Simulated User Studies for Explanation Evaluation

Anonymous Author(s)
Affiliation
Address
email

Abstract

The goal of evaluating explanations is often to determine suitable explanations for a desired real-world use case. Traditionally, evaluation of explanations follows two steps: first, proxy metrics (an algorithmic evaluation based on desirable properties) and then, human user studies (an experiment with real users that puts explanations to the test in real use cases). The former is efficient to compute but is typically disconnected from the use case itself. Meanwhile, the latter is time-consuming to organize and often difficult to get right. We argue for the inclusion of a new step in the evaluation workflow that capitalizes on the strengths of both proxy metrics and user studies called *simulated user evaluations*, an algorithmic evaluation grounded in real use cases. We provide a two-phase framework to conduct simulated user evaluations and demonstrate that, by instantiating this framework for local explanations, we can use simulated user evaluations to recreate findings from existing user studies for two use cases (identifying data bugs and performing forward simulation). Additionally, we demonstrate the ability to use simulated user evaluations to provide insight into the design of new studies.

1 Introduction

There exists a gap between the desired real-world use cases of interpretability and the technical objectives for which interpretable machine learning (IML) methods are optimized [1]. As a result, it is difficult for (a) end-users to identify suitable IML methods for their use cases and (b) researchers to ground their technical developments in real applications. Towards bridging this gap, one would ideally conduct *user studies* in realistic settings (Figure 1, Right), and, indeed, a few recent works have started running these types of studies (e.g., [2–6]). While user studies are the gold standard for IML evaluation [7], they are often infeasible for most IML researchers to run because: (1) they are resource intensive, requiring access to real users, data, and an environment where one would typically find these users; (2) they are time-consuming to conduct, requiring the coordination of many participants; and (3) they are difficult to set up correctly in a way that is carefully controlled to isolate the effect of explanations, and consequently often yield noisy results.

Due to the difficulties of running user studies, explanations are more commonly first evaluated using *proxy metrics* (Figure 1, Left), which measure the extent to which an explanation captures seemingly desirable properties (e.g., faithfulness to model [8, 9], stability over runs [10, 11]). The algorithmic nature of proxy metrics is advantageous for researchers because they make experiments easy to run and reproduce, but such metrics fail to reflect desired aspects of the use case that would have been captured in a user study [1, 7].

To help contextualize this discussion, we provide a running example based on a use case studied in [3], where a researcher is interested in studying whether explanations can help ML engineers identify bugs in an ML pipeline. Specifically, the bug is that the data used to train the model may have been corrupted (e.g., a feature was imputed incorrectly). One one hand, the researcher could conduct an expensive user study, where the researcher would have to recruit many ML engineers to participate

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

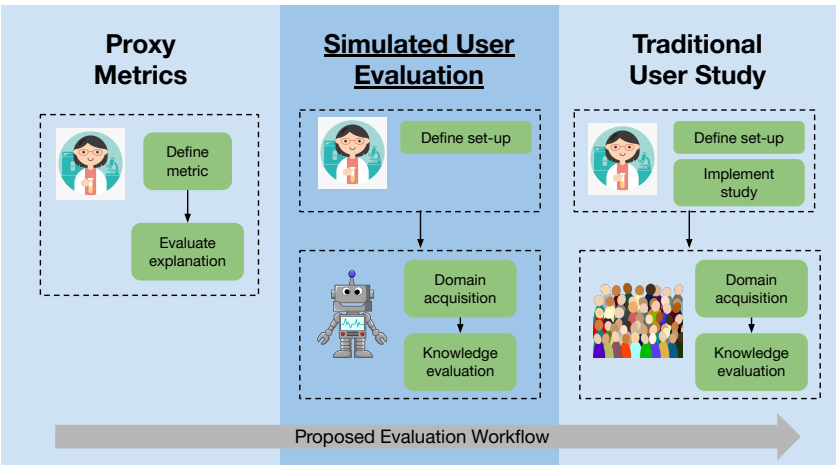

Figure 1: Our proposed IML evaluation workflow, including current, common evaluation steps (Proxy Metrics and Traditional User Study) as well as our new evaluation step *Simulated User Evaluations*. For the researcher, running simulated user evaluation is a natural preceding step to a traditional user study because defining a use-case grounded set-up is necessary for both. However, for a traditional user study, there is the additional step of implementing the study, which would entail recruiting participants and setting up a realistic use case environment for the participants to interact in. Note that the exact way that humans would acquire domain knowledge might differ from the agent.

and properly ensure that each engineer is shown explanations in a controlled setting. On the other hand, the researcher could try to evaluate using proxy metrics but would get stuck because there is no known proxy metric for the particular bug in which they are interested.

In this work, we argue for the addition of a new step in the evaluation pipeline called *simulated user evaluations* (Figure 1, Middle). Simulated user evaluations leverage the strengths of both proxy metrics and traditional user studies by being both algorithmic (consequently, fast and reproducible) and use-case specific (i.e., helpful for the end user). The goal of simulated user evaluation is to encode as much of the relevant context of the use case as possible (typically designed with domain expertise in mind) and then replace the human with an algorithmic agent. We argue that simulated user evaluation should be regarded as a foundational step toward building more efficient and reproducible evaluations of explanation methods.

While algorithmic agent may not learn in exactly the same way as a human would, the agent's success at using an evaluation may be indicative of a human's success. That is, simulated user evaluation can provide signal on which explanation methods should be considered for a specific use case. In our running example, the researcher runs simulated user evaluations before investing time and effort into a traditional user study. To do so, the researcher could enlist the help of some ML engineers to define a realistic set-up of the use study, which would include a dataset, a specification of the bug that describes how features are often imputed, and the training procedure for the model, and then run simulated user evaluations. Using these evaluation results, the researcher could select candidate explanation methods for the actual user study.

To make the idea of simulated user evaluations more concrete, our first contribution is a general two-phase framework, described in Section 3, to train an algorithmic agent to simulate an end user that uses explanations. This framework consists of step (1) *domain acquisition*, where the agent learns the relevant domain knowledge for the use case, and step (2) *knowledge evaluation*, where the agent is evaluated on unseen examples of the use case as a way to measure explanation utility.

Our second contribution, in Section 4, instantiates our general framework to evaluate the utility of *local explanations* (e.g. [8, 12]). In Section 5, we demonstrate that our framework is able to corroborate existing findings from two user studies papers [3, 4] related to local explanations for two use cases in an algorithmic fashion. We also showcase how, once the set-up has been done for a given use case, our framework can be used to easily provide additional insights. For example, we find that certain explanation methods perform just as well as the ones from the studies.

2 Related Work

Connecting Methods and Real-world Use Cases: [13] was one of the first to highlight the disconnect between the many goals of interpretability that are potentially discordant with existing methods, and as such [1, 14] stressed the need for more studies that explicitly study connections between methods and a specific downstream use case. However, evaluations to establish these connections come in various forms, as illustrated in the taxonomy of evaluations proposed by [7] where evaluations range in level of human involvement. These ideas are clarified in [1], which discusses the importance of distinguishing evaluations of faithfulness (i.e., conclusions drawn from proxy metrics) from evaluations of usefulness (i.e., conclusions found from user studies). These goals motivate the introduction of simulated user evaluations in the evaluation workflow.

User studies on explanations: There have been an increasing number of user studies including [2–5] which were conducted to understand whether explanations are helpful for humans in a variety of real-world use cases ranging from fraud detection [5] to model debugging [3]. The findings from these user studies are mixed, with some finding that explanations are not always helpful, particularly when compared to more naive baselines like providing the score or model prediction to the user. As such, we also consider these naive baselines in our study. We reproduce the general findings for use cases studied in [3, 4], but with an automated algorithmic approach, which also allowed us to dive deeper into the use case by performing follow-up (simulated) experiments.

Algorithmic Evaluation Frameworks: One direction of existing evaluation frameworks focus on desirable properties of explanations relating to faithfulness of the explanation to the model [15–17]. These evaluations are typically disconnected from the downstream use case of interest. In terms of algorithmic evaluations that are grounded in a use case, we were able to identify only a handful of such examples (e.g., [6, 18, 19]). In this work, we present a framework for performing a more general simulated user evaluation, and we demonstrate in Section 3 how these examples can be cast in our general framework.

While we study whether explanations can be helpful for a downstream use case, by simulating a user evaluation, another line of work studies whether explanations contain additional information [20–22], typically to improve the model training process. The most salient difference between these two paradigms is the former assumes that explanations might help humans on a use case that they might not have been able to do as easily otherwise while the latter does not include a human in the loop.

3 General Framework

We provide a general framework for training an agent that simulates a human using explanations for a downstream use case. Our two-part framework[1] is as follows:

First, our framework begins with the phase of *domain acquisition*. The domain acquisition phase is where the agent sees labeled training examples for the downstream use case as a way to learn information about the use case (e.g., what data-points and explanations look like with a bug or not) and also the dataset context. This step is motivated by our observation that, in user studies, participants are sometimes provided some examples to convey the relevant knowledge (e.g., presented example explanations and expected conclusions). Otherwise, they are assumed to already have the relevant knowledge for the use case.

Once the agent has learned the relevant knowledge, as evidenced by a high accuracy on the training examples, we move onto the second phase, *knowledge evaluation*. This phase is analogous to the main portion of typical user studies, where humans are put to the test on new examples to see if they are able to use a given explanation effectively. Similarly, our agent is evaluated on unseen test examples and its accuracy on these examples determines the utility of the explanation for the use case. We note that existing one-off examples of simulated user evaluations can also be cast under these two steps as described in Table 1.

With an algorithmic agent, the researcher would be able to more carefully control the evaluation process and get a "clean slate" in each condition that one would want to consider in the evaluation

[1] Note on terminology: in the following sections, we refer to the *framework* as the process to train the agent through these two phases and the *agent* as the simulated user.

Table 1: Unifying existing adhoc algorithmic evaluations under our framework. These algorithmic agents had considerably simpler domain acquisition processes, which were largely heuristic based as opposed to learning from data.

	Expo[6]	Influence Function[19], RPS[18]
Use Case	Determine how to modify instance to achieve target prediction	Fixing mislabeled examples
Domain Acquisition	Randomized, greedy heuristic to pick largest feature	Use heuristic such as pick a point with larger influence
Knowledge Evaluation	Number of modifications used to reach target prediction	Number of correct points selected to inspect

process. This would be like recruiting a new batch of participants but without dealing with per-subject variations, which might be a side-effect of different subjects in different conditions.

4 Simulated User Evaluations for Local Explanations

We begin by instantiating our general framework to study the utility of local explanations (a local explanation is one that assigns a score to each feature for each data-point.)

We apply our local explanation framework to two broad use cases: detecting data bugs and performing forward simulation. Both are set-ups are modeled exactly after ones considered in previous user studies [2–4]. In these user studies, the humans were generally allowed to look at multiple data-points (and their corresponding local explanations) because it helps with the use cases (e.g., it might be easier to detect that a model was trained with label errors if given more than one data-point). We also give our agent the ability to do this by defining the input as a *set* of data-point / explanation pairs.

The agent is given a dataset of size N of sets of data points and explanation scores (which produce a vector of length $2d$ where d is number of features) as training input. Then, we train our agent to predict an outcome related to the use case (e.g., Does the base model which produced these explanation have a bug? What is the base model's prediction for this X value?).

4.1 Data Bugs

We train an agent to predict whether a base dataset contains a systematic error. That is, given as input a set of data-points and corresponding explanations for each point, predict whether this set of data-points and explanations came from a model that was trained on a corrupted base dataset. This set-up is one that was studied in multiple user studies [2, 3].

We generate such a dataset of data-point sets by making half of the data-point sets have explanations from bugged models and the other half from unbugged models. To produce each data-point set, we first subsample the original base dataset and then add the bug to half of them. We produce explanations for each data-point set by training a base model on each subsampled dataset and extracting explanations for randomly sampled data-points. The reason why we subsample the base dataset is so the agent sees more variants of the use case, and potentially slightly different model behavior. Given this dataset of data-point sets, we split it into a train and test set, where the train set is provided to the agent in the domain acquisition phase and the test set is used to evaluate the agent in the knowledge evaluation phase. A more detailed algorithm is provided in Appendix A.

We consider bugs that were studied in prior user studies, where the base dataset has been tampered with or corrupted in some way (we provide the motivation for each from the original paper):

- **Missing value [3]:** Methods to deal with missing values can inject bias into the models (e.g. impute using the mean value of a feature).
- **Redundant feature [3]:** The same feature may be incorporated in several ways, and this can distribute importance among them.
- **Label Error [2]:** A few incorrectly labeled data-points can cause the model to learn wrong associations.

4.2 Forward Simulation

We train an agent to predict model behavior, with or without an explanation. This setting was motivated by [4], where the authors test whether providing explanations helps humans better predict model behavior than if no explanation is provided. Following the procedure from [4], we split the base dataset into train, test, and validation sets. In the domain acquisition phase, the agent sees data-points from the train set. Then, in the knowledge evaluation phase, the agent is evaluated on data-points from the validation set, which it has not seen before. We can compare the utility of different explanation methods for this forward simulation use case by measuring the agent's ability to predict the base model's output with explanation and without. A more detailed algorithm is provided in Appendix A.

5 Experiments

We provide experimental details for both use cases[2].

Base Datasets: Both papers for which we replicate results ran their experiments on the UCI Adult dataset[3]. For diversity of results, we additionally run experiments with three additional UCI datasets (Bank Marketing[4], Occupancy Detection[5], and Online Shopper Purchasing Intention[6]) for the label errors bug. In Appendix B.1, we discuss how each dataset was processed. All datasets considered had a binary target label and had at least $10,000$ points.

Base Models: For the data bugs use case, we use LightGBM [23] as the base model, following [3]. For the forward simulation use case, we use a Random Forest as the base model, following [4].

Agent Model: We leverage DeepSets [24], a model which is permutation invariant on its input, for the agent model. This property is desirable for our agent to have because the sets of data-points/explanations that are provided as input do not have a pre-defined ordering. The exact model architecture is provided in Appendix B.2. We use Adam with learning rate $1e-4$ for our optimizer and binary crossentropy for the loss function.

Explanations: For data bugs, we provide the model prediction as a baseline. For forward simulation, the baseline receives no additional information (i.e., only the data-point). We focus on feature-based explanation methods (e.g. LIME [8] and SHAP [12]). However, our framework can also handle global explanation methods, such as Anchors [25] and GAMs [26] which were considered in the user study, if a local explanation can be derived from them. We discuss how we convert Anchors and GAMs to local explanations in Appendix B.3.

Framework Hyperparameters: For data bugs, we generate a training set for knowledge acquisition of size 1000 and perform knowledge evaluation on a dataset of size 250. For forward simulation, we generate a dataset of up to size 2000 and evaluate on 250 new data-points.

5.1 Data bugs

Missing Values: The missing values bug was originally implemented by replacing the 'Age' value with the mean value of 38 for 10% of adults with > 50k income. We note that while we attempted to recreate the same setting used in [3], the exact hyperparameters of their LightGBM model and dataset preprocessing were not provided. Thus, we had to artificially increase the bugged percentage to 30% to reproduce the middle plot in Figure 1 from [3] (for more details, see Appendix B.4).

Sim-User-Eval Findings: Similar to [3], we found that SHAP and GAMs are good choices for identifying the missing values bug. Furthermore, as shown in Table 2, we find that accuracy of the algorithmic agent starts to drop if there are too few explanations (i.e., set size of 10 or less). This means that too few explanations do not contain sufficient signal to detect the bug. As such, the researcher would need to figure out how to present a large number of explanations to a human user in a human-interpretable manner. This suggests the need for a user interface where the explanation

[2] Anonymized code for the experiments can be found here.

[3] https://archive.ics.uci.edu/ml/datasets/Adult

[4] https://archive.ics.uci.edu/ml/datasets/Bank+Marketing

[5] https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+

[6] https://archive.ics.uci.edu/ml/datasets/Online+Shoppers+Purchasing+Intention+Dataset

Table 2: Recreating [3]'s missing values setting, showing that both explanation types were successful in finding the bug with high accuracy. We additionally vary the size of the set that is observed on the effect of test accuracy.

Explanation	1	10	100	1000	2000
SHAP	47.9%	83.4%	99.3%	99.8%	99.5%
GAM	60.4%	86.6%	99.6%	99.5%	99.8%
LIME	52.5%	51.3%	62.5%	82.1%	82.8%
Model Prediction	52.9%	57.5%	55.4%	56.2%	52.5%

Table 3: Performing additional experiments on the Adult dataset by varying the strength of the missing values bug for a fixed set size of 1000, where we observe that similar patterns as Table 2 hold in terms of what explanations were successful.

Explanation	5%	10%	15%	20%	30%
SHAP	77.9%	92.5%	99.1%	99.8%	99.8%
GAM	61.8%	97.5%	99.5%	99.1%	99.5%
LIME	52.9%	55.8%	61.2%	59.2%	82.1%
Model Prediction	51.7%	52.9%	53.5%	55.1%	56.2%

feature values are presented in an aggregated fashion, which is indeed what the approach taken by [3] for the user interface. Further, as seen in Table 3, we find that a researcher could make the bug more subtle and the explanations would still contain enough signal to indicate a bug.

Redundant Features: The redundant features bug was implemented by including two features, 'Education' and 'EducationNum,' that represent the same information. To create a version where there is no bug, we randomize the value of one of the features so there is no correlation between the two and only one feature contains the original information. Note that, in the original paper, they did not have a *strength* of the bug, but we are easily able to study this variant of the bug with our algorithmic framework. We increase the strength of the bug by increasing the number of data-points that have the two correlated features.

Table 4: We vary the strength of the redundant features bug on the Adult dataset for a fixed set size of 1000 and corroborate results from [3].

Explanation	10%	30%	50%	70%	90%
SHAP	61.2%	74.1%	86.2%	99.5%	99.5%
GAM	59.1%	56.2%	81.3%	99.8%	99.7%
LIME	57.1%	52.0%	55.6%	54.6%	51.3%
Model Prediction	57.9%	52.1%	51.5%	52.0%	52.9%

Sim-User-Eval Findings: Using our algorithmic agent, we found that the two explanations considered in the user study, SHAP and GAM, were good for finding the redundant features bug as shown in Table 4. This matches findings from [3], where a fraction of users were able to use these explanations to detect the bug. Additionally, we found virtually no benefit in using methods like LIME and the model prediction baseline to detect redundant features beyond random guessing.

Label Error: The label error bug was implemented by randomly flipping the label of a fraction of the dataset as motivated by [2]. We experimented with the same set-up by running the same experiment with a fixed percentage of label corruptions on multiple UCI datasets. We first used the Adult dataset to select a setting where the best explanations did reasonably well (but not perfect) to ensure a sufficiently difficult task: 1% and set of 5 explanations.

Sim-User-Eval Findings: We are able to draw the same conclusions as [2] (discussed in Appendix B.5). However, the experiments in [2] were on saliency maps for image datasets, and we found different results for the same type of bug in a tabular setting. As shown in Table 5, we find that explanations like SHAP or GAMs could be potentially useful for detecting label errors. Additionally, as shown in Table 6, the same explanation methods do not have the same utility for the same exact

Table 5: We study the label errors bug on the Adult dataset by varying the strength $\{1\%, 5\%, 10\%\}$ of the bug for sets of sizes $\{1, 5, 10\}$ (note that this bug required far smaller set sizes than the previous two data bugs).

Explanation	1%			5%			10%		
	1	5	10	1	5	10	1	5	10
SHAP	63.5%	81.2%	87.5%	92.5%	98.5%	97.9%	96.7%	99.2%	99.5%
GAM	68.8%	85.8%	86.7%	93.3%	99.2%	99.1%	95.4%	99.5%	99.8%
LIME	55.0%	62.5%	53.3%	58.7%	56.8%	56.7%	51.5%	55.8%	55.4%
Model Prediction	56.8%	55.8%	60.0%	53.3%	50.8%	52.1%	49.5%	57.5%	52.9%

Table 6: We compare the same instantiation (1% label error and set of 5) of the label error bug on four different UCI datasets and find that the same explanation method has varying utilities on different datasets.

Explanation	Adult	Bank	Shopping	Occupancy
SHAP	81.2%	71.6%	93.3%	99.5%
GAM	85.8%	66.7%	68.3%	99.6%
LIME	62.5%	53.3%	55.8%	58.3%
Model Prediction	55.8%	62.5%	54.5%	52.1%

bug on different datasets, so there is no single best explanation for all use cases. This further suggests the need for a researcher to first perform simulated user evaluations to guide the selection of candidate explanation methods for a real user study.

5.2 Forward Simulation

To recreate the original setting from [4], we compare the prediction accuracy of our agent on the 'No explanation' setting where it receives either 16 or 32 randomly sampled validation data-points without explanation with its prediction accuracy on 16 or 32 validation data-points when given an explanation. With our algorithmic framework, we were also able to explore what happens if the participants had more time to consider more examples (i.e., more training time) by increasing the number of training data-points from 16-32 up to 2000. However, a limitation of this additional question is that an algorithmic agent would have superior cognitive working capacities than humans, so in order to replicate our findings in a human study, the large number of explanations would need to be presented in a distilled or aggregated manner. We then tested on 3 different samples of 32 from the 250 validation points, where results are shown in Table 7.

Table 7: We vary the number of training data-points the agent receives to perform forward simulation, recreating results from [4] in the 16 and 32 column and providing additional results for increased number of training data-points. LIME and Anchors were explanations considered in the original paper while we additionally evaluated SHAP and GAM.

Explanation	16	32	100	1000	2000
LIME	89.3%	89.2%	91.9%	94.8%	95.6%
Anchors	81.9%	83.4%	96.7%	99.1%	99.4%
SHAP	91.2%	94.8%	97.1%	99.7%	99.5%
GAM	81.7%	86.2%	96.1%	98.0%	98.3%
No explanation	77.8%	78.6%	85.1%	91.2%	92.4%

Sim-User-Eval Findings: As shown in Table 7, LIME is a good choice for the forward simulation use case as an agent exhibits a significant increase in accuracy when using it over the baseline. Anchors is only a good choice if the agent (or human) has more training examples. This corroborates findings from [4], where they found that only LIME had a significant difference over the no explanation baseline when they provided 16 or 32 training examples. However, if they had performed simulated user evaluations, they would have noticed that SHAP could have also been a good option to try. While

SHAP is not traditionally an approximation method, its feature importance values can be used to inform a human's prediction.

6 Conclusion

In this work, we argued for the inclusion of simulated user evaluations as an integral part of the IML evaluation workflow. To make this new type of evaluation more concrete, we proposed a two-phase framework for describing the general simulated user evaluation problem, that endows an algorithmic agent with necessary domain knowledge and then evaluates it on the downstream use case. We then instantiated the framework for local explanations to evaluate the utility of a set of local explanations on two use cases based on real user studies. We demonstrate that, with this algorithmic agent, we are able to corroborate findings about which explanation methods are best able to help a human and, in general, how many explanations a human would need to see (i.e., the size of the set) to be able to solve the use case.

There are several exciting directions for future work on simulated user evaluations including running a simple user study to verify some of the hypotheses that arose from the algorithmic evaluation process on these two use cases, expanding the instantiated framework to include other modalities like images, and adding other explanation methods to the evaluated set.

References

[1] Valerie Chen, Jeffrey Li, Joon Sik Kim, Gregory Plumb, and Ameet Talwalkar. Towards connecting use cases and methods in interpretable machine learning. *CoRR*, abs/2103.06254, 2021. URL `https://arxiv.org/abs/2103.06254`.

[2] Julius Adebayo, Michael Muelly, Ilaria Liccardi, and Been Kim. Debugging tests for model explanations. In *NeurIPS*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/075b051ec3d22dac7b33f788da631fd4-Abstract.html`.

[3] Harmanpreet Kaur, Harsha Nori, Samuel Jenkins, Rich Caruana, Hanna Wallach, and Jennifer Wortman Vaughan. *Interpreting Interpretability: Understanding Data Scientists' Use of Interpretability Tools for Machine Learning*, page 1–14. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450367080. URL `https://doi.org/10.1145/3313831.3376219`.

[4] Peter Hase and Mohit Bansal. Evaluating explainable ai: Which algorithmic explanations help users predict model behavior? In *ACL*, 2020. URL `https://arxiv.org/abs/2005.01831`.

[5] Sérgio Jesus, Catarina Belém, Vladimir Balayan, João Bento, Pedro Saleiro, Pedro Bizarro, and João Gama. How can i choose an explainer? an application-grounded evaluation of post-hoc explanations. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 805–815, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383097. doi: 10.1145/3442188.3445941. URL `https://doi.org/10.1145/3442188.3445941`.

[6] Gregory Plumb, Maruan Al-Shedivat, Ángel Alexander Cabrera, Adam Perer, Eric Xing, and Ameet Talwalkar. Regularizing black-box models for improved interpretability. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 10526–10536. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/770f8e448d07586afbf77bb59f698587-Paper.pdf`.

[7] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[8] 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, 2016.

[9] Gregory Plumb, Denali Molitor, and Ameet Talwalkar. Model agnostic supervised local explanations. *arXiv preprint arXiv:1807.02910*, 2018.

[10] David Alvarez-Melis and Tommi S. Jaakkola. Towards robust interpretability with self-explaining neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 7786–7795, Red Hook, NY, USA, 2018. Curran Associates Inc.

[11] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3681–3688, 2019.

[12] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems*, pages 4768–4777, 2017.

[13] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[14] Maximilian Idahl, Lijun Lyu, Ujwal Gadiraju, and Avishek Anand. Towards benchmarking the utility of explanations for model debugging. *arXiv preprint arXiv:2105.04505*, 2021.

[15] Joon Sik Kim, Gregory Plumb, and Ameet Talwalkar. Sanity simulations for saliency methods. *arXiv preprint arXiv:2105.06506*, 2021.

[16] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. *arXiv preprint arXiv:1810.03292*, 2018.

[17] Mengjiao Yang and Been Kim. BIM: towards quantitative evaluation of interpretability methods with ground truth. *CoRR*, abs/1907.09701, 2019. URL `http://arxiv.org/abs/1907.09701`.

[18] Chih-Kuan Yeh, Joon Kim, Ian En-Hsu Yen, and Pradeep K Ravikumar. Representer point selection for explaining deep neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/8a7129b8f3edd95b7d969dfc2c8e9d9d-Paper.pdf`.

[19] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894, 2017.

[20] Danish Pruthi, Bhuwan Dhingra, Livio Baldini Soares, Michael Collins, Zachary C Lipton, Graham Neubig, and William W Cohen. Evaluating explanations: How much do explanations from the teacher aid students? *arXiv preprint arXiv:2012.00893*, 2020.

[21] Sayna Ebrahimi, Suzanne Petryk, Akash Gokul, William Gan, Joseph E. Gonzalez, Marcus Rohrbach, and trevor darrell. Remembering for the right reasons: Explanations reduce catastrophic forgetting. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=tHgJoMfy6nI`.

[22] Joseph D Viviano, Becks Simpson, Francis Dutil, Yoshua Bengio, and Joseph Paul Cohen. Saliency is a possible red herring when diagnosing poor generalization. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=c9-WeM-ceB`.

[23] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[24] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf`.

[25] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[26] Trevor Hastie and Robert Tibshirani. Generalized additive models: some applications. *Journal of the American Statistical Association*, 82(398):371–386, 1987.

A Use Case Algorithms

A.1 Data bugs

- Given base dataset and bug specification
- Let N be size of desired dataset, x be the set size
- For i in N
  - Subsample base dataset and split into train/test/validation
  - With probability 0.5, add bug to corrupt the subsampled dataset
  - Train the base model on the training set
  - Sample from the x points from the subsampled dataset
  - Generate an explanation for each of the x points
  - Let this set of data-points and their respective explanations be the ith entry in the final dataset

A.2 Forward simulation

- Given base dataset, split into train/test/validation
- Train the base model on the train set
- Let N_T be size of desired training dataset and N_V be size of the desired validation dataset
- Sample N_T points from the train set and also generate an explanation for each
- Sample N_V points from the validation set and also generate an explanation for each
- Repeat the same process without explanation for the baseline

B Experiments

B.1 Dataset preprocessing

In the adult dataset, we dropped the finalweight feature. In the occupancy dataset, we dropped the date feature. In the bank dataset, we dropped the poutcome and contact features. For all datasets, we first encoded categorical variables and MinMax scaled all features.

B.2 Deepset Architecture

The deepset architecture [24] is characterized by two parts, the feature extractor ϕ and then a standard network ρ, which takes in a summed representation from ϕ. Our ϕ network has a sequence of Dense layers of size ($2d$ where d is number of features, 200, 100), each followed by an ELU activation with the exception of the last layer. The ρ network is also a sequence of Dense layers of size (100, 30, 30, 10), each followed by an ELU activation with the exception of the last layer which is followed by a Sigmoid activation.

B.3 Converting Global Explanations to Local

For both GAM and Anchors, we used official Github implementations published by the authors. To convert GAM to a local explanation, there was already a method in the implementation to 'explain_local', so we used that out of the box.

The Anchors implementation works by returning a set of anchors for a given local instance. For categorical variables, the anchor is one of the values that the feature can take on. For continuous variables, the anchor could take the form of a lower and upper bound. As such, we needed to modify the input from $2d$ number of features to include one additional feature for each continuous variable. For example, if there are 5 continuous variables as in the Adult dataset, the input is now size $2d + 5$.

B.4 Missing Values Adjustment

Figure 2 and Figure 3 show our attempt to recreate the original bug described in [3] as closely as possible. We observe that the 30% bug for both explanation methods much more closely resembles the figure that was presented in the original paper.

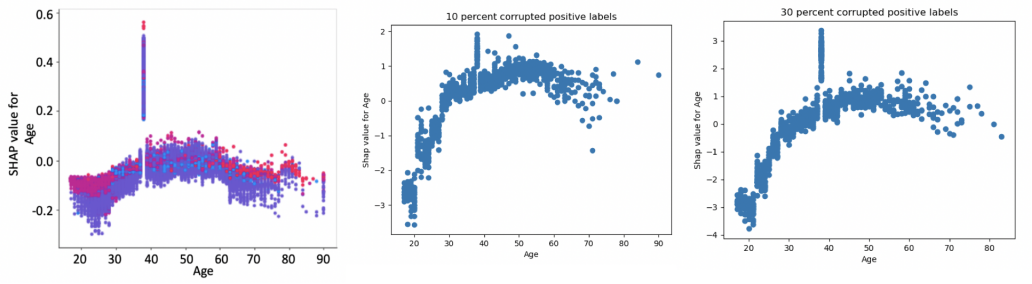

Figure 2: (Left) Image taken from the user interface from [3] that plots the distribution of SHAP values for the age feature (Middle/Right) Distribution of SHAP values for 1000 points for 10% and 30% corruption.

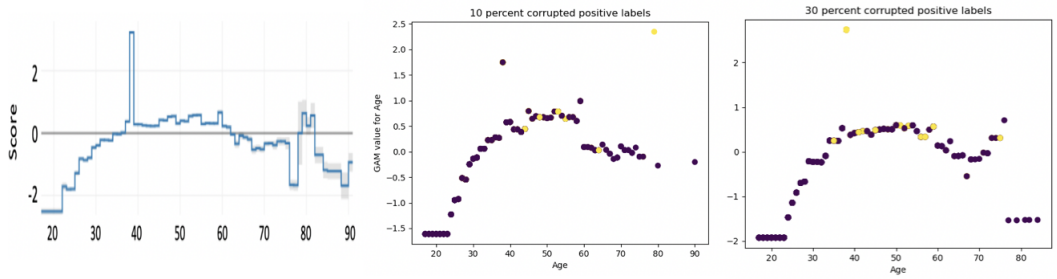

Figure 3: (Left) Image taken from the user interface from [3] that plots the distribution of GAM values for the age feature (Middle/Right) Distribution of GAM values for 1000 points for 10% and 30% corruption.

B.5 Image Results for Label Error

The experiments for [2] were performed on a dog breed dataset, where they considered a few different types of bugs. One in particular was a label error bug, where the dog breed label was randomized in the dataset before the base model was trained. For all explanation methods considered for this bug, participants said they were able to detect the bug overwhelmingly because of the incorrect label as opposed to using the explanation. Since the dataset modality is images as opposed to tabular, we modified the DeepSet architecture so that the ϕ network is a convolutional architecture. Following the experimental details otherwise from Section 5, we found that with 100% accuracy, the model prediction alone was sufficient for the agent to detect the bug. We plan to explore variations of the label error bug in the image setting to try to find a version where explanations might actually be useful.