14

K. P. Jantke   S. Kobayashi
E. Tomita   T. Yokomori (Eds.)

Algorithmic Learning Theory

4th International Workshop, ALT '93
Tokyo, Japan, November 8-10, 1993
Proceedings

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editor

Jörg Siekmann
University of Saarland
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
D-66123 Saarbrücken, Germany

Volume Editors

Klaus P. Jantke
Fachbereich Informatik, Mathematik und Naturwissenschaften
Hochschule für Technik, Wirtschaft und Kultur Leipzig
Postfach 66, D-04251 Leipzig, Germany

Shigenobu Kobayashi
Interdisciplinary Graduate School of Science and Engineering
Tokyo Institute of Technology
4259 Nagatsuta, Midori-ku, Yokohama, 227 Japan

Etsuji Tomita
Department of Communications and System Engineering
University of Electro-Communications
1-5-1 Chofugaoka, Chofu, Tokyo, 182 Japan

Takashi Yokomori
Department of Computer Science and Inform. Math.
University of Electro-Communications
1-5-1 Chofugaoka, Chofu, Tokyo, 182 Japan

63502

CR Subject Classification (1991): I.2.6, I.2.3, F.1.1

ISBN 3-540-57370-4 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-57370-4 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera ready by author
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

PREFACE

This volume contains all the papers that were presented at the Fourth Workshop on Algorithmic Learning Theory (ALT '93), which was held at University of Electro-Communications in Tokyo from November 8th to 10th, 1993. In addition to 3 invited papers, 29 papers were selected from among 47 submitted extended abstracts, which represent the highest number of papers submitted to an ALT workshop, exceeding 46 which were recorded in 1990.

This workshop was the fourth in a series of ALT workshops, whose focus is on theories of machine learning and the application of such theories to real world learning problems. The ALT workshops have been held annually since 1990, sponsored by the Japanese Society for Artificial Intelligence. In the past, ALT alternated between an English-language international conference and a Japanese-language domestic workshop. (ALT'90 was international whereas ALT'91 and ALT'92 were more or less domestic.) Starting with ALT'93, all the future ALT workshops will be English-language international conferences, and researchers from throughout the world will be invited to present papers and to attend the conference.

This year, we are fortunate to include three invited papers by distinguished researchers: "Identifying and Using Patterns in Sequential Data" by Dr. P. Laird, NASA Ames Research Center, "Learning Theory Toward Genome Informatics" by Prof. S. Miyano, Kyushu University, and "Optimal Layered Learning : A PAC Approach to Incremental Sampling" by Prof. S. Muggleton, Oxford University. It goes without saying that the three researchers are distinguished theoreticians, but at the same time the issues they addressed were highly practically relevant. It is our hope that the future ALT workshops will continue to bring together researchers from both theoretical and practical sides of machine learning to provide a forum for truly worthwhile research interactions.

We would like to extend our sincere gratitude to the many individuals who made this workshop possible. These include the invited speakers, all the presenters and participants at the workshop, the members of the steering committee, the members of the program committee, many referees who helped ensure the quality of the accepted papers, and many others.

Last but not least, we gratefully acknowledge the support of the Telecommunications Advancement Foundation.

Tokyo, November 1993

K.P. Jantke
S. Kobayashi
E. Tomita
T. Yokomori

Sponsored by Japanese Society for Artificial Intelligence (JSAI)

In Cooperation with

Information Processing Society of Japan (IPSJ)
Institute of Electronics, Information and Communication
Engineers of Japan (IEICE)
Japanese Cognitive Science Society (JCSS)
Japan Society for Software Science and Technology(JSSST)
Japan Neural Network Society (JSNN)
Society of Instrument and Control Engineers of Japan (SICE)

List of Referees

N. Abe
S. Akaho
K. Akama
T. Akutsu
D. Angluin
S. Arikawa
J. Arima
H. Arimura
H. Asoh
A. Biermann
J. Case
R. Daley
R. Freivalds
K. Furukawa
M. Hagiya
M. Haraguchi
M. Harao
H. Iba
Hideki Imai
Hiroshi Imai
Y. Inagaki
B. Indurkhya
M. Ishikawa
H. Ishizaka
K. Jantke
Y. Katayama
E. Kinber
J. Kivinen
Satoshi Kobayashi
Shigenobu Kobayashi
Y. Kodratoff
A. Konagaya
T. Koshiba
S. Kuhara
S. Kunifuji
T. Kurita
P. Laird
S. Lange
M. Li
W. Maass
A. Maruoka
O. Maruyama
M. Matsuoka
D. Michie
T. Miyahara
S. Miyano
K. Morik
H. Motoda
T. Motoki
S. Muggleton
Y. Mukouchi
A. Nakamura
T. Nishida
T. Nishino
S. Nishio
T. Nishizawa
M. Numao
N. Ohtsu
H. Onda
H. Ono
L. Pitt
Masahiko Sato
Masako Sato
T. Sato
A. Sakurai
A. Sharma
S. Shimozono
A. Shinohara
T. Shinohara
Y. Shirai
H. Simon
C. Smith
M. Suraj
E. Suzuki
Y. Takada
H. Takahashi
J. Takeuchi
K. Tanatsugu
S. Tangkitvanich
N. Tanida
M. Tatsuta
A. Togashi
H. Tsukimoto
E. Ukkonen
O. Watanabe
S. Weinstein
R. Wiehagen
S. Yamada
A. Yamamoto
T. Yokomori

TABLE OF CONTENTS

Identifying and Using Patterns in Sequential Data

Philip Laird

NASA Ames Research Center, Moffett Field, CA 94035-1000, U.S.A.

Abstract. Whereas basic machine learning research has mostly viewed input data as an unordered random sample from a population, researchers have also studied learning from data whose input sequence follows a regular sequence. To do so requires that we regard the input data as a stream and identify regularities in the data values as they occur. In this brief survey I review three sequential-learning problems, examine some new, and not-so-new, algorithms for learning from sequences, and give applications for these methods. The three generic problems I discuss are:

- Predicting sequences of discrete symbols generated by stochastic processes.
- Learning streams by extrapolation from a general rule.
- Learning to predict time series.

1 Introduction

Algorithmic Learning Theory treats both the theory of learning and the design of practical algorithms. Over the years many useful and interesting algorithms have derived from the assumption of *data independence*, that is, that the observations be considered an unordered set of examples. Indeed, we consider an algorithm that learns successfully regardless of the presentation order of the examples to be more robust than one that depends on assumptions about the sequence of examples. For example, given an algorithm that infers the grammatical structure of a language from examples of sentences and non-sentences, we would prefer that the examples not have to be given in order of size or in some order depending on the structure of the grammar.

In practice, however, the requirement of order independence is a strong one: much information can be conveyed by carefully choosing the sequence of examples–*too much* information for some theoretical models, where the learning problem becomes trivial if the presenter is free to choose the presentation order. More significantly, many algorithms *depend* on the assumption that the examples come from random sampling of a fixed population; such algorithms usually turn in poor results when this assumption fails to hold.

In some applications the real problem is to learn the pattern responsible for generating the sequence, rather than to learn to attach a label to the individual examples. And in situations where examples arrive as a continual stream of symbols without any natural divisions or termination, we have little choice, at least initially, but to treat the incoming values as a sequence until enough is known to segment it into its parts. In the next section I relate some situations I have encountered where sequential effects cannot be overlooked in the learning problem.

This paper briefly surveys learning problems and algorithms for sequential data. I distinguish among three kinds of learning problems: stochastic sequence prediction, sequence extrapolation, and modeling time series. For each I review the characteristics of the problem, discuss some learning algorithms, and note several applications. In the conclusion I summarize some common features of these problems and the challenges for continuing research.

2 Some Sequential-Learning Applications

The descriptions below abstract some learning applications I have encountered where the sequential nature of the data is important to the problem.

Events in Telemetry Streams

Let $\{f(t), t \geq 1\}$ be a stream of numbers obtained by sampling a physical process at regular time intervals. This stream is our only way of observing the process since it is at a remote site. Most of the time this telemetry data conveys nothing of interest, but now and then an important event occurs. We recognize this fact by the pattern (or *signature* of successive values of $f(t)$ over some fixed time interval.

Humans learn quickly to recognize and identify these significant events visually by looking at a plot of $f(t)$. Suppose, however, we want to automate the process of signaling these interesting events. We obtain samples of $f(t)$ with labels by an expert of the interesting events. Thereupon the process of training a program to recognize and label interesting events is apparently straightforward: any number of static classification algorithms can be used—Bayesian discriminants, neural networks, decision trees, etc. After training, the system is turned on, and as expected, it performs quite well identifying these events.

As time continues, however, we discover that it reports successively fewer events of interest. To find out why, we examine the telemetry stream and discover a small but significant trend in the signatures, due perhaps to physical wear or some hysteresis effect. Unless this trend can be identified and corrected, no amount of random sampling will make our classifier work for more than a short time.

Engineering Monitoring and Maintenance

Replacing parts of a complex piece of machinery can be done by schedule or by need. Scheduled maintenance means that the part is replaced after a fixed time period, regardless of its actual condition. Other parts, especially expensive ones, are removed and examined frequently and replaced as soon as they show signs of deterioration. Sometimes for very critical parts, or for parts that are expensive to remove, sensors are attached to the part to indicate its condition without the need to remove it.

In practice, scheduled maintenance is expensive, and sensors are failure-prone. As a result the true condition of a critical part must often be inferred indirectly from a combination of sensors and observations. To automate the process of monitoring critical components, we need to be able to learn from sequences of multiple noisy observations $\{\mathbf{X}_t, t \geq 1\}$ when to replace the part.

Clustering

Clustering programs examine datasets and try to group the data into "meaningful" classes from which useful conclusions can be drawn. For example, in an attempt to discover some of the structure hidden in a huge database of spectral measurements from stars and galaxies, a clustering program groups the many thousands of observations into about fifty distinct classes. Astronomers then study these classes, looking for common physical properties of the objects in the classes.

The clustering program treats the observations as a set, and the time of the observation is not normally included as an attribute. But in actuality the observations occurred in time, and unexpected correlations between the measurement times of the objects in a class need to be explained. In the case of spectral imaging, temporal correlations could result from the fact that consecutive observations tend to be directed at objects physically proximate to each other and hence closely related. The correlations could, however, be a consequence of temporal effects in the use or adjustment of the apparatus rather than of any astrophysical phenomena.

Operator Training

Drivers, pilots, and other operators of machinery are often trained on simulators. By presenting a realistic set of scenarios on a model of the machine, the trainee can develops the motor and judgment skills to operate safely without any risk that a serious accident will result from his errors.

Often, however, the trainee also learns to predict the simulator. In one case a pilot learned that he could always avoid a collision in a simulated near-miss by reducing his altitude and banking to the right. This strategy might not be effective in a real encounter, so the designers of the simulator were worried that the pilots were learning the simulator rather than good piloting skills. They warned the trainee of the risk of relying on apparently consistent patterns in the simulator, and then reinforced this warning by "training" the pilot to expect a certain predictable response from the simulator— only to change this response suddenly after the pilot's expectations (and responses) had been conditioned. This tactic, however, does little to mitigate the human ability to learn sequential patterns, and this ability seriously limits the usefulness of simulators in training.

3 Models of Sequential Effects

The three models given below, while related, differ in the nature of the models and the learning objectives and hence call for different kinds of algorithms.

Stochastic Sequence Prediction

In the most basic form, the input sequence is an infinite stream of individual symbols with no meaning or internal structure. By assumption the symbols are produced by some unknown stochastic process that may or may not produce each symbol independently of its predecessors. The learning task is to construct a model of the process and to predict as accurately as possible each symbol in the stream.

For example, consider the stream fragment,

```
..., T + + + $ % T + $ % T + + + + $ % ...
```

After reading this sequence from left to right, most people would agree that the most likely symbol to follow is `T`. The basis for this opinion is the heuristic that the future tends to follow the past, and in both previous occurrences the character `%` has been succeeded by `T`. The confidence in this prediction may be low since the evidence is meager, but in the absence of any knowledge about the process generating this stream, humans seem to default to this kind of statistical prediction.

Trying to predict beyond the next two or three symbols degenerates quickly into a simple frequency analysis—what one would predict with the assumption that all symbols were selected independently at random. Evidently the sequential information decays quite rapidly with time, something characteristic of state-based stochastic models. In general we can model the observed sequence as a deterministic or probabilistic function of some unobservable random state or "context". If the state were known, the prediction task would reduce to one where the symbols stream is a sequence of independent random variables. The task, then, is to model both the underlying state structure and the random process generating the symbols in each state.

Discrete-state Markov models are a popular way of modeling state dependencies. Hidden Markov models [31] consist of a discrete-state, discrete-time Markov process in which a multinomial process is attached to each state. Fairly efficient offline algorithms, known as *reestimation algorithms*, (e.g, [39]) are known for finding models from observations.

Markov trees are a different representation for Markov chains. Instead of representing the individual states of the process, the tree represents transitions from the steady state. (See Figure 1.) Each level of the tree represents a context: the root node ignores all previous input, nodes at the first level represent changes based on the single previous input symbol, and so forth. At any time a prediction can be based on any of these contexts, and different algorithms choose the appropriate context differently (e.g., [53]). Currently the best known

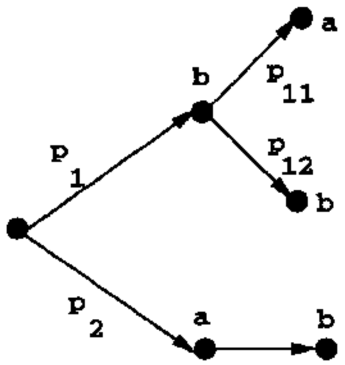

Fig. 1. A two-level Markov tree. The nodes are labeled by the input symbols, the arcs by probabilities. The root represents the steady-state or empty context.

general-purpose text compression algorithms are based on Markov trees ([5] surveys these algorithms), but exactly why they consistently outperform Markov models and other techniques is not clear.

Finding an optimal Markov process is in general an intractable problem, but when the underlying dynamics of the state-transition process are known, Kalman filters are an optimal procedure for estimating the state and making predictions ([9]). When the input observations are numeric, one can estimate the parameters of and quantify the effects of noise quite precisely.

We can point to a few of the many applications of stochastic sequence prediction. Some are based on the dual relationship between learning and compression. Learning algorithms become compression algorithms by forming a model of the sequence, predicting the next input, and encoding the difference between what is predicted and what actually occurs. Only these differences need be retained: there is no need to encode the model if the uncompression process is deterministic and undergoes exactly the same learning, forming the same models at the same point in the stream.

Conversely, a text-compression utility can be "taken from the box" and used as a learning algorithm Vitter and Krishnan [47, 48] used the Lempel-Ziv algorithm for data compression as the learning element in a database module that predicts record requests during queries. If they predict a request for a record that is not online, a prefetch is issued in anticipation thereof.

My colleague Ronald Saul and I [24] analyzed and generalized the Markov tree PPM algorithm for text compression [5]. Our variant, known as *TDAG*, is well suited to a variety of tasks, such as managing the cache memory of a mass storage system and dynamically optimizing Prolog programs [23]). Others have since applied TDAG to trend analysis and protein sequence prediction.

Hidden Markov models are widely used for speech recognition, and the carefully engineered system due to Kai-fu Lee and his colleagues [30] is one of the most successful examples. By drastically reducing the number of possibilities for the next input item, the learning element enabled them to advance substantially the state of the art in real-time speech recognition.

When the input stream is naturally segmented into "sentences", formal-language methods can also be used to predict shorter segments of the input stream. For example, Hermens and Schlimmer [19] used zero-reversible finite automata and a learning algorithm due to Angluin [2] to help predict the user's keyboard input in an electronic forms assistant. Stochastic context-free grammars are a more powerful model, and with new algorithms for inferring such grammars from the sequences they generate, they are finding applications in predicting the secondary structure of proteins [12].

Let us note some of the distinctive features of stochastic sequence prediction algorithms. First, we can usually characterize the number of degrees of freedom in the models by two numbers: the number of distinct symbols and the number of "states" (or the *order*) of the model. The models are usually not aware of "noise" (random disturbances of the input symbols) because the noise is incorporated into the model as modifications of the probabilities. On the other hand, the lack of a distinct noise model makes it hard to separate the signal from the noise, as sometimes can be done with other learning methods, e.g., the Kalman filter, where Gaussian white noise is assumed to affect both the dynamics and the observations. The ability of a model to compress the input stream serves as a useful comparative measure its predictive success. Finally, as for most forms of learning from examples, there is usually a tradeoff between the expressiveness of the model and the number of observations required to converge to a model.

Structured Sequence Extrapolation

When the symbols in an incoming stream are not atomic but instead have known relationships among them, we can make stronger models than statistical ones. For example, consider the following sequence of strings: $S =$ *, !**?, !!*!**??, !!!*!!*!**???, ... Although we can ignore the string boundaries and predict symbol by symbol, we can also use what we know about the algebra of strings to look for regular substructure in the sequence. Observe, for example, that the simple sequence $R = *?^0, *?^1, *?^2, *?^3, \ldots$ forms a suffix of S. Hence we can write $S = L \cdot R$, and then focus on the subsequence L. After a few moments, most people find a reasonably simple rule for L and then are able to make a reasonable prediction for the next string in this sequence: !!!!*!!!*!!*!**????.

The knowledge that the elements of this sequence are strings, and that strings can be decomposed into substrings by the non-commutative concatenation operation, is the key to this approach. Other data types besides strings have similar properties: the sequence of integers 8, 9, 11, 15, 23, ... can be written as the sum of two simpler sequences 7, 7, ... and 1, 2, 4, 8, ...; and the sequence of binary lists

$$S = \langle a, [a, a], [[a, a], a], [[[a, a], a], a] \ldots \rangle,$$

(where [,] is the ordered-pair operation) can be represented by the recurrence expression $S_{n+1} = [S_n, a]$. The idea, then, is to extrapolate a sequence using prior knowledge about the data type of the elements in the sequence.

Humans exhibit remarkable skill at this task. "Tests of intelligence" often include integer extrapolation problems. Curiously, while there is usually no unique way to extrapolate a sequence (e.g., 1, 2, 3, 4, ... can continue with 5 or 97 or any other value depending on one's assumptions), there is typically a simplest or "most elegant" rule that most people will agree upon, or that, when shown to someone unable to find it, results in expressions of "Oh, of course!" In the case of integer sequences, the underlying assumptions are that each element S_{n+1} of the stream can be obtained from "a few" (one or two) of its predecessors (S_n, S_{n-1}) by means of only a "few" additions or multiplications, involving perhaps some small constants (0, ±1, ±2).

Moreover, when more than one rule of roughly equal complexity can account for the values seen so far, humans often hedge their prediction by offering more than one, along with a rough confidence estimate. As the number of possible explanations diminish, the confidence in the remaining rules increases correspondingly. Given that the input stream is deterministic, not stochastic, one can ask how these confidence estimates arise.

Most of all, that all this seems to be a common skill suggests this learning/generalization ability is somehow fundamental.

The number of principled algorithms for sequence extrapolation is small. Kotovsky and Simon [21] explored a model of human sequence extrapolation in the case of Thurstone letter sequences. Pivar and Finkelstein [37] implemented a Lisp program that extrapolates integer sequences based on the so-called *method of differences*, which reportedly goes back to Gauss. This idea is based on the fact that a polynomial sequence $\{f(n) \mid n = 1, 2, \ldots\}$ reduces to a constant sequence $\{c, c, c, \ldots\}$ by computing the successive k'th differences, defined recursively as

$$\begin{aligned} D_k f(n) &= D_{k-1} f(n+1) - D_{k-1} f(n) \\ D_0 f(n) &= f(n), \end{aligned}$$

where k is the degree of the polynomial f. Then the matrix of first, second, ..., k'th differences can be turned directly into a finite-difference representation or into a closed-form polynomial. Best of all, this requires only $k+1$ consecutive examples (values) of the polynomial.

This simple method applies only for sequences that are polynomials, but some have tried to develop algorithms extending the method to incorporate factors, constants, and other options along with differences. But the need to search through a number of possible ways to decompose the numbers limits the approach. With some clever heuristics, Feenstra (cited in [10]) used an extended difference method to achieve an "IQ" of about 160 on a published intelligence test.

Genetic programming [22] has also been used to search for both polynomial forms and recurrence relations. While reasonably effective, this method has two drawbacks: it requires many more examples than necessary, and it comes up with rules that are more complex than necessary.

For other than numerical domains, few algorithms have been suggested than can be presented formally and analyzed. Many researchers, however, have been aware of the need to study sequence extrapolation in a way that is not restricted to just one data type (see, for example, [36, 18]). Intuition says that essentially the same algorithm should work for strings, integers, lists, queues, etc., with specific differences that depend on the properties of that type. Since general algorithms are more useful than specialized ones, we are encouraged to look for extrapolation algorithms not restricted to a single data type.

Recently my colleague Ronald Saul and I have developed such an algorithm [25]. The main features of this algorithm are the use of abstract data types, a recursive language for representing streams, and the ability to provide a confidence measure with each prediction. A data type is a set of expressions containing a finite set of generators (atoms) and closed under a finite set of operators. Congruences group the expressions into equivalence classes (e.g., $(2 * 3) = (3 + 3)$. The algorithm depends on an efficient algorithm for factoring elements into subelements—e.g., writing the string abc as $\lambda \cdot (abc)$, $(a) \cdot (bc)$, $(ab)(c)$, and $(abc) \cdot \lambda$. There are a few additional restrictions, but the model covers most of the commonly used data types, including multisorted ones like pairs of strings.

The representation language is called *elementary stream descriptions* and does not depend on the type. Streams are recursively defined in one of three forms:

- An *initial-value* form, giving the first element in the stream and an elementary description of its tail;
- A *functional* form, expressing the stream as a functional combination of two or more streams, which are in turn defined by elementary descriptions;
- A *recursive* form, stating that the stream is recursively equal to a stream within whose definition it occurs.

For example, the Fibonacci stream $F = \langle 1, 1, 2, 3, 5, \ldots \rangle$ is defined: $F = \langle 1 \mid F_2 \rangle$, $F_2 = \langle 1 \mid F_3 \rangle$, $F_3 = (F_4 + F_5)$, $F_4 = F$, $F_5 = F_2$.

The algorithm breaks each incoming example into its parts and uses these either to create new descriptions or to test existing ones. Descriptions that are inconsistent with the example are discarded. Hypotheses are stored in such as way that the simplest ones, especially the ones with the fewest initial values, can be found efficiently.

Another feature of the algorithm is that it provides confidence estimates with its prediction of the next value, estimates that agree qualitatively with our own. We make the simplifying assumption, that normally is not true, that any hypothesis H has a fixed probability p_H of incorrectly predicting a symbol in the next input example. p_H depends on H but not on the symbol or where it occurs in the input. The longer the string of correct predictions, the lower our estimate of p_H, which we can compute explicitly with Bayes' rule.

We have done some analysis of the algorithm, including a proof of correctness (identification in the limit). For the special case of freely generated types, the algorithm is efficient, running in time bounded by a polynomial in the sizes of the input values and revising its hypotheses in polynomial time. For more general types the algorithm is not efficient with respect to time or space. We have implemented the algorithm and are now experimenting with it.

Sample size analysis for elementary stream descriptions is difficult. Our only formal result so far [26] is to show that, for the family of streams whose type is dotted pairs (non-associative pair algebra) with at most one initial value, the number of input values required in the worst case is exactly $h + 3$, where h is the height of the first example. Thus if the first example is an atomic value, say a, then 4 input examples are necessary and sufficient to identify the stream uniquely. This tends to confirm our observation that sample sizes required for sequence extrapolation is extremely small compared to statistical models.

Because of the lack of good algorithms, we cannot point to many actual applications where sequence extrapolation has played an important role, but we can suggest applications where such an algorithm could be of value.

In the late 1960's and early 1970's there was a flurry of interest in automatic programming, including programming by examples [17, 42, 43]. Some methods looked for patterns in successive examples that could be turned into recurrence relations and thence into recursive programs. For example, given the following examples of the function f on lists:

$$\begin{aligned} f([a]) &= a \\ f([a, b]) &= b \\ f([a, b, c]) &= c \end{aligned}$$

Summers's program [43] finds two sequences: the sequence of input forms `cons(a, nil)`, `cons(a, cons(b, nil))`, `cons(a, cons(b, cons(c,nil)))`, and the sequence of value forms expressed in terms of the input x: `car(x)`, `car(cdr(x))`, `car(cdr(cdr(x)))`. Then, using templates for certain kinds of recurrences, he derives the function (recall that `nil` is an atom in LISP):

$$\begin{aligned} f(\mathtt{cons}(X, Y)) &= X \quad \text{if } Y \text{ is an atom} \\ &= f(Y) \quad \text{otherwise} \end{aligned}$$

The ability to generalize from $\mathtt{cons}(a, b)$ to $\mathtt{cons}(X, Y)$ (where X and Y are variables) derives from his (arbitrary) assumption that the functional form f depends only on the structure of the input list, not on the specific values.

Shapiro's Model Inference System shifted the approach to programming by examples away from finding recurrences. His algorithm is based on generalizing and specializing formulas by instantiating variables to terms and adding clauses to conjunctive-form logical sentences. Since then, most of the research—notably, the work on inductive logic programming [38, 32, 35] – has concentrated on inductive generalization—generalizing specific formulas until they are general enough to cover the examples— instead of looking for patterns and recurrences. While inductive generalization by itself is usually a sufficient technique, it does not take advantage of information about the target function or relation that is revealed by sequential patterns.

For example, consider the arithmetic function $F(X,Y)$ defined by the following examples:

X	0	1	0	2	1	0	3	2	1	0	4	3	...
Y	0	0	1	0	1	0	0	1	2	3	0	1	...
F	2	2	2	2	3	2	2	4	4	2	2	5	...

The structure of this concept becomes much more discernible if arranged so that the sequential patterns emerge:

Y	0	1	2	3
X				
0	2	2	2	2
1	2	3	4	5
2	2	4	6	8
3	2	5	8	11

The patterns $F(X,0) = 2$, $F(X,1) = F(X-1,1)+1$, $F(X,2) = F(X-1,2)+2$, ... are all easy extrapolations. Then by generalizing these patterns (instead of the original examples) we obtain

$$F(X,Y) = F(X-1,Y) + Y,$$

with an initial value of $F(0,Y) = 2$. Inductive programming currently suffers from the need for a very large sample size; by contrast, extrapolation requires small sample sizes. In the preceding example, fewer than twenty values of F sufficed to infer a simple hypothesis for the concept. Whereas sequence extrapolation alone is not sufficient for programming by example, in combination with inductive generalization it may enable us to resurrect and incorporate the good ideas from some of the past research.

Numerical discovery is the term often used to describe the problem of finding a simple formula to account for numerical data. For example, given (noisy) measurements of pressure, volume, and temperature, one looks for a "simple" relation $f(P,V,T) = k$ that agrees "adequately" with the data. (Interpreting the quoted terms is the hardest part.) Current algorithms feature regression and knowledge-based discovery techniques, such as [27, 14].

At first glance sequence extrapolation seems not to apply, since the observations need not be in order and since they are noisy. But the formulas one seeks are generally rational functions with integer coefficients, and in many cases (e.g., economic measurements) one or more of the variables is evenly spaced (e.g., $t = 1, 2, \ldots$). Two kinds of noise affects sequence extrapolation: discrete noise, where an input symbol is occasionally changed to another, unrelated symbol, and continuous noise, where numerical input values differ from their true values by an amount whose probability decreases with its absolute value. The assumption of integral coefficients means, for example, that neither the formula $IR/E = 1$ nor $2IR/E = 1$ will fit the data exactly, but as more input values accrue, one's confidence in the one will outstrip that in the other. Moreover, one can quantify that confidence, as discussed above.

As a final potential application, let us mention the large class of formal program transformation techniques wherein a functional or logic program with undesirable features is transformed into one with desirable ones. For example, the unfold-fold technique suggested by Burstall and Darlington [7, 45] entails unfolding (i.e., partially evaluating) a recursive program some number of times, applying some equivalence transformations to the expanded form, and folding the program into a more efficient recursive form. Automating these transformations is difficult because a certain amount of insight into which transformations improve performance seems to be required. But many transformations, when applied to a sequence of unfoldings, lead to a sequence of forms that a sequence extrapolator can then fold back into a recursive form. Testing this resulting form for improved properties (speed, termination, correctness, etc.) is no small task, but the refolding process itself can be instantiated as sequence extrapolation.

Finally, we note some characteristics of the known sequence extrapolation algorithms. Like stochastic sequences, stream descriptions have two ways to characterize their degrees of freedom: the *order* (also called the delay, or latency), and what I shall call the breadth (not a standard term). The order measures how many preceding values determine the next value. The breadth depends on the representation and counts the number of substreams that are required to define the observed stream. (A close analogy would be the number of variables in a context-free grammar or the number of predicate symbols in a logic program.) As noted, there are two flavors of noise: metric and non-metric. Both are most damaging if they occur early in the stream, since then they are most likely to cause false hypotheses to be proposed and correct hypotheses to be rejected.

As with stochastic sequences the effectiveness of an extrapolation algorithm can again be measured by how well it compresses a data stream. Note that a perfect extrapolator can compress an infinite stream into a finite number of bits, but with noise or a weak representation language, the infinite stream can only be compressed into a smaller, but still infinite, stream.

Hypotheses are rules rather than descriptions of stochastic processes. Hence it is not clear how uniform convergence results can be applied to sequence extrapolation. But just as stochastic models typically exhibit uniform convergence of the likelihood of the possible hypotheses to their means, the confidence (Bayesian posterior) of our models likewise appears to converge uniformly, although we have not done any analysis on this.

Time Series

A time series is a vector function of time (or other continuous scalar variable), $\mathbf{X}(t)$. Observations consist of samples of $\mathbf{X}$ at regular or irregular intervals, $\mathbf{X}_t$, $\mathbf{X}_{t+\tau_1}$, $\mathbf{X}_{t+\tau_2}$, ... The learning task is to construct a model of $\mathbf{X}(t)$ so that certain predictions can be made about the course of its future values.

The dream is that one can infer $\mathbf{X}_t$ and use it to predict stock prices, weather patterns, cardiovascular functions, and similarly important series. However, without strong assumptions about $\mathbf{X}$ this problem is hopelessly difficult.

Nevertheless a number of fairly general and useful algorithms are available for inferring the properties of time series.

Formal analysis of time series began about seventy-five years ago with efforts to separate the stationary (uncorrelated) component of the series from the trend (non-stationary, deterministic component). An elegant theory developed for a family of linear stationary models, known as ARMA (autoregressive moving-average) models, that fit the data to the form

$$\mathbf{X}_n = \sum_{i=1}^{m} a_i \mathbf{E}_{n-i} + \sum_{j=1}^{d} b_j \mathbf{X}_{n-j}.$$

Here $\mathbf{E}_n$ is a white-noise (uncorrelated) process that combines with a finite number of recurrent values of the series $\mathbf{X}$ to produce successive values. The "deconvolution" problem is to infer the coefficients of this process from the input examples. This turns out to be possible because from the spectrum of the series one can estimate the autocorrelation function, and from that one can solve for the coefficients. An iterative linear modeling procedure developed by Box and Jenkins [6] first applies differencing to subtract a polynomial non-stationary component of the series and then deconvolves, repeating the process until the best fit is obtained.

Despite its elegance, this "ARIMA" model often gives poor results with real data. Extensions to higher-order powers of $\mathbf{X}$ lead to integral equations that are difficult to solve. An analytical *theory of inverse problems* has arisen to study general mathematical issues of extracting a function f from discrete values of $\mathbf{B}f$, where $\mathbf{B}$ is an operator in a general family of operators [46]. Other nonlinear modeling methods such as regressive splines, Padé approximants, and maximum entropy are effective for specific problems; [13] contains good summaries of many of these.

Recently two research areas—chaos and neural networks—have contributed new ideas to the learning of time series. Statistical models decompose a stationary series into random and deterministic components:

$$X = A * R + D,$$

where R is a white-noise process and D is a deterministic process ($*$ is the convolution operator and A is a constant filter). A theorem due to Wold [54] states that, under very general conditions, any stationary sequence can be so decomposed. The novel insight is the "deterministic" and "predictable" are not the same: chaos theory has demonstrated that most nonlinear deterministic systems exhibit complex behavior that is difficult to predict, in the sense that system trajectories diverge exponentially in time no matter how close their initial position. Computing the behavior of such a system requires $O(t)$ space and quickly exceeds the computational capacity of real machines. In the end such systems are as unpredictable as purely random ones.

Yet just as random processes have predictable properties (e.g., their moments), so do chaotic ones. For example, the well-known logistic recurrence,

$X_{n+1} = 4X_n(1 - X_n)$, yields a complex, uncorrelated time series. If $0 \leq X_1 \leq 1$, the values of X_n remain between 0 and 1. Given successive values of this sequence, how could we detect that it is generated by a simple deterministic process rather than a stochastic one? If we graph the *return map* X_{n+1} versus X_n, we obtain an extremely simple curve: a parabola. Since no random process would exhibit such regularity, we are sure that the process is deterministic. More generally, to distinguish a chaotic process from a random one, we can look for some deterministic function of the process. But how do we find such a property, and what can we do if the process is a mixture of random and chaotic processes?

As yet the answers to these questions are not fully known, but some intriguing hints have emerged. For sampled chaotic processes the *k-th order return map*, X_{n+1} as a function of $X_n, \ldots, X_{n-k+1}$ enjoys some remarkable topological properties. Roughly speaking, as k increases from 1, there exists a value k_0 such that for all $k \geq k_0$ the topological dimension of the surface of the return map embedded in the k-dimensional space R^k is much smaller than k. The conditions are (1) that the time interval, or *lag*, between the measurements be large enough that the chaos has a chance to eliminate most of the correlations between values, and (2) that the values $X(t)$ we observe be derived from those of the underlying process $\mathbf{Y}_t$ by a diffeomorphic coordinate transformation $X(\mathbf{Y}(t))$. This so-called *embedding dimension* can be estimated numerically from the sample entropy of the sequence and serves as a measure of the inherent complexity ("degrees of freedom") of the underlying process. [44, 40]. Experimentally, graphical procedures are often effective in determining a bound on the embedding dimension for chaotic time series [33, 34]. One cannot help being impressed when an apparently random process—be it water dripping from a faucet or the planet Pluto moving in a gravitational potential—is coaxed into showing us its basic simplicity.

Scargle [41] has generalized Wold's theorem by further decomposing the deterministic component of a stationary time series:

$$X = A * R + B * Y + C,$$

where Y is an uncorrelated chaotic series and C is a non-chaotic deterministic series. He is developing deconvolution techniques whereby the components of X can be estimated from the observations.

Most algorithms for analyzing time series seek to predict the future course of the sequence, if not exactly, at least within specified ranges. Recently researchers have develop a number of new approaches. Their effectiveness evidently depends on the complexity of the underlying process and on whether the short-term or long-term behavior of the series is to be modeled. Weather-prediction methods, for example, are very different depending on whether one wants to know the weather a few hours or days hence or whether one is interested in three- to six month temperature trends. For short-term analysis, neural network methods have proved relatively successful, especially for high dimensional processes. The networks are trained to approximate the embedded surface $X_{t+1} = f(X_t, \ldots, X_{t-k+1})$ in "lag space" with a smooth, non-linear surface. Successful architectures have included sigmoid and radial-basis nets [28, 52] and *finite impulse response (FIR)* nets [49], in which simple connections are replaced by FIR filters and trained using a parallel backpropagation technique. As with other connectionist methods, the effectiveness of the algorithms is difficult to quantify, and the complexity increases rapidly with the size of the problem.

Recently a competition to predict the future of time series [51] received a number of connectionist entries, some quite successful and other not. Reportedly the successful ones required a lot of experimentation and analysis to determine the appropriate embedding delay, network structure, and training procedure for the task. In view of this per-problem empirical analysis, it is still a stretch to refer to these procedures as "algorithms."

Besides connectionist models, researchers have proposed algorithms that model small neighborhoods of the embedded surface as hyperplanes. These so-called *local-linear models* are based on the idea that the surface is approximately linear in a sufficiently small neighborhood. The hyperplane can be constructed using the nearest neighbors in R^k and used to project the vector $(X_n, \ldots, X_{n-k+1})$ to $(X_{n+1}, \ldots, X_{n-k+2})$ [15, 8]. (The ARMA model is a "global" linear model in that a single hyperplane represents the entire surface.) The larger the approximating surface, the greater the scale of the prediction: a local model based on a very small neighborhood is best for short-term predictions, whereas a longer time scale requires a coarser neighborhood model.

Let us review briefly some of the characteristics of time-series algorithms. The order (delay, etc.) of the series—i.e., the number of immediately preceding values upon which the next one depends—seems a fundamental value to establish. For a series obtained by sampling a continuous process, the *lag* or time between observations can also affect the results: if it is too short, one may overlook the effects of chaos that can eliminate any apparent predictability over short time intervals; if too long, one may not be able to extrapolate on a sufficiently short time scale. Related is the question of overfitting: if we fit the observations too closely, our model may incorporate noise and other spurious effects as non-random components of the series, leading to poor extrapolation results. One way that neural network models avoid overfitting is by halting training when cross-validation scores (obtained by testing the predictions against data withheld from training) stop improving and begin to worsen [50]. Another is to prune away elements that contribute little to reducing the error [29]. Still another is to build up the network incrementally, increasing the size only when doing so significantly improves the performance. The true issue here is how much generalization can justified by the data for a particular family of hypotheses; and while the fundamentals of this question are understood fairly well for concept-learning problems [3, 20], it is very much an open problem for time series.

Decomposition has been a recurring strategy for time-series analysis, whether separating the stationary from the non-stationary, the chaotic from the simple deterministic, or one local neighborhood from another. Finally, although we have not mentioned compression, the entropy of the underlying source process plays an important role in the ergodic theory of time series. Learning reduces the rate of increase in the information provided by the source process, and this in turn translates directly into greater compression of the sample.

4 Summary

Although the three types of sequential learning problems are very different one from the other, we should at least pick out some common aspects:

- They try in some fashion to reduce the sequence to simpler subsequences: by identifying common contexts or states, by expressing the sequence as a combination of other sequences, or by separating random, chaotic, and simple deterministic contributions.
- They predict or represent the next input value based on a finite number of its recent predecessors. Determining the number of those predecessors (the order of the sequence) seems to be a crucial part of any algorithm.
- The success of the learning algorithm is measured by prediction accuracy, or equivalently, by the ability to compress the information in the examples.
- The notion of convergence of random variables applies, but it has not been exploited as effectively as in concept learning. For stochastic sequences the likelihood of the hypothesis converges to its limit value, so that maximum likelihood strategies are effective [4, 1]. For sequence extrapolation —which is not a random process— our interpretation of the performance or likelihood of a hypothesis depends on our forming some probabilistic assumptions about the occurrences of prediction errors. To do so may seem rather arbitrary since these errors are deterministic rather than probabilistic, but evidently we humans do something like that in order to estimate the confidence in our hypotheses. For time series problems, Bayesian and maximum entropy techniques converge rapidly and as such discriminate with great sensitivity among different models when strong models about the underlying process are available. See [11], especially the article by Gull [16].

I conclude with these observations. Concept-learning and clustering research has had its greatest impact with relatively simple, general-purpose algorithms (decision trees, networks, hierarchical clustering, etc.) that apply broadly in the absence of strong models about the data. Similarly I expect that the most influential sequence learning algorithms to be simple even if naïve, effective if not rigorous, for divers types of data streams. For stochastic sequences we have such, but general-purpose algorithms are still lacking for the other two. Also, all the algorithms and procedures described here are first-order algorithms—in effect, search optimization algorithms. Aside from obtaining confidence estimates, little research is given to learning the properties of the solutions themselves.

Acknowledgment

Doug Fisher kindly reviewed a draft of this paper and returned helpful suggestions.