Machine Learning Programs and Compilers

Alexander Gray, Georgia Institute of Technology

I describe here a long-term project of mine called Algorithmica, whose goal is to make the creation of state-of-the-art machine learning programs possible at the expert level, both statistically and computationally, with less time and fewer errors [1, 2, 3]. It is inspired by my experience with the the pioneering AutoBayes system begun by Wray Buntine [4].

Goal. Data analysis has always been a fundamental pillar of science, engineering, and business, and has increased in importance in recent decades as data has increased in availability and scale in virtually all areas, as observed by countless new applications every year.

How is data analysis performed today? First, the person on a team with the most familiarity or inclination toward statistics (we will refer to multivariate statistics, machine learning, and data mining interchangeably) becomes the de facto data analyst. Note that data analysts with formal training in state-of-the-art data analysis are extremely rare, even in the sciences, let alone other areas. This person is generally also not an expert programmer or computer scientist. Armed with a textbook on data analysis and a library of statistical *methods* (perhaps underneath an interactive environment such as R, Matlab, or SAS) written by professional statistician/programmers, the analyst can try a number of methods on his/her data (even more if all the downloadable codes on the internet are collected – though in practice not all of the codes will be reliable or easily usable together).

Note that this is *very far* from from what an *expert* data analyst can do. An expert has a deep, rather than superficial, understanding of the *principles* upon which the methods in the textbook are based and understands that the selection of methods in any textbook or library is simply a small finite sample of the infinite number of combinations and possibilities that the principles allow. Even the methods that exist in a library are unlikely to be implemented using the best known *algorithms* that have been shown somewhere in the vast research literature and are thus unlikely to be usable on a modern dataset which happens to be massive. Given a dataset and problem, a number of abstract principles, and enough time, an expert can derive a custom method (which can achieve more accurate prediction or modeling than a library can) and a custom algorithm for that method (which can make the method tractable on massive data where the library might not be), taking advantage of, for example, closed-form solutions to subproblems, parallelization opportunities, Monte Carlo methods, online optimizers, and advanced data structures. If the analyst, in addition to being a statistical expert, is also an expert *programmer*, he/she would also write extremely tight code which would do things like tune parameters such as cache sizes for the target (multicore) machine architecture. An *army* of expert data analysts could in principle each try different paths in the immense search space of possible statistical methods, algorithms for those methods, and programs for those algorithms, comparing their statistical and computational performance to arrive at a best method-/algorithm/program. Furthermore, the expert analysts' methods would come with oft-overlooked professional-level statistical *validation*, i.e. confidence bands and statistical significance tests.

Approach. We are building a system for interactive statistical modeling which, given a statistical model declaratively specified by the user, interactively or (semi-)automatically derives correct and efficient algorithms for solving them via iterative problem reformulation. With respect to previous automatic derivation and code generation systems, our proposed system represents a pragmatic point in the design space, toward the goal of *correctness by construction* and one we introduce – *performance by construction* – by building automatic experimentation and performance logging into the (interactive or automatic) search process of our rewrite system.

The project has two main parts. The first is a new approach to the development of technical (mathematical) software, based on performance-driven synthesis based on schema (templates) and

a domain-specific typed embedded language, which we believe will allow the creation of an algorithmic derivation system of unprecedented scope and power. The second is the first large-scale type-theoretic formalization (syntax, type system, semantics) of the foundations of probability/statistics and optimization. Note that the level of formality needed to operationalize mathematics for automated proofs is typically much higher than that needed in research papers and textbooks written and digested by humans, which can tolerate high levels of ambiguity and imprecision. Such a formalization thus has mathematical significance in its own right. It is also portable, in principle, to other proof systems such as Coq.

The amount of complexity (number of mathematical and algorithmic concepts) needed to ascend beyond toy problems, toward utility by actual data analysts, is immense. The greater the complexity/scope of the system, the more important it becomes to build in mechanisms for ensuring correctness wherever possible. On the other hand, extreme approaches to correctness, such as verifiability of mathematical assertions all the way down to axioms (as in a full-blown theorem prover like Coq) hampers extendability – the ability of system developers to add large numbers of concepts to the system to achieve the scope needed. The schema-based approach represents a practical point on the spectrum of correctness possibilities, which we believe is sensible. The goal of "correctness by construction" is achieved to the extent that human-written schemas are correct. We propose enforcing greater correctness by defining a *type system* for the input language. This requires the development of a language which "understands" the meaning of critical domain concepts, such as probabilities and optimization problems, so that only valid operations can be performed on them. Our approach is fully typed and strictly logical, unlike Prolog. The system we propose, called *Algorithmica*, is based on one language, called *OM (Operationalized Mathematics)* which will serve as the language for specifying problems declaratively and for specifying (possibly partially instantiated) pseudocode; it also serves as the final output language which can be compiled and run, though we will also generate target code in other languages such as C++ and Matlab in the later phases of the project.

Impact. The broader impacts are potentially enormous. As noted, most of today's data analysts operate far from the expert level. If their capabilities can be augmented, the consequent improvements in data modeling performance and extraction of knowledge from potentially massive data, as well as human productivity across all areas of science, engineering, and business, would be incalculable. The impact on technical (mathematical) education models is also potentially revolutionary, with the capability of engaging young minds with mathematics, algorithms, and programming in ways that were never before possible.

References

[1] S. Bhat, A. Agarwal, R. Vuduc, A. Gray, A Type Theory for Probability Density Functions, in: 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), 2012.

[2] A. Agarwal, S. Bhat, A. G. Gray, I. E. Grossman, Automating Mathematical Program Transformations, in: Practical Aspects of Declarative Languages (PADL), 2010.

[3] S. Bhat, A. Agarwal, A. Gray, R. Vuduc, Toward Interactive Statistical Modeling, in: International Conference on Computational Science (ICCS): Workshop on Automated Program Generation for Computational Science, 2010.

[4] A. G. Gray, B. Fischer, J. Schumann, W. Buntine, Automatic Derivation of Statistical Algorithms: The EM Family and Beyond, in: S. Becker, S. Thrun, K. Obermayer (Eds.), Advances in Neural Information Processing Systems (NIPS) 15 (Dec 2002), MIT Press, 2003.