Qin and Ye *Journal of Big Data* (2024) 11:31
https://doi.org/10.1186/s40537-024-00889-7

Journal of Big Data

RESEARCH Open Access

Algorithms of the Möbius function by random forests and neural networks

Huan Qin[1†] and Yangbo Ye[2*†]

[†]Huan Qin and Yangbo Ye: These authors contributed equally to this work.

*Correspondence: yangbo-ye@uiowa.edu

[1] San Diego State University-Imperial Valley, 720 Heber Ave, Calexico 92231, CA, USA
[2] Department of Mathematics, The University of Iowa, Iowa City 52242, IA, USA

Abstract

The Möbius function $\mu(n)$ is known for containing limited information on the prime factorization of n. Its known algorithms, however, are all based on factorization and hence are exponentially slow on log n. Consequently, a faster algorithm of $\mu(n)$ could potentially lead to a fast algorithm of prime factorization which in turn would throw doubt upon the security of most public-key cryptosystems. This research introduces novel approaches to compute $\mu(n)$ using random forests and neural networks, harnessing the additive properties of $\mu(n)$. The machine learning models are trained on a substantial dataset with 317,284 observations (80%), comprising five feature variables, including values of n within the range of 4×10^9. We implement the Random Forest with Random Inputs (RFRI) and Feedforward Neural Network (FNN) architectures. The RFRI model achieves a predictive accuracy of 0.9493, a recall of 0.5865, and a precision of 0.6626. On the other hand, the FNN model attains a predictive accuracy of 0.7871, a recall of 0.9477, and a precision of 0.2784. These results strongly support the effectiveness and validity of the proposed algorithms.

Keywords: The Möbius function, The algorithm of the Möbius function, Machine learning, Random forests, Neural networks

Introduction

The era of big data has revolutionized various industries and enabled significant advancements in data science, leading to transformative applications in fields such as healthcare, finance, and artificial intelligence. However, along with the tremendous potential of big data comes the paramount concern for data security during transmission and within applications. Despite rapid progress in data science, the theoretical foundation of data security has been somewhat overlooked, leading to a gap between data science development and robust security infrastructure. To address this issue, our study aims to take the first step in proposing machine learning solutions to bolster our understanding of data security in the era of big data.

It is a common belief that the prime factorization of a large integer n cannot be computed in the polynomial time of log n. This belief forms the basis for the security of prominent cryptographic systems like the Rivest-Shamir-Adleman (RSA) public-key cryptosystem (Rivest et al. [1]) and now has become the security foundation of much of the internet communication and monetary transactions, online commerce, digital

financial instruments, and an important part of daily life. This so-called "not in class P" belief, however, has no theoretic or scientific evidence, while an increasing number of number theorists believe that the the opposite is actually true (cf. Sarnak [2]). It is our long-term goal to seek scientific evidence to support this disbelief.

The Möbius function is defined as

$$\begin{aligned} \mu(n) &= (-1)^k \quad \text{if } \mathrm{n} = \mathrm{p}_1 \cdots \mathrm{p}_\mathrm{k} \text{ for distinct primes } \mathrm{p}_1, \ldots, \mathrm{p}_\mathrm{k}; \\ &= 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise.} \end{aligned}$$

In other words, $\mu(n)$ vanishes if n is not square-free, i.e., if there is a prime p with p^2 dividing n. On the other hand for square-free n's, $\mu(n)$ provides the factorization parity of n. Consequently, the Möbius function $\mu(n)$ detects two features of n: (1) whether n is square-free, and (2) if n is square-free, what its factorization parity is.

Question (1) is well studied with a known algorithm without using factorization of n by Booker, Hiary, and Keating [3]. Their algorithm runs in a deterministic subexponential time on $\log n$. Although its computational speed is not the fastest among known algorithms, it is the first non-factorization algorithm of $\mu(n)$ when n is not square-free.

This paper attempts to answer Question (2). Since square-free moduli are used in the RSA cryptosystem, Question (2) may have a huge potential impact on cyberspace security. While $\mu(n)$ contains much less information than the prime factorization of n, all existing algorithms of $\mu(n)$ for square-free n still rely on factorization.

Finding properties, randomness, and/or faster algorithms of $\mu(n)$ are seemingly easier problems than finding a factoring algorithm in class P. In recent years, much theoretical efforts on $\mu(n)$ for square-free n's have be focused on the randomness of $\mu(n)$ and its dynamic behaviors. In this paper, we will take a different approach. While current theoretical approaches cannot produce new algorithms of $\mu(n)$ for square-free n's, we propose novel algorithms by machine learning.

Training machine learning models for $\mu(n)$ and other multiplicative number theoretic functions may take a long time and require large datasets, and hence may only be done for n's of a moderate size. We hope the machine learning models obtained in this paper, however, are "true" algorithms of $\mu(n)$ in the sense that they can be applied to much larger n's of the size currently used in the RSA cryptosystem, say about 10^{300}. It is in this sense that our machine learning models might provide efficient algorithms of $\mu(n)$. This is the work in progress of the authors which requires machine learning techniques for high precision integers.

The function values of $\mu(n)$ constitute an infinite dataset, presenting typical, if not more complex, challenges for big data analysis and algorithms. The current study also offers an illustrative approach to address the broader challenges posed by big data analysis and algorithms.

Related work

Mathematical research has evolved notably since the 1960s, with the integration of computers aiding in pattern discovery and conjecture development, crucial for establishing theorems. A prime example is the Birch and Swinnerton-Dyer conjecture, a Millennium Prize problem, highlighting this blend of mathematics and computational tools.

Although publications in this field are few, existing research showcases the innovation in this interdisciplinary domain.

Some research in various pure mathematical fields can be categorized as employing primitive machine learning models. Davies et al. [4] utilize fully connected feed-forward neural networks in Knot Theory to analyze hyperbolic knots, employing various datasets with a focus on predicting algebraic invariants. In the field of Algebraic Geometry, He, Hirst, and Peterken [5] apply deep neural networks to study dessins d'enfants, investigating their connections with modular subgroups and Seiberg-Witten curves. Bao et al. [6] employ machine learning techniques, such as neural networks and a Random Forest Classifier, in Geometry to analyze Hilbert series within the framework of quantum field theory, using datasets derived from geometric sources and the Graded Ring Database. Additionally, Bao et al. [7] implement multiplayer perceptions and convolutional neural networks in their Combinatorial Geometry study to predict the properties of lattice polytopes by analyzing data from 2D and 3D polygons. Moreover, He, Lee, and Oliver [8] use Bayesian classifiers in Number Theory to study the arithmetic of hyperelliptic curves, utilizing datasets comprising elliptic curves and Sato-Tate groups. Finally, Lample and Charton [9] demonstrate that neural networks, traditionally known for handling statistical or approximate problems, can excel in complex mathematical tasks like symbolic integration and solving differential equations, proposing a new syntax for mathematical problem representation and dataset generation methods.

Advanced machine learning models, particularly transformers [10], have shown impressive capabilities in mathematical applications. Notably, transformers excel in tasks like symbolic integration, solving differential equations, and cryptosystem attacks. Charton [11] explores the ability of small transformers to calculate the greatest common divisor of two positive integers, achieving 98% accuracy by optimizing training distribution and representation base. Additionally, Wenger, Chen, Charton, and Lauter [12] train transformers to perform modular arithmetic and combine them with statistical cryptanalysis to develop SALSA, a novel machine learning attack on cryptographic schemes based on the Learning with Errors problem.

Number theoretic background

The RSA public-key cryptosystem [1] uses two distinct primes p and q as private keys. The product $n = pq$ and an exponent $k \in \mathbb{Z}_+$ are made public. Anyone can encode a message a by computing $b \equiv a^k \pmod{\mathrm{n}}$, which can be done by the method of successive squaring in polynomial time of $\log n$. To decode b, a decoder needs to use p and q to compute the Euler ϕ-function $\phi(n) = (p-1)(q-1)$, solve the congruence $ku \equiv 1 \pmod{\phi(\mathrm{n})}$ by the Euclidean algorithm, and compute $b^u \pmod{\mathrm{n}}$ by successive squaring to recover the original message $a \equiv b^u \pmod{\mathrm{n}}$. All these steps are in polynomial time of $\log n$, except the factorization $n = pq$ if one does not know the private keys p and q. The belief that factorization cannot be done in class P makes the RSA crypto scheme secure.

Integer factorization is believed to be slow because all known algorithms use trial division to detect prime factors of n, or use trial checking to find enough suitable smooth numbers. The latter approach is called a sieve method. The most advanced sieve method is the general number field sieve (cf. Pomerance [13]) which can factor n in about

$$c_1 \exp\left(\left(\frac{64}{9}\log \mathrm{n}\right)^{\frac{1}{3}} (\log\log \mathrm{n})^{\frac{2}{3}}\right) \tag{1}$$

time, for some constant $c_1 > 0$, which is much slower than a polynomial time $c_2(\log n)^{c_3} = c_2 \exp\left(\mathrm{c}_3 \log\log \mathrm{n}\right)$ for some constants $c_2, c_3 > 0$. For comparison, a factorization method by trial division has a computational time around

$$c_4 \frac{n^{\frac{1}{2}}}{\log n} = c_4 \exp\left(\frac{1}{2}\log \mathrm{n} - \log\log \mathrm{n}\right)$$

for some constant $c_4 > 0$.

The squared Möbius function $\mu^2(n)$ detects whether n is square free or not. An algorithm of $\mu^2(n)$ based on factorization requires (1) of time. Booker, Hiary, and Keating [3] proposed an algorithm of $\mu^2(n)$ based on the random matrix theory with a time of $\exp\left((\log \mathrm{n})^{\frac{2}{3}+\varepsilon}\right)$ for any $\varepsilon > 0$. In Luo and Ye [14], additive relationships among values of $\mu^2(n)$ are explored based on work by Carlitz [15], Hall [16], Heath-Brown [17], Tsang [18, Theorem 1], etc. These additive relationships suggest the existence of additive and hopefully efficient algorithms for $\mu^2(n)$.

For square-free n's, additive relationships among values of $\mu(n)$ are identified in Luo and Ye [14]. More precisely, for finite X and $1 \le h \le 1,000$, [14] shows that the conditional expectation of $\mu(n+h)$ on $\mu(n) = 1$ for $1 \le n \le X$ is not equal to the conditional expectation of $\mu(n+h)$ on $\mu(n) = -1$ for $1 \le n \le X$. Note that Chowla's conjecture [19, (341)] (cf. also Matomäki, Radziwiłł, and Tao [20]) as modified for square-free integers, predicts that these two conditional expectations converge to each other when $X \to \infty$. Since for algorithms and computational complexity of $\mu(n)$ the integers n are always finite, the former case rules and hence serves as a theoretical foundation of the present study.

Random forests and neural networks

Decision trees and random forests

General principles

A great variety of random forest methods are currently used in supervised machine-learning classification problems. For the purpose of this study, we will implement the random forest algorithm proposed by Breiman [21]. The underlying principle of random forests is to aggregate a collection of random decision trees. First of all, to establish a classification model utilizing a decision tree algorithm, the set of all feasible values for feature variables is partitioned into distinct and non-overlapping regions. The prediction for a given observation can be made by identifying the class that occurs most frequently among the training observations in the region where it falls. The objective is to find these regions that minimize the error rate of classification regions that correspond to the fraction of the training observations that do not belong to the most commonly occurring class in that region. Decision trees can be easily explained, displayed graphically, and outperformed on cases assumed non-linear decision boundaries than some commonly used linear models like regressions. However, these tree models can be non-robust and suffer from high variances. One technique that is applied to overcome these disadvantages is through bootstrap by taking repeated samples from the training set, building a

separate prediction model using each sample, recording the class predicted by each tree, and taking a majority vote (James et al. [22]).

By introducing random perturbations to individual decision trees, the forest can extensively explore a broader spectrum of potential tree predictors, which, in practice, yields enhanced predictive performance. Specifically, the supervised classifier and random forests can be set up as follows ([21] and Genuer and Poggi [23]). Let $\mathcal{L}_n = \{(\mathbf{X_1}, Y_1), \ldots, (\mathbf{X_n}, Y_n)\}$ be a learning sample composed of n couples of independent and identically distributed observations, coming from the same common joint unknown distribution $(\mathbf{X}, Y)$. Assuming $\mathbf{X} \in \mathcal{X}$, a space of dimension p and $Y \in \mathcal{Y} = \{1, 2, \ldots, C\}$, the classifier $\hat{h} : \mathcal{X} \to \mathcal{Y}$ is a Borel measurable function which associates a prediction $\hat{y}$ of the response variable Y corresponding to any given input observation $x \in \mathcal{X}$. Let $(\hat{h}(., \Theta_1), \cdots, \hat{h}(, .\Theta_q))$ be a collection of classification trees, with $\Theta_1, \Theta_2, \ldots, \Theta_q$ be q independent and identically distributed random variables independent of the learning sample $\mathcal{L}_n$. The random forests predictor $\hat{h}_{RF}$ in classification is obtained by aggregating this collection of classification trees as the majority vote among individual trees, i.e.

$$\hat{h}_{RF}(x) = \arg\max_{1 \le c \le C} \sum_{l=1}^{q} 1_{\hat{h}(x, \Theta_l) = c}.$$

Random forest with random inputs (RFRI)

We implement RFRI to our target Möbius function, which exhibits two significant characteristics. Firstly, during the construction of each tree, a subset of *mtry* variables is randomly chosen at each node. This random selection is achieved by uniformly drawing mtry variables, without replacement, from the pool of p available input variables. Among these selected variables, the optimal split is determined by considering all possible splits. Secondly, the RFRI trees are not pruned. To summarize, the algorithm for random forests classification with RFRI trees can be outlined as follows:

1. Draw *ntree* bootstrap samples from the original data.
2. For each of the bootstrap samples, grow an unpruned classification tree in the following manner: at each node, randomly sample mtry of the predictors and choose the best split from among those variables.
3. Predict new data by aggregating the predictors of the ntree trees, which is obtaining the majority votes for classification.

There are two tuning parameters involved in building the RFRI and predicting values of the Möbius function. The first parameter, ntree, represents the number of trees in the model. A larger number of trees generally leads to better performance, and thus the value of ntree is selected based on the computational cost of the model. It is considered sufficiently large when further increases in the number of trees do not result in significant improvements in prediction accuracy [23]. The second parameter, mtry, determines the number of variables chosen at each node. The tuning process of mtry will be described in "Implementations and model validations" section. It will be shown that different values of mtry may lead to different prediction results.

Feedforward neural network (FNN)

The non-parametric nature of neural networks makes them attractive choices for learning tasks where the underlying functional form is unknown, such as in the case of the Möbius function. A typical neural network architecture consists of layers of elementary processing units called neurons, interconnected based on the specific type and purpose of the network. The first layer, known as the input layer, consists of neurons equal to the number of distinct features in the input data. Subsequently, the data passes through neurons of custom-sized hidden layers. Finally, the output layer, containing one neuron for each predicted feature, completes the network architecture. Upon entering the input layer of a neural network, data is transformed into output by propagating through neurons via connections (edges) between them. In FNNs, data propagates strictly towards the output layer, i.e. neurons output exclusively to neurons of subsequent layers.

Consider the a-th neuron in the L-th layer of a given neural network, which we will denote by n_a^L. Based on the topology and architecture of the neural network, n_a^L receives signals from m neurons and transmits signals to q other neurons. Then, the neuron n_a^L can be characterized by two parameters: an m-dimensional weight vector $(w_{1a}, w_{2a}, \ldots, w_{ma})^T$ and a bias term b_a. When n_a^L receives signals $x_1, x_2, \ldots, x_m$ from neurons $n_1, n_2 \ldots, n_m$ in the preceding layer, the signal o_a transmitted by n_a^L to each of the q output connections is calculated using the following formula (Warner and Misra [24]):

$$o_a = \sigma_L\Big(\sum_{i=1}^{m} w_{ia}x_i - b_a\Big),$$

where σ_L is an activation function specified for all neurons of layer L, and $\sum_{i=1}^{m} w_{ia}x_i - b_a$ is referred to as the input signal. This transformation is depicted in Fig. 1.

Application of machine learning to the Möbius function

The goal of this study is to find the Möbius function value $\mu(n)$, for a given large square-free number, n.

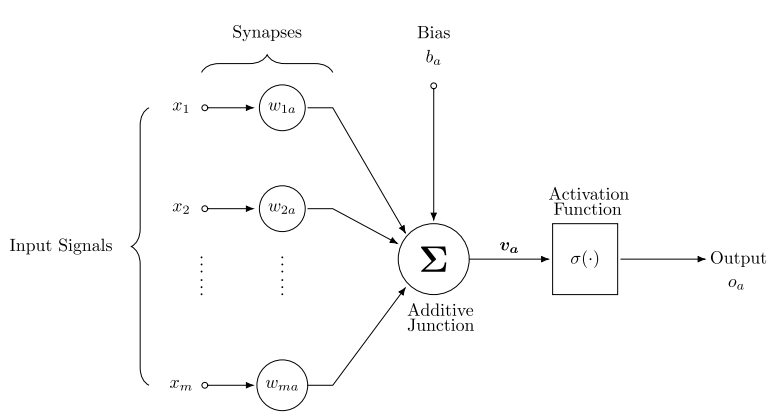

Fig. 1 Feedforward propagation in a single hidden layer neuron.

Construction of a database

Let P be the set of some primes $\leq X$ where X is a fractional power of n. Choose positive integers k and ℓ, and pairwise coprime positive integers $m_1, \ldots, m_\ell$. The database consists of records of $\ell + 2$ variables. The first variable is m which is either a prime in P, the product of two distinct primes in P, or the product of k distinct primes in P. Consequently, the number of records in this database is $\sum_{1 \leq j \leq k} \binom{|P|}{j}$. The second variable of a record is $\mu(m) = \pm 1$ which is known by the construction of m. The rest ℓ variables are remainders of m divided by some chosen integers m_ν, $\nu = 1, \ldots, \ell$.

Data balancing

The data will be imbalanced with a severe difference between the two classes, Class 1 which is the set of m with $\mu(m) = 1$ of $\sum_{1 \leq j \leq k/2} \binom{|P|}{2j}$ records and Class $-$ 1 which is the set of m with $\mu(m) = -1$ of $\sum_{1 \leq j \leq (k+1)/2} \binom{|P|}{2j-1}$ records. Synthetic Minority Oversampling Technique (SMOTE) is applied to address this imbalance (Chawla, Bowyer, Hall, and Kegelmeyer [25]). Instead of simply oversampling the minority class, SMOTE first selects examples from the minority class and finds a certain number of the nearest neighbors for an example in the $(\ell + 2)$-dimensional feature space. Then, a randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in the feature space.

Data splitting

In this study, a random selection of 80% of the data was utilized for training purposes, leaving the remaining 20% for testing. Consequently, the training dataset consists of 317,284 observations, while the test dataset contains 79,322 observations.

Implementations and model validations

RFRI classification

The experiment employs the RFRI classification model, which is implemented using the programing language R and the R library caret (Kuhnm [26]) and randomForest (Liaw and Wiener [27]). The dataset used in this study consists of 396,606 observations of six variables, with the response variable $\mu(n)$ generated by the algorithm described in "Construction of a database" section. A list of prime numbers up to 263 is obtained and used to compute the products of two, three, and four distinct primes, resulting in integer values that are included in the "n"column. The maximum value recorded in the "n"column is 4,088,647,181, which is the product of prime numbers 241, 251, 257, and 263. The choice of other features is based on several considerations, including relevance, computational efficiency, and training difficulty. The selected features include integer values n from the "n"column and the congruent values of n modulo 4, modulo 9, modulo 25, and modulo 49.

To prepare the data for analysis, the feature values are pre-processed using the standardization procedure, which involves subtracting the mean and dividing by the standard

deviation. To enhance the robustness and generalization performance of the model, the training set is subjected to three rounds of 10-fold cross-validation conducted on the training set with SMOTE resampling technique. This approach helps to address the issue of class imbalance and improve the model's ability to generalize to new data.

We use a tuning algorithm to aid in managing the training process and improving model outcomes. Recall that mtry refers to the number of variables that are randomly selected to be sampled at each split, while ntree pertains to the number of trees in the random forest model. In this experiment, ntree is set to be 500 and the highest training accuracy is achieved when the mtry is set to 3, see Fig. 2.

Single-layer FNN classification

To facilitate comparison and validation, we employed a FNN to analyze the identical training and test data sets utilized in our random forests experiment. The same SMOTE resampling technique and standardization procedures were utilized and three repeated 10-fold cross-validations were performed.

The FNN was developed using the R package nnet (Venables and Ripley [28]), incorporating a single hidden layer. The results from the repeated cross-validation showed that the model attained its peak ROC value of 0.92 when configured with a single hidden unit, as depicted in Fig. 3. The refined model comprises one hidden layer with a single unit, employing the Sigmoid activation function. The optimization process employed the Broyden-Fletcher-Goldfarb-Shannon (BFGS)method, coupled with a least-squares loss function, a full batch, and only one epoch. The weighted decay is 0.1.

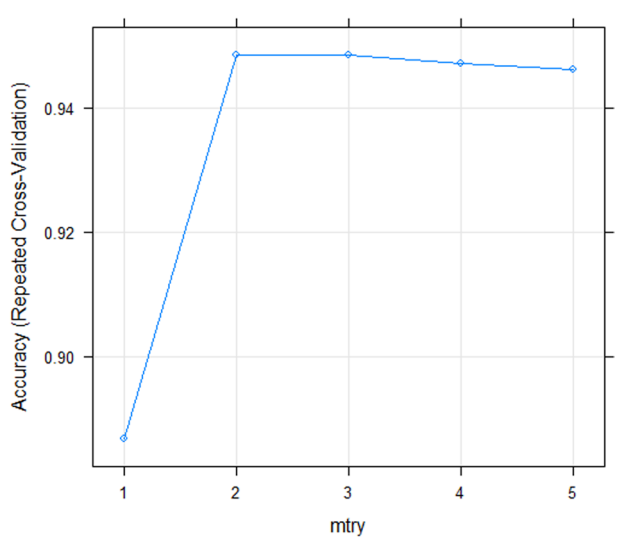

Fig. 2 Model Tuning Results for RFRI Model: Achieving Peak Accuracy of approximately 0.96 with mtry = 2 after Repeated Cross-Validation.

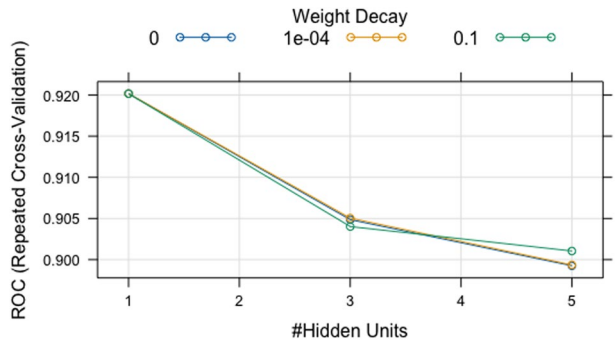

Fig. 3 Model Tuning for the Single-layer FNN: Receiver Operating Characteristic (ROC) Reaches Maximum Value with One Hidden Unit after Repeated Cross-Validation and Various Weight Decay Options.

Learning ideas

The database will be constructed in a way to avoid factorization. In particular, we will not seek to have records to cover an entire neighborhood of n. There will be smaller values of m in the database, which we believe is an important feature of the database, because these m's and their Möbius function values may provide easier hints for the model to learn.

Residue classes modulo $m_1, \ldots, m_\ell$ are incorporated in the database because values of the Möbius function have hidden additive properties as pointed out in Luo and Ye [14]. Values of the Möbius function are deterministic and are not random. For a given modulus m_i, the distribution of $\mu(m)$ on the arithmetic progression $m \equiv a \pmod{\mathrm{m_i}}$ are presumably different from the distribution of $\mu(m)$ on $m \equiv b \pmod{\mathrm{m_i}}$, when $a \not\equiv b \pmod{\mathrm{m_i}}$. This difference itself manifests an additive property of the Möbius function.

The multiplicative properties of $\mu(m)$ are easy to understand based on its multiplicative definition. Its additive properties seem to be complicated and beyond human comprehension so far. It is our hope that a machine learning model may discover some of these additive properties and use them to formulate a fast algorithm.

A central issue for an algorithm of $\mu(n)$ is its computational complexity, which has two stages, the training time complexity and the run-time complexity. For a random forests model, the run-time complexity is simply the depth of the trees. The training time complexity, on the other hand, is estimated to be $O(NDT \log N)$, where N is the number of points in the training set, D is the dimension of the data, and T is the number of decision trees (Kumar [29]). These complexity bounds will be used in dataset construction and model selection.

On the other hand, it is known (Kearns and Valiant [30]) that for a pseudorandom function $f(n)$, even if it is polynomial-time computable, there is no way to learn it from examples in polynomial time (cf. Arora and Barak [31, 9.5.5]). It is our surmise that the Möbius function is not pseudorandom.

The proposed algorithm is of course a probabilistic algorithm in the sense that its accuracy is based on the metrics to be discussed in "Metrics for prediction performance" section below. We hope that the accuracy may be improved by adjustments to the database structure and model parameters.

Table 1 Confusion matrix of the testing dataset

Reference		
Predicted	**Positive ("−1")**	**Negative ("1")**
Positive ("−1")	TP: 3311	FP: 1686
Negative ("1")	FN: 2334	TN: 71991

Table 2 Prediction performance metrics for two classifiers: RFRI and single-layer FNN

Metric	**Formula**	**Explanation**	**RFRI**	**FNN**
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	Percentage of correct classifications	0.9493	0.7871
TPR/Sensitivity/Recall	$\frac{TP}{TP+FN}$	Rate of correctly classified positives	0.5865	0.9477
FPR	$\frac{FP}{FP+TN}$	Rate of incorrectly classified positives	0.0229	0.2248
Precision	$\frac{TP}{TP+FP}$	Fraction of positive predictions thatwere actually positives	0.6626	0.2384
F_1-Score	$\frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$	Harmonic mean of the precision and recall	0.6223	0.3809

Metrics for prediction performance

Performance metrics for the RFRI classifier

To evaluate the performance of our models comprehensively, we considered essential metrics, including Accuracy, True Positive Rate (TPR), False Positive Rate (FPR), Precision, F_1-Score, ROC, and Area Under the ROC Curve (AUC).

For the RFRI classifier, Class − 1 was designated as the positive class. The Accuracy metric, which indicates the percentage of correct predictions out of the total number of predictions made, achieved an overall rate of 0.9493. This implies that 94.93% of instances in the test set were correctly classified by the model. However, it is important to note that the test set was imbalanced, with significantly more instances in Class 1 than in Class − 1 (see Table 1). In such cases, relying solely on Accuracy may not provide a complete picture of prediction performance. Classifiers that constantly predict the majority class could still achieve high Accuracy, even if their performance in the minority class is poor. Therefore, additional metrics like TPR, FPR, Precision, and F_1-Score are crucial, especially in imbalanced datasets. These metrics take into account the number of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN), providing a more nuanced evaluation of the classifier's performance. The definitions and corresponding results of these metrics for the RFRI model can be found in Table 2.

TPR, also known as Sensitivity or Recall, assesses how well a model can identify true positives. Specifically, it represents the percentage of actual Class − 1 instances that are correctly predicted by the model. In this case, the TPR of the classifier is 0.5865, meaning that 58.65% of the actual Class − 1 instances in the test set are correctly identified by the model. The TPR metric is particularly important in this case because correctly classifying Class − 1 instances is more crucial than correctly classifying Class 1 instances due to the imbalanced test dataset, where Class − 1 instances are the minority class.

Precision and F_1-Score are alternative metrics that can be used to evaluate the performance of a predictive model, especially in the context of class imbalance. Precision

measures the proportion of the TP among all positive predictions made by the model. In other words, it counts the percentage of positive predictions that are correct. In this experiment, when the trained model predicts Class − 1, it is correct 66.26% of the time. High Precision is desirable because it means that the model is highly accurate when predicting positive instances, even if it may miss some positive cases. For the high TPR model, it succeeds well in finding all the positive cases in the test dataset, even though it may also wrongly predict some negative cases as positive cases. Both high Precision and high TPR are preferred, but in reality, there is often a trade-off between them. Increasing one metric often results in a decrease in the other. Therefore, it is crucial to find a balance between Precision and TPR based on the specific requirements of the problem. The F_1-Score is computed by taking the harmonic mean of Precision and TPR. High values in F_1-Score are desirable since they indicate both high Precision and high TPR. In this case, the F_1-Score is 0.6223 (Fig. 4).

The ROC curve is a probability curve with a horizontal axis from 0 to 1 of the FPR, and a vertical axis from 0 to 1 of the TPR. A perfect classifier would have a TPR of 1 and an FPR of 0, implying it can correctly classify all positives and negatives. This ideal classifier would closely hug the upper left corner of the ROC curve. In contrast, a random classifier would have a diagonal line from the bottom-left to the top-right corner, indicating that it performs no better than random guessing (cf. Fawcett [32]). The AUC metric measures a binary classifier's ability to distinguish between positive and negative classes. A perfect classifier would have an AUC of 1, signifying complete separation of the two classes.

In our case, we observed a near-perfect ROC curve with an AUC very close to 1, despite some misclassification cases. The AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. It is important to note that the AUC only measures the ranking of probabilities and not the actual probability values themselves. As such, it is possible to have a perfect AUC even if the probabilities are poorly calibrated or biased. Additionally, using a decision threshold above 0.5 can result in a zero error rate for a perfect AUC score. However, selecting the optimal threshold may vary depending on the specific application and the relative costs of FP and FN errors (cf. [32]).

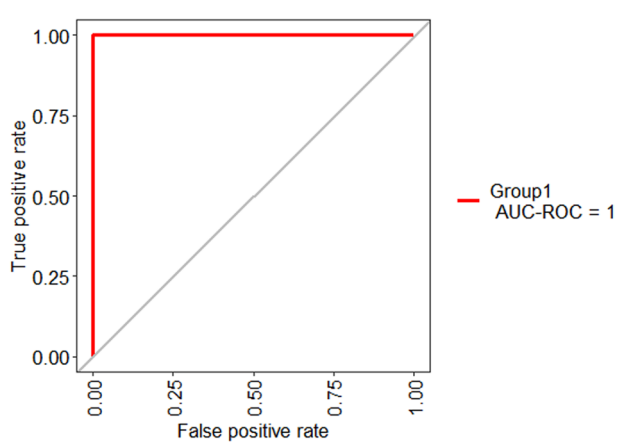

Fig. 4 ROC curve of the RFRI classifier

Performance metrics for the single-layer FNN

In terms of performance metrics (as shown in Table 2), the FNN model exhibits an overall Accuracy rate of 0.7871, indicating that it accurately classified 78.71% of the instances in the test set. The TPR (Sensitivity/Recal)l value is 0.9477, signifying that it correctly identified 94.77% of the actual Class − 1 instances in the test set. On the other hand, the FPR value is 0.2248 suggests that 22.48% of Class 1 instances were incorrectly classified as Class − 1. The Precision score of 0.2384 indicates that when the model predicts Class − 1, it is accurate 23.84% of the time. Comparatively, the F_1-Score of 0.3809 is much lower than that achieved by the RFRI model.

Discussion

Conclusion

The proposed algorithms utilize the neighboring values of the Möbius function to predict $\mu(n)$ based on the additive relationships discovered in Luo and Ye [14]. Instead of factorization, the algorithms generate these neighboring values by multiplying a set of primes. To encourage the learning process towards additive structures, the algorithms select congruent values of the neighboring integers modulo 4, 9, 25, and 49 as features.

The algorithms yield promising results with satisfactory performance metrics, indicating that the learning model has successfully uncovered hidden additive properties of the Möbius function. This outcome also suggests that a similar approach could be applied to other multiplicative number-theoretic functions and may pave a way towards developing efficient machine learning algorithms for integer factorization.

The novelty of this paper includes (1) application of machine learning technology to the study of the Möbius function, (2) machine learning models as algorithms of $\mu(n)$ with satisfactory performance metrics, (3) a route map towards efficient algorithms of $\mu(n)$, and (4) potential application to cyberspace security.

Limitation

The use of the R programing language imposed a constraint on the scale of this study. To extend the algorithms' applicability to larger integers, a different software solution may be necessary. Additionally, the choice of programing language limited the number of primes that could be used to generate neighboring integers through multiplication. Allowing for products of more primes would provide greater flexibility and reduce data imbalances.

The current study does not address the computational complexity and speed of the algorithms. Since this is the first of its kind, a significant amount of time was dedicated to model research and fine-tuning. However, we believe that with further development, the learning process can be formalized, and a model could be quickly constructed when presented with a large integer n.

Other than the constraints imposed by the programming language R, training of models for multiplicative number theoretic functions may be inherently complex computationally in time and memory. The present paper is an attempt to address this difficulty.

Future work

Our ultimate objective is to develop efficient machine learning algorithms for integer factorization. Initially, we experimented with deep learning using feedforward neural networks with multiple hidden layers but we did not incorporate the resampling and standardization processes. Regrettably, the performance results indicate that the predictions were essentially random, with a chance of only around 50% being classified as either positive or negative, even after fine-tuning. On the other hand, the implementation of a random forests model and a single-layer neural networks model exhibited significantly better predictive performance. In our future studies, we also intend to explore the potential of deep learning techniques to get other machine learning algorithms. We believe that combining deep learning techniques with approaches such as support vector machines or gradient boosting might yield better results for integer factorization. Additionally, we plan further refine the current algorithms for the Möbius function and focus on developing algorithms for other multiplicative number-theoretic functions.

The potential of our machine learning models lies on their possible ability to be applied to large integers well beyond the scope of the training datasets. This is work in progress of the authors and might have huge impact on cyberspace security.

Abbreviations

AUC	Area Under the ROC Curve
FN	False Negatives
FP	False Positives
FPR	False Positive Rate
RSA	Rivest-Shamir-Adleman
FNN	Feedforward Neural Network
RFRI	Random Forest with Random Inputs
ROC	Receiver Operating Characteristic
SMOTE	Synthetic Minority Oversampling Technique
TN	True Negatives
TP	True Positives
TPR	True Positive Rate
BFGS	Broyden-Fletcher-Goldfarb-Shannon

Acknowledgements
The authors would like to thank David E. Stewart for helpful discussions.

Author Contributions
All authors contributed equally to the study. All authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
The datasets generated and analyzed in this study may be available from the corresponding author upon request.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests in this study.

Received: 27 July 2023 Accepted: 28 January 2024
Published online: 21 February 2024

References

1. Rivest RL, Shamir A, Adlemany L. A method for obtaining digital signatures and public-key cryptography. Comm ACM. 1978;21(2):120–6.
2. Sarnak P. Three lectures on the Möbius function randomness and dynamics. Institute for Advanced Study. 2011. https://www.math.ias.edu/files/wam/2011/PSMobius.pdf. Last retrieved on February 4, 2024.
3. Booker AR, Hiary GA, Keating JP. Detecting squarefree numbers. Duke Math J. 2015;164(2):235–75.
4. Davies A, Veličković P, Buesing L, Blackwell S, Zheng D, Tomašev N, et al. Advancing mathematics by guiding human intuition with AI. Nature. 2021;600(7887):70–4.
5. He YH, Hirst E, Peterken T. Machine-learning dessins d'enfants: explorations via modular and Seiberg-Witten curves. J Phys A Math Theor. 2021;54(7): 075401.
6. Bao J, He YH, Hirst E, Hofscheier J, Kasprzyk A, Majumder S. Hilbert series, machine learning, and applications to physics. Phys Lett B. 2022;827: 136966.
7. Bao J, He YH, Hirst E, Hofscheier J, Kasprzyk A, Majumder S. Polytopes and machine learning. arXiv preprint. 2021. arXiv:2109.09602.
8. He YH, Lee KH, Oliver T. Machine-learning the Sato-Tate conjecture. J Sym Comput. 2022;111:61–72.
9. Lample G, Charton F. Deep learning for symbolic mathematics. arXiv preprint. 2019. arXiv:1912.01412.
10. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. Adv Neural Inform Process Syst. 2017;30.
11. Charton F. Can transformers learn the greatest common divisor? arXiv preprint. 2023. arXiv:2308.15594.
12. Wenger E, Chen M, Charton F, Lauter KE. Salsa: attacking lattice cryptography with transformers. Adv Neural Inform Process Syst. 2022;35:34981–94.
13. Pomerance C. A tale of two sieves. Not Amer Math Soc. 1996;43(12):1473–85.
14. Luo Q, Ye Y. Distribution of neighboring values of the Liouville and Möbius functions. arXiv preprint. 2024; arXiv:2401.18082.
15. Carlitz L. On a problem in additive arithmetic II. Quart J Math. 1932;3:273–90.
16. Hall RR. Square-free numbers on short intervals. Mathmatika. 1982;29(1):7–17.
17. Hearth-Brown DR. Square sieve and consecutive square-free numbers. Math Ann. 1984;266:251–9.
18. Tsang KM. The distribution of r-tuples of square-free numbers. Mathematika. 1985;32:265–75.
19. Chowla S. The Riemann Hypothesis and Hilbert's Tenth Problem. New York: Gordon and Breach; 1965.
20. Matomäki K, Radziwi M, Tao T. An averaged form of Chowla's conjecture. Alg Number Theor. 2015;9(9):2167–96.
21. Breiman L. Random forests. Mach Learn. 2001;45(1):5–32.
22. James G, Witten D, Hastie T, Tibshirani R. An introduction to statistical learning with application in R (springer texts in statistics). New York: Springer; 2017.
23. Genuer R, Poggi JM. Random Forests with R (Use R!). Cham: Springer Nature; 2020.
24. Warner B, Misra M. Understanding neural networks as statistical tools. Amer Stat. 1996;50(4):284–93.
25. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: synthetic minority over-sampling technique. J Artif Intell Res. 2002;16(1):321–57.
26. Kuhnm M. Building predictive model in R using the caret package. J Stat Softw. 2008;28(5):1–26. https://doi.org/10.18637/jss.v028.i05.
27. Liaw A, Wiener M. Classification and Regression by randomForest. R News. 2002;2(3):18–22.
28. Venables WN, Ripley BD. Modern applied statistics with S. New York: Springer; 2002.
29. Kumar P. Computational complexity of ML models. December 4, 2019 https://medium.com/analytics-vidhya/time-complexity-of-ml-models-4ec39fad2770. Last retrieved on February 4, 2024.
30. Kearns M, Valiant L. Cryptographic limitations on learning Booleann formulae and finite automata. J ACM. 1994;41(1):67–95.
31. Arora S, Barak B. Computational complexity a modern approach. New York: Cambridge University Press; 2009.
32. Fawcett T. An introduction to ROC analysis. Pattern Recognit Lett. 2006;27(8):861–74.

Publisher's Note