Title: Inductive programming

URL: https://en.wikipedia.org/wiki/Inductive_programming

PageID: 41644056

Categories: Category:Machine learning, Category:Programming paradigms

---

Inductive programming (IP) is a special area of automatic programming , covering research from artificial intelligence and programming , which addresses learning of typically declarative (logic or functional) and often recursive programs from incomplete specifications, such as input/output examples or constraints.

Depending on the programming language used, there are several kinds of inductive programming. Inductive functional programming , which uses functional programming languages such as Lisp or Haskell , and most especially inductive logic programming , which uses logic programming languages such as Prolog and other logical representations such as description logics , have been more prominent, but other (programming) language paradigms have also been used, such as constraint programming or probabilistic programming .

Definition

Inductive programming incorporates all approaches which are concerned with learning programs or algorithms from incomplete (formal) specifications. Possible inputs in an IP system are a set of training inputs and corresponding outputs or an output evaluation function, describing the desired behavior of the intended program, traces or action sequences which describe the process of calculating specific outputs, constraints for the program to be induced concerning its time efficiency or its complexity, various kinds of background knowledge such as standard data types , predefined functions to be used, program schemes or templates describing the data flow of the intended program, heuristics for guiding the search for a solution or other biases.

Output of an IP system is a program in some arbitrary programming language containing conditionals and loop or recursive control structures, or any other kind of Turing-complete representation language.

In many applications the output program must be correct with respect to the examples and partial specification, and this leads to the consideration of inductive programming as a special area inside automatic programming or program synthesis , [1] [2] usually opposed to 'deductive' program synthesis, [3] [4] [5] where the specification is usually complete.

In other cases, inductive programming is seen as a more general area where any declarative programming or representation language can be used and we may even have some degree of error in the examples, as in general machine learning , the more specific area of structure mining or the area of symbolic artificial intelligence . A distinctive feature is the number of examples or partial specification needed. Typically, inductive programming techniques can learn from just a few examples.

The diversity of inductive programming usually comes from the applications and the languages that are used: apart from logic programming and functional programming, other programming paradigms and representation languages have been used or suggested in inductive programming, such as functional logic programming , constraint programming , probabilistic programming , abductive logic programming , modal logic , action languages , agent languages and many types of imperative languages .

History

Research on the inductive synthesis of recursive functional programs started in the early 1970s and was brought onto firm theoretical foundations with the seminal THESIS system of Summers [6] and work of Biermann. [7] These approaches were split into two phases: first, input-output

examples are transformed into non-recursive programs (traces) using a small set of basic operators; second, regularities in the traces are searched for and used to fold them into a recursive program. The main results until the mid-1980s are surveyed by Smith. [8] Due to limited progress with respect to the range of programs that could be synthesized, research activities decreased significantly in the next decade.

The advent of logic programming brought a new elan but also a new direction in the early 1980s, especially due to the MIS system of Shapiro [9] eventually spawning the new field of inductive logic programming (ILP). [10] The early works of Plotkin, [11] [12] and his " relative least general generalization (rlgg) ", had an enormous impact in inductive logic programming. Most of ILP work addresses a wider class of problems, as the focus is not only on recursive logic programs but on machine learning of symbolic hypotheses from logical representations. However, there were some encouraging results on learning recursive Prolog programs such as quicksort from examples together with suitable background knowledge, for example with GOLEM. [13] But again, after initial success, the community got disappointed by limited progress about the induction of recursive programs [14] [15] [16] with ILP less and less focusing on recursive programs and leaning more and more towards a machine learning setting with applications in relational data mining and knowledge discovery. [17]

In parallel to work in ILP, Koza [18] proposed genetic programming in the early 1990s as a generate-and-test based approach to learning programs. The idea of genetic programming was further developed into the inductive programming system ADATE [19] and the systematic-search-based system MagicHaskeller. [20] Here again, functional programs are learned from sets of positive examples together with an output evaluation (fitness) function which specifies the desired input/output behavior of the program to be learned.

The early work in grammar induction (also known as grammatical inference) is related to inductive programming, as rewriting systems or logic programs can be used to represent production rules. In fact, early works in inductive inference considered grammar induction and Lisp program inference as basically the same problem. [21] The results in terms of learnability were related to classical concepts, such as identification-in-the-limit, as introduced in the seminal work of Gold. [22] More recently, the language learning problem was addressed by the inductive programming community. [23] [24]

In the recent years, the classical approaches have been resumed and advanced with great success. Therefore, the synthesis problem has been reformulated on the background of constructor-based term rewriting systems taking into account modern techniques of functional programming, as well as moderate use of search-based strategies and usage of background knowledge as well as automatic invention of subprograms. Many new and successful applications have recently appeared beyond program synthesis, most especially in the area of data manipulation, programming by example and cognitive modelling (see below).

Other ideas have also been explored with the common characteristic of using declarative languages for the representation of hypotheses. For instance, the use of higher-order features, schemes or structured distances have been advocated for a better handling of recursive data types and structures; [25] [26] [27] abstraction has also been explored as a more powerful approach to cumulative learning and function invention. [28] [29]

One powerful paradigm that has been recently used for the representation of hypotheses in inductive programming (generally in the form of generative models) is probabilistic programming (and related paradigms, such as stochastic logic programs and Bayesian logic programming). [30] [31] [29] [32]

Application areas

The first workshop on Approaches and Applications of Inductive Programming (AAIP) Archived 2016-03-03 at the Wayback Machine held in conjunction with ICML 2005 identified all applications where "learning of programs or recursive rules are called for, [...] first in the domain of software engineering where structural learning, software assistants and software agents can help to relieve programmers from routine tasks, give programming support for end users, or support of novice programmers and programming tutor systems. Further areas of application are language learning,

learning recursive control rules for AI-planning, learning recursive concepts in web-mining or for data-format transformations".

Since then, these and many other areas have shown to be successful application niches for inductive programming, such as end-user programming , [33] the related areas of programming by example [34] and programming by demonstration , [35] and intelligent tutoring systems .

Other areas where inductive inference has been recently applied are knowledge acquisition , [36] artificial general intelligence , [37] reinforcement learning and theory evaluation, [38] [39] and cognitive science in general. [40] [32] There may also be prospective applications in intelligent agents, games, robotics, personalisation, ambient intelligence and human interfaces.

See also

Evolutionary programming

Inductive reasoning

Test-driven development

References

Further reading

Flener, P.; Schmid, U. (2008). "An introduction to inductive programming". Artificial Intelligence Review . 29 (1): 45– 62. doi : 10.1007/s10462-009-9108-7 . S2CID 26314997 .

Kitzelmann, E. (2010). "Inductive Programming: A Survey of Program Synthesis Techniques" (PDF) . Approaches and Applications of Inductive Programming . Lecture Notes in Computer Science. Vol. 5812. pp. 50– 73. CiteSeerX 10.1.1.180.1237 . doi : 10.1007/978-3-642-11931-6_3 . ISBN 978-3-642-11930-9 .

Partridge, D. (1997). "The case for inductive programming". Computer . 30 (1): 36– 41. doi : 10.1109/2.562924 . S2CID 206403583 .

Flener, P.; Partridge, D. (2001). "Inductive Programming". Automated Software Engineering . 8 (2): 131– 137. doi : 10.1023/a:1008797606116 . S2CID 6675212 .

Hofmann, M.; Kitzelmann, E. (2009). "A unifying framework for analysis and evaluation of inductive programming systems" . Proceedings of the Second Conference on Artificial General Intelligence : 55– 60.

Muggleton, S.; De Raedt, L. (1994). "Inductive Logic Programming: Theory and methods" . The Journal of Logic Programming . 19– 20: 629– 679. doi : 10.1016/0743-1066(94)90035-3 .

Lavrac, N. ; Dzeroski, S. (1994). Inductive Logic Programming: Techniques and Applications . New York: Ellis Horwood. ISBN 978-0-13-457870-5 . https://web.archive.org/web/20040906084947/http://www-ai.ijs.si/SasoDzeroski/ILPBook/

Muggleton, S.; De Raedt, Luc.; Poole, D.; Bratko, I.; Flach, P.; Inoue, K.; Srinivasan, A. (2012). "ILP turns 20" . Machine Learning . 86 (1): 3– 23. doi : 10.1007/s10994-011-5259-2 .

Gulwani, S.; Hernandez-Orallo, J.; Kitzelmann, E.; Muggleton, S.H.; Schmid, U. ; Zorn, B. (2015). "Inductive Programming Meets the Real World" . Communications of the ACM . 58 (11): 90– 99. CiteSeerX 10.1.1.696.3800 . doi : 10.1145/2736282 . hdl : 10251/64984 . S2CID 425881 .

External links

Inductive Programming community page , hosted by the University of Bamberg.

v

t

e

Jackson structures

Block-structured

Modular
Non-structured
Procedural
Programming in the large and in the small
Design by contract
Invariant-based
Nested function
Class-based , Prototype-based , Object-based
Agent
Immutable object
Persistent
Uniform function call syntax
Recursive
Anonymous function (Partial application)
Higher-order
Purely functional
Total
Strict
GADTs
Dependent types
Functional logic
Point-free style
Expression-oriented
Applicative , Concatenative
Function-level , Value-level
Monad
Flow-based
Reactive (Functional reactive)
Signals
Streams
Synchronous
Abductive logic
Answer set
Constraint (Constraint logic)
Inductive logic
Nondeterministic
Ontology
Probabilistic logic

Query
Algebraic modeling
Array
Automata-based (Action)
Command (Spacecraft)
Differentiable
End-user
Grammar-oriented
Interface description
Language-oriented
List comprehension
Low-code
Modeling
Natural language
Non-English-based
Page description
Pipes and filters
Probabilistic
Quantum
Scientific
Scripting
Set-theoretic
Simulation
Stack-based
System
Tactile
Templating
Transformation (Graph rewriting , Production , Pattern)
Visual
Actor-based
Automatic mutual exclusion
Choreographic programming
Concurrent logic (Concurrent constraint logic)
Concurrent OO
Macroprogramming
Multitier programming
Organic computing
Parallel programming models

Partitioned global address space
Process-oriented
Relativistic programming
Service-oriented
Structured concurrency
Attribute-oriented
Automatic (Inductive)
Dynamic
Extensible
Generic
Homoiconicity
Interactive
Macro (Hygienic)
Metalinguistic abstraction
Multi-stage
Program synthesis (Bayesian , Inferential , by demonstration , by example)
Reflective
Self-modifying code
Symbolic
Template
Aspects
Components
Data-driven
Data-oriented
Event-driven
Features
Literate
Roles
Subjects