End-to-end Verification of Code for Statistical Tests

Eliot Moss*, Michael Norrish**, Jared Yeager*

*University of Massachusetts Amherst
**Australian National University

Fairness in Machine Learning

- Fairness in machine learning is an increasingly pressing issue
  - Ramifications of unfair learned functions are increasingly impactful
  - Examples: Giving advice on loan applications, parole hearings, etc.
  - Our formulation also covers *safety* using the same mathematics
- How we formalize fairness of a learned model
  - Apply a statistical test to a validation data set
  - Validation data is separate from training/test data used to build the model
  - The learned model will be applied to some population of cases in the future
  - We *assume* the validation data is drawn randomly from that same population
    - They have the same distribution
  - We do *not* assume that we *know* what that distribution is

The Validation Procedure

- For each validation case x_i (i = 1, ... , n)
  - Apply the learned model to get predicted result $f(x_i)$
  - Compute an error err_i as some difference between $f(x_i)$ and the desired result y_i
  - For example, err_i = 0 if the loan decision agrees, err_i = 1 if it does not
- Compute the sum/average of the err_i, Aerr
- We desire E[Aerr] ≤ some chosen limit, e.g., .01 (1% error rate)
  - This is what “fair” means in this context
- Using the concentration inequality we can test whether E[Aerr] meets that limit, within some probability p

The Validation Procedure: Hoeffding's Inequality

Let $X_i, ..., X_n$ be independent random variables such that $\forall i : 1 \leq i \leq n \rightarrow \exists a_i, b_i : P(X_i \in [a_i, b_i]) = 1$

$$\forall t : P(E(\overline{X}) - \overline{X} \geq t) \leq exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

So if each $X_i = Err_i$, is 0/1 error, then:

$$\forall t : P(E(\overline{Err}) - \overline{Err} \geq t) \leq e^{-2nt^2}$$

The Validation Procedure: Example

Say our validation error on 500 samples was 0.04
Say we want $E(\overline{Err}) \leq 0.05$ with probability 0.95

$$P(E(\overline{Err}) - 0.04 \geq 0.01) \leq e^{-2(500)(0.01)^2} = 0.9048$$

$$P(E(\overline{Err}) \leq 0.05) > 1 - e^{-2(500)(0.01)^2} = 0.0952$$

$$P(E(\overline{Err}) \leq 0.05) > 1 - e^{-2(14979)(0.01)^2} = 0.9500$$

Goal: formally verified code for fairness tests

- Formally verified = checked by a machine proof checker
  - We use HOL4, an interactive theorem prover
- Verified math: concentration inequalities, e.g., Hoeffding's Inequality
  - Requires background theories: measure theory, probability, etc.
- Verified algorithm (for computing the desired inequality)
- Verified implementation of the algorithm
  - Must address lower level concerns
  - Notably floating point computation - which induces additional accuracy bounds

Extending Verified Mathematics

- Hoeffding's Inequality is well accepted and understood
- But people keep inventing new concentration inequalities
  - Their correctness may not be as trusted
- We may wish to bound the variance or other measures, not (just) the mean
- Proof about statistics necessitate measure theory
- Some measure theory results have yet to be shown in HOL4
  - Fubini's Theorem
  - Extensions from semi-algebras to measure spaces

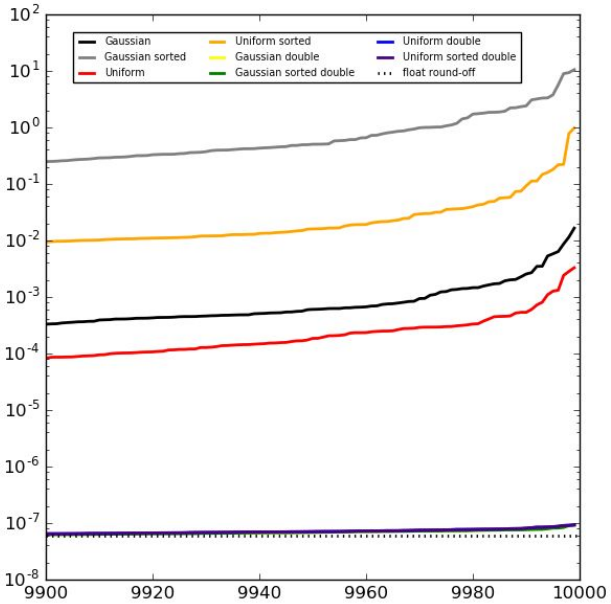
Proving algorithms that compute with random data

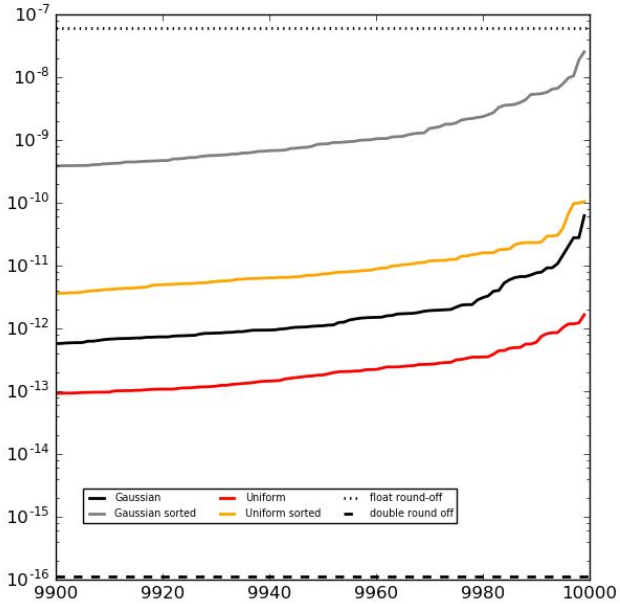
- Most program logics deal with definite data and deterministic computation
- Here, we need to say something about the err_i coming from a *distribution*
- The summation algorithm has deterministic *control flow*, but some algorithms may follow different paths depending on the nature of the data
    - These algorithms requires deeper probabilistic reasoning

More about floating point numbers

- “Poster child”: Summation on *n* floats
- Inaccuracies can accumulate over arithmetic operations
  - No upper bound to these inaccuracies
- We ran an experiment to assess potential error
  - Draw 1,000,000 floating point numbers from a distribution
  - Compute relative error of direct sum vs accurate sum
  - Total of 10,000 trials, show 100 worst trials (worst 1%)
- Distributions used
  - Uniform over most values (excluding extremes to prevent overflow)
  - Gaussian with $\mu = 0$ and $\sigma^2 = 10$
  - Both have sorted and unsorted variants

Why careful floating point algorithms matter





IEEE float and double summation of 1,000,000 random numbers, 10,000 trials, worst 1% of the trials.
Relative error of direct sum vs accurate sum. Black/gray drawn Gaussian ($\mu = 0$; $\sigma^2 = 10$); red/orange drawn uniformly over all floats, omitting top ¼ of exponent values (to avoid overflow). Black/red added in order drawn; gray/orange added from most negative to most positive. Blue/purple compare using doubles vs accurate floats. Dotted = float round off; dashed = double round off.

About accurate floating point computation

- Algorithms are known for computing this sum to the floating point value closest to the *actual* sum of the real numbers that the input floats represent
  - Accurate multiplication and dot product algorithms exist as well
- *But*, people don't tend to use them
  - They involve more instructions
  - They're tricky
  - (Maybe) a lot of people just say 'I'll compute with doubles and that will be good enough'
  - New IEEE instructions that return the rounded result and the exact error will help - but this is not yet approved, and will take a while to happen in practice
- However, we need a rigorous *proof* of accuracy!
  - We must use the accurate methods
- Floating point instructions *have* been formalized - but not the algorithms

End-to-end Verification: Top to Bottom

- The math
- The ideal algorithm (over the reals)
- The real algorithm (over the floats)
- The code produced by a compiler
  - For small-ish pieces of code this process is understood
  - For some languages, there are verified *compilers* (CakeML for ML, CompCert for C), so we can also just use them
  - We may wish to push toward GPU implementations

Why? For algorithms where we need to trust a statistical result produced with a computer, e.g., for legal or safety reasons.

Thanks to our team

Prof. Eliot Moss, UMass, Amherst, moss@cs.umass.edu

Dr. Michael Norrish, Australian Nat'l University and Data61/CSIRO, Michael.Norrish@data61.csiro.au

Jared Yeager, UMass, Amherst, jyeager@cs.umass.edu