GENERAL REPORT ON MACHINE LEARNING EXPERIMENTS FOR THE MÖBIUS FUNCTION

DAVID LOWRY-DUDA

LAST UPDATED: 2024.10.19

ABSTRACT. Last week, I was at the Mathematics and Machine Learning program at Harvard's Center of Mathematical Sciences and Applications. The underlying topic was on number theory and I've been studying various number theoretic problems from a machine learning perspective.

Initially, I sought to get Int2Int [Int2Int] to work. Then I set it on various examples. I describe some of them here.

CONTENTS

1. INTRODUCTION

I've been computing several experiments related to estimating the Mobius function $\mu(n)$. I don't expect $\mu(n)$ to be easily approximable; all earlier attempts to study μ using machine learning have resisted much success. This is related to *Mobius Randomness* (see for example [Sarnak]).

Previous machine learning experiments on studying $\mu(n)$ have used neural networks or classifiers. Francois Charton made an integer sequence to integer sequence transformer-based translator, Int2Int, and I thought it would be fun to see if this works any different. I'm splitting my description into two parts: a general report and a technical report [DLD-Technical]. This is the general report. The technical† report includes technical details for running or re-running Int2Int experiments and other programming-related aspects.

†By "technical" here, I mean *pertaining to technology* (i.e. to programming). Both notes are nonelementary. But I acknowledge that there are very few people who are experts in both number theory and machine learning.

2. MOBIUS FUNCTION

Recall that the mobius function $\mu(n)$ is 0 if the square of any prime divides n, and otherwise is $(-1)^{\omega}(n)$, where $\omega(n)$ is the number of prime divisors of n. For example, $\mu(1) = 1, \mu(2) = \mu(3) = -1, \mu(4) = 0, \mu(5) = -1, \mu(6) = 1$, and so on.

Int2Int takes as input a sequence of integers, and the output is a sequence of integers. I struggled to make sense of studying many outputs, but this is really my own problem.

2.1. **Inputs and Outputs for Möbius.** Int2Int takes sequences of integers as input and produces sequences of integers as output. I tried several variations to estimate $\mu(n)$, including

1. Input just n and output $\mu(n)$. (Or rather, make sure I can get Int2Int to process anything at all with the simplest possible example).
2. Input $n \bmod p$ and p for the first 100 primes.
3. Input $n \bmod p$ and p for the *second* 100 primes.
4. Input the Legendre symbol (n/p) for the first 100 primes.
5. Input n, $n \bmod p$, and (n/p) for the first 100 primes.

For each of these, I estimated $\mu(n)$, $\mu^2(n)$, and $\mu(n+1)$. The input n were sampled uniformly randomly from n between 2 and 10^{13} (with a few larger experiments here and there), using training sets between $2 \cdot 10^6$ for initial runs and $5 \cdot 10^7$ to investigate further. I also trained over n restricted to be squarefree.

3. Better than Random: squarefree detection

I quickly saw that Int2Int could guess $\mu(n)$ better than random guesses. But the reason *why* was because it was determining if n was squarefree or not with reasonable accuracy.†

†This is similar to a pattern observed by Jordan Ellenberg when attempting to train neural networks to estimate $\mu(n)$. The network seemed to figure out eventually that $4 \mid n \implies \mu(n) = 0$, and then sometime later that $9 \mid n$ also implies $\mu(n) = 0$. Presumably it would figure out other squares later, eventually.

The Int2Int models were determining whether n was squarefree or not with very high accuracy, and then guessing $\mu(n) = \pm 1$ randomly when it thought n was squarefree. Some of these models were guessing $\mu(n)$ correctly around 60 percent of the time: far better than chance.

Looking closer, the best-performing model (which also had the most data: $n, n \bmod p$, and (n/p) for the first 100 primes p) correctly recognized almost 92 percent of squareful numbers,† but only correctly recognized whether $\mu(n) = \pm 1$ about 40 percent of the time. Using that the density of squareful numbers is about 0.39, this gave the overall correctness at

†Be careful with what is the condition here. In particular it *doesn't* say that the model computes $\mu(n)$ correctly 92 percent of the time.

$$0.39 \cdot 0.92 + 0.61 \cdot 0.4 \approx 0.6,$$

recovering the approximately 60 percent overall correctness. The model tended to overestimate the number of squareful numbers and guessed that several squarefree numbers were squareful.

This occurred quickly when trained using quadratic residue symbols. I wasn't initially surprised by this because of course Legendre symbols include information about squares. Thus it should be possible to quickly train a network to recognize most squares given (n/p) for the first 100 primes (most numbers are divisible mostly by small primes, and hence checking small prime behavior usually suffices).

But here we're looking at numbers that are or are not squarefree: multiplying a square by a squarefree number mixes up all the quadratic residues and nonresidues.

With a bit more training, having only $n \bmod p$ for the first 100 primes produced very similar behavior. How was it doing this?

> This is an interesting **purely mathematical** question: how would you guess whether n is squarefree or not given $n \bmod p$ for lots of primes p?

One way would be to perform the Chinese remainder theorem, reconstruct n, and then actually check. Is the model recognizing something like this?

To test, I ran several experiments along the following lines:

1. Given (n mod p) for the first 100 primes, output if n is in the interval $[10^6, 2\cdot 10^6]$.
2. Given (n mod p) for the first 100 primes but *excluding* 7, output n mod 7.

These probe CRT-type knowledge. I sample input n uniformly at random from large intervals. The frequencies of each residue class should be approximately uniformly randomly distributed.

But the model never did better than random guessing on either of this type of experiment. I guess the model isn't recovering CRT-like information.

Remark 1. I'm also looking to determine behavior mod p^2 or p^3 using this type of transformer model. This is similar to CRT-like information, but slightly different. I'll talk about this later.

4. How to guess if $\mu(n) = 0$ given n mod p

After talking with Noam Elkies and Andrew Sutherland, I think I know how the model is guessing when $\mu(n) = 0$ with such high accuracy. The point is that numbers that are not squarefree are probably divisible by a small square and thus likely to be 0 mod a small prime. Numbers that *are* squarefree might be 0 mod a small prime, but not as often.

Let's look at this in greater detail.

The zeta function associated to squarefree numbers is

$$\zeta_{\mathrm{SF}}(s) = \prod_p \left(1 + \frac{1}{p^s}\right) = \zeta(s)/\zeta(2s).$$

Thus the ratio of numbers up to X that are squarefree is about

$$\mathrm{Res}_{s=1}\zeta(s)/\zeta(2s) = 1/\zeta(2) = \frac{6}{\pi^2} \approx 0.6079$$

The default algorithm to use would be to guess that every integer is squarefree: this is right just over 60 percent of the time. We need to do better than that.

The zeta function associated to *even* squarefree numbers is

$$\frac{1}{2^s}\prod_{\substack{p\\ p\neq 2}} \left(1 + \frac{1}{p^s}\right) = \frac{1}{2^s}\frac{\zeta^{(2)}(s)}{\zeta^{(2)}(2s)} = \frac{1}{2^s}\frac{(1-1/2^s)}{(1-1/4^s)}\frac{\zeta(s)}{\zeta(2s)}.$$

It follows that the ratio of numbers up to X that are even and squarefree is about

$$\frac{1}{2}\frac{1/2}{3/4}\frac{6}{\pi^2} = \frac{1}{3}\frac{6}{\pi^2}.$$

This implies that the remaining $\frac{2}{3}\frac{6}{\pi^2}X$ squarefree integers up to X are odd. We could see this directly by noting that the corresponding zeta function is

$$\prod_{\substack{p\\ p\neq 2}} \left(1 + \frac{1}{p^s}\right) = \frac{(1-1/2^s)}{(1-1/4^s)}\frac{\zeta(s)}{\zeta(2s)},$$

and computing the residue as $(1/2)/(3/4)\cdot\frac{6}{\pi^2} = \frac{2}{3}\frac{6}{\pi^2}$.

A squarefree integer is twice as likely to be odd as even.

For this classification problem, we're interested in the converse conditional: what is the probability that n is squarefree given that it is even (or odd)? Basic probability shows that

$$P(\text{sqfree}|\text{even}) = \frac{P(\text{even and sqfree})}{P(\text{even})} = \frac{\frac{1}{3}\frac{6}{\pi^2}}{\frac{1}{2}} \approx 0.4052$$

and

$$P(\text{sqfree}|\text{odd}) = \frac{P(\text{odd and sqfree})}{P(\text{odd})} = \frac{\frac{2}{3}\frac{6}{\pi^2}}{\frac{1}{2}} \approx 0.8105.$$

This already gives a better-than-naive strategy: if n is even, guess that it's not squarefree (correct about $1-0.4052 \approx 0.6$ of the time); if n is odd, then guess squarefree (correct about 0.8105 of the time). This should be correct about $0.5\cdot(1-0.4052)+0.5\cdot(0.8105) \approx 0.7$ (or actually $0.7026423\ldots$) of the time.

As $0.7 > 6/\pi^2$, this type of thinking is an improvement.

This readily generalizes to other primes. The Dirichlet series for squarefree numbers that are divisible by a fixed prime q is

$$\frac{1}{q^s}\prod_{\substack{p \\ p\neq q}}\left(1+\frac{1}{p^s}\right) = \frac{1}{q^s}\frac{(1-1/q^s)}{(1-1/q^{2s})}\frac{\zeta(s)}{\zeta(2s)},$$

and the series for squarefree numbers that aren't divisible by a fixed prime q is the same, but without q^{-s}. Thus the percentage of integers that are squarefree and divisible by q or not divisible by q are, respectively,

$$\frac{1}{q+1}\frac{6}{\pi^2} \quad \text{and} \quad \frac{q}{q+1}\frac{6}{\pi^2}.$$

Playing conditional probabilities as above shows that

$$P(\text{sqfree}|\text{q-even}) = \frac{P(\text{sqfree and q-even})}{P(\text{q-even})} = \frac{q}{q+1}\frac{6}{\pi^2}$$
$$P(\text{sqfree}|\text{q-odd}) = \frac{P(\text{sqfree and q-odd})}{P(\text{q-odd})} = \frac{q^2}{q^2-1}\frac{6}{\pi^2}.$$

I use the adhoc shorthand q-even to mean divisible by q, and q-odd to mean not divisible by q.

The differences are the largest when the prime q is small. A good strategy would then be to look at a couple of small primes q and then predict whether n is squarefree based on divisibility rules for the primes q.

I've ignored all the joint probabilities. These are explicitly computable by computing the local densities at the appropriate primes, as above. But the point is that divisibility by primes q correlates nontrivially with being squarefree, and this sort of table correlation is something that we should expect machine learning to figure out.

Explicit computation shows that using the first 20 primes and guessing `squarefree` or `not squarefree` based on which divisibility pattern of those primes is more common yields an overall correct rate of 70.3 percent, only 0.1 percent higher than using 2 alone.

We might hope that machine learning could learn to do better. Computing the table of cross correlations given sufficient data isn't hard. But ML models should also determine weights to use for better outcome prediction. Predicting what ML can or can't do is much harder.

5. Modified Experiments

Inspired by the above, I tried other experiments.

5.1. **Pure squarefree detection.** With the same inputs, I looked at guessing $\mu(n)^2$. That is, I tried to look just at the squarefree detection powers.

Overall, the models were correct about 70 percent of the time. This is consistent with the above behavior and with the heuristic that it could only use mod 2 information.

5.2. **Restricting to squarefree n.** In the other direction, I also restricted all inputs to squarefree n. This balances the expected outputs: about 50 percent each should correspond to -1 and about 50 percent should correspond to 1. Any prediction with accuracy greater than 50 percent would be a major achievement.

None of these models did any better than 50 percent consistently.

5.3. **Removing 2.** Still input n mod p for 100 primes, but use the 100 primes *after* 2. As we saw above, 2 has the most explanatory power using pure Bayesian probability. This asks: is the machine learning doing anything else *other* than the 2-based cross correlations described above?

In short, the performance plummeted to less than 50 percent accuracy for guessing $\mu(n)$. The performance was consistent with determining whether n was squarefree correctly about 60 percent of the time, and then guessing randomly between $+1$ and -1 when n was determined to be squarefree.

And this is consistent with using the pure Bayesian probabilistic approach on exactly the prime 3. Indeed, the probability that n is squarefree given that 3 divides n is $(3/4)\cdot 6/\pi^2 \approx 0.4559$, and the probability that n is squarefree given that 3 doesn't divide n is $(9/8)\cdot 6/\pi^2 \approx 0.6839$. Thus 1/3 of the time, we would guess "not squarefree" with accuracy $1-0.4559$ and the rest of the time we would guess "squarefree" with accuracy 0.6839, giving a total accuracy around

$$(1/3)\cdot(1-\tfrac{3}{4}6/\pi^2)+(2/3)\cdot\tfrac{9}{8}6/\pi^2 \approx 0.6372.$$

References

[DLD-Technical] David Lowry-Duda, *Technical Report on Machine Learning Experiments for the Möbius Function*. 2024 October 21. (Cited on page 1)

[Int2Int] *Int2Int* Github Repository, https://github.com/f-charton/Int2Int. Accesssed 2024 October 20. (Cited on page 1)

[Sarnak] Peter Sarnak, *Three lectures on Möbius randomness*, 2011. See https://publications.ias.edu/sites/default/files/MobiusFunctionsLectures(2).pdf. (Cited on page 1)