A Rule-authoring System to Enable End-User In-The-Loop Machine Learning

Soya Park
MIT CSAIL
http://people.csail.mit.edu/soya/
soya@mit.edu

Kerry Chang, Edward Benson
Instabase
[kerry, ted]@instabase.com

Abstract

Improving a machine learning (ML) model is difficult without ML knowledge. In this paper, we propose a model wherein user-constructed rules can work together with a learned model to achieve better outcomes than either working alone. Informed by expert interviews, we introduce a user interface and a query language that enables end-users to perform an error analysis of pre-trained models. This interface then enables the users to explore their dataset and help to gain better insights. Then, users can create and iteratively refine rules targeted at areas of poor model performance.

2012 ACM Subject Classification Human-centered computing → User interface programming

Keywords and phrases Mixed-initiative interface, Debugging ML, Rule-based

Digital Object Identifier 10.4230/OASIcs.PLATEAU.2019.23

1 Machine Learning: Everyone wants, only some can use

Many people want to apply machine learning (ML) to their domain, but not everyone understands how ML works. As a result, a burgeoning market for pre-trained models has emerged. In this market, a small group of ML practitioners train and supply models to technical, but non ML-savvy, domain experts who use them in workflows. For example, financial firms use third-party models to classify incoming financial documents attached to customer emails.

While the users of these pre-trained models are technical experts in their own domains, they often do not have the knowledge to train and improve the models themselves. This creates a disconnect: these users are the key stakeholders of their model's output, but they must rely on third parties to make any improvements related to performance.

In this paper, we propose a new system that enables end-users to work together with ML. In our system, end-users construct rules that can be used to tactically *overwrite* an ML model's decision when the model's confidence is low. This allows end-users to continue treating the learned model as an unmodified black box—as they do today—but begin to directly put their domain knowledge to use towards performance improvement. We enable this by letting users craft rules. Rules are logical statements about an `(input, output)` pair from a learned model that can override that model's choice of output. Unlike typical ML systems, rules are declarative, relatively easy to comprehend for end-users, and efficient for capturing domain knowledge [**?**].

The following are example use cases in different data domains:

- **Text Classification**. A text classifier trained to detect spam might learn to automatically flag emails containing the phrase `Housing Refinance`, which could be appropriate across a general population but incorrect for employees at a mortgage company. Our system would permit members of this mortgage company to visually observe that `Housing`

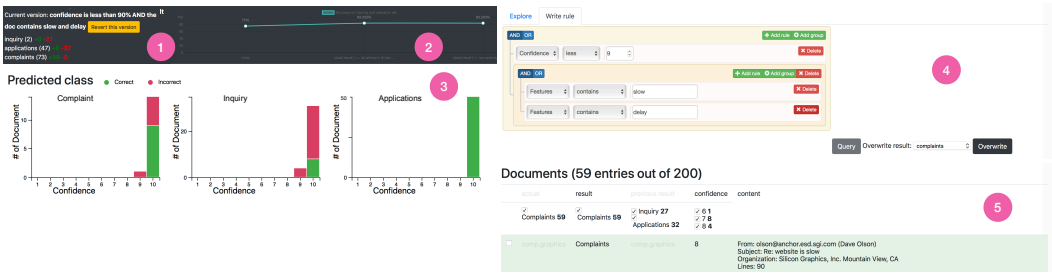

Figure 1 Interface of Toby: a user can explore and overwrite ML results based on their observation. Each rule becomes a version at Toby and Toby provides an interface to aid version controls (a) rule version-control panel (b) accuracy plot of users' versions (c) model's decision and confidence distribution (d) rule editor (e) input previews table with a filtering feature

`Refinance` was a problematic classification feature and create a custom rule to override the learned spam model for this special case.

- **Image Classification**. In practice, many image classifiers tend to heavily rely on pre-trained neural network weights from benchmark datasets. This can over-bias classification in ways unanticipated by a downstream practitioner. For example, an "occupation guessing" model might pick up on eyeglasses and over-bias results toward outputs such as `librarian`, simply because the training dataset had an abundance of stock imagery playing to the archetype. Our model would enable a practitioner to associate the intermediate feature of eyeglasses with the mis-classification and craft a rule to suppress this edge case.

Research has shown that end-users who are domain experts can craft well-performing rule-based systems, but it is very time consuming [**?**, **?**]. To simply rule construction, we use the presence of a black-box ML model to our advantage: guiding rule creation by applying the model to labeled datasets and using an interactive error analysis system to prioritize rule creation.

Informed by expert interviews, we designed a mixed initiative system of machine learning and rule-based heuristics, **Toby** (Tweaking yOur model By using Your domain knowledge) that focuses on authoring post-processing rule incorporating their own domain knowledge.

Toby provides an user interface where users can explore their dataset and author post-processing rules. We propose a data model that contains results from a model and features of individual inputs in a relational database.

Our paper makes the following contributions:

- In-depth interviews with machine learning experts for insights on designing a machine learning debugging interface.
- A rule-based programming language that enables end-users to specify rules as a post-processing step of a machine learning model, using features of a data type (e.g., text, image) independent of the machine learning model.
- A user interface that ties rule creation to interactive error analyses showing the impact of users' rules and rule editing history. Our user interface is tied with a matching query and supports smart auto-complete queries that are of interest to the user. Users can simply click button to see corresponding data.

2 Related work

We discuss previous work on debugging machine-learning models, with specific focus on mixed-initiative systems of intelligent systems and visualizing machine learning systems in the HCI community.

Rule-authoring tool for end users

Previous work has focused on building rule-based tools to facilitate task automation. These tools allow [**?**, **?**] end-users to program automation of different systems such as their geo-location and email. There are tools to aid automation of enterprise and customer relations [**?**, **?**, **?**]. Users can customize their marketing strategy using the tools' rule-authoring interface.

Our rule-authoring system for ML shares a common theme with email filtering since both overwrite results from ML models. Email users can write simple actions (e.g., move to a folder, archive) for individual messages based on their attributes such as sender and subject line. Users write email filters that overwrites results of ML-based spam filters. Similarly, our system overwrites results of users' ML models. Our system allows users to incorporate both ML model decisions and confidence in those decisions in their rule-authoring.

Mixed-initiative systems of intelligent systems

Recent work in intelligent systems combines two intelligent systems to capture the benefits of both [**?**]. In the program-by-example domain, users can get their desired program by disambiguating programs [**?**] and also providing sub-programs [**?**]. Mixed-initiative systems in machine learning rely on various initiatives including rule-based heuristics [**?**] and crowdsourcing [**?**]. Our work combines ML-based and rule-based systems to enhance the accuracy of ML models.

Programming ML

Some work extend programming paradigms in the area of machine learning and let ML engineers to *debug* their model [**?**]. Data-programming [**?**] provides a new approach to programmatically generate training sets. While data-programming improves a ML model in a *pre*-training phase by generating richer training set, our work focuses on a *post*-training phase by letting users author rules based on the results of a ML model.

Interactive exploration of ML results

Researchers have proposed several ways to explain model behavior to end-users [**?**, **?**, **?**]. For example, novel visualizations show a ML model's strength and weaknesses to help users understand the model's performance [**?**, **?**, **?**]. Another approach is *model-agnosticism* [**?**, **?**, **?**], which fits a statistical approximation to understand model decisions.

Previous work provides valuable insight into the workings of models, but does not enable end-users to act on this understanding with action that improves performance. Furthermore, modern deep learning techniques often pose significant interpretability challenges to experts, let alone end-users [**?**].

3 Expert Interviews

We conducted semi-structured interviews with seven ML practitioners (2 females, 5 males, mean age=26) to understand how experts work to improve the result of an ML model. Our goal is that by learning ML experts' common practices, we can design an interface that enables end-users to do so.

Among our interviewees, five of them worked in the industry, and the remaining two were academic researchers. All used machine learning as part of their work across the domains of document analysis, education, and health care.

Each interview was 45 minutes long and driven by a questionnaire that posed questions related to (1) model design workflow, (2) model tuning workflow, and (3) how the participant incorporates domain knowledge into the structure of their model. While approaches varied, and many standard maxims ("just add more data") were present, so too were common themes that lend themselves toward adaption for end-user empowerment.

It begins with understanding the data

Experts reported spending a significant amount of time exploring their data before constructing of a model. They often start by inspecting basic statistics of a dataset such as the mean and deviation. One interviewee said, "*One of the common mistakes of novice machine learning engineers are they spent too much time thinking about building model but not much time on input and understanding of data itself.*"

Domain knowledge plays an important role in the data exploration phase. Since domain knowledge provides an opportunity for ML experts to do a sanity check of their model performance, it helps experts to notice if their model is working as desired and gives feedback of early stage of model designing. An interviewee said, "*Knowing domain knowledge does greatly help designing a ML model such as how to pre-process and format the training set*". For example, knowing that a mean of 98.0 is a human body's temperature provides the human modeler an intuition that 32.0 than if the numbers were void of semantics. Interviewees list different ways of gaining domain knowledge such as exploring datasets and reading research papers. Modelers sometimes consult domain experts (e.g. course instructor, financial expert, medical doctor) if the domain is a specialized area.

A final data exploration procedure is understanding the format of the dataset. Questions they ask while probing the formats are 1) how data is created 2) how data is stored and 3) what role data cleaning and wrangling play in the data that will be experienced by the model.

Data partitioning is a common strategy

All experts reported common strategies and usefulness of dividing data into a custom partition. They said it is necessary to partition and group data into not only train, test, and validation sets which are in randomized manner but also sensible manner based on different data attributes. For example, one of the experts who is working on health care divides their dataset into different races to ensure the model's fairness: "it is important to know if my model works fairly (..) One of my practice is dividing my dataset into demographics across different races to test fairness."

Echoing the findings of previous work [**?**], how ML practitioners evaluate the performance of a model depends on the need of their task. For example, one practitioner who worked on medical data said, "For us, false positive is not as severe as false negative." The evaluation

method used also depends on the availability of the data data. One interviewee said, "we are testing on toy data which possesses the same characteristics of our target domain since there is no existing data in this domain." Another interviewee said: "we tried our two candidate models on customers and A/B test to find which is better."

It also turns out there is no *super* tool or interface that is being commonly used among ML practitioners. However, they write programming scripts (mostly python among the interviewees) for their ML tasks.

4 Design Goals

From our literature review and expert interviews, we identified the following design decisions for a post-processing rule-authoring interface for machine learning.

Facilitate sensemaking with easy dataset exploration

One main design goal of Toby is easy navigation of different attributes due to interviewees' need to explore data carefully. Data exploration includes not only viewing the input data before training, but also the data as it relates to the trained model (e.g. how the model classified inputs that contain certain keywords) [**?**]. In order to provide an easy and expressible outlet to explore and access data of interest, we provide a *query* feature. We have designed our rule-authoring interface to enable queries on any subset of a dataset and the model's results on that subset. Having this query feature would help not only exploring interesting subsets of datasets easily but also explore and overwrite rule on data partition that is mentioned during our expert interviews.

Track performance of rules

Another goal of Toby is to allow domain experts to leverage their expertise by writing rule-based heuristics while avoiding the intractability of purely rule-based systems. Although rule-based heuristics are often almost guaranteed to hold true, a purely rule-based system would have too many heuristics to reasonably keep track of. Once our system has helped a user navigate to a region of the dataset in which a common category of error seems to dominate (for example, a mis-classification from one label onto another), it permits them to craft targeted rules applying only to these error regions.

5 TOBY: a tool for crafting rules to improve classifiers

This section introduces the interface of Toby (Figure 1). Toby provides features for domain experts to explore a dataset and craft rules based on their observations and domain knowledge. We starts this section with a usage scenario to describe how a user could interact with Toby. We then discuss Toby's key features in detail.

5.1 Usage Scenario

Sam works in a bank and is responsible for handling customer emails. She uses a classifier that puts incoming emails into one of the three categories: complaints, inquiries, and applications. The classifier was created by an engineer in the bank a while ago. Over time, Sam has found that many emails were mis-classified. Sam has no experience with machine learning, but she has been doing this job for a while and has a good sense of what an email for a category generally looks like.

Sam collects a set of emails, and puts the email body and their actual categories in an Excel sheet. She uploads the data and the classifier model - a text file that is a Naive Bayes classifier using bi-grams as features - to Toby.

Comparing the classifier output with the actual category data, Toby shows that the current classifier has a 75% accuracy (Figure 1.(b), the first data point in the line chart). Below that, Toby shows three bar charts. Each chart is a histogram showing for all the documents predicted as a certain category, how confident the classifier is for the predictions, and if the predictions are correct. From these charts, Sam found that many emails were falsely classified as "inquiry". Even when the classifier is 100% confident, the majority of the emails it classified as "inquiry" should be something else.

Sam clicks on the red bar in the middle bar chart for "inquiry" (Figure 4) to inspect the falsely categorized emails. Toby supports an editor to let users write queries to explore the data (Figure 1.(d)). After Sam clicks on the red bar, Toby automatically brings up the corresponding data in a document pane (Figure 1.(e)) and fills the query editor with the query that generated this selection.

Sam found that most of the emails falsely classified as inquiries are actually complaints. She also notices that those emails contain words that often appear in the complaints emails in her department, such as "slow" and "delay". To correct this, Sam clicks on to the "Write rules" tab (Figure 1 at d) in the editor which lets her construct a query and assign a class to the data returned by the query to form a rule (Figure 2). She builds a rule that sets all emails predicted as inquiry and containing the word "slow" or "delay" to complaints, and hits a "Overwrite" button to apply this rule.

Toby applies Sam's newly set rule, and compares the final result to the actual category data again to generate a new accuracy value. Toby updates the accuracy line chart (Figure 1.(b)) with a second data point and shows in text how the rule impacts the result (Figure 1.(a), and Figure 2 at the bottom). Sam's new rule actually makes things worse. She realizes that her rule might be too aggressive, so she modifies her rule to apply only the model confidence is lower than 90%. She reapplies the rule. This time the accuracy increases!

Sam sets up a few more rules following the similar procedures - identifying the classifier's weakness through the prediction bar charts, exploring the falsely classified emails through queries, setting a rule and testing the rule by shuffling the data to ensure its robustness. Her rules now work together with the machine learning model made by her engineer colleague to become a new ensemble model that performs better than either the rules or the ML model working alone.

5.2 Key features

Data model

Toby's UI is backed up by a data model that includes the input data, the model's predicted values and confidence, and the actual values. Every time when the user modifies a rule, Toby stores the predicted values to support showing the accuracy tuning history.

The last thing we added to Toby's data model is what we called "data feature" that allows users to construct rich, meaningful queries for a type of data. Note that this "data feature" is not the features that the ML model uses to make decisions, as ML model features sometimes do not make sense to an end-user (such as vectors or pixel values). Here, data feature is related to the type of data and is independent of the ML model used.

In our prototype, Toby supports data feature for two types of data: text and images of handwriting. For text, words is the obvious data feature choice. Users are able to write

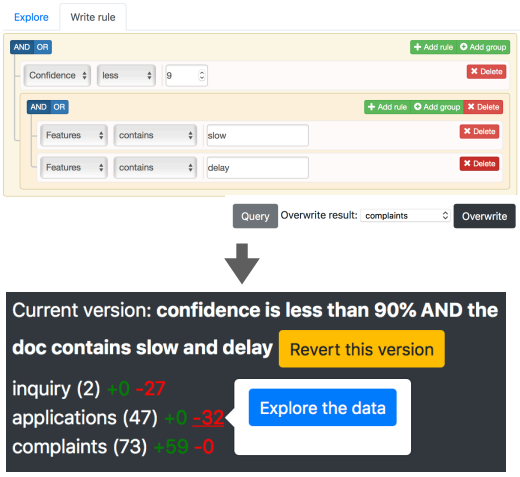

Figure 2 Toby users can author overwriting rules with simple `AND` and `OR` logic (top). Each rule becomes a version. As a user creates a new version, Toby displays the summary of the new version so the user can check the impact of the rule (bottom).

actual	result	previous result	confidence	content
label/class of the given input	Predicted label by the model	Predicted label by the model at a previous version	How confident about this decision (1-10)	Input preview

Figure 3 Toby data model

queries such as “documents that model is less than 80% confident and contain the word "TIFF" and "VGA"“. For images of handwriting, we choose the data feature to be the number of connected components in the image. This enables users to write rules such as “assign handwriting that have two connected components to be the number 8“ while users are almost impossible to describe this rule using raw pixel values. The data feature is created as the user uploads the dataset into Toby. In the UI, Toby has a document pane that displays the data model using a table (Figure 1.(e)).

Prediction bar charts

Toby displays a bar chart for each class of the model with the frequency on the y axis and confidence on the x axis (Figure 1.(c)). We design the prediction bar charts to help users identify under what situation it is better to use their domain knowledge to make a decision rather than using the ML model’s prediction. For example, if below a certain confidence value the ML model’s accuracy is worse than random guesses (i.e. 33% in our usage scenario), than instead of randomly guessing, make the decision using rules crafted by a domain expert.

Rule-authoring mechanism

As described in the usage scenario, a rule in Toby consists of a query and a value to assign to data returned by the query. User-defined rules are applied after the ML model runs. We used jQuery QueryBuilder [**?**] for the authoring interface. Users can build a query using the query editor directly, or they can generate a query by clicking on the prediction bar charts

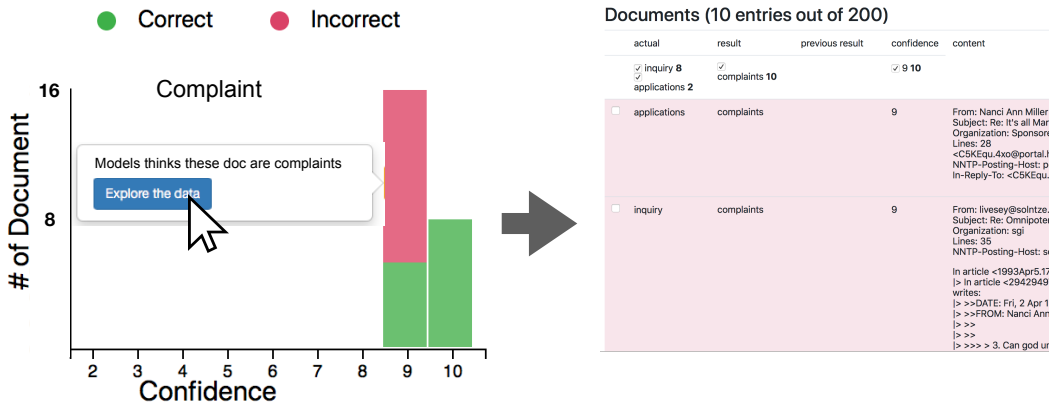

Figure 4 Toby plots the dataset with respect to the confidence probability of the decision (left). As a user clicks the "explore the data" button, Toby auto-completes the query and one can query the corresponding data (right). In this example, it will show the documents that are mis-classified as complaints with 90% of confidence.

(Figure 4) or the result summary text (Figure 2 at the bottom). Users can further modify the auto-generated query to add more searching criteria.

Version control of user-generated rules

We represent rules as *versions* in Toby and provide features of version-control systems such as tracing difference between versions and reverting versions. Each rule is equivalent to a version in Toby. Toby displays a summary of the impact of each version compared to the previous version. The summary shows how many data are classified as each class and the difference between the current and previous version. As users craft rules, they can cancel rules if the rule is not working as intended (Figure 2).

Saving queries

Reflecting findings from our interviews, we add a query-saving feature so that users can save queries they want to keep track of. Users can flexibly define the query e.g. `data containing features X & Y`, or even manually select data of interest.

6 Conclusion & Future Work

Machine learning is a critical piece of today's technical infrastructure, but the fine-tuning of learned models is often either difficult or out-of-reach. For experts, a lack of mature tooling and labeled data makes model tuning difficult despite the technical knowhow. For non-ML experts, learned models are often a black box.

In paper explores a system intended to broker a productive compromise between learned models and rule-based systems. One in which users can take a trained model, inspect its performance on real-world data, and then refine its behavior by applying rules that can elect to override the model.

Rules enable explicit domain-specific knowledge to be added in environments without enough data to learn them implicitly. They allow error hot-spots to be quickly targeted in production environments while better models are built offline. And they provide a mechanism

for integrating model tuning with the traditional manner by which most programmers learn to approach problems.

In future work, we intend to further explore new categories of rules and objects that could be incorporated into our system, such as posterior constraints, feature boosting, and more complex predicates. We believe there is value in extending an interactive environment such as ours to model training and run-time monitoring.

References

1 Saleema Amershi, Bongshin Lee, Ashish Kapoor, Ratul Mahajan, and Blaine Christian. Cuet: human-guided fast and accurate network alarm triage. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 157–166. ACM, 2011.

2 Laura Chiticariu, Yunyao Li, and Frederick R Reiss. Rule-based information extraction is dead! long live rule-based information extraction systems! In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 827–832, 2013.

3 Seth Grimes. Text/content analytics 2011: user perspectives on solutions and providers. *Alta Plana*, 2011.

4 Sumit Gulwani and Prateek Jain. Programming by examples: Pl meets ml. In *Asian Symposium on Programming Languages and Systems*, pages 3–20. Springer, 2017.

5 Peter Hase, Chaofan Chen, Oscar Li, and Cynthia Rudin. Interpretable image recognition with hierarchical prototypes. *arXiv preprint arXiv:1906.10651*, 2019.

6 Hubspot, 2005. `https://www.hubspot.com`.

7 IFTTT. Ifttt gmail, 2010. `https://ifttt.com/gmail`.

8 jQuery QueryBuilder. jquery querybuilder, 2014. `https://querybuilder.js.org`.

9 Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. Model assertions for debugging machine learning. In *NeurIPS MLSys Workshop*, 2018.

10 Todd Kulesza, Simone Stumpf, Weng-Keen Wong, Margaret M Burnett, Stephen Perona, Andrew Ko, and Ian Oberst. Why-oriented end-user debugging of naive bayes text classification. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 1(1):2, 2011.

11 Gierad Laput, Walter S Lasecki, Jason Wiese, Robert Xiao, Jeffrey P Bigham, and Chris Harrison. Zensors: Adaptive, rapidly deployable, human-intelligent sensor feeds. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1935–1944. ACM, 2015.

12 Mailchimp, 2001. `https://mailchimp.com`.

13 Marketo, 2016. `https://www.marketo.com`.

14 Mikaël Mayer, Gustavo Soares, Maxim Grechkin, Vu Le, Mark Marron, Oleksandr Polozov, Rishabh Singh, Benjamin Zorn, and Sumit Gulwani. User interaction models for disambiguation in programming by example. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 291–301. ACM, 2015.

15 Microsoft. Microsoft flow, 2018. `https://flow.microsoft.com/en-us/`.

16 Besmira Nushi, Ece Kamar, and Eric Horvitz. Towards accountable ai: Hybrid human-machine analyses for characterizing system failure. In *Sixth AAAI Conference on Human Computation and Crowdsourcing*, 2018.

17 Hila Peleg, Sharon Shoham, and Eran Yahav. Programming not only by example. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 1114–1124. IEEE, 2018.

18 Stephan Rabanser, Stephan Günnemann, and Zachary C Lipton. Failing loudly: An empirical study of methods for detecting dataset shift. *arXiv preprint arXiv:1810.11953*, 2018.

19 Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. Data programming: Creating large training sets, quickly. In *Advances in neural information processing systems*, pages 3567–3575, 2016.

20 Donghao Ren, Saleema Amershi, Bongshin Lee, Jina Suh, and Jason D Williams. Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE transactions on visualization and computer graphics*, 23(1):61–70, 2016.

21 Marco Tulio Ribeiro, Sameer Singh, and 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*, pages 1135–1144. ACM, 2016.

22 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

23 Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206, 2019.

24 Cynthia Rudin and Yaron Shaposhnik. Globally-consistent rule-based summary-explanations for machine learning models: Application to credit-risk evaluation. *Available at SSRN 3395422*, 2019.

25 Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the gpdr. *Harv. JL & Tech.*, 31:841, 2017.