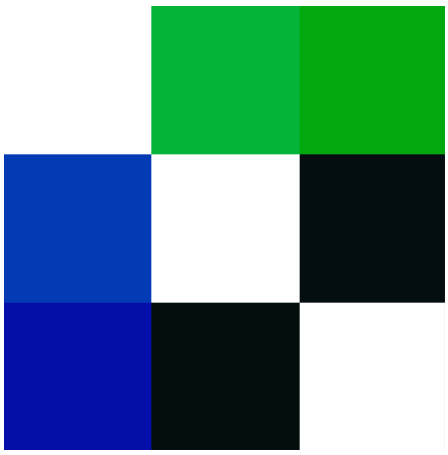
Black Tree

AutoML



NOTICES

SECURITIES: This booklet is ***not*** an offer, or a solicitation for an offer, to enter into any transaction. It is solely for informational purposes, only to describe a set of algorithms that implement machine learning and deep learning (the “algorithms”).

Vectorized Deep Learning

Charles Davi

March 14, 2022

Abstract

In a series of lemmas and corollaries, I proved that under certain reasonable assumptions, you can classify and cluster datasets with literally perfect accuracy (see Lemma 1.1). Of course, real world datasets dont perfectly conform to the assumptions, but my work nonetheless shows, that worst-case polynomial runtime algorithms can produce astonishingly high accuracies: This results in run-times that are simply incomparable to any other approach to A.I. of which I'm aware, with classifications at times taking seconds over datasets comprised of tens of millions of vectors, even when run on consumer devices. Below is a summary of the results of this model as applied to benchmark datasets, including UCI and MNIST datasets, as well as several novel datasets rooted in thermodynamics. All of the code necessary to follow along is linked to below.

1 Introduction

In a series of lemmas and corollaries (see, "Analyzing Dataset Consistency" [1]), I proved that given certain reasonable assumptions about a dataset, simple algorithms can classify and cluster with literally perfect accuracy (see, specifically, Lemmas 1.1 and 2.3 of [1]). Of course, real world datasets don't always conform to the assumptions, but my work nonetheless shows, that worst-case polynomial runtime algorithms can produce astonishingly high accuracies, as a general matter. This results in run-times that are simply incomparable to any other approach to deep learning of which I'm aware, with classifications at times taking seconds over datasets comprised of tens of millions of vectors, even when run on consumer devices. Below is a summary of the results of this model as applied to UCI and MNIST datasets, as well as several novel datasets rooted in thermodynamics. All of the code necessary to follow along is linked to below, together with links to a free version of commercial software I've written that implements this work, Black Tree AutoML.

For a mathematically rigorous, theoretical explanation, of why these algorithms work, see [1]. For an in-depth, practical explanation of how these algorithms work, including applications to other datasets, see "A New Model of Artificial Intelligence" [2].

1.1 Results

The results and runtimes in Table 1 below were generated using Black Tree's, "Supervised Delta Classification" algorithm, running on an iMac, with a 3.2 GHz Core i5 processor. Both a Free Version of Black Tree, capable of producing these results on any Mac with OS 10.10 or later, and Octave (which is also free) can be downloaded through the Black Tree Website.

Dataset	**No. Rows**	**Accuracy**	**Total Runtime (Seconds)**
UCI Credit	2,500	98.46%	217.23
UCI Ionosphere	351	100.0%	0.7551
UCI Iris	150	100.0%	0.2379
UCI Parksinsons	195	100.0%	0.3798
UCI Sonar	208	100.0%	0.5131
UCI Wine	178	100.0%	0.3100
UCI Abalone	2,500	100.0%	10.597
MNIST Fashion	2,500	100.0%	25.562
MRI Dataset	1,879	98.425%	36.754

As a general matter, my work seeks to make maximum use of data compression, and parallel computing, taking worst-case polynomial runtime algorithms,

producing, at times, best-case constant runtime algorithms, that also, at times, run on a small fraction of the input data. The net result is astonishingly accurate and efficient Deep Learning software, that is so simple and universal, it can run in a point-and-click GUI:

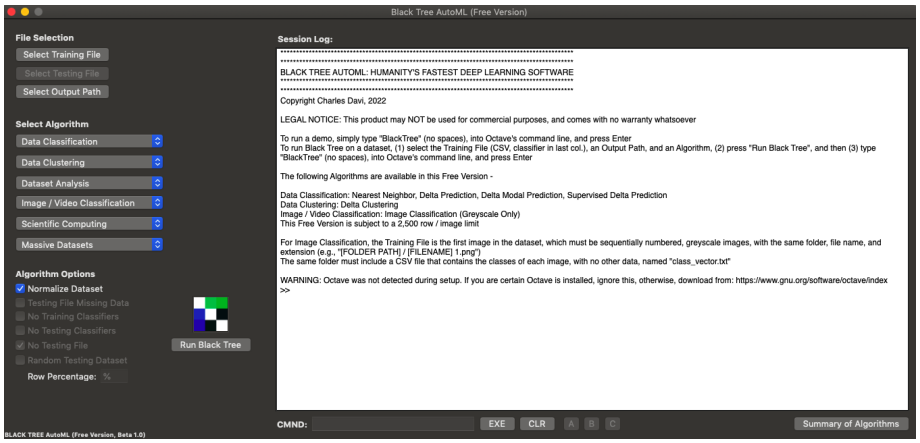


Figure 1: Black Tree's Graphical User Interface, available for free on the Black Tree Website

Even when running on consumer devices, Black Tree's runtimes are simply incomparable to typical Deep Learning techniques, such as Neural Networks, and Figure 2 below shows the runtimes (in seconds) of Black Tree's fully vectorized Delta Clustering algorithm, running on a MacBook Air 1.3 GHz Intel Core i5, as a function of the number of rows, given datasets with 10 columns (left) and 15 columns (right), respectively. In the worst case (i.e., with no parallel computing), Black Tree's algorithms are all polynomial in runtime as a function of the number of rows and columns.

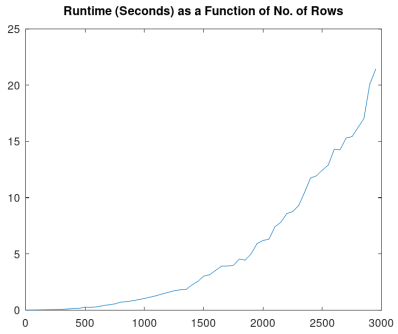


Figure 2: Runtime with 10 Columns

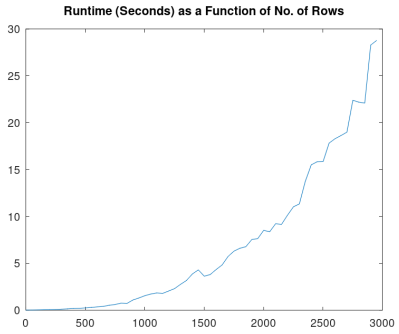


Figure 3: Runtime with 15 Columns

1.2 Spherical Clustering

Almost all of my classification algorithms first make use of clustering, and so I'll spend some time describing that process. My basic clustering method is in essence simple:

1. Fix a radius of delta (the calculation is described below);

2. Then, for each row of the dataset (treated as a vector), find all other rows (again, treated as vectors), that are within delta of that row (i.e., contained in the sphere of radius delta, with an origin of the vector in question).

This will generate a spherical cluster, for each row of the dataset, and therefore, a distribution of classes within each such cluster.

The iterative version of this method has a linear runtime as a function of $(M-1) \times N$, where M is the number of rows and N is the number of columns (note, we simply take the norm of the difference between a given vector, and all other vectors in the dataset). The fully vectorized version of this algorithm has a constant runtime, because all rows are independent of each other, and all columns are independent of each other, and you can, therefore, take the norm of the difference between a given row and all other rows, simultaneously. As a result, the parallel runtime is constant.

1.3 Calculating Delta

My simplest clustering methods use a supervised calculation of delta: simply increase delta some fixed number of times, beginning at delta equals zero, using a fixed increment, until you encounter your first error, which is defined by the cluster in question containing a vector that is not of the same class as the origin vector (see Section 2.2 below). This will of course produce clusters that contain a single class of data, though it could be the case, that you have a cluster of one for a given row (i.e., the cluster contains only the origin).

This requires running the Spherical Clustering algorithm above, some fixed number of K times, and then searching in order through the K results for the first error. And so for a given row, the complexity of this process is, in parallel, $O(K \times C + K) = O(C)$, again producing a constant runtime. Because all rows are independent of each other, we can run this process on each row simultaneously, and so you can cluster an entire dataset in constant time.

What turns these algorithms into a tool of basically all humanity, is that even consumer devices have some capacity for parallel computing, and languages like Matlab and Octave, are apparently capable of utilizing this, producing the astonishing runtimes above. These same processes run on a truly parallel machine

would reduce Deep Learning to something that can be accomplished in constant time, by basically anyone, using a GUI, as a general matter. The prospect of hardware designed for this specific purpose would democratize access to basically instantaneous medical diagnosis, credit decisions, and Deep Learning problem solving generally, on a mass scale. Again, the Free Version of Black Tree is already publicly available, and can process up to 2,500 rows of data (including gray scale images). The commercial versions will have no limits on the number of rows, and as described below, will be able to process roughly one-hundred-million rows on a consumer device.[1]

2 Application to Specific Datasets

2.1 Unsupervised Clustering

UCI Iris Dataset

- **Size**: 150×4.
- **Task**: Unsupervised Clustering.
- **Average Accuracy**: 94.168%.
- **Runtime**: 0.14465 seconds.[2]

The average accuracy reported above is the average accuracy across all clusters, and there is exactly one cluster generated for each row of the dataset. Note that the clusters are not mutually exclusive. For a given cluster, the accuracy is calculated by counting the number of classification errors in the cluster, and dividing by the number of rows in the cluster. This ratio is then subtracted from 1. The average number of elements per cluster is 13.053, the minimum number of elements is 0, and the maximum is 35. The minimum accuracy is 0%, and the maximum accuracy is 100%.

Summary of the Clustering Algorithm

This algorithm effectively answers the question of how different two points in a dataset need to be in order to be clustered separately. Using fully vectorized

[1] Note that my software may NOT be used for commercial purposes, unless you purchase a commercial license. Further, this article is subject to my Copyright Policy, available here. Though not a binding statement, in summary of the Copyright Policy, this paper is itself in the public domain, whereas I retain all rights to the underlying algorithms themselves.

[2] All runtimes referenced in Section 2 were generated on an iMac 3.2 GHz Intel Core i5.

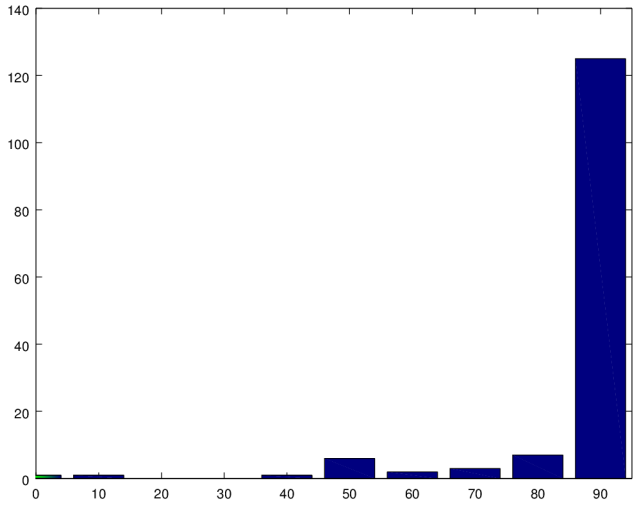


Figure 4: UCI Iris Dataset: The number of clusters with an accuracy of at least $x\%$.

processes, this algorithm calculates a single value, δ, that allows us to say, if the distance between any two vectors in the dataset x and y, exceeds δ, then they should be clustered separately. The runtime is, as far as I'm aware, unparalleled, though the accuracy is comparable to other deep learning algorithms. Note the algorithm is totally unsupervised, with no training data at all, and is not specialized in anyway, and can instead cluster any dataset of Euclidean vectors.

UCI Wine Dataset

This algorithm is the same as the one used for the Iris Dataset above, with an additional step that first normalizes the dataset.

- **Size**: 178×13.
- **Task**: Normalization; Unsupervised Clustering.
- **Average Accuracy**: 94.792%.
- **Runtime**: Normalization, 0.925937 seconds; Clustering, 0.0356669 seconds.

The average accuracy reported above is the average accuracy across all clusters, and there is exactly one cluster generated for each row of the dataset. Note

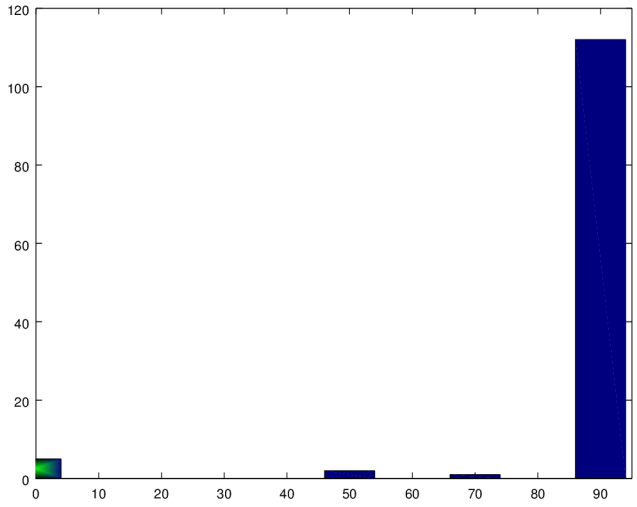


Figure 5: UCI Wine Dataset: The number of clusters with an accuracy of at least $x\%$.

that the clusters are not mutually exclusive. For a given cluster, the accuracy is calculated by counting the number of classification errors in the cluster, and dividing by the number of rows in the cluster. This ratio is then subtracted from 1. The average number of elements per cluster is 1.6067, the minimum number of elements is 0, and the maximum is 10. The minimum accuracy is 0%, and the maximum accuracy is 100%.

UCI Ionosphere Dataset

This algorithm is the same as the one used for the Iris and Wine Datasets above.

- **Size**: 351×34.
- **Task**: Unsupervised Clustering.

- **Average Accuracy**: 96.835%.
- **Runtime**: 0.14465 seconds.

The average accuracy reported above is the average accuracy across all clusters, and there is exactly one cluster generated for each row of the dataset. Note

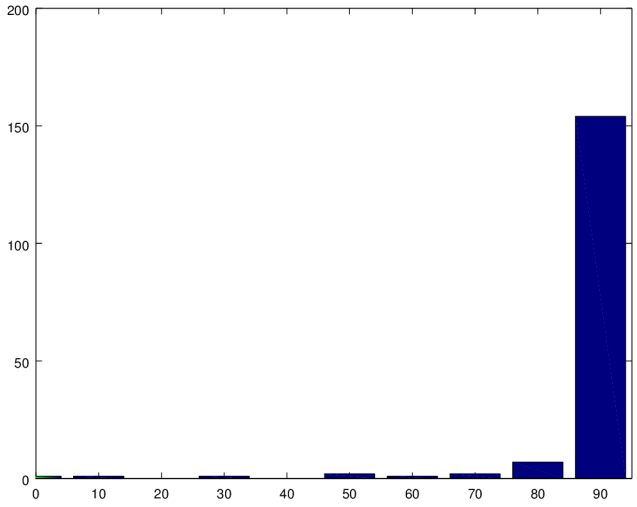


Figure 6: UCI Ionosphere Dataset: The number of clusters with an accuracy of at least $x\%$.

that the clusters are not mutually exclusive. For a given cluster, the accuracy is calculated by counting the number of classification errors in the cluster, and dividing by the number of rows in the cluster. This ratio is then subtracted from 1. The average number of elements per cluster is 6.6382, the minimum number of elements is 0, and the maximum is 48. The minimum accuracy is 0%, and the maximum accuracy is 100%.

2.2 Supervised Prediction

MNIST Numerical Dataset

- **Size**: Training Dataset, $5,000 \times 121$; Testing Dataset, $5,000 \times 121$.
- **Task**: Supervised Classification Prediction.
- **Accuracy**: 99.971%.
- **Runtime**: Training, 257.271 seconds; Prediction, 28.8227 seconds.

Summary of Supervised Prediction Algorithm

Prior to training and testing, $10,000$ images from the dataset are first loaded into memory, and then processed, generating two $5,000 \times 121$ datasets (i.e., the training and testing datasets). The runtimes listed above are the runtimes for only the training and prediction algorithms. The actual image processing algorithms that generate the datasets are explained in Section 1.3 below, which includes the applicable runtimes.

For each point in the dataset, the training step of this algorithm treats each point as the origin of a sphere. It then iterates through radii of increasing lengths, until it finds the longest radius, for which all points within the resultant sphere have the same classifier. As a consequence, if we go beyond that sphere, the classifiers of the points change. Said otherwise, this is the largest sphere for a given point in the dataset within which all points have the same classifier. This radius (i.e., the value δ discussed above) is calculated separately for each point (i.e., row) of the dataset, on a supervised basis.

Then, for the testing step, a partially vectorized implementation of the nearest neighbor algorithm is used to make predictions,[3] taking each row of the testing dataset as an input vector, and searching for the nearest neighbor of that input vector in the training dataset. If the nearest neighbor of testing row i is training row j, and the distance between the vectors in rows i and j exceeds the applicable value of δ, then the prediction is rejected, as beyond the scope of the training dataset. As a result, only predictions that are within the applicable value of δ are considered for purposes of calculating accuracy.

The accuracy reported above is calculated by counting (x) the total number of prediction errors, and dividing by (y) (a) the number of rows in the testing

[3]Using fully vectorized implementations will cause personal computers to run out of memory for a task with this many rows, though on a truly parallel machine, there is no reason to make use of anything less than full vectorization, which would improve runtimes.

dataset minus (b) the number of rejected predictions. This ratio is then subtracted from 1. This formulation of the divisor (y) is consistent with the idea that the number of rejected predictions (b) is not considered at all for purposes of calculating error, and should therefore be subtracted from the number of rows in the testing dataset (a). In this case, 30.080% of the training rows were rejected by the prediction algorithm.

2.3 Image Processing

I've also developed generalized image processing algorithms that allow any basic, single object image dataset, to be quickly and reliably transformed into a data structure, that can then be used for clustering and classification. Specifically, the algorithms generate a super-pixel representation of each image in the dataset, that can then be fed to a classifier or clustering algorithm. This is done by first processing a representative image from the dataset, and the information generated from the representative image is then used to process the remaining images in the dataset, at a much more efficient rate.

These algorithms would be particularly useful in real-time image classification problems, where a single object is presented to a camera, or otherwise fed to a computer for identification, and for whatever reason, the hardware is inexpensive or low-energy.

iPhone Photo Dataset

- **Task**: Initial Analysis (Single Image).
- **Original Image Size**: $3264 \times 2448 \times 3$ pixels.
- **Runtime**: 10.2044 seconds.

- **Task**: Process Dataset (30 images)
- **Original Image Size**: $3264 \times 2448 \times 3$ pixels.
- **Runtime**: 0.069453 seconds, on average, per image.

Below are three images related to the iPhone Picture Dataset, for context, which are simply photos I took of grocery items from Whole Foods, using my iPhone: The first is an image of a grapefruit, the second is the super-pixel image fed to the classifier algorithm, and the third shows the boundary data that generated the super-pixel image.

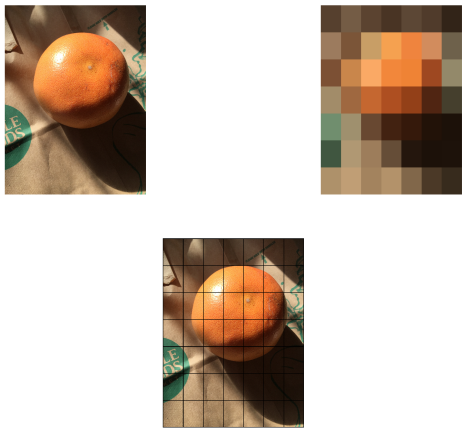


Figure 7: The original (left), the super-pixel image (right), and the boundaries (bottom).

MNIST Numerical Dataset

- **Task**: Initial Analysis (Single Image).
- **Original Image Size**: 28 × 28 pixels.
- **Runtime**: 0.304519 seconds.

- **Task**: Process Dataset (10,000 images)
- **Original Image Size**: 28 × 28 pixels.
- **Runtime**: 0.0041957 seconds, on average, per image.

MNIST Fashion Dataset

- **Task**: Initial Analysis (Single Image).
- **Original Image Size**: 28 × 28 pixels.
- **Runtime**: 0.16334 seconds.

- **Task**: Process Dataset (7,500 images)
- **Original Image Size**: 28 × 28 pixels.
- **Runtime**: 0.0036458 seconds, on average, per image.

2.4 Massive Datasets

The algorithms used in this section were developed to allow for deep learning techniques to be applied to thermodynamic systems, making use of both fully and partially vectorized algorithms. They are, however, generalized algorithms that likely have applications in other areas of study.

Two-State Gas Dataset

This dataset consists of two classes of collections of vectors:

(a) one representing the particles of a gas in a compressed volume, and

(b) another representing the particles of the gas in an expanded volume.

These two classes are intended to represent the two possible macrostates of the gas, compressed or expanded. Each class consists of 50 configurations, for a total of 100 configurations, intended to represent the microstates of the gas. Each configuration consists of $15,000$ vectors. The classification task is to cluster the 100 microstate configurations in a manner that is consistent with the two hidden macrostate classifiers, compressed or expanded.

The algorithm applied to this dataset also iterates through increasing levels of discernment, like the algorithms used in Section 1.1 above. However, this algorithm makes use of a vectorized operator that can quickly compare two large collections of vectors, as single operands, in turn allowing for the efficient comparison of microstates of complex systems.

- **Size**: $1,500,000 \times 3$.
- **Task**: Identify the macrostates of a gas.
- **Accuracy**: 100%.
- **Runtime**: 15.6721 seconds.

The accuracy is calculated by counting the number of classification errors, and dividing by the number of rows in the dataset. This ratio is then subtracted from 1.

Expanding Gas Dataset

This dataset consists of two classes of sequences:

(a) one representing the particles of a gas expanding at a slow rate, and

(b) another representing the particles of the gas expanding at a fast rate.

Each sequence of expansion consists of 15 observations, and there are 600 sequences. Each observation consists of $10,000$ vectors, representing the particles of the gas. The classification task is to cluster the 600 sequences in a manner that is consistent with the two hidden classifiers, slow or fast.

The algorithm applied to this dataset first compresses the dataset, by sorting and then embedding the dataset on the real number line. The sorting algorithm again makes use of a vectorized operator that can quickly compare two large collections of vectors. Then, a clustering algorithm similar to the one used in Section 1.1 above is applied to the embedded dataset. The bulk of the work done by the algorithm is sorting the dataset, ultimately allowing the dataset to be compressed from a $90,000,000 \times 3$ matrix, into a 600×15 matrix.

- **Size**: $90,000,000 \times 3$.
- **Task**: Identify the different rates at which a gas expands.
- **Accuracy**: 100%.
- **Runtime**: Sorting, 21.242 minutes; Embedding, 49.8727 seconds; Clustering, 0.0923202 seconds.

The accuracy is calculated by counting the number of classification errors, and dividing by the number of rows in the dataset. This ratio is then subtracted from 1.

Statistical Spheres Dataset

This dataset consists of some fixed number of K spheres in Euclidean 3-space, each of which consists of some number of points, producing shapes that are not solid, but nonetheless visually distinct. The classification task is to cluster the points in a manner that is consistent with the K hidden classifiers, representing the K distinct objects in the space.

The algorithm applied to this dataset also iterates through increasing levels of discernment, like the algorithms used in Section 1.1 above. However, this algorithm makes use of a different technique that can quickly cluster large collections of low-dimensional vectors.

- **Size**: $1,048,724 \times 3$.

- **Task**: Identify and cluster objects in Euclidean 3-Space.

- **Accuracy**: 100%.
- **Runtime**: 114.036 seconds.

The accuracy is calculated by counting the number of classification errors, and dividing by the number of rows in the dataset. This ratio is then subtracted from 1.

2.5 Other Algorithms

The balance of my work includes applications of these algorithms and others to image and video classifications, object tracking, shape classification, function approximation, as well as other algorithms specific to physics, including algorithms capable of estimating object velocities, and predicting projectile paths, given 3-D point data.

3 Financing Options

I would entertain financing for either -

1. An outright sale of my entire library of A.I. software, as it stands; or

2. Capital for a sales team and office space, to market the software, in exchange for equity.

In either case, I would likely insist on an opinion from counsel that the commercial distribution of the software does not violate all applicable laws, though I could be comfortable with a more reasoned opinion from competent counsel expert in the laws that relate to the distribution of software that could be used for military purposes.

Moreover, in either case, I would likely insist on the right to continue to develop my work, though I would entertain reasonable restrictions, such as a right of first refusal before any commercial offering, or perhaps developing my work outside of the public domain. In the latter case, I would likely insist on some form of explicit protection for the intellectual property I develop, since in that case, I would no longer be able to rely on copyright through publication.

4 About Me

I am a mathematician that worked in financial services for eight years, most recently at BlackRock, spending a significant portion of my free time conducting research in information theory. I spent the last five years conducting this research full-time, and the last two years coding and writing full-time.

In addition to my scientific writing, I've also published in The Atlantic,[4] and elsewhere, writing about banking, finance, and economics, which was widely cited by bank regulators, and other legal and financial professionals, including Judge Richard Posner.[5]

I'm a fairly prolific composer of both classical and contemporary music,[6] having studied piano and voice at Manhattan School of Music Prep; I was a professional audio engineer through an apprenticeship to a family friend; and I also wrote a loosely autobiographical epic poem during the Covid-19 quarantine in New York City.[7]

I received my J.D. from New York University School of Law, and my B.A. in Computer Science from Hunter College, City University of New York.

[4] My byline at *The Atlantic*.
[5] See page 50, footnote 5 of, "*The Crisis of Capitalist Democracy" (2011)*.
[6] A collection of my most recent recordings on *Soundcloud*.
[7] My book, *"Sketches of the Inchoate"*.