TIC – A Toolkit for Validation in Formal Language Learning

Volker Dötsch*
Universität Leipzig
Institut für Informatik
Augustusplatz 10–11
04109 Leipzig
Germany

Klaus P. Jantke**
Deutsches Forschungszentrum
für Künstliche Intelligenz
Stuhlsatzenhausweg 3
66123 Saarbrücken
Germany

Abstract

Quite often, heuristics and common sense suggest directions for improving well-known learning algorithms. However it seems not an easy task to verify that the modifications are indeed helpful.

This is made more complicated through various additional influences inherent in different application domains. In order to obtain a faithful impression of phenomena that are intrinsic to the algorithms, the role of specific domains should be minimized.

Our validation toolkit TIC allows to explore the behaviour of various algorithms for learning formal languages. This is a well-examined and standardized application domain.

TIC is operated by interactive as well as automatic control.

Motivation and Introduction

Today, a lot of different learning approaches and algorithms do exist. There are "classical" as well as "brand new" approaches, and all of them come in many versions and refinements. On the one hand this indicates a desirable improvement of methods, but on the other hand it makes their relative strengths and weaknesses more and more difficult to compare. This also hinders the transfer of results and ideas from one approach to another.

Quite often, heuristics and common sense suggest directions for improving well-known learning algorithms. However it seems not an easy task to verify that the modifications are indeed helpful.

In TIC two prototypical types of learning algorithms were implemented: inductive algorithms and case-based algorithms. For introductions to inductive learning, see (Angluin & Smith 1983) and (Jantke 1989); for an overview of case-based reasoning, see (Riesbeck & Schank 1989) and (Kolodner 1993).

We decided to restrict our attention to the area of learning formal languages (Hopcroft & Ullman 1979), for three, rather independent, reasons: (1) In practice it is difficult to characterize and compare different application domains. In addition, such domains always have characteristic features that dominate the specific properties of the useable learning algorithms. The influence of specific domains should be minimized in the interest of the study of fundamental phenomena. (2) Formal languages are widely used to represent knowledge in computer science. So the concept is general enough to supply practically interpretable results. (3) Formal languages, in particular in their connection with acceptor and generator concepts, are theoretically well examined. Thus a rich reservoir of results is available, into which the results of validation and evaluation of investigated approaches and algorithms can be embedded and discussed.

(Gold 1967) is the seminal paper underlying our learning paradigm. The main task in learning is to learn from incomplete information. This incompleteness approximates practical problems.

There are several ways to present information about formal languages to be learnt. The basic approaches are defined via the concepts *text* and *informant.* A text is just any sequence of words exhausting the target language L. An informant for L is any sequence of words labelled by 1 or 0 such that all the words labelled by 1 form a text for L whereas the words labelled by 0 form a text of the complement of L.

Learning algorithms have to express their hypotheses in some particular form. Case-based learners generate bases of selected cases and tune similarity concepts; cf. (Jantke & Lange 1995). Because a similarity measure can be refined during the learning process, each case-based hypothesis consists of the case base and the similarity measure. A small number of case-based learning algorithms (Aha, Kibler, & Albert

* volkerd@informatik.uni-leipzig.de
** jantke@dfki.de

1991) have been published that reflect the standard case-based reasoning paradigm.

Inductive learning algorithms construct explicit generalizations from examples (processed information). Such a hypothesis could be a regular expression, pattern, automaton, etc.

In one learning step the learning algorithm outputs one hypothesis. In the next learning step, if further information is offered, another (refined) hypothesis possibly is output. Thus, during the complete learning process, a sequence of hypotheses is produced.

TIC Overview

TIC is a tool for validating algorithms that are supposed to learn formal languages. We did implement 21 algorithms: 6 basic algorithms and several versions of it. One can examine one algorithm or compare some algorithms in the batch processor. Typical questions addressed are as follows:

- Do the algorithms show the expected behaviour (e.g. consistence)?
- Do the algorithms stabilize on a correct hypothesis on increasing information provided?
- How fast do the hypothesis sequences stabilize?
- Do algorithms need a repetition or a special ordering of the information provided?
- How does the size of the hypotheses depend on the amount of information processed?

Validation may be seen as the process of inspecting a given learning algorithm for answering questions like these.
The following algorithms have been implemented:

- the basic algorithm for learning finite, deterministic automata described by Trakhtenbrot and Barzdin (Trakhtenbrot & Barzdin 1973) and 9 variants of it,
- IB1, IB2, IB3 by David Aha (Aha, Kibler, & Albert 1991) for case-based learning, slightly adapted for learning formal languages, and three different types of similarity measures for each IB-Alg. (cf. (Dötsch & Jantke 1996)),
- the algorithm FIND developed by Sakakibara and Siromoney (Sakakibara & Siromoney 1992) for inductive learning *Containment Decision Lists* (CDL for short),
- the algorithm by D. Angluin (Angluin 1980) for learning regular patterns.

TIC is providing a framework for interactively inspecting these algorithms including visualization and documentation of the results.

Validation Scenarios

With TIC we focus on interactive approaches to the validation of learning algorithms. Based on the ideas in (Grieser, Jantke, & Lange 1998) a validation scenario consists of test case generation, experimentation, evaluation, and assessment. Test case generation depends on some intended target behaviour, possibly with respect to peculiarities of the system under validation. Interactive validation is performed by feeding test data into the system and receiving system's response. The results of such an experimentation are subject to evaluation. The question is whether a learning algorithm is able to learn all target objects with respect to the considered criterion of success. Thus, a validation problem for a learning algorithm is a triple of a set U of representers (e.g. automata or CDLs), some learning algorithm S and a criterion of success.

Test Data Generation

After the selection of one algorithm for evaluation it is necessary to fix the used alphabet. In TIC all upper case and lower case characters and the digits 0 to 9 are allowed. One has to select U, the set of representers[1] which the learning algorithm is supposed to learn. Maybe U is infinite, but one can test finitely many representers only. Here the idea is to select representers that are typical for U, and test only these. Another strategy is implemented for finite automata only. One can generate automata randomly and use these for validating the learning algorithm.

A test case consists of one specific element of U (one representer) and a subset of all possible information (test data) for it. As test data, the use of informant is default, but the user also can restrict it for using text only. The definition of test data is a two-phase process. In the first phase the user defines a list of words (a subset of all words over the fixed alphabet, duplicates are allowed), in the second phase TIC classifies each word via the selected representer.

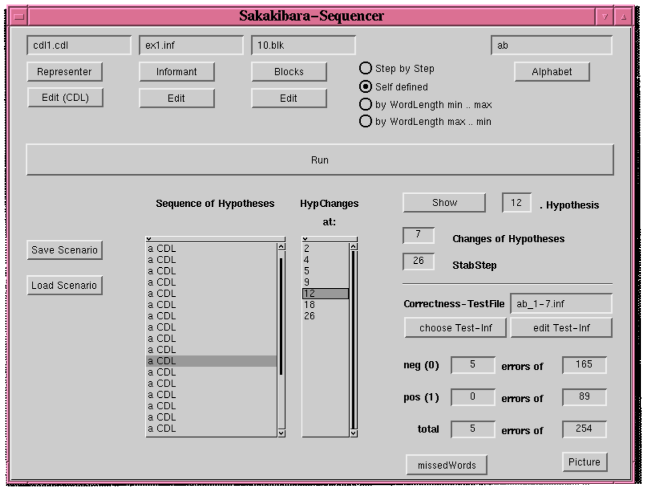
To define the list of words, we implemented a comfortable editor in TIC. It generates words over the fixed alphabet. All the words will be classified (0 or 1) during the learning process before feeding them into the learning algorithm. This guarantees the consistency between the selected representer and the test data. The editor for the learning information makes available the following functionality: Modify the list

[1]Possible are regular patterns, containment decision lists or finite deterministic automata. There exist editors for defining representers of these three types. The functionalities *save, copy* and *load* are available for each type of representers; *minimize, generate randomly* and *test for equality* only for some.

by hand (insert, add, delete), Generate n words from word length i to length j randomly, Generate all words between word length i and j, Load and save a list, Append a complete list to the current list, Remove duplicates in the current list, Order the current list (randomly, alphabetically, by word length, ascending/descending). With this features the user can make sure that the list of test data is appropriate to show the behaviour of the learning algorithm the user wants to validate. For example: to validate that the algorithm learns correctly without duplicates in the learning information, one can remove all duplicates in the test data. To validate that the algorithm learns correctly from order-independent information, one can feed in randomly ordered test data.

Experimentation

Up to now we worked with learning algorithms that generate an output on every possible input. Before one can start the experimentation, one has to define the special parameter ***stepwise*** for the learning process. This parameter specifies how TIC divides the list of test data in initial segments of a certain length. Each segment is processed by the learning algorithm. So, we get a sequence of hypotheses over increasing information during the learning process. Possible options are "step by step", "by word length" or "self defined" (fig. 1). "Step by step" means the learning algorithm sees



cdl1.cdl | ex1.inf | 10.blk | ab

( ) Step by Step
(●) Self defined
( ) by WordLength min .. max
( ) by WordLength max .. min

12 . Hypothesis
7 Changes of Hypotheses
26 StabStep

Sequence of Hypotheses: a CDL, a CDL, a CDL, a CDL, a CDL, a CDL, a CDL, a CDL, a CDL, a CDL, a CDL, a CDL, a CDL, a CDL, a CDL, a CDL, a CDL

HypChanges at: 2, 4, 5, 9, 12, 18, 26

Correctness-TestFile ab_1-7.inf

neg (0)	5	errors of	165
pos (1)	0	errors of	89
total	5	errors of	254

Figure 1: TIC after learning and testing

the first example (word and its classification) and produces a hypothesis in the first learning step. In the next learning step, the algorithm sees the first two examples and produces a hypothesis. In the third learning step the algorithm sees the first three examples, and so on. If the stepwise is "by word length", in the first learning step the algorithm works on all examples with the shortest word length in the given test data, and produces a hypothesis. In the second step, the algorithm works on all examples with the shortest and the next length. And so on, up to the maximal word length in the information provided. Also it is possible to start with the maximal word length and go down to the shortest length. If one choose "self defined", one can define how much examples the algorithm works on in each learning step (named "blocks" in fig. 1); for example, in the first step the first three examples, in the next step the first 50 examples, in the next step the first 175 examples. There is only one condition: the number of examples must increase in each step.

During an experiment, TIC produces a sequence of hypotheses and the list of numbers of those steps, at which the hypotheses were changed. If the stabilization on a correct hypothesis is decidable, that stabilization point in the sequence of hypotheses is shown.

A test scenario is defined by any test case and a stepwise. One can save the complete scenario with all settings[2] and load it for continuing later.

Evaluation

Each test case and its appropriate hypothesis is subject for evaluation. One can check whether the hypothesis is correct (correctness is the criterion of success). If the target language is regular, it can be decided whether or not the actual hypothesis of the learning algorithm describes the target language correctly. For regular patterns and finite automata such a test is implemented, and TIC uses it automatically. For CDLs and case-based hypotheses we go another way. We define a ***test set*** T of words over the used alphabet. Now the toolkit classifies correctly each $w \in T$ via the given representer and compares these classification via the selected hypothesis. When any misclassification occurs, the actual hypothesis can not be correct. In the other case (no error) one should check the hypothesis by hand or use another test set. For most of our experiments, a test set T of more than 20 000 words was sufficient to check the incorrectness of hypotheses. The misclassified words can be shown and saved in a separate list for further research.

If hypotheses are not correct, it is useful to know the ratio of the quality between hypotheses. Therefore TIC uses a given test set T and counts the number of misclassifications w.r.t. the target language. The result is shown in the main window (fig. 1) separate for words of the target language (1 classification via the given representer), words of the complement of the target

[2]used algorithm, similarity measure for case-based learning and, when finished the learning process, the list of hypotheses and the stabilization point

language (0 classification) and all words of T. To evaluate the behavior of the used learning algorithm or the strategy of the algorithm, TIC can generate a picture of these error rates for the whole sequence of hypotheses (fig. 2 and 3). It is possible to make visible one, two or all error rates. Each test case and its appropriate

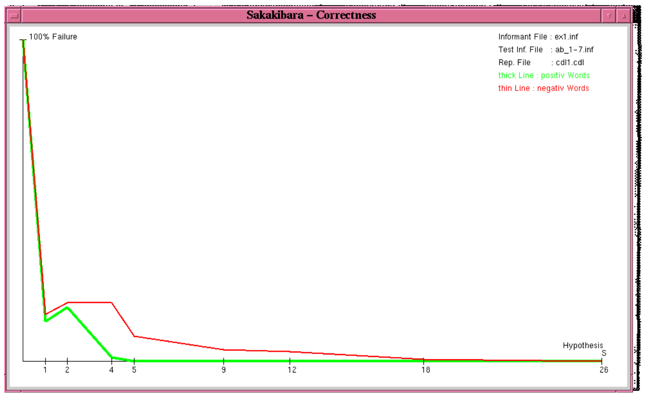


Figure 2: Error rate for correct learning

hypothesis is evaluated (marked) by a number between 0 and 1, expressing the opinion whether or not the experiment witnesses the learning algorithm's ability to learn the target language.

In case the user is validating a learning algorithm for consistency, it is enough, to test the hypotheses with the same information that the algorithm learned from.

TIC also shows the size of hypotheses. So the user can evaluate the relation between size of hypotheses and the increasing information during the learning process. One can open a separate window for each hypothesis for a deeper analysis. Also one can save each hypothesis to use it as a representer later.

Assessment

When systems are valid with respect to the criterion of success, experiments on the basis of larger and larger test data sets are evaluated (marked by a number between 0 and 1) and the evaluation converges to the value 1.

As said above, the experiments provide only initial segments of potentially infinite experiments. But the implemented learning algorithms have a useful property, which can be exploited to derive validity statements even from a small number of experiments. The algorithms do not change their actual hypothesis any further in case the current one correctly classifies the given data. This is called conservativeness and allows to extrapolate the algorithm's behaviour. Once a conservative learning algorithm outputs a correct hypothesis for the target language, it is guaranteed that the algorithm learns this particular language.

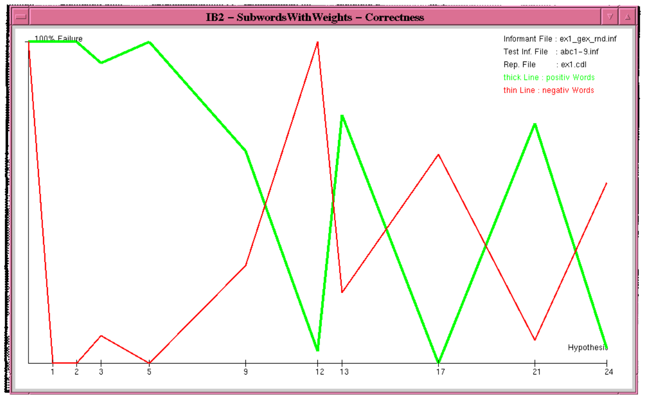


Figure 3: Not learning on randomly arranged good examples

Figure 3 is showing the rates of misclassifications during one run of IB2' on a randomly ordered good example list (cf. section *An Application Example*). It is plain to see how the algorithm is "changing its mind". If the user (the domain expert) had arranged the learning information in a particular order, IB2' would learn correctly.

In the present setting, validation means to run the algorithm under inspection on certain input data which are either randomly chosen or systematically synthesized to represented prototypical learning situations. The expected learning result is known to the validator in advance. Valid systems are expected to clearly demonstrate their success in learning on all input data fed in.

The behavior displayed in figure 3 above does not indicate any success of learning. This bears evidence of the systems ***invalidity***.

Batch processing

For comparing algorithms, batch processing is very useful. The user gives a set R of representers and a set D of lists of test data. So we have the set $C = R \times D$ of test cases. Only one stepwise, a set S of test sets and a set A of learning algorithms is necessary also.

The batch processor runs $R \times D$ learning scenarios for each algorithm. Furthermore the batch processor tests correctness and quality (the three error rates; via the set S of test sets) of each hypothesis. The size of the hypothesis and the result of the tests are saved in a report file in ASCII format, ordered by the used test case and algorithm. One can load the report file into some tools for statistical analysis or generating trend curves for size or quality of hypotheses.

If the user want this, also the batch processor saves all $R \times D \times A$ scenarios in separate scenario files. So one can load each test-scenario and continue or refine the evaluation in the toolkit TIC later.

An application example

Our approach uses the paradigma of case-based learning which can be expressed briefly as follows: *Given any CBR system, apply it. Whenever it works successfully, do not change it. Whenever it fails on some input case, add this experience to the case base. Don't change anything else.* We will show that this is ***practically valid*** only in the presence of ***substantial user guidance.***

The target class of formal languages to be learnt is specified via Containment Decision Lists. The learning theoretic investigation in (Sakakibara & Siromoney 1992) has drawn our attention to this quite simple type of decision lists.[3] A CDL is a finite sequence of labelled words (w_i, d_i) $(i = 1, \ldots, n)$, where the labels d_i are either 0 or 1. Such a list can be easily understood as an acceptor for words as follows. Any word w fed into a CDL is checked at node (w_1, d_1) first. If any check tells us that w_i is a subword of w, than this word is classified as determined by d_i, i.e. w is accepted exactly if $d_i = 1$. If otherwise w does not contain w_i, the input word w is passed to w_{i+1} and so on. By definition, w_n equals the empty word ε, is a subword of any w, and thus the process described terminates in a well-defined manner.

The CDL $L = [(aab, 1), (aa, 0), (\varepsilon, 1)]$ is an illustrative example. Roughly speaking, the language accepted by L contains all words containing aab or not containing a square of a. Words in the complement are containing aa, but not containing aab.

Within case-based reasoning, case-based learning as investigated in (Aha, Kibler, & Albert 1991) is a natural way of designing learning procedures. There are even normal form results (cf. (Globig *et al.* 1997)) explaining that all learning procedures of a certain type may be rewritten as case-based learning procedures. The first task of case-based learning is to collect good cases which will be stored in the case base for describing knowledge and classifying unknown examples. Case-based learning algorithms do not construct explicit generalizations from examples. Their hypotheses consist of case bases together with similarity concepts. Both constituents may be subject to learning, i.e. the second task of case-based learning might consist in suitably tuning the similarity measure in use.

Due to (Sakakibara & Siromoney 1992), arbitrary containment decision lists are known to be learnable. In other words, the knowledge contained in any CDL L can potentially be acquired by processing finitely many cases describing the target language accepted by L.

[3]All languages accepted by CDLs are regular, but not all regular languages can be accepted by a CDL.

In (Aha, Kibler, & Albert 1991) there has been presented a simple algorithm named IB2 for acquiring knowledge from finitely many cases. IB2 is selectively collecting cases which are subsequently presented, in case there is any need to do so. It is exactly following the paradigmatic idea described above.[4]

A lot of example CDLs were used for series of experiments. We have performed more than 1 000 000 particular test cases. They exhibit a catastrophic behavior of IB2'. It turns out that algorithms like IB2 and IB2' do essentially depend on user guidance.[4]

Validation results

For case-based learning languages of CDLs, we first tried completely unsupervised learning experiments. Every individual experiment was an attempt to learn from a sequence of correctly classified cases, but they failed completely.

From critical inspection of the difficulties, we have been lead to the concept of *good example lists.*[5] Those lists are known to be sufficient for learning. On the one hand, they are algorithmically well-defined and can be generated automatically. On the other hand, they might be difficult to find, if the target phenomenon is not sufficiently well-understood. Even if everything needed to build those lists of good examples is known, it might be an additional problem to arrange this knowledge appropriately. We performed experiments, to explore the importance of finding an appropriate ordering of information presented as a basis for learning. The results illuminate the sensitivity of case-based learning to the ordering of information quite well; a ratio of success of 10% or below usually is unacceptable in realistic applications.

We are convinced that case-based learning of CDLs is considerably simpler than most problems of knowledge acquisition in "real life". Thus, user guidance for acquiring knowledge in a case-based manner is practically at least as important as exhibited in the prototypical domain of our present investigations, it is *just inevitable.* The toy application domain chosen for the present work is extremely simple. Intuitively almost every other application domain of some proper relevance is of an larger complexity. It is quite unlikely that in those realistic domains very simple algorithmic ideas should succeed, that fail in our toy domain. This circumscribes our understanding of a "lower bound" provided by the present findings.

[4]For our purpose, we extend IB2 (to IB2') to allow for an adaptation of similarity concepts. The reader is directed to (Jantke & Dötsch 1997a) and (Jantke & Dötsch 1997b) for more and detailled information.

[5]Learning from good examples was introduced in (Freivalds, Kinber, & Wiehagen 1989).

Conclusion

TIC is a toolkit aimed at validating the behavior of various algorithms and to generalize some effects. TIC is able to produce and to show a sequence of hypotheses out of given examples. It makes visible correctness and quality of hypotheses and some more information of interest. In the system, a few algorithms and variants of these are implemented.

TIC is a fully object-oriented implementation. So other algorithms for validation or other strategies for analysis and evaluation can be plugged in. Also one can extend the system for learning in other domains, e.g. learning functions.

Acknowledgment

The implementation of the first version of TIC was part of the project IND–CBL. It has been supported by the German Research Fund (DFG) under grants no. Ja 566/2–1, Ja 566/2–2 and Ja 566/2–3. Our ideas have been inspired by many fruitful discussions with Michael M. Richter, Christoph Globig and Steffen Lange. The first version of TIC (cf. (Burghardt, Dötsch, & Frind 1996)) was developed and implemented by the first author in cooperation with Udo Burghardt and Stephan Frind. Today TIC is developed and used by the authors in cooperation with Gunter Grieser and Steffen Lange.

References

Aha, D. W.; Kibler, D.; and Albert, M. K. 1991. Instance-Based Learning Algorithms. *Machine Learning* 6(1):37–66.

Angluin, D., and Smith, C. H. 1983. A Survey of Inductive Inference: Theory and Methods. *Computing Surveys* 15:237–269.

Angluin, D. 1980. Finding Patterns Common to a Set of Strings. *Journal of Computer and System Science* 21:46–62.

Burghardt, U.; Dötsch, V.; and Frind, S. 1996. TIC - ein Testrahmen für IND-CBL. Technical Report CALG-01/96, HTWK Leipzig.

Dötsch, V., and Jantke, K. P. 1996. Solving Stabilization Problems in Case-Based Knowledge Acquisition. In Compton, P.; Mizoguchi, R.; Motoda, H.; and Menzies, T., eds., *Proc. of PKAW'96*, 150–169. University of NSW, Australia.

Freivalds, R.; Kinber, E.; and Wiehagen, R. 1989. Inductive Inference from Good Examples. In Jantke, K. P., ed., *Proc. of AII'89*, volume 397 of *LNAI*, 1–17. Springer.

Globig, C.; Jantke, K. P.; Lange, S.; and Sakakibara, Y. 1997. On Case-Based Learnability of Languages. *New Generation Computing* 15(1):59–83.

Gold, E. M. 1967. Language Identification in the Limit. *Information and Control* 14:447–474.

Grieser, G.; Jantke, K.; and Lange, S. 1998. Towards the Validation of Inductive Learning Systems. In Richter, M.; Smith, C.; Wiehagen, R.; and Zeugmann, T., eds., *Proc. of ALT'98*, volume 1501 of *LNAI*, 409–423. Springer.

Hopcroft, J. E., and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley.

Jantke, K. P., and Dötsch, V. 1997a. Extended Experimental Explorations of the Necessity of User Guidance in Case-Based Learning. Technical Report DOI-TR-135, Kyushu University, Fukuoka, Japan.

Jantke, K. P., and Dötsch, V. 1997b. The Necessity of User Guidance in Case-Based Knowledge Acquisition. In Dankel II, D., ed., *Proceedings of the FLAIRS'97*, 312–336. Florida AI Research Society.

Jantke, K. P., and Lange, S. 1995. Case-Based Representation and Learning of Pattern Languages. *Theoretical Computer Science* 137(1):25–51.

Jantke, K. P. 1989. Algorithmic Learning from Incomplete Information: Principles and Problems. In Dassow, J., and Kelemen, J., eds., *Machines, Languages, and Complexity*, volume 381 of *LNCS*, 188–207. Springer.

Kolodner, J. L. 1993. *Case-Based Reasoning.* San Mateo: Morgan Kaufmann.

Riesbeck, C., and Schank, R. 1989. *Inside Case-Based Reasoning.* L. Erlbaum Assoc.

Sakakibara, Y., and Siromoney, R. 1992. A Noise Model on Learning Sets of Strings. In *Proc. of COLT'92*, 295–302. ACM.

Trakhtenbrot, B., and Barzdin, Y. 1973. *Finite Automata - Behavior and Synthesis.* Fundamental Studies in Computer Science. Amsterdam: North-Holland.