© andriano_cz | AdobeStock

Opening the Black Box: Defendants' Rights to Confront Forensic Software

Despite this country's commitment to fair and open trials, people are being convicted on the basis of secret computer code. When neither the public nor the accused is allowed to look at how the software operates, it undermines the legitimacy of the judicial system and can send innocent people to prison or to their execution.

Forensic software is used in the criminal justice context to make assertions about the presence and nature of DNA, to deploy police resources to certain areas, or to guide bail and sentencing determinations.

Software, however, is far from impartial or infallible. It is simply a set of instructions to a computer, programmed by fallible humans or trained on flawed historical data sets. Errors both intentional and unintentional are routinely discovered when independent experts are able to analyze these tools.

This article provides advice for understanding and confronting software-based evidence in criminal prosecutions. The advice falls primarily into two categories. First, from a computer science perspective, the article describes different types of review that attorneys and judges might seek in understanding software-based evidence. Second, from a legal perspective, the article explains why law and public policy require disclosure to the public and independent experts, such as those working with the defense, of the relevant software source code and other software development records, including any training data sets.

In particular, the article explains why courts must reject the idea that a vendor's purported commercial interest in trade secrets should override the rights of a defendant who is at risk of imprisonment or death, or the public's right to the open and fair administration of justice.

I. What Information Do Defense Experts Need to Evaluate Forensic Software?

A. Source Code and Executables: What Does It Mean to Evaluate Software?

Generally, software does what it is programmed to do, including any bugs and biases programmed into it by its creators. Everyone has experienced glitchy software, and everyone has been frustrated when software does not behave the way they expect it to or does not give them the options they need. Software often evolves over time, removing bugs and adding or

BY STEPHANIE J. LACAMBRA, JEANNA MATTHEWS, AND KIT WALSH

removing features in response to feedback from users or shareholders.

Forensic software is no different in its vulnerability to bugs or biases, but the stakes are much higher because the people most impacted by its operation have no power over the decision to use it and very limited opportunities to review and understand how it works. A wide variety of forensic software is deployed in the criminal justice system, including probabilistic DNA software like TrueAllele and STRmix, recidivism risk software like COMPAS, and predictive policing software like PredPol, Hunchlab, and Civicscape.

Defendants, defense teams, and the general public do not typically have access to the source code that defines these programs. They do not have information on how the software was constructed or the degree to which it is reliable. Independent, third-party review of the system is often prohibited by the vendor. Even requests for expert witnesses to review the details of the system under protective order have often been denied.

Because such denials tend to reflect a lack of understanding of the significance of independent expert analysis and public scrutiny, it is important to discuss different types of review that must take place to verify or discredit a software tool. This ranges from testing the "black box" software that is distributed to customers, to design and testing documentation, to documentation of errors reported and repaired, to source code level review and more.

Examining the normal user-facing *executable* version of software is one important level of review for expert witnesses evaluating the suitability of evidence produced by a software system. An executable is a series of ones and zeroes designed to be interpreted by a computer rather than read by a human. When a user runs an app, it "executes" that set of computer instructions and often generates graphics and an interface with which the user can interact. Figure 1 shows an example of the user interface presented by the executable for LRMix, one probabilistic genotyping software package.

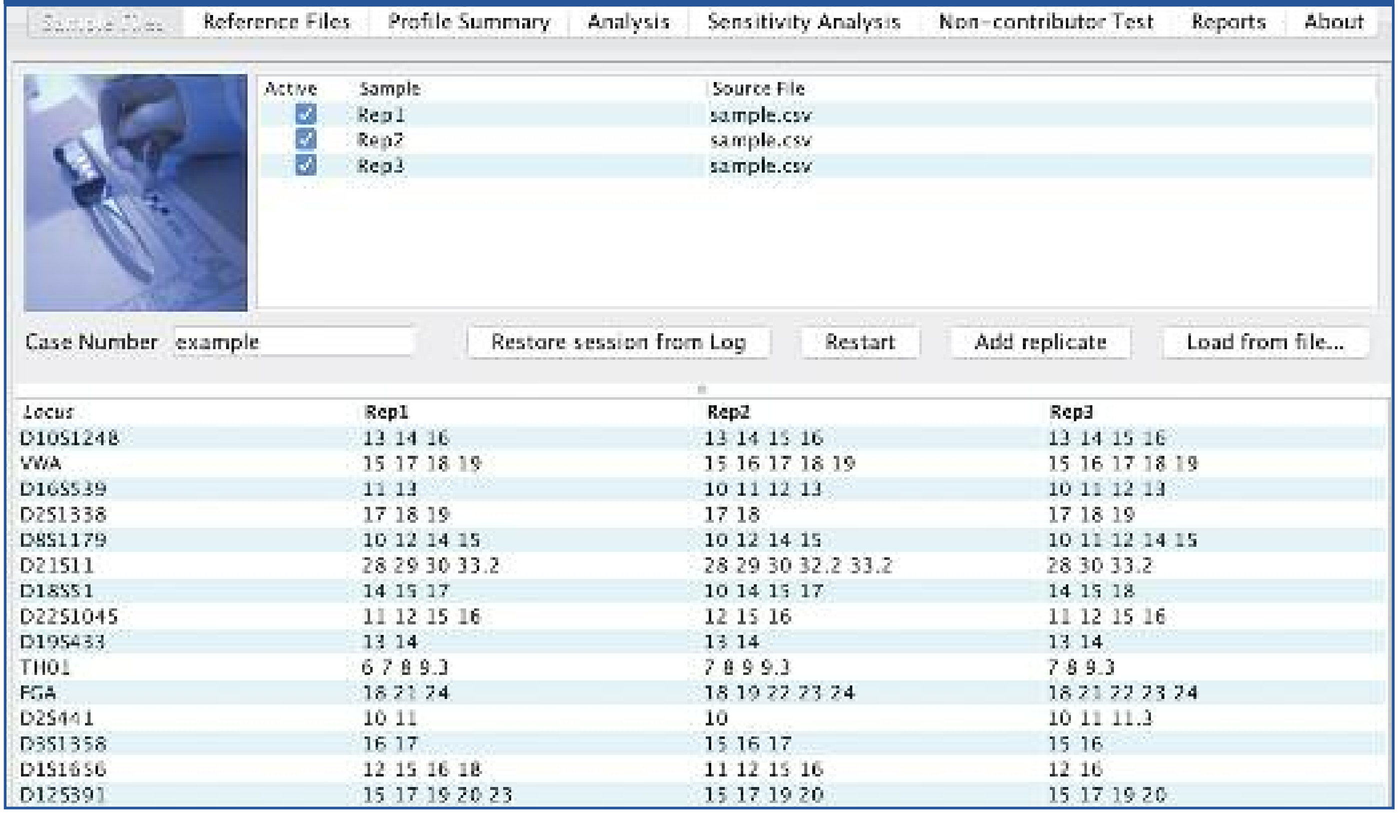
Directly engaging with the executable form of the software, like the one depicted in Figure 1, allows for a limited but important evaluation of the software. Experts can run the program using different inputs and different options provided by the program and observe how the results change, which may reveal important insights into any biases or assumptions within the program. They can run the software on test cases similar to the case at hand to see if the program reliably reports appropriate outputs.

"Source code" review provides a much more useful analysis than just engaging with the executable version of software. Source code is the human-readable list of commands (e.g., "if the user enters X, then do Y," "repeat trying possible solutions until one works") written by programmers and used to generate the executable version itself. It is like being able to see the blueprints for the house instead of just walking through it. You get to see the inner workings of how the house is put together and how different systems like plumbing, heating, and lighting function within the house's structure. Figure 2 shows an example of what source code might look like.

Because source code clearly defines the behavior of the software in all cases in a human-readable way, analyzing it provides more and better results than just running a finite number of test cases through the executable interface. Reviewing the source code can both explain why software is incorrect and reveal the presence of errors that an expert would have been unlikely to guess are present just from using the executable interface.

Reviewing source code is a careful and logical process, tracing multiple possible paths through the program to identify problems. It is a bit like a highly technical version of a "build your own adventure" book where the reader has to follow all possible outcomes, such as "If the girl knocks on the door of the scary house, turn to

Figure 1: Screenshots of the executable of LRMix, a probabilistic genotyping software system.



Active	Sample	Source File
✓	Rep1	sample.csv
✓	Rep2	sample.csv
✓	Rep3	sample.csv

Case Number: example

Locus	Rep1	Rep2	Rep3
D10S1248	13 14 16	13 14 15 16	13 14 15 16
VWA	15 17 18 19	15 16 17 18 19	15 16 17 18 19
D16S539	11 13	10 11 12 13	10 11 12 13
D2S1338	17 18 19	17 18	17 18 19
D8S1179	10 12 14 15	10 12 14 15	10 11 12 14 15
D21S11	28 29 30 33.2	28 29 30 32.2 33.2	28 30 33.2
D18S51	14 15 17	10 14 15 17	14 15 18
D22S1045	11 12 15 16	12 15 16	11 12 15 16
D19S433	13 14	13 14	13 14
TH01	6 7 8 9.3	7 8 9 9.3	7 8 9.3
FGA	18 21 24	18 19 22 23 24	18 21 22 23 24
D2S441	10 11	10	10 11 11.3
D3S1358	16 17	15 16 17	15 16
D1S1656	12 15 16 18	11 12 15 16	12 16
D12S391	15 17 19 20 23	15 17 19 20	15 17 19 20

page 57. But if the girl heads home without knocking, turn to page 234."

Figure 2 is an example from the FST source code where, if the sum of frequencies is greater than 0.97, a row in the raceTable is removed. An expert witness reviewing these lines of source code has the opportunity to validate and question the behavior of the software at a much deeper level than would be possible by simply using the program's executable version.

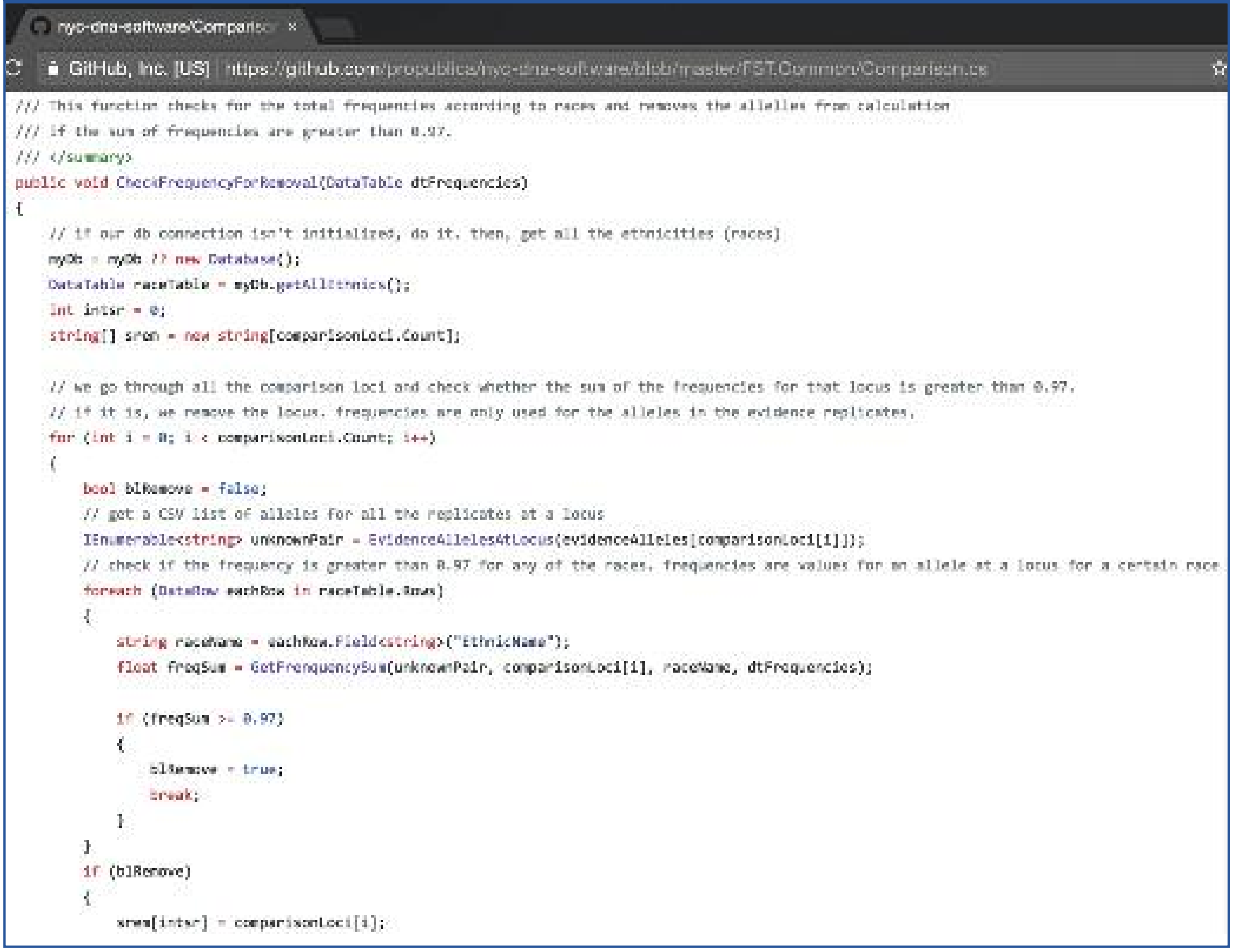
Software errors are extremely common. While most mistakes in software are caught before products are released, many are not. As software becomes ever more complex, and interacts with increasingly complex systems, errors become harder to prevent.[1] Some bugs are fairly easy to discover, such as when a bug causes a program to crash. But for other errors, the software will appear to function properly but will output incorrect results that may escape notice.

To take a famous and venerable example, the hole in the ozone layer went undiscovered for years because NASA's software was programmed to ignore outlier data that the original programmers had assumed was unrealistic.[2] A recent software error in Ireland's National Integrated Medical Imaging System "meant potentially thousands of patient records from MRIs, X-rays, CT scans, and ultrasounds were recorded incorrectly."[3] The error involved a misplaced less-than (<) symbol and may have led to thousands of unnecessary medical procedures. A large Australian bank recently admitted a software error had caused it to fail to report certain transactions for almost three years, leading to widespread money laundering.[4] In rare cases, software is intentionally deceptive, as when Volkswagen programmed its vehicles to cheat at emissions tests.[5]

Forensic technology is not immune to software errors, and increasing complexity carries a greater risk of error.[6] Independent public scrutiny and testing is the best way to discover and fix such problems.[7] In criminal cases, the Constitution guarantees defendants the right to do so. The Supreme Court said: "Confrontation is one means of assuring accurate forensic analysis. … Confrontation is designed to weed out not only the fraudulent analyst, but the incompetent one as well. Serious deficiencies have been found in the forensic evidence used in criminal trials."[8]

Ideally, law enforcement agencies that use forensic software should establish incentives and requirements for software vendors to be "open source" by publishing source code and making their design development information and data available for public review when they are considering software systems for procurement. Vendors could provide detailed information on the testing conducted, e.g., how extensive the testing was, on what populations the software was tested, and with what results. Vendors could make executable copies freely available to enable independent third-party testing. They could be required to

Figure 2: Screenshots of the source code of FST, a probabilistic genotyping software system.


nyc-dna-software/Comparis
GitHub, Inc. [US] https://github.com/propublica/nyc-dna-software/blob/master/FST.Common/Comparison.cs

```
/// This function checks for the total frequencies according to races and removes the alleles from calculation
/// if the sum of frequencies are greater than 0.97.
/// </summary>
public void CheckFrequencyForRemoval(DataTable dtFrequencies)
{
    // if our db connection isn't initialized, do it. then, get all the ethnicities (races)
    myDb = myDb ?? new Database();
    DataTable raceTable = myDb.getAllEthnics();
    int intsr = 0;
    string[] srem = new string[comparisonLoci.Count];

    // we go through all the comparison loci and check whether the sum of the frequencies for that locus is greater than 0.97.
    // if it is, we remove the locus. frequencies are only used for the alleles in the evidence replicates.
    for (int i = 0; i < comparisonLoci.Count; i++)
    {
        bool blRemove = false;
        // get a CSV list of alleles for all the replicates at a locus
        IEnumerable<string> unknownPair = EvidenceAllelesAtLocus(evidenceAlleles[comparisonLoci[i]]);
        // check if the frequency is greater than 0.97 for any of the races. frequencies are values for an allele at a locus for a certain race
        foreach (DataRow eachRow in raceTable.Rows)
        {
            string raceName = eachRow.Field<string>("EthnicName");
            float freqSum = GetFrenquencySum(unknownPair, comparisonLoci[i], raceName, dtFrequencies);

            if (freqSum >= 0.97)
            {
                blRemove = true;
                break;
            }
        }
        if (blRemove)
        {
            srem[intsr] = comparisonLoci[i];
```

disclose bugs reported against the software publicly and their response to those bugs. They could offer bug bounties for third-party groups that report problems that can be verified. In general, subjecting the software to more rigorous public review during the procurement phase would allow defense teams to focus their expert witness review resources not on general software quality but on the specifics of how the software applies to the individual case at hand.

B. Case Study: Probabilistic DNA Analysis Software

In recent years, software has been introduced to interpret DNA evidence that was previously considered too complex for manual analysis. Accordingly, it is not possible to manually verify the results generated by such software.

For example, "probabilistic genotyping" software is used when the evidence is a mixture of contributions from many people, when only trace amounts of evidence are collected, or when the evidence has been substantially degraded by environmental factors. Such software generates an estimate of the probability that the defendant contributed to the DNA sample, as opposed to some unknown person. Different methodologies may produce radically different results, even in the same case. In one such example, *Commonwealth v. Foley*,[9] three different government experts testified that the probability that another Caucasian could be the contributor instead of the defendant was 1 in 13,000; 1 in 23 million; or 1 in 189 billion. That is the difference between having a thousand other people in a city who might have contributed to the DNA instead of the accused, versus no one else on Earth.

Defendants are being sent to prison or to their death based on the results of such proprietary software,[10] even in the absence of other physical evidence, despite substantial, open questions about its accuracy and reliability. In 2016, the President's Council of Advisors on Science and Technology (PCAST) issued a report emphasizing the need for independent review of probabilistic DNA programs, in part to determine "whether the software correctly implements the methods" on which the analysis is based.[11]

Consider, for example, the case of Mayer Herskovic. He was convicted in 2013 for beating up a black student, Taj Patterson, in Brooklyn with a group of Hasidic men. Police recovered a DNA sample from Patterson's sneaker and forensic software developed in-house determined that it was more likely than not that the remainder belonged to Herskovic. Herskovic had no criminal record. He voluntarily provided a DNA sample against the advice of his lawyers believing that it would help him. There was no surveillance footage. The victim and those who witnessed the crime did not identify Herskovic at trial. It is reasonable to question whether the DNA testing software used is even accurate in distinguishing members of a genetically insulated population such as Hasidic Jews, who may be more likely to share common ancestors, and therefore DNA. Despite all this, Herskovic was convicted and his defense team had no access to the source code or development records of the software that generated the primary evidence against him.

In fact, source code review has routinely revealed critical flaws in DNA analysis software. A Forensic Statistical Tool (FST) was developed in 2010 by New York City's Office of the Chief Medical Examiner (OCME). For years, OCME fought any independent review of FST's source code and other software development materials, even under a protective order. In a 2016 criminal case,[12] a federal judge ordered OCME to turn over FST's source code for review.

In that initial review of FST source code, defense expert Nathan Adams found many problems. A secret function (see Figure 2) was present in the software, tending to overestimate the likelihood of guilt. The actual functioning of the software, revealed upon inspection of the source code, did not use the methodology publicly described in sworn testimony and peer-reviewed publications. Adam's review was conducted under protective order, but in late 2017, in response to filings by ProPublica and Yale's Media Freedom and Information Access Clinic, the judge unsealed these findings as well as the entire FST source code, which ProPublica then published online.[13]

The FST source code, now available online, contains over 30,000 lines of code, the equivalent of roughly 550 pages of dense technical information. In Adams' initial review, he could only scratch the surface of what is possible to determine with source code level review. Similarly, when STRmix, commercial software that is replacing FST in New York, was analyzed by independent researchers, they found programming errors that created false results in 60 out of 4,500 cases in Queensland, Australia.[14]

The problems caused by nondisclosure are especially acute in the context of the latest generation of probabilistic DNA analysis programs because there is no objective baseline truth against which they may be evaluated. These programs analyze samples that are thought to have DNA from multiple sources (such as a swab from a handbag or weapon)[15] according to the assumptions programmed into them, and then they produce a ratio of statistical probability of a match.

Because the output of newer DNA analysis tools depends on speculative assumptions,[16] different products like STRmix and TrueAllele provide drastically different estimates from one another — a discrepancy that can mean the difference between pointing toward guilt or innocence.[17] As a result, these programmed assumptions, and the way they are coded into the software, must be reviewed at the source code level for reliability and accuracy.

C. Beyond Code: Testing Materials, Revision History, and More

When programmers develop software, they do more than write the source code. They generate internal testing plans and record the results of those tests. Programmers also keep a database of bug reports and revisions.

The revision control history of a system reveals when code has been introduced to improve the system or fix reported bugs or problems. It is a well-known issue in software development that introducing a fix for one problem can introduce new problems. Experts for both the prosecution and the defense should be able to review the exact software version used for testing in a particular case and consider the relevance of flaws present in that version and how those flaws may or may not have impacted the results.

Testing plans and the results of internal testing would allow defense teams to investigate whether a particular system has been used successfully in cases similar to the case at hand. For example, if the case involved DNA from multiple contributors in the same family or DNA from a specific subpopulation, it would be appropriate to review the test results to see if the tool has proven effective in those particular scenarios. As one concrete example of accuracy that varies for different demographics, Buolamwini and Gebre evaluated three commercial gender classification systems and showed that while the error rates for lighter-skinned males was only 0.8 percent, the error rates for darker-skinned females was up to 34.7 percent.[18]

D. Unique Considerations for Machine Learning Systems

"Machine learning" is a technique whereby software developers do not write out all of the operations for their software, but instead enable the system to infer rules from a set of training data that has been tagged — by humans — with "correct" answers or classifications.

This approach relies on having a large body of data. Developers manually classify the data and then invite the machine learning algorithm to learn how to apply those classifiers. The machine learning algorithm looks for any patterns present in the training set that are effective at predicting past decisions. For example, a machine learning system might conclude from historical data that women are less likely to commit violent crimes.

The magic and the danger of these systems are that it is not obvious what criteria the program is looking at to make its determinations. As an example, researchers trained a machine learning classifier to differentiate between photos of dogs and wolves.[19] They took a training set of photos of dogs and wolves and labeled them correctly by hand. They used half of the photos to train the machine learning classifier. When tested on the other half of the photos, the system matched the manual classification with a high degree of accuracy. However, when the researchers configured the system to provide explanations of the decisions, they could see that the system decided that the best way to determine the difference between dogs and wolves was the presence of snow in the image. If there was snow in the image, then the system classified the image as a wolf and if not, the system classified the image as a dog.

The dog/wolf classifier illustrates three important points.

First, a machine learning tool is quick to leap to a proxy (like snow) that is easy to see, instead of making more difficult distinctions (between dog or wolf). In the context of the criminal justice system, this might mean using a zip code (which might be highly correlated with race or socio-economic status) as a proxy for the likelihood of recidivism.

Second, a machine learning tool trained on a particular data set might be completely useless as a tool to analyze new information. In the dog/wolf example, if one gave that system photographs of dogs and wolves against a blank background, it would have no predictive power at all because it was trained on an artifact of the data set (snow) and not on actual features of dogs and wolves.

Third, the use of biased training data is one way to introduce errors into software systems that might not be visible to experts even at the level of source code review. The researchers in this case skewed the decision of the system by using a training set in which wolves were photographed in the snow. Only with access to the training data does the problem become apparent.

E. What Information Is Needed to Evaluate a Machine Learning System?

When machine learning tools are used to make judgements about the legal rights of people, it is critical for the public and defense experts to have the ability to dig into the software's decision-making process and challenge its accuracy and fairness. One good place to start is providing information on the training data used to build systems of this kind.

Disclosing the specific data available to the decision-making program is important for transparency and accountability. As in the dog/wolf example, it can expose serious problems with extending the use of the tool to make judgments about data that was not part of the training set.

When the system is using protected attributes like race or gender, then it is especially important to understand the impact of that information on decisions. It is also important to remember that there are many proxies for protected attributes that may not seem particularly sensitive. For example, in some areas, zip code can be an effective proxy for race or socioeconomic status. The richer the data set used (social media posts, consumer reports, etc.), the easier it is to make decisions based on sensitive attributes even if the system can honestly claim not to be looking at them directly.

Machine learning algorithms are fundamentally based on the idea of using historical data to inform future decisions. This makes perfect sense because the past is all people have to learn from. However, it is important to remember that people do not always want the future to be like the past. The past is often unjust and flawed, and if the machine learning algorithm is trained on past information, it will only perpetuate the same injustices and flaws.

For example, as the Human Rights Data Analysis Group discovered in its experiment using PredPol predictive policing software fed with drug arrest history from Oakland, California, the algorithm only reinforced police bias in reporting when making its predictions. Due to inequalities in which neighborhoods were being policed in the first place, there was an inequality in the data being produced and input into the training set. Locations that were heavily patrolled by police, like lower income communities of color, were over-represented in the police report data that was being used to train the algorithm. As a result, the algorithm learned not about patterns in actual crime, but about patterns in how police record crime, and used these patterns to predict and deploy patrols to the same areas overrepresented in the police data — reinforcing and hiding biased police practices by using a supposedly "impartial" software program.

Programmers should not create software systems that perpetuate the past's mistakes into the future and then call them "logical" or "unbiased" decisions made by a computer. Programmers must understand the limits of the tools they build and not apply them outside the range of these limits. Allowing defense teams to examine and question the assumptions built into software tools used in the criminal justice system is an important step in the right direction.

People know what it feels like to use beta software and be frustrated by its bugs and inconsistencies. The process of reporting and fixing bugs improves software over time, but software manufacturers often resist such criticism when it impacts their sales. When a person is accused of committing a crime, the criminal justice system cannot allow bugs to go undiscovered or unremedied when they have the potential to adversely impact that person's life or liberty. Defendants should have the opportunity to challenge the evidence against them and meaningfully argue that a particular conclusion based on high-level historical statistics does not apply in their case.

Professionals in computing and software engineering have developed best practices for assessing the quality and reliability of software products. It is important that these standards be rigorously followed in developing forensic software. Manufacturers should be made to follow best practices for software development, including documentation of requirements, design, testing, version control, and error reporting. Also, there are best practices for algorithmic accountability and transparency. In 2017, the Association for Computing Machinery's U.S. and European policy arms published a joint statement on algorithmic transparency and accountability that included seven primary principles: (1) awareness, (2) access and redress, (3) accountability, (4) explanation, (5) data provenance, (6) auditability, and (7) validation and testing.[20]

F. What to Ask for in Discovery

The following outline lists types of software materials that defense teams should request, along with a brief definition and issues to consider. It is not an exhaustive list. If manufacturers of forensic software do not produce or maintain some of these materials, this suggests they are not following professional software development best practices, which would in itself be a substantial cause for concern about the quality and accuracy of the resulting system.

1. Executable Program: Working program as run by users of the system

a. Materiality/Usefulness:

i. Ability to find flaws in system through systematic testing

ii. Ability to identify sensitivity to parameters not well defined or routinely recorded when documenting results used for evidence

iii. Software manufacturers less protective of executable program

b. Challenges/Issues:

i. Even though made available to all users of the system, it can still be difficult for defense teams and independent experts to get access (expensive to buy, only available to law enforcement or other restrictive terms of use)

2. Design Documentation: Documentation describing the intended operation of the system from high-level requirements documents to low-level system plans

a. Materiality/Usefulness:

i. Can identify flaws in the underlying premise of system, not just errors in implementation

b. Challenges/Issues:

i. May not match actual working system

3. Test Plans and Results: Information about what testing the software developers did to validate the correct operation of the system

a. Materiality/Usefulness:

i. Opportunity to identify when the system has not been tested in particular cases or on particular demographics

ii. Opportunity to identify patterns of errors found and fixed

b. Challenges/Issues:

i. Testing conducted by the program development team is not independent testing because of the team's bias towards showing that the system does work rather than pointing out potential flaws

ii. Important to complement this with independent and adversarial testing

4. Bug reporting/ resolution database: Reports of errors discovered by end users or in testing along with responses (source of error, how and when fixed)

a. Materiality/Usefulness:

i. Provides information on the type and frequency of errors and how the software vendor has responded

ii. Patterns of past errors often suggest errors that still remain

iii. Opportunity to consider whether error identified in the past was present in the system when it was used to produce evidence in a specific case

b. Challenges/Issues:

i. Vendors may not maintain a formal bug reporting/ resolution database. Requiring that they do so in the procurement phase of software would be a good idea

5. Revision Control History: Detailed information about when and how the software has changed (e.g., when features are added or flaws found and fixed)

a. Materiality/Usefulness:

i. Provides insight into how the software has evolved

ii. Opportunity to ask whether evidence in a particular case would have been the same with previous or future versions

6. Training Data Used: For machine learning/expert systems, software may be trained using a specific set of training data

a. Materiality/Usefulness:

i. Opportunity to identify potential biases built into the system

ii. Opportunity to ask whether specific case at hand is well-covered by the training set

b. Challenges/Issues:

i. Training data often contains private information about individuals, and access to individually identifiable information must be carefully managed

7. Source Code: Human readable listing of all instructions followed

a. Materiality/Usefulness:

i. Step by step details of how the system works

ii. Opportunity to spot mistakes/errors

b. Challenges/Issues:

i. Some decisions made by the system may not be spelled out in source code (e.g., classifications made by machine learning classification)

ii. It is important to determine if the version of source code being reviewed is the same as that used to produce evidence (see revision control information)

iii. Software manufacturers are especially protective of source code and will likely try to assert a trade secret privilege. Many of the other artifacts suggested can be a good place to begin as well as provide unique information to complement source code access.

II. The Law Requires Disclosure of Forensic Software Code and Documentation

When the government chooses to use evidence created or manipulated by forensic software, it must disclose to the defense the software's source code and other software development records to comply with the Constitution's guarantee of fairness and due process and the strong public interest in overseeing the integrity of court proceedings. Yet the prosecution often seeks to hide such information by invoking a private party's commercial interest in keeping it secret.

At most, a company seeking to protect a trade secret could seek a protective order from the court ensuring that the information would not be used for competitive advantage or disclosed outside the defense team. Such an order may issue only if the company can show that its financial interest in keeping the source code secret somehow outweighs the interests of the public and the defendant, but this should be the rare exception and not the rule. Such disclosures are common in civil litigation where the stakes are merely economic and the chance of misappropriation is higher because the parties are direct competitors.

Criminal defendants deserve *better* access to relevant technical information about the evidence against them. Certainly they do not deserve *less* access than civil litigants.

A. Due Process Requires Disclosure of Forensic Program Source Code and Software Development Materials

Generally, U.S. criminal court proceedings are open to the public under both Supreme Court precedent and common law tradition.[21] The Constitution guarantees an accused the right to review and meaningfully confront the prosecution's evidence, and prohibits the prosecution from shifting its burden of proof to the defense.[22]

Due process entitles the defense to review the prosecution's evidence. Defendants have both a constitutional and statutory right to receive and review the evidence against them. When that evidence is the product of proprietary forensic software, the source code must be produced to the defense under both the Fourteenth Amendment guarantee of due process and the Sixth Amendment right to a fair trial and to "be informed of the nature and cause of the accusation; to be confronted with the witnesses against him; [and] to have compulsory process for obtaining witnesses in his favor."[23]

State evidence codes also typically require the prosecution to produce relevant evidence to the defense prior to trial. The default rule is production,[24] and commercial business interests in maintaining a purported trade secret cannot justify abridging these important constitutional and statutory rights.

Otherwise, when the prosecution's key forensic evidence is based on software code and program methodology that is hidden from the defense, the defendant's fate could be determined by a black box that the defense has no opportunity to examine or challenge. As discussed above, forensic software has no special immunity from the bugs and mistakes that plague software in other fields. Probabilistic DNA-matching programs are only one example of a forensic technology that embodies potentially flawed assumptions that could cause the wrong person to be imprisoned or executed.

Forensic software should be disclosed in criminal prosecutions and subjected to rigorous independent testing to discover any program flaws or bugs that can be exploited. Without access to the program's source code, methodology, design documents, testing plans, training data and bug reports, there is no way for the defense to meaningfully confront the assumptions embedded within the program or the accuracy, functionality, and credibility of its outcomes.

Exclusion is the only appropriate remedy for nondisclosure. The justice system cannot tolerate convictions based on secret evidence.[25] To do so would undermine the common law right to access criminal proceedings[26] and the constitutional rights to due process and a fair trial.[27]

Due process prohibits burden shifting to the defense. Due process dictates that "a State must prove every ingredient of an offense beyond a reasonable doubt, and … may not shift the burden of proof to the defendant. …"[28] At the same time, "a presumption which, although not conclusive, [has] the effect of shifting the burden of persuasion to the defendant," is likewise unconstitutional.[29]

Thus, any framework that requires the defense to make a showing of materiality before granting access to the forensic software materials runs contrary to these basic constitutional guarantees.

B. Common Practice and Equity Require Disclosure of Trade Secret Information When It Is Material to the Defense

As a general rule, when the prosecution seeks to hide evidence by claiming a trade secret privilege, it has to prove that the evidence qualifies as a trade secret.[30] Even if the government meets that burden, courts should still require disclosure, but may impose a protective order if the government proves one is warranted.[31]

It is so common in civil cases to disclose trade secrets under protective orders that many federal district courts have adopted a model protective order for the disclosure of source code to opposing counsel and experts retained by the party who agree to be bound by the order.[32] Disclosure subject to a protective order is routinely required even when the parties are direct competitors with an interest in profiting from the proprietary information of the other.[33]

Yet prosecutors consistently urge courts to divert from this established practice and deprive criminal defendants of access to forensic software even subject to a protective order. This higher barrier to discovery is backwards. It should be *easier* for a defendant trying to defend his life and liberty to access and assess forensic software, as compared to a party with a mere economic interest.

C. The Prosecution Typically Cannot Establish That Disclosure Subject to a Protective Order Would Cause Harm

When disclosure is sought under a protective order, the court weighs the risk of harm from disclosure *subject to the protective order*, rather than presuming disclosure to the public.[34] It is very unlikely for harm to result from disclosure to attorneys and retained experts subject to a protective order.[35] Thus, even in the rare case when a company's economic interest is found to outweigh the public's interest in transparency, access and the fair administration of justice, source code and software development documentation should at the very least still be made available to defense teams subject to a protective order, not withheld completely.

Conclusion

Technology has the potential to make the administration of justice fairer, but it has the opposite effect if it is shielded from inspection and accountability. Computer software can be flawed or embody dangerous biases that will harm innocent people unless defendants and the public are granted the opportunity to conduct independent testing. Educating defense attorneys and the courts about the need for transparency and accountability in the review of forensic software used to generate evidence is the only way to safeguard clients from a technologically obscured miscarriage of justice.

Notes

1. Roger A. Grimes, *Five Reasons Why Software Bugs Still Plague Us*, CSO Online (July 8, 2014), https://www.csoonline.com/article/2608330/security/5-reasons-why-software-bugs-still-plague-us.html.

2. Michael King & David Herring, *Research Satellites for Atmospheric Sciences, 1978-Present, Serendipity and Stratospheric Ozone* (Dec. 10, 2001), https://earthobservatory.nasa.gov/Features/RemoteSensingAtmosphere/remote_sensing5.php.

3. Jack Power, *Software Company Behind HSE Scan Glitch Begins Investigation*, Irish Times (Aug. 5, 2017), https://www.irishtimes.com/news/ireland/irish-news/software-company-behind-hse-scan-glitch-begins-investigation-1.3178349.

4. Allie Coyne, *CBA Blames Coding Error for Alleged Money Laundering*, itnews (Aug. 7, 2017), https://www.itnews.com.au/news/cba-blames-coding-error-for-alleged-money-laundering-470233.

5. Sonari Glinton, *How a Little Lab in West Virginia Caught Volkswagen's Big Cheat*, NPR Morning Edition (Sept. 24, 2015), http://www.npr.org/2015/09/24/443053672/how-a-little-lab-in-west-virginia-caught-volkswagens-big-cheat.

6. Andrea Roth, *Machine Testimony*, 126 Yale L.J. 1972, 1983-93 (2017); Christian Chessman, *A 'Source' of Error: Computer Code, Criminal Defendants, and the Constitution*, 105 Cal. L. Rev. 179 (2017).

7. Edward J. Imwinkelried, *Computer Source Code: A Source of the Growing Controversy Over the Reliability of Automated Forensic Techniques*, 66 DePaul L. Rev. 97 (2016).

8. *Melendez-Diaz v. Massachusetts*, 557 U.S. 305 (2009).

9. *Commonwealth v. Foley*, 38 A.3d 882 (Pa. 2012).

10. L. Kirchner, *Thousands of Criminal Cases in New York Relied on Disputed DNA Testing Techniques*, ProPublica, Sept. 4, 2017, https://www.propublica.org/article/thousands-of-criminal-cases-in-new-york-relied-on-disputed-dna-testing-techniques.

11. President's Council of Advisors on Science and Technology (PCAST), Report to the President: Forensic Science in Criminal Courts: Ensuring Scientific Validity of Feature-Comparison Methods 78 (2016) [hereinafter PCAST Report], https://obamawhitehouse.archives.gov/sites/default/files/microsites/ostp/PCAST/pcast_forensic_science_report_final.pdf.

12. *See United States v. Johnson*, 15-CR-565 (VEC) (S.D.N.Y. June 7, 2016).

13. L. Kirchner, *ProPublica Seeks Source Code for New York City's Disputed DNA Software*, ProPublica, Sept. 25, 2017, https://www.propublica.org/article/propublica-seeks-source-code-for-new-york-city-disputed-dna-software; L. Kirchner, *Federal Judge Unseals New York Crime Lab's Software for Analyzing DNA Evidence*, ProPublica, Oct. 20, 2017, https://www.propublica.org/article/federal-judge-unseals-new-york-crime-labs-software-for-analyzing-dna-evidence; *Forensic Statistical Tool Source Code*, https://github.com/propublica/nyc-dna-software.

14. David Murray, *Queensland Authorities Confirm 'Miscode' Affects DNA Evidence in Criminal Cases*, Courier Mail (Mar. 20, 2015), http://www.couriermail.com.au/news/queensland/queensland-authorities-confirm-miscode-affects-dna-evidence-in-criminal-cases/news-story/833c580d3f1c59039efd1a2ef55af92b.

15. *See People v. Collins*, 49 Misc.3d 595, 613-616 (Kings Co. Sup. Ct. 2015).

16. *See, e.g.*, Paolo Garofano, et al., *An Alternative Application of the Consensus Method to DNA Typing Interpretation for Low Template-DNA Mixtures*, Forensic Sci. Int'l: Genetics Supp. Series 5, at e422–e424 (2015).

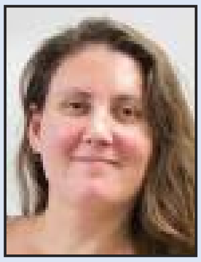
17. PCAST Report at 79 n.212. *See, e.g.*, *People v. Hillary*, Court No. 2015-15 (N.Y. Sup. Ct. St. Lawrence Co. 2016), http://www.northcountrypublicradio.org/assets/files/08-26-16DecisionandOrder-DNAAnalysisAdmissibility.pdf; *see* Jesse McKinley, *Oral Nicholas Hillary Acquitted in Potsdam Boy's Killing*, N.Y. Times, Sept. 28, 2016, http://www.nytimes.com/2016/09/29/nyregion/oral-nicholas-hillary-potsdam-murder-trial-garrett-phillips.html; *see also* PCAST, *An Addendum to the PCAST Report*, at 8, https://obamawhitehouse.archives.gov/sites/default/files/microsites/ostp/PCAST/

(Continued on page 66)

About the Authors

Stephanie J. Lacambra is the Criminal Defense Staff Attorney at the Electronic Frontier Foundation. Prior to joining EFF, she worked as a Federal Defender in San Diego and a Public Defender in San Francisco, handling cases ranging from drug and alien smuggling to attempted murder.

Stephanie J. Lacambra
Electronic Frontier Foundation
San Francisco, California
415-436-9333
Email stephanie@eff.org



Jeanna Matthews is an Associate Professor of computer science at Clarkson University. Her research is in computer security, algorithmic accountability and transparency. She is a 2017-18 Fellow at Data and Society and a member of the Executive Committee of ACM U.S. Public Policy Council.

Jeanna Matthews
Clarkson University
Potsdam, New York
315-268-6288
Email jnm@clarkson.edu

Kit Walsh is a Senior Staff Attorney at the Electronic Frontier Foundation, working on free speech, net neutrality, coders' rights, and other issues that relate to freedom of expression and access to knowledge. Prior to joining EFF, she led the civil liberties and patent practice areas at Harvard's Cyberlaw Clinic.

Kit Walsh
Electronic Frontier Foundation
San Francisco, California
415-436-9333
Email kit@eff.org

Hughes & Richard Torres, *Mixing It Up: Legal Challenges to Probabilistic Genotyping Programs for DNA Mixture Analysis*, The Champion, May 2018 at 12.

3. See Ford & Krane, *supra* note 1.

4. ISO/IEC/IEEE, *Systems and Software Engineering — Vocabulary*, ISO/IEC/IEEE *24765:2010(E)*, vol. 2010. at 1–418, 2010.

5. *Id.*

6. *Id.*

7. *Id.*

8. Black box software is "a system or component whose inputs, outputs, and general function are known but whose contents or implementation are unknown or irrelevant." *Id.*

9. White box software is "a system or component whose internal contents or implementation are known." *Id.*

10. *About ACM* (2017), https://www.acm.org/about-acm (last visited June 28, 2018).

11. Press Release, Association for Computing Machinery U.S. Public Policy Council (USACM), USACM Issues Statement on Algorithmic Transparency and Accountability (2017), https://www.acm.org/articles/bulletins/2017/january/usacm-statement-algorithmic-accountability (last visited June 28, 2018); Press Release, Association for Computing Machinery U.S. Public Policy Council (USACM), Statement on Algorithmic Transparency and Accountability (2017), https://www.acm.org/binaries/content/assets/public-policy/2017_usacm_statement_algorithms.pdf (last visited June 28, 2018).

12. N. Diakopoulos & S. Friedler, *How to Hold Algorithms Accountable*, MIT Technology Review (2016), https://www.technologyreview.com/s/602933/how-to-hold-algorithms-accountable (last visited June 28, 2018).

13. C.D. Steele & D.J. Balding, *Statistical Evaluation of Forensic DNA Profile Evidence*, 1 Annu. Rev. Stat. Its Appl. 361–384 (2014).

14. Scientific Working Group on DNA Analysis Methods, *Guidelines for the Validation of Probabilistic Genotyping Systems* (2015); M.D. Coble et al., *DNA Commission of the International Society for Forensic Genetics: Recommendations on the Validation of Software Programs Performing Biostatistical Calculations for Forensic Genetics Applications*, 25 Forensic Sci. Int. Genet. 191–197 (2016).

15. Institute of Electrical and Electronics Engineers, IEEE Std 1012-2012 - IEEE Standard for System and Software Verification and Validation (May 2012); Institute of Electrical and Electronics Engineers, IEEE Std 829-2008 - IEEE Standard for Software and System Test Documentation (July 2008).

16. M.W. Perlin, Second Declaration of Mark W. Perlin in Response to Defense Motion to Compel (2016), https://www.cybgen.com/information/newsroom/2016/apr/files/B-Perlin-second-declaration.pdf (last visited June 28, 2018).

17. D. Balding, likeLTD (likelihoods for low-template DNA profiles), https://sites.google.com/site/baldingstatisticalgenetics/software/likeltd-r-forensic-dna-r-code (last visited June 28, 2018).

18. See Ford & Krane, *supra* note 1. ■

About the Author

Nathaniel Adams is a Systems Engineer at Forensic Bioinformatics, a company that reviews cases involving forensic DNA testing.

Nathaniel Adams
Forensic Bioinformatic Services
Fairborn, Ohio
937-426-9270
Email adams@bioforensics.com

Confronting Forensic Software

(Continued from page 39)

pcast_forensics_addendum_finalv2.pdf; P. Garofano et al., *An Alternative Application of the Consensus Method to DNA Typing Interpretation for Low Template-DNA Mixtures,* Forensic Sci. Int'l: Genetics Supp. Series 5 at e422–e424 (2015).

18. Joy Buolamwini & Timnit Gebru, Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification. Proceedings of the Fairness, Accountability and Transparency Conference, New York, N.Y. (2018), http://proceedings.mlr.press/v81/buolamwini18a/buolamwini18a.pdf.

19. Marco Tulio Ribeiro, Sameer Singh & Carlos Guestrin, 'Why Should I Trust You?': Explaining the Predictions of Any Classifier. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). ACM, New York, N.Y. 1135-1144 (2016). DOI: https://doi.org/10.1145/2939672.2939778, https://dl.acm.org/citation.cfm?id=2939778.

20. S. Garfinkel, J. Matthews, S. Shapiro & J. Smith, *Toward Algorithmic Transparency and Accountability,* 60 Communications of the ACM at 5, 10.1145/3125780 (Sept. 2017).

21. *Richmond Newspapers, Inc. v. Virginia,* 448 US 555, 580, n.17 (1980) (upholding presumption that criminal trials be open to the public and recognizing the common law tradition "that historically both civil and criminal trials have been presumptively open").

22. U.S. Const. amend. VI, amend. XIV.

23. *Id.*

24. *See Bridgestone/Firestone Inc. v. Sup. Ct.* (1992) 7 Cal.App.4th 1384, 1393 (finding that a court is required to order disclosure of a trade secret unless, after balancing the interests of both sides, it concludes that under the particular circumstances of the case, no fraud or injustice would result from denying disclosure); *Agricultural Labor Relations Bd. (ALRB) v. Richard A. Glass Co.,* 175 Cal.App.3d 703 (1985) (allowing the trade secret privilege to stand would tend to work an injustice on the agricultural workers involved).

25. *See* U.S. Const. amend. VI ("In all criminal prosecutions, the accused shall enjoy the right to a speedy and public trial … and to be informed of the nature and cause of the accusation; to be confronted with the witnesses against him; to have compulsory process for obtaining witnesses in his favor …"); *Richmond Newspapers*, 448 U.S. at 580 (First Amendment requires criminal trials be open to the public).

26. *Richmond Newspapers,* 448 U.S. at 580 n.17 (recognizing the common law tradition "that historically both civil and criminal trials have been presumptively open").

27. U.S. Const. amend. XIV; *State v. Schwartz,* 447 N.W.2d 422, 427-28 (Minn. 1989) (court reasoned that prejudicial failure to disclose information such as forensic methodology on the basis that the method was a trade secret provided grounds for excluding the evidence because "access to the data, methodology, and actual results is crucial so a defendant has at least an opportunity for independent expert review").

28. *Patterson v. New York,* 432 U.S. 197, 215 (1977).

29. *Sandstrom v. Montana,* 442 U.S. 510, 524 (1979); *see generally Mullaney v. Wilbur,* 421 U.S. 684 (1975).

30. *See* Evid. Code § 1061(b)(1); *see Bridgestone/Firestone*, 7 Cal.App.4th at 1393 (the party claiming the privilege has the burden of establishing its existence).

31. *See* Evid. Code § 1061(b)(4).

32. *See* http://www.cand.uscourts.gov/model-protective-orders.

33. Benham, 71 Wash. & Lee L. Rev. at 2240-2241.

34. *Coca-Cola Bottling Co. of Shreveport v. Coca-Cola Co.* 107 F.R.D. 288, 293 (D. Del. 1985).

35. *See United States v. United Fruit Co.,* 410 F.2d 553, 556 (5th Cir. 1969) (*cert. denied*) (disclosure is less likely when made to a party that is not a competitor). ■