JULY 30, 2024

Improving the resilience of machine learning in financial systems through synthetic data

By Baxter Eaves

The stability of a financial system requires the ability to recognize and recover from catastrophic events quickly. To ensure the stability and reliability of data backups and their connected systems, we must be able to determine whether the data in these backups are consistent. This inference problem requires a model of why acceptable differences exist to detect when inconsistencies arise. Synthetic data that systematically generate acceptable and unacceptable inconsistencies can improve the financial system's resilience. This brief outlines an interpretable procedure using Bayesian probabilistic models to generate synthetic data. This approach allows one to develop machine learning tools to detect inconsistencies in federated backups. We show how this synthetic data approach can reveal the conditions under which a machine learning tool may fail and how we can exploit that information to build a more robust tool for detecting potential operation outages or cybersecurity threats.

INTRODUCTION

"Episodes of acute financial stress in 2008 and 2020 have exposed several major gaps and deficiencies in the range and quality of data available to financial regulators to identify emerging risks in the financial system."

FSOC 2021 annual report, section 5.5.5

The stability of financial systems requires the ability to recognize and recover from catastrophic events—operation outages, cyberattacks, etc.—quickly. Yet, significant financial system elements are federated: they do not have a unified representation of the current state. For example, foreign exchange trades 24 hours a day worldwide; trades and transactions are typically distributed geographically via brokers, and due to the temporal aspect of finalizing transactions and geographic constraints, data at different locations are typically different. With federated data, recognizing an adverse event cannot simply be looking for disagreements, and recovery requires inferring the most likely truth underlying the unaffected data.

Cyberattack damage is estimated to exceed $10.5 trillion annually by 2025, with cyber security expenditures increasing 12.4% per year[1]. The financial sector is a popular target for attack; these attacks on the financial sector scale, from small-scale jackpotting

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

attacks on ATMs, to far-reaching attacks such as the 2020 attack on SolarWinds, which impacted financial regulators. Machine learning (ML) systems are increasingly assessing stability and resilience in financial systems to detect events that could portend failures to combat increasing threats. For example, fraud and cyber threat detection are treated as classification problems (is this payment activity anomalous? is this access pattern malicious?); determining the mutual consistency of federated data can be formalized as a probabilistic inference about consistency (are multiple datasets statistically identical)[2]. ML systems deployed to monitor the consistency of various transaction streams may have been able to detect and attribute unusual order flow (such as generating and canceling large sell orders) like those that resulted in the 2010 flash crash.

ML systems require training data. To learn to classify outages, attacks, or other adverse events a detection system must have access to many examples, and any detection system is only as good as its training data. Though ML systems are finding broader use in financial systems via cyber security and fraud analysis, the performance characteristics, stability, and resilience of these ML systems are poorly understood.

A key bottleneck in ensuring the performance of ML systems is the availability of ample, high-quality training data. By definition, catastrophic events are rare. Because ML systems are optimized based on available training data, rarity is a significant challenge to ensuring the financial system's stability. Human experts can imagine other possibilities not yet present in the data, but human experts' time and knowledge are costly and ultimately limited. Existing synthetic data methods, such as deep neural networks, do not support systematic manipulation of causal factors that might give rise to inconsistencies or facilitate human understanding of any failure, which is critical to improving the system. The financial system's stability would be greatly improved by systematic methods for synthetically generating plausible failures, affordably and scalably, to understand limitations and improve the performance of ML systems.

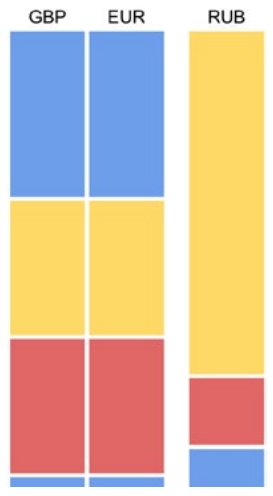
This brief describes a method for automatically generating ample, high-quality synthetic data for evaluating systemic events. We demonstrate the usefulness of this approach by using synthetic data to iteratively design, evaluate, and improve a machine learning system designed to detect inconsistencies in distributed backups of financial data. The approach is extensible to any domain where machine learning is trained on tabular data—the vast majority of applications in the financial system—and therefore represents a general approach to systematic evaluation and improvement of the resilience of ML systems deployed in finance.

PROBABILISTIC MODELS AND SYNTHETIC DATA

Financial data typically follow a tabular format, with rows and columns as one could represent in an Excel spreadsheet. For example, stock market data is often represented as a table in which the rows represent a timestamp and columns represent the high, low, open, and close prices for a fixed amount of time at that timestamp. There are several approaches to synthetic tabular data, the most popular being deep learning architectures (e.g., the Conditional Tabular Generative Adversarial Network (CTGAN)) and bespoke probabilistic model-based approaches. Deep learning is used in automated trading for text sentiment analysis, asset pricing, risk management, and trade execution and is used to predict risk in underwriting and lending[3]. Deep learning allows the user to drop a dataset and generate a black box model to generate synthetic data. The ease of use of the deep learning approach comes at a cost: the deep learning approach learns an arbitrary black box function that cannot be understood by users and thus cannot be manipulated in meaningful ways. While probabilistic approaches are generally interpretable and emit tunable models, users must motivate, design, and implement those models.

An ideal synthetic data approach allows for learning intuitive and tunable probabilistic models from arbitrary data, such as Probabilistic Cross-Categorization (PCC)[4]. PCC learns probability distributions over tables of data by clustering statistically dependent columns into views and, within each view, clustering similar rows into categories (see **Figure 1**). Views capture a generalized notion of correlation by finding

Figure 1. An example of structure learned under probabilistic cross-categorization



Notes: Hypothetical Forex example demonstrating a three-column PCC structure with two views. The leftmost view captures the British Pound (GBP) and the Euro (EUR) columns, which are correlated currencies, while the right view contains the uncorrelated Russian Ruble (RUB). Within the GBP-EUR view, PCC has captured several categories of activity, denoted by colored blocks. Within the RUB view, the categories represent different classes of activity over different time points.

Source: Author's creation.

columns whose rows follow similar categorizations. The categories within a view represent groups of rows that follow a similar probability distribution.

Monitoring which columns in the PCC table occupy the same views offers a fast way to assess the statistical structure of the data: which things predict or depend on which other things. Monitoring which rows within those views occupy the same categories provides a way to determine activity similarity at different time points.

To help us better understand, let us construct a hypothetical and unrealistically simple Foreign Exchange (Forex; FX) dataset. Our dataset contains a few years of minute-level price data (USD at close) for three currencies: the British Pound (GBP), the Euro (EUR), and the Russian Ruble (RUB). Each row in our PCC table represents the price of these currencies at a given timestamp. In this example (see Figure 1), PCC would cluster features describing correlated currencies into the same view; the Pound and the Euro would likely be in the same view, and likely not in the same view as the Ruble. Categories within these views may describe trading periods with similar behavior (e.g., various common trends), such as wild volatility around a central bank announcement. Our view containing the Ruble may have categories for the flat activity leading up to Russia's war against Ukraine, the sharp drop in the following weeks, the recovery between March and June 2022, and the high volatility period following.

PCC has features that make it more general than deep learning approaches. First, whereas deep learning requires all features (columns) to be continuous (or transformed into continuous data), PCC features can be modeled explicitly as continuous, categorical, or any type that a probabilistic model can explain. Second, whereas deep learning requires all entries in the data table to be occupied, PCC embraces missing data and can even explicitly model missing-not-at-random data, which is vital when the absence of data is informative about other quantities.

PCC creates a joint probability distribution of our dataset from which we can derive conditional distributions. For example, we can ask about the likelihood of an entire record (row in the data table) or we can ask about the likelihood of specific columns of a record given the values (observed or hypothetical) of other columns. Continuing with our Forex example, we can ask how likely the activity at a particular tick is across all currencies, or we may ask how likely the spread in the Singapore dollar is given the current spread of the Hong Kong Dollar and the British Pound. We ask the system to simulate these inferred distributions to generate synthetic data.

Standard synthetic data methods generate all data simultaneously from the learned data model. This all-or-nothing generation does not allow straightforward data manipulation for testing purposes. On the other hand, PCC gives us access to conditional distributions, allowing us to generate data given certain constraints imposed by other features. For example, we may wish to simulate the Euro given that the Pound is trading above a certain range or simulate S&P500 activity given the VIX has shown a sharp increase over the past five trading days. Conditional distributions are the primary mechanism by which we manipulate the synthetic data. By manipulating the conditions provided to the simulator along interpretable dimensions and evaluating machine learning performance on the resultant simulated data, we can determine under what conditions the machine learning system fails and succeeds and use that information to build more robust systems.

In the following sections, we will work through an in-depth example of how synthetic data may be used to identify failure modes and improve ML systems by designing, testing, and improving an ML system for detecting inconsistencies in distributed financial data systems.

INCONSISTENCY DETECTION

Now that we have described a framework for generating and manipulating synthetic data let us walk through an example of how we might apply synthetic data to improve an ML tool designed to maintain stability and resilience in a financial market. For example, we will develop an approach to detect inconsistencies in federated financial data backups and improve it via synthetic data.

We would like to maintain a backup of financial data, but storing all the data in one place makes our backup vulnerable to attack or failure. We can store the data on multiple distributed or federated backups to add robustness[5]. The issue is that the financial system moves so fast and geographically distributed that none of these backups will ever contain the same data. How, then, do we determine whether the data on these backups are *consistent*, that the data on all backups reflect the same state of affairs, or that one or more backups have experienced an error, failure, or attack?

Determining whether backups are consistent is determining whether the same causal model generated their data. If the same process generates the data stored in each backup, it stands to reason that probabilistic models trained on each backup should be similar. Similar models attribute similar likelihood to identical data, which means that models trained on consistent backups should attribute similar likelihood to any datum.

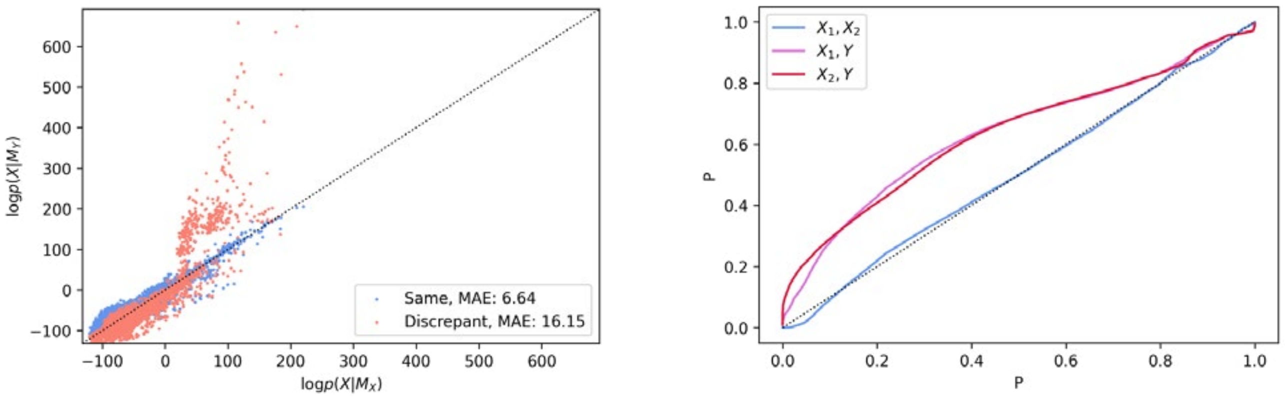
To assess consistency, we may train a probabilistic model, M, on each backup or dataset and directly evaluate the similarity between models using a standard mathematical measure of divergence, for example, Kullback–Leibler (KL) divergence or the more general Jensen-Shannon (JS) divergence. KL divergence can be interpreted as the expected difference in surprisal when interpreting data generated by a model M1 under a different model M2. Given a set of models, JS divergence is the average KL divergence between each model and the average (mixture) of the models.

KL and JS divergence are intractable to estimate for complex classes of models, but one may extract similar information by comparing the likelihoods of data under multiple models. If two models, M1 and M2 are identical, they will attribute the same likelihood to identical data, i.e., $p(x|M\_1) = p(x|M\_2)$ for all data x. We would not expect models trained on different consistent data sources to be completely identical due to noise in the data-generating process and noise in the model-fitting process. Still, they should be similar and thus produce similar likelihoods on an evaluation dataset.

The combined set of data across datasets can be used as the evaluation dataset. For example, computing the likelihoods of the evaluation data under two models yields a scatter plot (**Figure 2, left**) where identical likelihoods lie on a straight diagonal line; points farther from the diagonal have higher error. Or, rather than producing a scatter plot, we can create a set of univariate distributions over the likelihoods for each model. We can then compare these likelihoods using standard statistical tests (e.g., Kolmogorov–Smirnov; **Figure 2 , right**).

This framework reduces multiple datasets to a single consistency metric, which we will call the Offline Multiple Model (OMM) approach. Practically speaking, OMM fits a model on each backup at each evaluation. This algorithm's glaring limitation is that model fitting happens offline in batches while actual data arrive more or less continuously.

Figure 2. Inconsistency as the difference between likelihood distributions



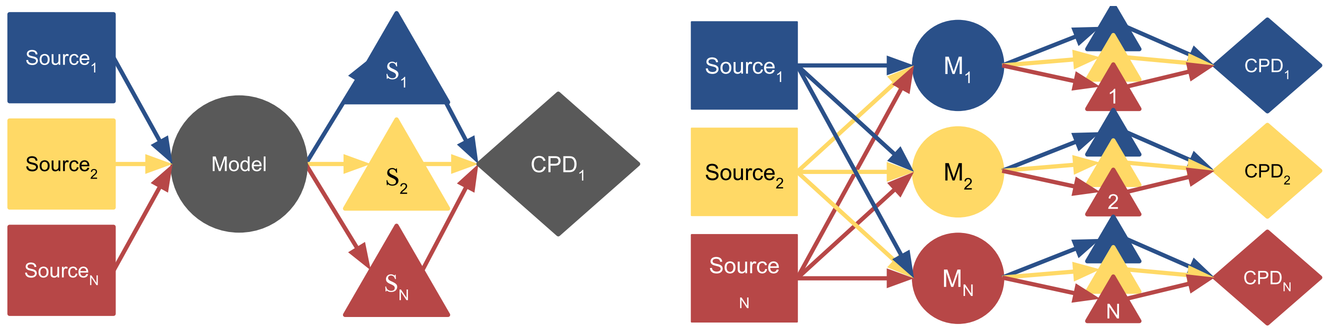
Notes: Inconsistency in practice. Left) Scatter plots show the relationship between the surprisal (negative log-likelihood) of data within a dataset under two models when the models are trained on consistent data (blue) and inconsistent data (orange). Note that in the inconsistent case, one model emits more high-surprisal values than the other, meaning it finds much of the data unlikely, hence inconsistent. Right) P-P plots show the error between models trained on consistent data (blue) and inconsistent data (red). We see that the inconsistent data deviated strongly from the main diagonal, a signal that can be used to identify inconsistency in practice.

Source: Author's creation.

Moreover, models require access to batches large enough to enable reliable statistical testing. Delays in evaluating inconsistency lead to delays in detecting and handling inconsistencies, during which time the backups may be unreliable, or some component of financial systems is behaving improperly.

To rectify this, let us propose a streaming algorithm based on the observation that identical data will have identical surprisal under a single model. Surprisal, sometimes referred to as self-information or information content, is a quantity from information theory that describes the level of surprise in a particular observation. It is defined as the negative log-likelihood of an observation *-log p(x)*. High surprisal indicates low likelihood. Surprisal does not have a standard scaling across all models, so it must be evaluated relative to a set of surprisal values. For example,

Figure 3. Two inconsistency detection architectures



Notes: Streaming inconsistency detection frameworks. Left) Streaming Single Model (SSM) approach. Each data source (backup; square) flows into a single, pre-trained model (black circle). The model outputs a stream of surprisals (triangles) attached to each data source, passing a changepoint detector (black diamond). Right) The Streaming Multi-Model approach (SMM). Multiple adapting models (circles) receive data from each data source but only adapt to data from their source.

Source: Author's creation.

we may consider a surprisal value in the 99th percentile anomalous.

Applying this to the inconsistency detector, if $x = y$, $-\log p(x|M) = -\log p(y|M)$ for all x, y, and importantly all models, M, that support the data. Thus, the distributions of surprisals should be the same as long as the data are consistent and come from the same generative model. Now, we can train one PCC model on a set of known-consistent data, compute the surprisal of data as they come in, and monitor the stream of surprisals for changepoints using a changepoint detection algorithm like Bayesian Online Changepoint Detection. This method is called the *Streaming Single Model* (SSM; **Figure 3, left**) approach. SSM facilitates near real-time detection of inconsistencies, but what does one lose by limiting the perspective of our approach to a single model?

EVALUATING AN INCONSISTENCY DETECTOR USING SYNTHETIC DATA

To evaluate an inconsistency detector, one could attempt to hunt down real-world datasets exhibiting certain inconsistencies or develop a simulator and manually program in certain failure modes. Certainly, creating story-like scenarios helps facilitate a better understanding, but neither of these approaches is fast or easy, and neither tests our tool against unknown unknowns. To overcome this limitation, one may alternatively deploy a procedure that automatically generates many statistical inconsistencies. After evaluation, one can use the data from the generator to determine under what statistical conditions the inconsistency detector fails. Here, we evaluate the SSM inconsistency detection approach defined above using synthetic Forex data.

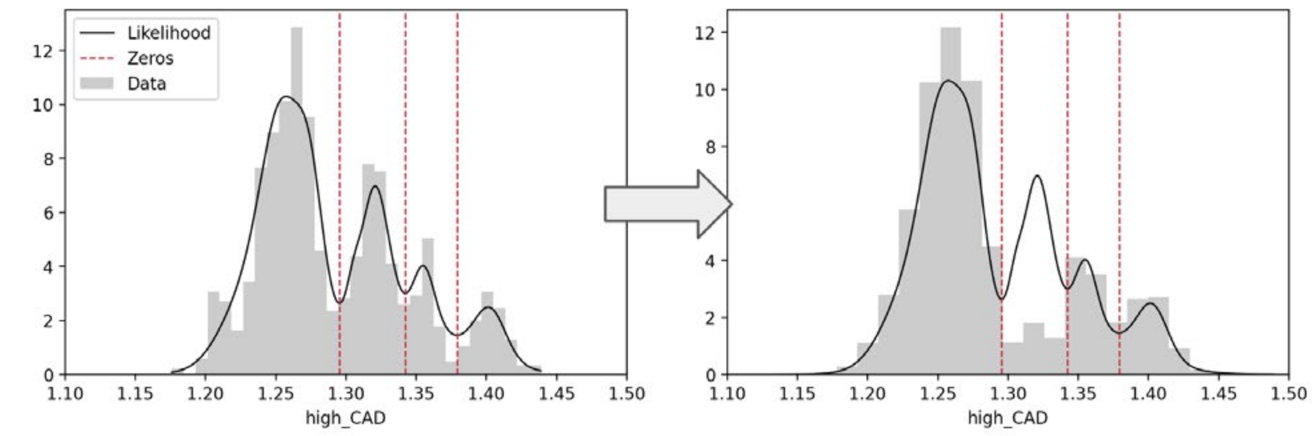
DATASET

Synthetic data generation requires a seed dataset. Following the previous Forex example, we choose minute-level Forex data (open, high, low, close) encompassing 29 major currencies for a total of 116 columns.

PCC is first trained on real data to generage the initial synthetic database. Once PCC has built a model of the data, the input data are dropped, and random synthetic data (which have the same distributional properties as the original data) are generated from the learned model. New ML systems are trained and evaluated only on the synthetic data. All data explored here are simulated.

AUTOMATICALLY GENERATING AND MANIPULATING SYNTHETIC DATA

Figure 4. Manipulating synthetic data by selectively removing probability mass



Notes: Synthetic data manipulation procedure. Left) Initial, consistent synthetic data. The troughs are found in the seed column likelihood (black), dividing the seed column into segments (red lines). Right) Synthetic data (gray histogram) in which the mass of an interior segment is reduced.

Source: Machine learning system outputs. Author's creation.

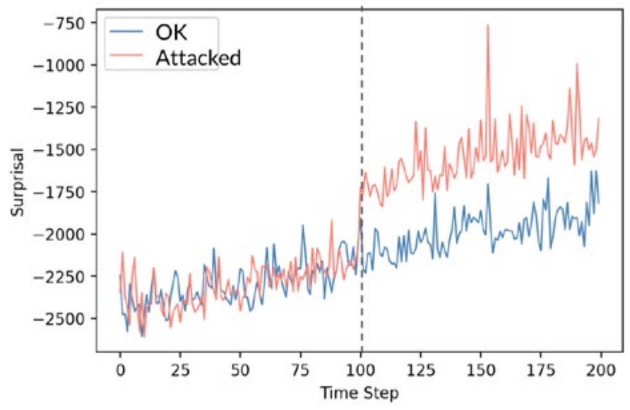
To automate synthetic data generation, we can deploy a simple method for tunably generating discrepant datasets. To do so, we first select a *seed* feature, x_s, and then find all the troughs in the distribution $p(x_s)$. Troughs separate modes in the distribution; modes correspond to different activity categories or latent causal factors (e.g., in price activity, a mode may be defined by the space between key support and resistance). We then segment the distribution by the troughs. For example, in a bi-modal distribution with one trough, there will be two segments containing the left and right tails of the distribution; for a tri-modal distribution, there would be a leftmost edge segment, an interior segment and a rightmost edge segment. To generate the discrepant column, $\hat{x}_S$, we remove mass from the segments.

For example, we may remove 0-100% of the mass from any segment (see **Figure 4**). We then simulate the remaining features conditioned on the discrepant feature distribution. This procedure gives us several statistical dimensions to manipulate that affect the synthetic data: the size, in probability mass, of the segment being manipulated; the proportion of that mass removed; whether the mass is from an edge or an interior segment; and the statistical importance of the seed column. The total amount of probability mass manipulated affects the frequency of manipulated data present in the discrepant dataset. Whether the manipulated segment is in the interior or lies on an edge determines the magnitude of surprisal: edge points are extreme values, which generally have higher surprisal (recall that higher surprisal indicates lower likelihood). The statistical importance of the seed column affects the magnitude of the effect manipulating the seed column has on the other columns. Manipulating a seed column that is statistically independent all other columns will affect only the synthetic data.

DIAGNOSIS

Running these experiments on synthetically generated data across all columns and segments revealed a weakness of the SSM approach. SSM has difficulty detecting inconsistencies derived from non-edge (internal) segments. As an example of an edge-affecting inconsistency, imagine an attack causing a particular asset's price volatility (variance) to increase at one broker. Increasing variance causes extreme values on both ends of the distribution, which causes higher surprisal (because extreme values are more surprising at baseline) and is easily detectable (see **Figure 5**). As an example of an internal-affecting inconsistency, imagine an attack that decreases volatility. The surprisals are now more likely to fall within the nominal range because the underlying data tends to be more nominal. Detecting that observations have become more extreme is far easier than detecting that they are unusually expected.

Figure 5. An attack that causes inconsistency causes surprisal streams to diverge



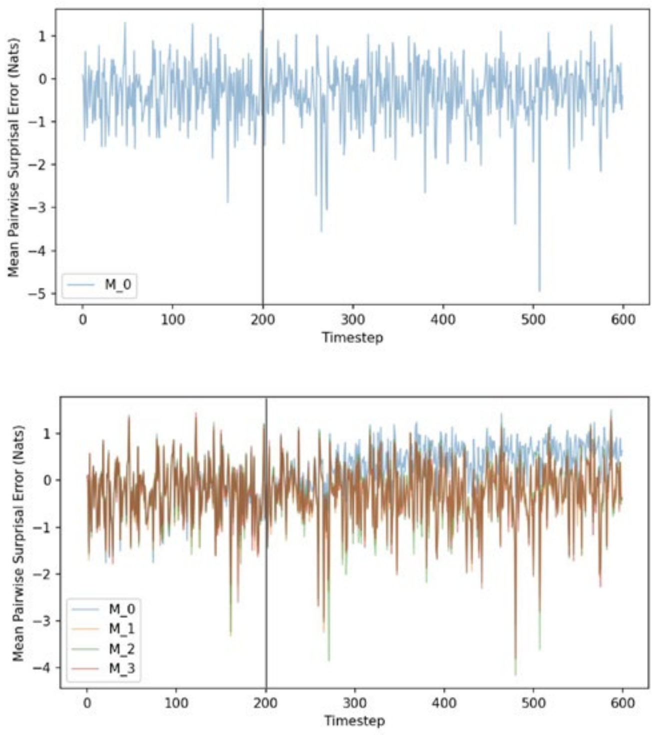
Notes: Surprisal streams from multiple consistent simulated data sources. An attack on the orange source occurs at step 100 (vertical line), which increases asset price volatility, which manifests as a noticeable increase in surprisal.

Source: Author's creation.

IMPROVING THE INCONSISTENCY DETECTOR WITH SYNTHETIC DATA

The synthetic data procedure described reveals failures in the SSM detection tool and has revealed under what conditions failures occur. We can now exploit this information to improve the inconsistency detector. Pulling from our volatility attack example, we would like to detect when the currency pair relationships are less volatile or more stable than usual. However, we are limited by the single perspective of the single-model approach described above. **Figure 6, top,** shows how the SSM approach fails to detect variance decreases. When evaluated on all data sources, a model trained on the narrower data would attribute higher surprisal to the normal data, making the attack stand out.

Figure 6. A multi-model approach facilitates detecting inconsistencies that decrease volatility



Notes: Performance of the streaming single model approach on simulated data (a) and the streaming multi-model approach (b) in detecting an inconsistency characterized by lower volatility in an asset price that occurs on step 200 (black line). On the top, we see that the single model approach stream does not appear to change after the inconsistency is introduced, but on the bottom, we see that the M0 stream (blue), which is stream of model connected to the attacked data source changes as the model adapts to a narrow range of values

Source: Author's creation.

One may achieve OMM's ability to detect model constriction in a streaming environment by using multiple models that update online. This new inconsistency detector will maintain a model trained on each data source, which will evaluate the surprisal of the data from multiple data sources (see Figure 3, left). This allows the surprisal of each data source to be evaluated from the perspective of each other data source.

Running the same experiments on this updated inconsistency detector indicates that the vulnerability has been closed: the online multiple-model approach can detect an inconsistent decrease in volatility. Figure 6 (bottom) shows that after the attack occurs, the model attached to that attacked data stream starts attributing higher surprisal to the consistent data in the stream. This approach results in a detector that identifies inconsistencies defined by unusually normal data by creating a model that is unexpectedly surprised by consistent data.

CONCLUSION

Training and validation are the core of developing machine learning tools, thus, machine learning tools are limited by the data they are trained and validated on. The financial system must be robust to unexpected and potentially catastrophic events, but unexpected events are rare, making real-world data hard to come by. People may be employed to create simulations of these events, but people are limited by time and imagination.

In this work, we explored an intuitive and general-purpose method using probabilistic modeling for generating synthetic data that can be manipulated along interpretable statistical dimensions. Using backup inconsistency detection as a testbed, we showed how validation led by such a synthetic data approach can help practitioners iteratively improve machine learning tools while simultaneously validating or invalidating their intuitions about the performance of their tools. The validation framework revealed flaws in the detector; it had difficulty detecting inconsistencies resulting in too much well-behaved data. This information revealed an opportunity to build a more robust inconsistency detector by combining multiple approaches. The resulting detector was greater than the sum of its parts: closing failure modes found in the speedier approach without sacrificing reaction time.

Financial systems are similar to other domains in that latent causal structure gives rise to observable data. Ensuring that financial systems are robust is a problem of identifying and simulating probable causes of instability, a natural application of the probabilistic approach we highlight.

ENDNOTES

1 McKinsey https://www.mckinsey.com/capabilities/risk-and-resilience/our-insights/cybersecurity/new-survey-reveals-2-trillion-dollar-market-opportunity-for-cybersecurity-technology-and-service-providers

2 Fraud detection encompasses detecting transaction fraud, identity theft, fake account generation, and other security risks such as phishing emails. Automated fraud detection has become so ubiquitous that cloud compute providers such as Amazon Web Services provide fraud detection examples as use cases for their hosted machine learning services. See https://aws.amazon.com/solutions/implementations/fraud-detection-using-machine-learning/.

3 For technical examples see https://arxiv.org/abs/1602.06561 and https://arxiv.org/abs/1810.03466

4 PCC learns a model of data by balancing the goals of fitting the data well and explaining the data simply with a model with few terms. This, along with PCC's ability to model missing data, and a variety of data types natively (instead of converting to and from real-valued vectors) give PCC a significant edge over existing methods in performance and usability.

5 All major cloud services offer distributed backups. Enabling distributed backups is often as simple as clicking a checkbox.