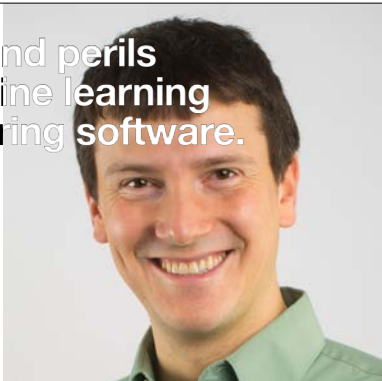


The promise and perils of using machine learning when engineering software.

Yuriy Brun



MaLTeSQuE 2022

Machine learning today



Resilient cities Cities

Predicting crime, LAPD-style

Cutting edge data-driven analysis directs Los Angeles patrol officers to likely future crime scenes - but critics worry that decision making by machine will bring 'tyranny of the algorithm'

- Join our live Q&A with Homicide Watch this Friday



Photo by investigator P. Jeffrey Brantingham at the United Concord Bank in Los Angeles. This is not Microsoft's logo. Photo credit: Jason Cummings/ISTOCK

<https://www.theguardian.com/cities/2014/jun/26/predicting-crime-lapd-los-angeles-police-data-analysis-algorithm-minority-report>

ACLU The Government Is Blacklisting People Based on Predictions of Future Crimes

By John Stanton, Executive ACLU National Security Project

Modern software uses machine learning to influence critical decisions



<https://www.aclu.org/blog/national-security/discriminatory-profiling/government-blacklisting-people-based-predictions>

Forbes The Algorithm That Beats Your Bank Manager

HAAS NEWS + NEWS CATEGORIES + RESEARCH NEWS

Minority homebuyers face widespread statistical lending discrimination, study finds

By Laura Counts | NOVEMBER 13, 2018

Face-to-face meetings between mortgage officers and homebuyers have been rapidly replaced by online applications and algorithms, but lending discrimination hasn't gone away.

A new University of California, Berkeley study has found that both online and face-to-face lenders charge higher interest rates to African American and Latino borrowers, earning 11 to 17 percent higher profits on such loans. All told, those homebuyers pay up to half a billion dollars more in interest every year than white borrowers with comparable credit scores do, researchers found.

The findings raise legal questions about the rise of statistical discrimination in the fintech era, and point to potentially widespread violations of U.S. fair lending laws, the researchers say. While lending discrimination has historically been caused by human prejudice, pricing disparities are increasingly the result of algorithms that use machine learning to target applicants who might shop around less for higher-priced loans.

"The mode of lending discrimination has shifted from human bias to algorithmic bias," said study co-author Adair Morse, a finance professor at UC Berkeley's Haas School of Business. "Even if the people writing the

HEALTH ITANALYTICS

Population Health Precision Medicine Quality & Innovation

MIT News

ON CAMPUS AND AROUND THE WORLD

SUBSCRIBE

Machine-Learning Model Detects Early-Stage Cancer

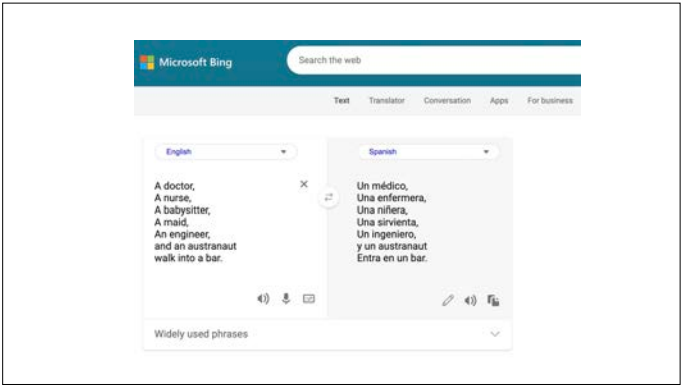
A new study suggests that machine-learned occult nodal metastasis in patients with a cavity cancer with more accuracy than sta

Using AI to predict breast cancer and personalize care

MIT/MGH's image-based deep learning model can predict breast cancer up to five years in advance.

Adam Conner-Simons and Rachel Gordon | CSAI, May 7, 2019

Software can make bad decisions. Software can discriminate!



Machine learning has great promise, but with that promise, come risks.

Today's goal:
Identifying and addressing the risks

Part I
Automated program repair

Part II
Software discrimination

Machine Learning in Software Engineering

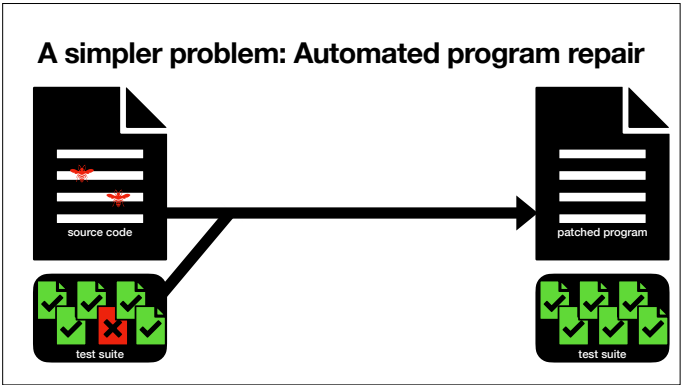
Your AI pair programmer

GitHub Copilot uses the OpenAI Codex to suggest code and entire functions in real-time, right from your editor.

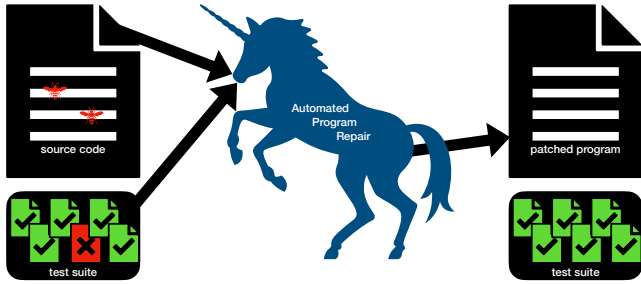
```

1 #!/usr/bin/env ts-node
2
3 import { fetch } from "https://api.github.com/graphql";
4
5 // Returns the sentiment of text in positive
6 // or negative
7
8 async function isPositive(text: string): Promise<boolean> {
9   const response = await fetch("https://api.github.com/graphql", {
10     method: "POST",
11     headers: {
12       "Content-Type": "application/json",
13     },
14   });
15   const json = await response.json();
16   return json.data.isPositive;
17 }
  
```

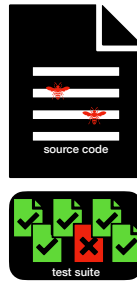
<https://github.com/features/copilot>



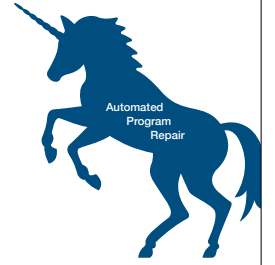
A simpler problem: Automated program repair



Program repair techniques



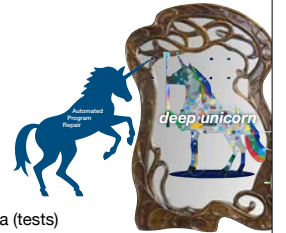
- Tweak the program
- Check if tests pass
- If not, repeat



Program repair techniques

APR is a form of machine learning

- first, many techniques rely on ML to learn
 - where to edit the code
 - how to edit the code
 - how to decide which patches are good
- second, the underlying problem is learning a function (program) using training data (tests)



How well does APR work?

- Evaluated 4 techniques
- GenProg
- Par
- TrpAutoRepair
- SimFix
- Measured patch quality
- Measured what affects patch quality

Quality vs. quantity

technique	GenProg	Par	SimFix	TRPAutoRepair	total
total					

When applied to real world bugs, APR produces patches for

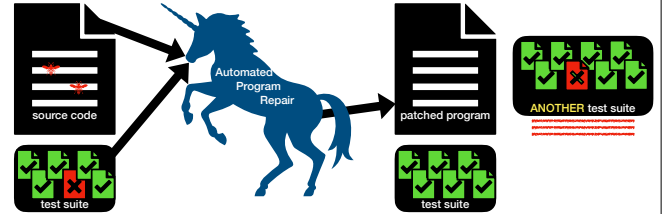
Quality vs. quantity

Potential problem: Overfitting

APR uses a set of tests to guide repair.
Tests are inherently partial.
No way APR can know if a patch captures intended behavioral constraints.



Quality vs. quantity



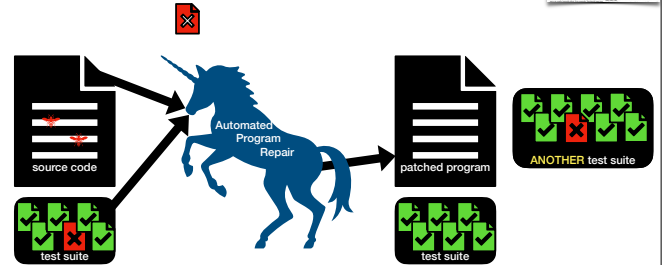
Quality vs. quantity

technique	minimum	patch quality mean	median	maximum	100%-quality patches
GenProg	64.8%	95.7%	98.4%	100.0%	24.3%
Par	64.8%	96.1%	98.5%	100.0%	13.8%
SimFix	65.0%	96.3%	99.9%	100.0%	46.1%
TrpAutoRepair	64.8%	96.4%	98.4%	100.0%	19.5%

Less than half (14-46%)
of the patches are correct



Does APR at least improve things a bit?



Does APR at least improve things a bit?



technique	change in quality due to patch			
	minimum	mean	median	maximum
GenProg	-30.9%	-1.7%	0.0%	2.6%
Par	-30.9%	-2.8%	0.0%	1.5%
SimFix	-24.9%	0.2%	0.0%	35.0%
TrpAutoRepair	-30.9%	-2.1%	0.0%	3.8%



Is the Cure Worse Than the Disease? Overfitting in Automated Program Repair

Edward K. Smith¹, Earl T. Barr², Claire Le Goues³, Yuriy Brun⁴
¹University of Massachusetts Amherst, MA, USA
²University College, London, UK
³Carnegie Mellon University, Pittsburgh, PA, USA
⁴tsdks, tsdks@cs.umass.edu, e.barr@ucl.ac.uk, clegoues@cmu.edu

ABSTRACT
 Automated program repair has shown promise for reducing the rigidity of manual error debugging practices. This paper addresses a critical barrier to the adoption of automated program repair: the quality of the patches it generates. We evaluate the quality of patches generated by four state-of-the-art APR techniques: GenProg, Par, SimFix, and TrpAutoRepair. We find that the quality of the patches is generally poor, with a median quality of 0.0%. We also find that the quality of the patches is generally poor, with a median quality of 0.0%. We also find that the quality of the patches is generally poor, with a median quality of 0.0%.



Takeaway: Tests are an imperfect oracle, so APR suffers, producing low-quality patches.

Can we find a domain with better oracles?

Formal verification allows proving software correct



Interactive theorem provers for formal verification

Formal verification comes with a built-in oracle:
The theorem prover

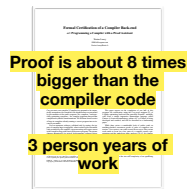


Industrial impact of theorem proving



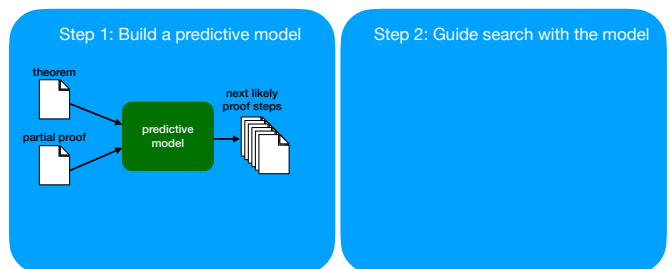
Prohibitively difficult

Verified software requires a lot of time and a lot of proofs in proportion to code

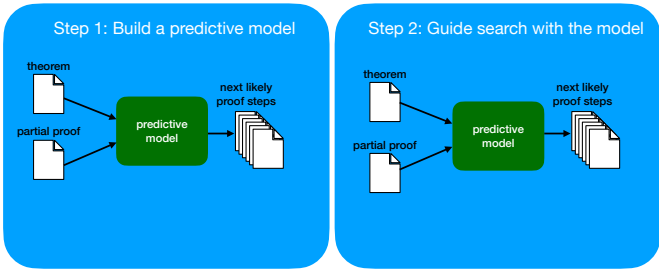


Virtually all software that ships today is unverified.

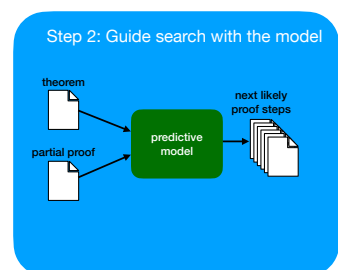
Proposal: Use APR-style technology to synthesize proofs



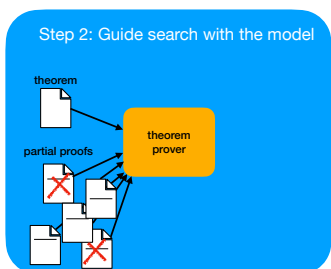
Proposal: Use APR-style technology to synthesize proofs



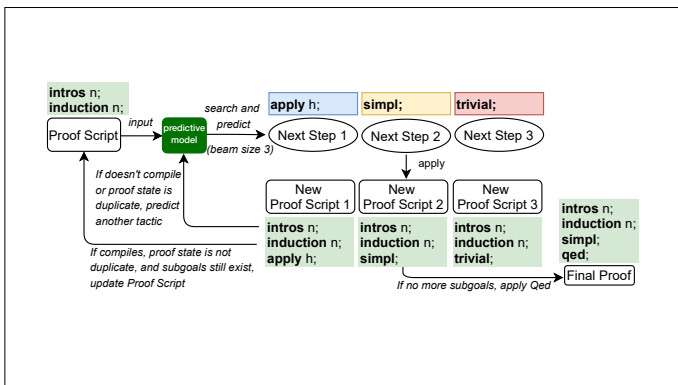
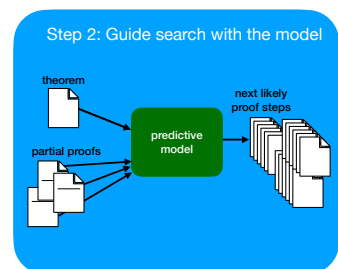
Proposal: Use APR-style technology to synthesize proofs



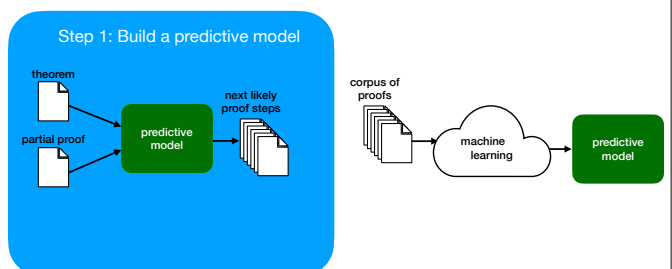
Proposal: Use APR-style technology to synthesize proofs



Proposal: Use APR-style technology to synthesize proofs



How to learn a predictive model



TacTok (OOPSLA'20)

TacTok: Semantics-Aware Proof Synthesis
 EMILY FIRST, University of Massachusetts Amherst, USA
 YURY BRUN, University of Massachusetts Amherst, USA
 ARJUN GUHA, University of Massachusetts Amherst, USA

Formally verifying software correctness is a highly manual process. However, because verification proof scripts often share structure, it is possible to learn from existing proof scripts to fully automate some formal verification. The goal of this paper is to improve proof script synthesis and enable fully automating more verification. Interactive theorem provers, such as the Coq proof assistant, allow programmers to write partial proof scripts, observe the semantics of the proof state that has, and then attempt more progress. Knowing the structure of the proof state, the current proof state, and the current tactic, we propose a neural network that models partial proof scripts and the current proof state, together.

Training Proofs, Training Instances, Tactic, AST

TacTok models partial proof and the current proof state, together

ASTactic [Yang and Deng, Learning to Prove Theorems via Interacting with Proof Assistants, ICML'19] modeled just proof state. [Hellendoorn, Devanbu, Alipour, On the naturalness of proofs, ESEC/FSE NIER'18] looked at predictability of proof sequences.

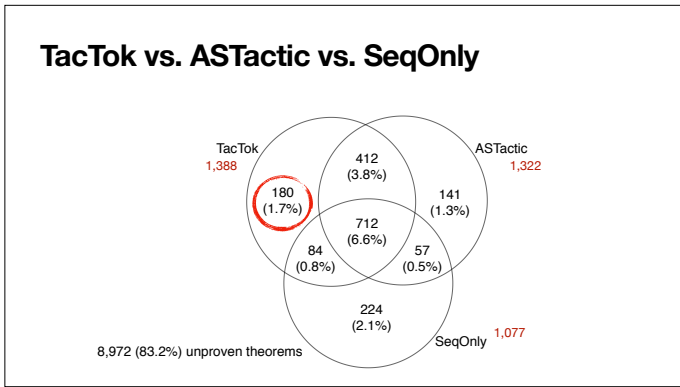
Emily First, Yury Brun, and Arjun Guha. 2020. TacTok: Semantics-Aware Proof Synthesis. Proc. ACM Program. Lang. 4, POPLA, Article 270 (December 2020), 13 pages. <https://doi.org/10.1145/3426209>

CoqGym Dataset

- 123 open-source software projects in Coq
- 70,856 theorems
- Broken down into 96 projects (57,719 proofs) for training and 27 projects (13,137 theorems) for testing

<https://github.com/princeton-vl/CoqGym>

[Yang and Deng, Learning to Prove Theorems via Interacting with Proof Assistants, ICML'19]



TacTok vs. ASTactic vs. CoqHammer

Works more frequently than most APR tools, and guaranteed correct!

APR produces patches for 10.6-19.0% of the defects

7,500 (69.6%) unproven theorems

Diva (ICSE'22)

2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)

Diversity-Driven Automated Formal Verification

Emily First, University of Massachusetts Amherst, USA
 Yury Brun, University of Massachusetts Amherst, USA

2 key observations:

- Machine learning is often noisy
- Theorem prover serves as an oracle to turn that noise into signal.

Diva (ICSE'22)

2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)

Diversity-Driven Automated Formal Verification

Emily First, University of Massachusetts Amherst, USA
 Yury Brun, University of Massachusetts Amherst, USA

- Vary:
 - proof tactic and token depth
 - learning rate
 - embedding size
 - number of layers
 - training order
 - access to proof state, partial proof, Gallina proof term

ABSTRACT
 Formally verified correctness is one of the most desirable properties of software systems. But despite great progress with interactive theorem provers, such as Coq, writing proof scripts for verification remains one of the most often-cited pain points for practitioners in difficult software development activities. Recent work has proposed tools that automatically synthesize proofs or proof scripts. For example, CoqHammer can prove 30% of theorems completely automatically by learning using programming languages like Haskell and Python, which are machine learning friendly proof languages, and then performs search through the proof script space. In this paper, we use 21.6% of the theorems in the CoqHammer dataset, those that are highly complementary together, they can prove 50% of the theorems fully automatically. Our key insight is that instead of the learning process can produce a diverse set of models, and that, due to the unique nature of proof synthesis (the existence of the theorem prover, an oracle that validity orders a proof), we can use the models to synthesize proofs that are designed to interactively synthesize proofs.

KEYWORDS
 Automated formal verification, language models, Coq, interactive proof assistants, proof synthesis

1 INTRODUCTION
 Building provable correctness in critical high-stakes domains, such as aerospace engineering and software for medical devices, however, most industrial verification tools either can't synthesize the verification proofs by themselves [11] or require manually writing the programming language in which the prover is capable [12]. A promising method for building correct software has been to use programming languages that are designed to interactively synthesize proofs.

Diva vs. state-of-the-art



Diversity inherent in ML increases the proving power 68%-77% over prior search-based synthesis tools, and 27% over CoqHammer.

<https://github.com/LASER-UMASS/Diva/>

Fully Automated Formal Verification

Machine learning and meta-heuristic search can fully automate some bug-repair and formal verification.

While APR underperforms because it is driven by an unreliable oracle, formal verification is a killer app for APR because the theorem prover provides a reliable oracle.

...let's talk about a different peril of machine learning that verification might help with.

Part II Software discrimination

**Part II
Software discrimination**

Data-driven systems can exhibit undesirable properties. Can we build systems to be safe and fair?

Testing systems for bias

Themis

automated test-suite generator



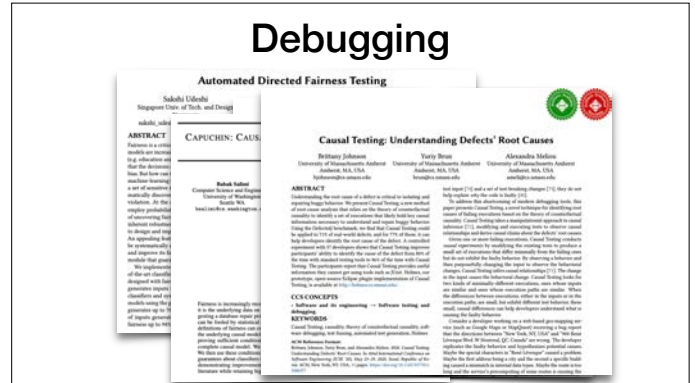
How much does my software discriminate with respect to ...?

Does my software discriminate more than 10% of the time, and against what?

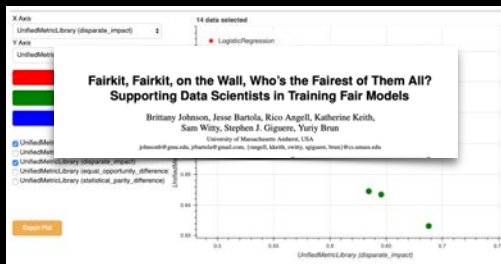
<http://fairness.cs.umass.edu>

Gahotra, Brun, and Melou, Fairness Testing: Testing Software for Discrimination, ESEC/FSE 2017

Debugging bias



fairkit-learn



Can we verify systems to be safe and fair?

How would that work?

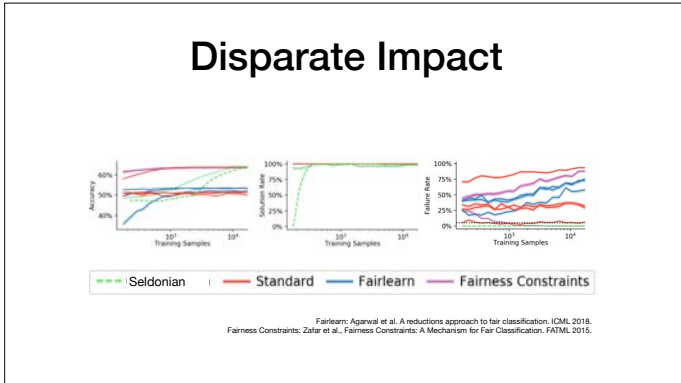
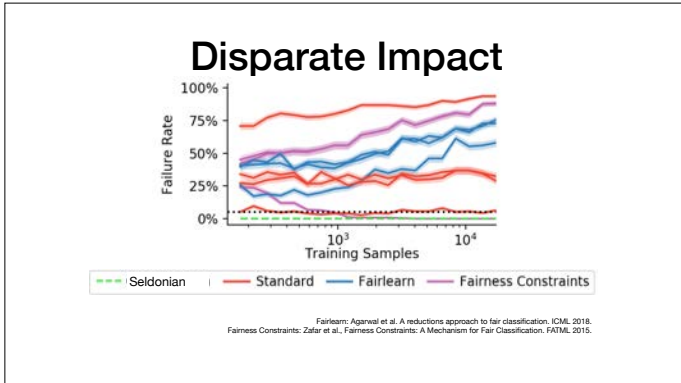
User specifies a definition of safe or fair behavior.

training testing safety

Train classifiers, selects one to satisfy fairness, verify safety on held-out suite.

Example scenario:

Suppose a university wants to train a model to predict student success from entrance exam scores, while ensuring the model is fair: roughly the same fraction of men and women are predicted to be successful. (This is called Disparate Impact.)



Equalized Odds

RESEARCH

COMPUTER SCIENCE

Preventing undesirable behavior of intelligent machines

Philip S. Thomas¹, Bruno Castro da Silva², Andrew G. Barto³, Stephen Giguere⁴, Yuri Brun⁵, Emma Brunskill⁶

Intelligent machines using machine learning algorithms are ubiquitous, ranging from simple data analysis and pattern recognition tools to complex systems that achieve superhuman performance on various tasks. Ensuring that they do not exhibit undesirable behavior—that they do not, for example, cause harm to humans—is therefore a pressing problem. We propose a general and flexible framework for designing machine learning algorithms. This framework simplifies the problem of specifying and regulating undesirable behavior. To show the stability of the framework, we used it to create machine learning algorithms that prevented the dangerous behavior caused by standard machine learning algorithms in our experiments. Our framework for designing machine learning algorithms simplifies the safe and responsible application of machine learning.

Machine learning (ML) algorithms are having an increasing impact on modern society. They are used by politicians to predict likelihood (1) and by biologists working to create a vaccine for HIV (2); they also influence criminal sen-

algorithm could output. Our framework mathematically defines what an algorithm should do in a way that allows the user to directly place probabilistic constraints on the solution, $\alpha(z)$, returned by the algorithm. This differs from the standard ML approach wherein the user can only indirectly constrain $\alpha(z)$ by restricting or modifying the feasible set \mathcal{H} or objective function f . Concretely, algorithms constrained using our framework are designed to satisfy constraints of the form $\text{Pr}(g(\alpha(z)) \leq \beta) \geq 1 - \delta$, where $g: \mathcal{H} \rightarrow \mathbb{R}$ defines a measure of undesirable behavior (as illustrated later by example) and $\beta \in [0, 1]$ limits the admissible probability of undesirable behavior.

Note that in these constraints, β is the only source of stochasticity; we denote random variables by capital non-algebraic letters to make clear which terms are random in statements of probability and expectation. Because these constraints define which algorithms are acceptable (rather than which solutions α are acceptable), they must be satisfied during the design of the algorithm rather than when the algorithm is applied. This shifts the burden of ensuring that the algorithm is well-behaved

algorithm could output. Our framework mathematically defines what an algorithm should do in a way that allows the user to directly place probabilistic constraints on the solution, $\alpha(z)$, returned by the algorithm. This differs from the standard ML approach wherein the user can only indirectly constrain $\alpha(z)$ by restricting or modifying the feasible set \mathcal{H} or objective function f . Concretely, algorithms constrained using our framework are designed to satisfy constraints of the form $\text{Pr}(g(\alpha(z)) \leq \beta) \geq 1 - \delta$, where $g: \mathcal{H} \rightarrow \mathbb{R}$ defines a measure of undesirable behavior (as illustrated later by example) and $\beta \in [0, 1]$ limits the admissible probability of undesirable behavior.

Note that in these constraints, β is the only source of stochasticity; we denote random variables by capital non-algebraic letters to make clear which terms are random in statements of probability and expectation. Because these constraints define which algorithms are acceptable (rather than which solutions α are acceptable), they must be satisfied during the design of the algorithm rather than when the algorithm is applied. This shifts the burden of ensuring that the algorithm is well-behaved

Thomas, Castro da Silva, Barto, Giguere, Brun, and Brunskill.
 "Preventing Undesirable Behavior of Intelligent Machines", Science 366 (6468), Nov 22, 2019

And this approach is very versatile: ...works for policy selection

Offline Contextual Bandits with High Probability Fairness Guarantees

Emmanuel Métévier¹, Stephen Giguere², Sarah Brockman³, Ad Kohen⁴,
 Yuri Brun⁵, Emma Brunskill⁶, Philip S. Thomas⁷

¹Department of Computer Science, University of Toronto
²College of Information and Computer Sciences, University of Massachusetts Amherst
³Department of Computer Science, University of Toronto
⁴Department of Computer Science, University of Toronto
⁵Department of Computer Science, University of Toronto
⁶Department of Computer Science, University of Toronto
⁷Department of Computer Science, University of Toronto

Abstract

We present *OfflineBandit*, an offline contextual bandit algorithm designed to satisfy a broad family of fairness constraints. Unlike previous work, our algorithm accepts multiple fairness definitions and allows users to constrain their own unique fairness definitions for the problem at hand. We provide a theoretical analysis of *OfflineBandit*, which includes a proof that it will not require an unfair solution to approximately guarantee user-specified fairness. We evaluate our algorithm on three applications: a learning system in which we conduct a user study and compare multiple simple fairness definitions; a loan approval setting using the Stanford Lending Club data set in which we know a fairness definition are applied, and contextual evidence using data released by ProPublica. In each setting, our algorithm is able to produce fair policies that achieve performance competitive with other offline and online contextual bandit algorithms.

Métévier, Giguere, Brockman, Kohen, Brun, Brunskill, Thomas. Offline Contextual Bandits with High Probability Fairness Guarantees. NeurIPS 2019.

Example scenario:

One source of ML bias comes from deploying a model on data that is fundamentally different from the data the model was trained on.

What if software is deployed on data fundamentally different from training data?

Published as a conference paper at ICLR 2022

FAIRNESS GUARANTEES UNDER DEMOGRAPHIC SHIFT

Stephen Giguere
 Department of Computer Science, University of Toronto at Austin
 sgi@cs.toronto.edu

Emmanuel Métévier, Yuri Brun, Bruno Castro da Silva, & Philip S. Thomas
 College of Information and Computer Sciences, University of Massachusetts

Sarah Brockman
 Department of Computer Science, University of Toronto at Austin

ABSTRACT

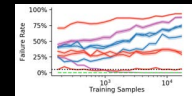
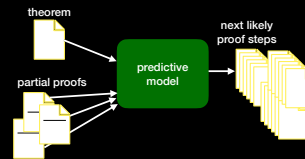
Recent studies found that using machine learning for social applications can lead to disparities in the form of racial, ethnic, and otherwise unfair and discriminatory outcomes. To address this challenge, recent machine learning algorithms have been proposed to ensure the data used for training is representative of what will be encountered in deployment, which is often unfair. In particular, fairness subgroups of the population become more or less probable in deployment in comparison to what demographic shifts, given such a fairness assumption and user feedback. In this paper, we consider the design, development, and generalization of algorithms, called *OfflineBandit*, that provide high-probability behavioral guarantees that hold under demographic shift when data from the deployment environment is unavailable during training. *OfflineBandit*, the first technique of its kind, demonstrates an effective strategy for designing algorithms that can be deployed in a wide range of settings. We present *OfflineBandit* using the U.S. Adult Census dataset (Kohler and Becker, 1998), as well as a real-world dataset of university entrance exams and subsequent student success. We show that the learned models avoid bias under demographic shift, while meeting methods. Our experiments demonstrate that our algorithm's high-probability fairness guarantees are able to prevent bias in our algorithms in an offline test for training models that are fair these demographic shift users.

Giguere, Métévier, Brun, Castro da Silva, Thomas, and Niekum, Fairness Guarantees under Demographic Shift, ICLR 2022.

Machine learning can result in unexpected, unintended behavior.

But machine learning can be leveraged to produce verified safe and fair models, avoiding such behavior.

Contributions



Rico Angell	Brittany Johnson	Stephen Giguere	Sarah Brockman	Blossom Metevier	Sainyam Gaihotra
Emily First	Alex Sanchez-Stern	Zhamna Kaufman	Manish Motani	Claire Le Goues	Talia Ringer
Alexandra Melou	Andy Barto	Bruno Castro da Silva	Emma Brunskill	Philip Thomas	Yury Brun