source symbol s_1 while the second one gets s_2. So, at this point, who is more likely to win the contest?

Since the first contestant still has to receive s_2 whose probability of occurrence is $\frac{1}{100}$ while the second one has to wait seeing s_1 whose probability of occurrence is $\frac{99}{100}$, it is intuitively clear that the second contestant is more likely to win than the first.

Thus, in some sense, the second contestant has received more *information* about $\mathcal{S}$ from the source symbol s_2 than did the first contestant. Therefore, it should be clear that, independently of how we define precisely the information obtained from a source symbol, *the* *less likely a source symbol is to occur, the more information we obtain from an occurrence of that symbol, and conversely*. Consequently, the information obtained from a source symbol cannot be a function of the symbol itself, but rather of the symbol's probability p of occurrence. Thus, we shall use $I(p)$ to denote the information obtained from a source symbol with probability p of occurrence. We shall make the following reasonable assumptions about the function $I(p)$, defined for all $0 < p \le 1$.

***Assumption* 0**. $I(p)$ is not the constant 0 function.

***Assumption* 1**. $I(p) \ge 0$ for all p.

***Assumption* 2**. $I(p)$ is continuous.

Moreover, we usually assume that the events of seeing s_i and s_j on different transmissions are independent. Thus, the information obtained from knowing that both s_i and s_j have occurred should be the sum $I(p_i) + I(p_j)$. Since the probability of both events occurring is p_ip_j we shall make

***Assumption* 3**. $I(p_ip_j) = I(p_i) + I(p_j)$.

Now, the remarkable fact about these four assumptions is that essentially one function satisfies them. This is expressed by the following theorem.

Theorem 7.1. *A function $I(p)$ is satisfying Assumptions 0 through 3 made above if and only if it has the form*

$$I(p) = c \log \frac{1}{p} \,,$$

where $c > 0$ is a constant and $\log$ is the logarithm to the base 2.

Proof. That the $\log$ function is satisfying Assumptions 1 through 3 is a well-known fact from calculus. For the converse direction, we start from Assumption 3, and obtain

$$I(p^2) = I(pp) = I(p) + I(p) = 2I(p) \,.$$

Now, it is easy to show by induction that

$$I(p^n) = nI(p) \text{ for all } n \in \mathbb{N}^+ \,. \tag{7.1}$$

Taking into account that (7.1) holds for all $p \in (0, 1]$ directly yields

$$I(p) = I(p^{\frac{1}{n}\cdot n}) = I((p^{\frac{1}{n}})^n) = nI(p^{\frac{1}{n}})$$

Consequently, we can conclude

$$I(p^{\frac{1}{n}}) = \frac{1}{n} I(p) \,. \tag{7.2}$$

Since (7.1) and (7.2) are valid for all $n \in \mathbb{N}^+$ we additionally have

$$I(p^{\frac{n}{m}}) = \frac{n}{m} I(p) \text{ for all } n,\, m \in \mathbb{N}^+ \,.$$

But this means that

$$I(p^r) = rI(p) \text{ for all positive rational numbers } q \,. \tag{7.3}$$

Now, recalling that for every positive real number r there is a sequence $(r_n)_{n\in\mathbb{N}}$ of positive rational numbers r_n such that $\lim_{n\to\infty} r_n = r$ we get by the continuity of the power function $\lim_{n\to\infty} p^{r_n} = p^r$. Hence, the continuity of function I (cf. Assumption 2) and (7.3) imply that

$$I(p^r) = I(\lim_{n\to\infty} p^{r_n}) = \lim_{n\to\infty} I(p^{r_n}) = I(p) \lim_{n\to\infty} r_n = rI(p)$$

for all positive real numbers r.

Now, let us fix any q with $0 < q \leq 1$. Since any p satisfying $0 < p \leq 1$ can be written as $p = q^{\log_q p}$ and since $I(p) \geq 0$ (cf. Assumption 1) as well as $I(p)$ not constant 0 (cf. Assumption 0), we have

$$I(p) = I(q^{\log_q p}) = I(q) \log_q p = c \log \frac{1}{p}$$

for some constant $c > 0$. ∎

As the proof of Theorem 7.1 shows, we additionally have $I(1) = 0$ nicely matching our intuition. Moreover, the arbitrary multiplicative constant c can be absorbed in the units of measurement of information.

7.3. Information, Entropy and Information Gain

First, we define information. Using the facts presented above, we naturally arrive at the following definition.

Definition 16. *The* ***information*** $I(p)$ *obtained from a source symbol* s *with probability* $p > 0$ *of occurrence is given by*

$$I(p) = \log \frac{1}{p} .$$

The unit of measurement of information is the bit, as already said above. The connection to the binary unit (also called bit) comes from the following observation. If the source is $\mathcal{S}$ and has alphabet $S = \{0, 1\}$ and both symbols are equally likely, i.e., $p_0 = p_1 = 1/2$, then the information given by either source symbol is $I(1/2) = \log 2 = 1$. Thus, if the source randomly emits one binary digit (bit), then the information obtained by a single emission is one binary unit (bit).

Example. A PC monitor is capable of displaying pictures made up of pixels at a resolution of 1024 columns by 768 rows (or higher). Hence, if each pixel can be in any one of 256 colors, there are a total of $2^{8\cdot 1024\cdot 768}$ different pictures. If each of these pictures is considered to be equally likely, the probability of a given picture is $2^{-6291456}$. Thus, the information obtained from a single picture is

$$I = \log 2^{6291456} = 6291456 \text{ bits} .$$

Next, consider a random speech of 1000 words from a 10 000 word vocabulary (what would be amazing for a politician). Then the probability of speaking any sequence of 1000 words from such a vocabulary is 10000^{-1000}. Consequently, the amount of information obtained from such a speech is

$$I = \log 10000^{1000} = 1000 \log 10000 < 14000 \text{ bits} .$$

This clearly shows that a picture is worth more than a thousand words.

Now, we are ready for defining the concept of entropy.

Definition 17. *Let $\mathcal{S} = (S, P)$ be a source with source alphabet $S = \{s_1, \dots, s_q\}$ and probability distribution $P = \{p_1, \dots, p_q\}$. The average information obtained from a single sample from $\mathcal{S}$ is*

$$H(\mathcal{S}) = \sum_{i=1}^{q} p_i I(p_i) = \sum_{i=1}^{q} p_i \log \frac{1}{p_i} = -\sum_{i=1}^{q} p_i \log p_i \ .$$

The quantity $H(\mathcal{S})$ is called ***entropy*** of the source. For the sake of convenience, we define in all calculations involving entropy $0 \log 0$ to be 0.

Now, given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this boolean classification is

$$\text{Entropy}(S) = -p_{\oplus} \log p_{\oplus} - p_{\ominus} \log p_{\ominus} \ , \tag{7.4}$$

where $p_{\oplus}$ is the proportion of positive examples in S and $p_{\ominus}$ is the proportion of negative examples in S.

Now, let S be a sample, let A be any attribute and let Values(A) be the set of all possible values for attribute A. Furthermore, let $S_v = \{s \mid \ s \in S \text{ and } A(s) = v\}$. Then we formally define the information gain as follows.

Definition 18.

$$\text{gain}(S, A) = \text{Entropy}(S) \ - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \cdot \text{Entropy}(S_v) \ .$$

Here Entropy(S) is the original entropy of the sample and the second term is the *expected* entropy after S has been partitioned using attribute A.

We continue by a having a closer look at Formula (7.4). First, let us assume that there are no positive examples, i.e., $p_{\oplus} = 0$. Consequently, all examples are negative and hence $p_{\ominus} = 1$. Then, recalling that we have defined $0 \log 0 = 0$, we get

$$\begin{aligned} \text{Entropy}(S) &= -p_{\oplus} \log p_{\oplus} - p_{\ominus} \log p_{\ominus} \\ &= 0 \log 0 - 1 \log 1 = 0 \ . \end{aligned}$$

Analogously, if all examples are positive, i.e., $p_{\oplus} = 1$ and $p_{\ominus} = 0$, we also obtain

$$\begin{aligned} \text{Entropy}(S) &= -p_{\oplus} \log p_{\oplus} - p_{\ominus} \log p_{\ominus} \\ &= -1 \log 1 - 0 \log 0 = 0 \ . \end{aligned}$$

Next, suppose that the sample contains an equal number of positive and negative examples, i.e., $p_\oplus = p_\ominus = 1/2$. Then,

$$\begin{aligned}\text{Entropy}(S) &= -p_\oplus \log p_\oplus - p_\ominus \log p_\ominus \\ &= -\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2} \\ &= \frac{1}{2} + \frac{1}{2} = 1\,.\end{aligned}$$

Note that we always have $p_\oplus + p_\ominus = 1$. Thus, we have to study the function

$$f(x) = -x\log x - (1-x)\log(1-x)$$

for $x \in (0,1)$. Recalling a bit calculus, it is easy to see that

$$f'(x) = -\log x + \log(1-x)\ ,$$

and that $f'(x) = 0$ if and only if $x = 1/2$. Since $f'(x) > 0$ for all $x < 1/2$, we see that $f(x)$ is monotonically increasing for all $x \in (0, 1/2)$. Furthermore, $f'(x) < 0$ for all $x > 1/2$, and thus $f(x)$ is monotonically decreasing for all $x \in (1/2, 1)$. Finally,

$$f''(x) = -\frac{1}{\ln 2}\left(\frac{1}{x(1-x)}\right) < 0$$

for all $x \in (0,1)$, and therefore $f(x)$ is concave. Putting it all together, we get the graph of the entropy function displayed in Figure 7.4.

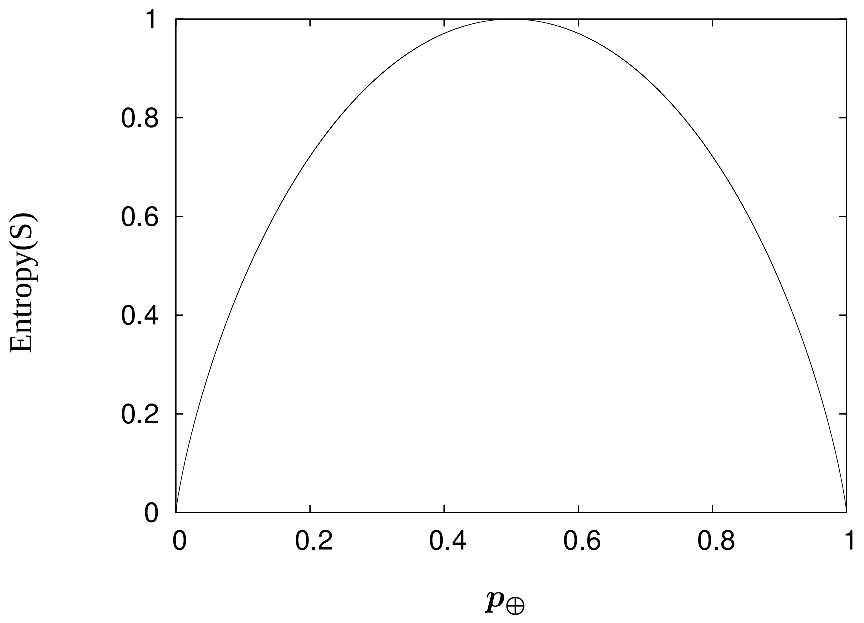

Figure 7.4: Entropy function in dependence of $p_\oplus$.

Next, we illustrate the use of the entropy function and information gain as given in Definition 18 by looking again at our weather data set S. We have a total of 14 examples and 9 of them are positive and 5 of them are negative. Consequently, $p_\oplus = 9/14$ and $p_\ominus = 5/14$. So, we can compute Entropy(S) and obtain

$$\begin{aligned}\text{Entropy}(S) &= -\frac{9}{14}\cdot\log\frac{9}{14} - \frac{5}{14}\cdot\log\frac{5}{14}\\ &= \frac{1}{14}\left[14\log 14 - 9\log 9 - 5\log 5\right] = 0.940\text{bits}\ ,\end{aligned}$$

i.e., the value we have already provided at Page 77. Note that is computationally advantageous to perform the calculation as described above, i.e., without working out the fractions and taking the logarithm of them.

Next, we compute the information gain for the attributes `outlook`, `temperature`, `humidity` and `windy` by using the formula provided in Definition 18. The attribute `outlook` can take the three values *sunny*, *overcast* and *rainy*. Looking at the weather data set, we directly get $|S_{sunny}| = 5$, $|S_{overcast}| = 4$ and $|S_{rainy}| = 5$. Furthermore,

$$\begin{aligned}\text{Entropy}(S_{sunny}) &= -\frac{2}{5}\log\frac{2}{5} - \frac{3}{5}\log\frac{3}{5} = 0.971\text{bits}\\ \text{Entropy}(S_{overcast}) &= 0\text{bits}\\ \text{Entropy}(S_{rainy}) &= -\frac{3}{5}\log\frac{3}{5} - \frac{2}{5}\log\frac{2}{5} = 0.971\text{bits}\end{aligned}$$

Putting it all together, we thus obtain

$$\text{gain}(S, \texttt{outlook}) = 0.940 - \frac{5}{14}\cdot 0.971 - \frac{4}{14}\cdot 0 - \frac{5}{14}\cdot 0.971 = 0.246\text{bits}$$

and analogously

$$\begin{aligned}\text{gain}(S, \texttt{temperature}) &= 0.029\text{bits}\\ \text{gain}(S, \texttt{humidity}) &= 0.152\text{bits}\\ \text{gain}(S, \texttt{windy}) &= 0.048\text{bits}\ ,\end{aligned}$$

i.e., again the values we already have provided at Page 77.

So far we have discussed entropy of a sample for the special case where the target classification is boolean. More generally, if the target attribute can take on c different values, then the entropy relative to this c-wise classification is defined as

$$\text{Entropy}(S) = -\sum_{i=1}^{c} p_i \log p_i\ ,$$

where p_i is the proportion of the sample S belonging to class i. Note that logarithm is still to the base 2, because entropy is measured in bits. Also note that maximum is now $\log c$.

This completes the description of the ID3 algorithm. So, what else can be said about the ID3 algorithm? For answering this question, let us look at the hypothesis space used by the ID3 algorithm.

7.4. ID3’s Hypothesis Space

ID3’s hypothesis space is the set of all decision trees. Thus, it is a *complete* space of finite discrete-valued functions, relative to the available attributes. Since every finite discrete-valued function can be represented by some decision tree, ID3 avoids one of the major risks of methods that search incomplete hypotheses spaces: That the hypothesis space might not contain a correct description of the target function.

Conceptually, we may think of ID3 as a learning algorithm that searches the hypothesis space of all decision trees for one that correctly describes the training data. The search is guided by using the information gain measure.

ID3 maintains only a single current hypothesis as it searches through the space of decision trees. This contrast, for example, our pattern language learner presented in Lecture 6 which conceptually maintains the set of all patterns that are consistent with the text seen so far in every learning step. From this set, it then chooses a descriptive pattern and outputs it. Thus, ID3 does not have the ability to determine which alternative consistent hypotheses are still available.

Moreover, ID3 in the form presented does not perform any backtracking in its search. Once an attribute has been selected as root in the relevant subtree, this choice is never reconsidered. While this improves the efficiency of the decision tree learner, it does not necessarily guarantee that the learned decision tree is the smallest possible. In other words, ID3 always selects the *locally optimal solution* and may thus converge to a locally optimal solution. However, this locally optimal solution may be less desirable than the *globally optimal solution* that would have been encountered along a different branch of the search.

ID3 uses all training examples at each step in the search to make a statistically based decision (i.e., by using the information gain) regarding how to refine its current hypothesis. The advantage of this approach is that the resulting search is much less sensitive to errors in individual training data than other methods using individual training data for making decisions.

Next, we turn our attention to another very important problem, i.e., the *inductive bias* used by the ID3 algorithm.

7.5. Inductive Bias in Decision Tree Learning

Roughly speaking the ***inductive bias*** is the set of assumptions that, together with the training data, deductively justify the classification assigned by the learner to future examples. For example, the weather data set contains 14 examples. However, there are 36 possible examples, and of course, besides explaining the training data, one is also interested in decision trees that classify the still unseen data correctly. Since there may be many possible decision trees that correctly explain the training data, every learner has to make a choice. The explicitely or implicitly made assumptions determining this choice are the inductive bias of the learner.

The ID3 algorithm chooses the first acceptable tree it encounters in its search through the hypothesis space. Roughly speaking, the ID3 algorithm selects in favor of shorter trees over longer ones, and selects the tree that places the attributes with highest information gain closest to the root. So, there is a subtle interplay between the attribute selection heuristics used by ID3 and the particular training examples it encounters .hus, it is difficult to characterize precisely the inductive bias used by ID3. Nevertheless, as a first approximation we may say:

> *Shorter trees are preferred over larger trees. Trees that place high information gain close to the root are preferred over those that do not.*

Of course, one has also to answer the question whether or not this is a good or not so good inductive bias. But it is beyond the scope of the introductory nature of our course to provide a thorough study of this problem.

References

[1] J.R. QUINLAN (1986), Induction of decision trees, *Machine Learning* **1**, No.1, 81 – 106.

[2] J.R. QUINLAN (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA.