Interpretable Feedback for AutoML and a Proposal for Domain-customized AutoML for Networking

Behnaz Arzani[‡], Kevin Hsieh[‡], Haoxian Chen[◇]
Microsoft[‡] and University of Pennsylvania[◇]

Abstract

The barrier to entry for network operators to use machine learning (ML) is high for operators who are not ML experts. Automated machine learning (AutoML) promises operators the ability to train ML models without requiring the expertise of data scientists or the need to learn ML. However, AutoML today: (a) is black-box; and (b) does not allow operators to leverage domain expertise. We start this paper by describing our broader vision for a domain-customized AutoML platform for networking and propose a set of potential solutions to realize that vision. As the first step, we introduce our feedback solution for AutoML that allows domain experts (who are not experts in ML) to better understand how to improve the input data to AutoML in order to achieve better accuracy.

ACM Reference Format:
Behnaz Arzani[‡], Kevin Hsieh[‡], Haoxian Chen[◇] . 2021. Interpretable Feedback for AutoML and a Proposal for Domain-customized AutoML for Networking. In *The Twentieth ACM Workshop on Hot Topics in Networks (HotNets '21), November 10–12, 2021, Virtual Event, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3484266.3487373

1 Introduction

Prior work demonstrate ML can provide remarkable benefits to production systems [7, 11, 12, 15, 20, 31, 55]. In many cases, simply employing traditional ML techniques instead of complex deep neural netowrks (DNNs) achieves significant gains [6, 7, 15, 20]. While the end-to-end benefits remain an open question, ML is a candidate for solving many networking use-cases. Unlike the heuristics (often approximations of integer programs) operators used in the past, ML can adapt over time to new conditions through retraining, leverage historical data to make optimized decisions based on past observations, and learn complex properties [11, 12, 31].

HotNets '21, November 10–12, 2021, Virtual Event, United Kingdom
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-9087-3/21/11…$15.00
https://doi.org/10.1145/3484266.3487373

However, building models for solving problems in networking often requires careful domain customization [6, 35, 36]. Today, this requires someone with expertise in both domains (networking and ML) to select the model, engineer features, identify corner cases where the model can fail, and optimize the model for performance and scale [1, 6, 15, 26, 34, 36, 51, 52, 55]. Such resources are scarce and the cost, in time and human resources, of tackling such problems is high. This is because of the scale, diversity, and complexity of the data we collect [5, 6, 42, 43, 57], the pace at which our networks evolve, and the number of questions we look to answer. Our conversations with operators of our large public clouds indicate the lack of operators with joint expertise is a bottleneck when building ML models to solve problems in networking. Others have made similar observations [1].

AutoML systems [18, 41] can address this problem and our community's early use of them show they can provide benefit [23]. However, they fall short on their promises because of two major shortcomings in existing AutoML systems:
(1) They do not allow operators to leverage their domain expertise. Our community's experience in developing ML solutions for networking problems indicates the significant role domain-expertise and domain-customization can play in building more effective ML solutions. For example, Park [35] leverages domain knowledge to reduce the state or action space for reinforcement learning solutions. PrivateEye [7] utilizes domain knowledge when engineering features which help it scale. Existing AutoML solutions, unfortunately, do not allow users to incorporate such domain knowledge.

Furthermore, priors play a crucial role in improving the accuracy of many ML models. Examples include: independence assumptions encoded in graphical models such as Bayes-Nets [39], kernels used in Gaussian processes [21], and prior distributions used for maximum likelihood estimations [40]. Current AutoML systems fail at efficiently leveraging such priors: they either try to greedily search the prior space [21] (at the cost of potentially using a sub-optimal one) or omit such models entirely [18, 41] at the cost of forgoing the benefits of such models in practice [2, 16, 28, 29].

A domain customized wrapper can allow operators to encode implicit or explicit priors into AutoML solutions. Take, as an example, the scenario where an operator *explicitly* specifies independence assumptions across features. A straw-man solution can take those assumptions and remove edges from the full-mesh Bayes-Net graph or alternatively add zeros in the covariance matrix for maximum likelihood estimators

with Gaussian priors. The wrapper can also *infer* these relationships from the operator's input. For example, the network's logical and physical topology can be an *implicit* indicator of such relationships. Given these *modified* models the AutoML framework can then include them in its search.
(2) They are black box. When AutoML fails to produce an accurate model, the user has no idea how to move forward. Even though many techniques exist to interpret what ML models have learned (e.g., [3]), these techniques do not show the user what to fix to improve the model's accuracy. Therefore, despite the many advances in ML and AutoML, non-ML experts find it challenging to use AutoML in practice [8, 20]. The complexity of AutoML, and the fact that its users are novices in ML increases the need for a feedback solution.

A feedback mechanism for AutoML should be able to detect when the problem is infeasible, what features may need to be changed, and what additional data can help the accuracy of the model. It is important for the non-ML expert to understand the justification for such feedback and to leverage their domain knowledge to address the issues identified.

Thus, we believe a domain-customized AutoML system with *interpretable* feedback can bridge the gap between network operators and AutoML. Realizing this vision requires extensive research in ML and networking. We take a first step in addressing the second problem (black box) and we defer the first problem (domain customization) to future work. Our solution focuses on a specific type of interpretable AutoML feedback: suggesting *new data points* the user can add to the training data in order to improve the accuracy of the model produced by the AutoML. Our contributions are as follows:

- We propose a vision on a domain-customized AutoML system with interpretable feedback for networking.
- An interpretable feedback solution for AutoML. To the best of our knowledge, this is the first work that proposes a feedback mechanism for AutoML that suggests new data points to improve AutoML accuracy. Our feedback solution is an interpretable data suggestion algorithm which targets a domain-expert with limited ML background. It not only provides explanations for the data being suggested (that non-ML experts can understand) but also in some cases outperforms existing active learning solutions.

Our solution is only the first step toward interpretable AutoML feedback. Other types of interpretable feedback such as feature modifications and identifying confounding variables require further research. We also need research by the HCI community to help with how best to present the outcomes of this feedback to a user with little to no ML background. We invite the community to help further this research and to help develop domain-customized AutoML solutions for networking that our operators can reliably use in practice.

2 Background and motivation

We first motivate our solution using running examples and then introduce relevant background material.

2.1 Motivation and Running Examples

The motivation for AutoML systems such as AutoSklearn [18] and TPoT [41] is to enable non-ML experts (such as network operators) to develop ML models for their domain-specific problems. AutoML has the potential to address a key bottleneck of developing and deploying ML solutions in production networks, which is the lack of ML expertise.

But the lack of a proper feedback mechanism for AutoML limits its usability: when it fails to produce a model with sufficient accuracy (the accuracy required by the domain expert for using the solution in deployment), the network operator is left with no follow up protocols — they would either need to acquire a deeper understanding of ML or to revert to hiring a data scientists to assist them in model development. Even ML experts may find it difficult to improve the output of AutoML as these systems typically produce an ensemble of models instead of a single model that is easier to understand. We use two running examples in this paper to illustrate both the need and the utility of interpretable feedback for AutoML:
(1) Detecting DDoS traffic using flow-level measurements. An operator wants to identify DDoS traffic and provides per-flow features such as: source and destination ports, source and destination IP addresses, and the number of packets/bytes sent and received. The "Internet Firewall Data Data Set" from the UCL KDD archives [25] is one such dataset.

In this example, the operator finds that the AutoML model fails to predict the DDoS traffic accurately and the model blocks a large fraction of legitimate customer traffic.
(2) A toy example in Congestion Control (CC). We construct a simple example where a developer would like to decide whether to use the Scream congestion control protocol [27]. The Scream congestion control algorithm was designed for latency sensitive applications but may not always be the best protocol. Our problem is to identify whether the application should use Scream to achieve the lowest end-to-end latency given the current network conditions. Although this is a toy problem, it is a problem that application developers encounter often in practice: picking the right congestion control protocol to use can be critical for achieving good performance [54] and depends on the properties of the network (e.g., queue sizes, bottleneck bandwidths, . . .).

In this example, the developer (who has limited ML background) provides AutoML with training data that identifies when Scream outperforms other congestion control protocols based on the network properties (bottleneck bandwidth, latency, loss rate, and number of concurrent flows). However, she finds that the AutoML model fails to predict the cases where Scream is not the best protocol.

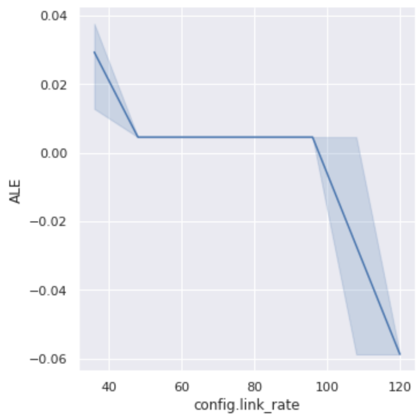


Figure 1: An ALE plot for a problem deciding whether to pick the Scream protocol.

Benefits of interpretable feedback. Figure 1 depicts how the network link rate affects the ML model decision for our second running example. In this figure, we plot Accumulate Local Effect (ALE), which is a technique that explains how features influence a ML model (§2.3). There is a lot of variance in ALE when link rates are low or high, which suggests the models in the AutoML ensemble cannot agree on these feature ranges. The operator can then try to add more data points with these feature ranges to improve AutoML accuracy.

Our solution returns per-feature feedback which by itself provides interpretability: a user's domain-knowledge of each feature allows them to decide which parts of the feedback to ignore and which to accept. But is the output of Figure 1 itself truly interpretable for a non-ML expert? The intuition for the solution helps explain why the answer is yes:

Imagine a scenario where *n* humans were blind-folded and asked to learn whether a given animal was a cat or a rabbit. These humans are presented with multiple examples of cats and multiple examples of rabbits and are then asked what they would think if they were presented with an animal whose ears were 3 inches long? Disagreement between the humans would indicate they were confused about whether 3 inch ears are more common in rabbits or cats — more examples of animals with 3 inch ears may help reduce the confusion.

Similarly, the values at each point of an ALE plot show what a black-box ML model has learned about the relationship between the value of a feature and the label it is trying to predict e.g., long ears are more indicative of a "rabbit" whereas shorter ears are more indicative of a "cat". The variance of ALE-values across different ML models at a given value of a feature is indicative of what these models have learned is similar/different: bigger differences indicate that particular feature at that point is "confusing" and more examples may help in reducing that confusion. This intuition allows non-ML experts to interpret the output of such feedback solutions.

2.2 Query by Committee

Query-by-Committee (QBC) [19, 47] is a popular active learning strategy that aims to minimize the data labeling cost for training. QBC proposes the use of a bag of (sufficiently diverse) ML models to label individual candidate points. The solution returns the data points which resulted in the most *disagreements* across these ML models, and labeling these data points are more likely to improve ML model accuracy.

QBC has synergies with AutoML solutions such as AutoSKlearn [18] and TPoT [41], which create an ensemble of ML models. A key assumption of QBC is that the committee consists of a number of sufficiently diverse ML models, which are non-trivial to identify even for ML experts [45]. However, this assumption can be met with AutoML systems that create an ensemble specifically when the models are individually strong and make uncorrelated errors [18]. Hence, we can re-purpose the models within the AutoML ensemble itself to form a QBC committee. Several challenges remain:
(1) Requiring candidate data points. QBC requires unlabeled candidate data points as input. The bag of models then votes on them to produce a set of points that are most promising. In the context of feedback for AutoML, if the user provides a set of candidate points they can bias the output based on their choice and limit the effectiveness of the data the solution produces. Alternatively, one could create a candidate set by sampling based on the distribution of the training set. However, the training set too can suffer from the same biases: in the context of our CC example, the developer may collect data from production and miss observing unique cases that only occur when the loss rate of the network is higher due to failures or congestion. Our solution does not rely on an input candidate set and provides samples from a subspace of the set of all valid feature values to the user.
(2) They are not interpretable. QBC and other active learning solutions are not interpretable by an operator who is unfamiliar with concepts such as decision boundaries and model confidence. Instead of using the model predictions, we use model-agnostic interpretation algorithms (such as ALE, §2.3) and the variance of the interpretations as a measure of disagreement. Thus, by returning the "aggregate interpretations" along with the standard deviation the user can understand how each feature contributes to the overall disagreement among the ML models in the AutoML ensemble.

2.3 ALE plots

Accumulate Local Effect (ALE) plots describe how features influence the prediction of a ML model on average [4] (See [4, 38] for more details). ALE computations result in a figure similar to Figure 1 and allow the user to understand what the ML model has learned about that feature.

3 Algorithm

The intuition for our algorithm is that the accuracy of AutoML systems such as AutoSKlearn and TPoT is in part due to their use of ensembles which contain a set of diverse ML models with uncorrelated errors [18]. Thus, these models are suited for QBC-like algorithms. In order to provide interpretability, we change our metric for model disagreements from prediction disagreements to the variance across their ALE plots. ALE plots (and other model-agnostic interpretation methods) are designed to provide insights to the user about what the model has learned. Through this simple change to QBC we are able to provide additional insights to the user about why they should sample those additional data points and add them to the training set. Our solution is as follows:

(1) The algorithm takes as input the ensemble produced by AutoML $\mathcal{M}$, a threshold $\mathcal{T}$ setting the variance we are willing to tolerate, the feature-set X and the domain of each feature in that set: $R(X_S)$ for each $X_S \in X$ (the range of values each feature can take in $\mathbb{R}$).

(2) The output of this algorithm is a set $\mathcal{K}$ of data points for the user to label and add to the training set.

(3) For each model in $\mathcal{M}$ we apply a model-agnostic interpretation algorithm. We use ALE in this work.

(4) Compute the standard deviation across the ALE values of models in $\mathcal{M}$ for feature $X_S \in X$ in its range $R(X_S)$.

(5) Return the subspace where the standard deviation is high (higher than $\mathcal{T}$) as the region for the user to sample more points from. These subspaces are essentially a collection of hyperplanes $\cup_i A_i x \leq b_i$ where x is an $| X | \times 1$ variable vector representing the features and A_i are $m \times | X |$ matrices of constants ($| X |$ is the cardinality of the set X). Similarly, b_i are $m \times 1$ vectors of constants. The magnitude of m depends on the regions where the standard deviation exceeds $\mathcal{T}$. The equations in $A_i x \leq b_i$ describe the regions in the ALE plot where the variance is higher than $\mathcal{T}$. Note, we need a union of $A_i x \leq b_i$ subspaces because the space need not be continuous: in our example in Figure 1, if we assume the bottleneck bandwidth is the only feature where the variance exceeds the threshold, our feedback returns $x \leq 45 \cup x \geq 99$ where x is the config.link_rate feature. The user can now sample more points from these regions and add them to the training set.

(6) Return the average ALE plots (along with error-bars) as explanations to the user (for example, see Figure 1).

Algorithm variants. A variant of the algorithm above runs AutoML multiple times and uses the disagreements across AutoML runs for feedback. This variant replaces the ensemble $\mathcal{M}$ with a set $\mathcal{M}'$ where each $m \in \mathcal{M}'$ is an ensemble returned by an AutoML run. AutoML runs are intrinsically non-deterministic due to their probabilistic search algorithms and thus, each run produces a different bag of models: this approach is more robust as it creates a more diverse bag of models but also is more expensive as each AutoML run can take a long time. We refer to this variant as Cross-ALE feedback and the earlier version as Within-ALE feedback. The Cross-ALE solution has an additional benefit: it allows us to extend our feedback solution to non-ensemble based AutoML systems. In our evaluations we use 10 AutoML runs for $\mathcal{M}'$.

4 Evaluation

We first describe the experimental setup. We aim to answer:

(1) Can our feedback solution help operators improve the accuracy of AutoML systems?

(2) How does our feedback solution work when the user has complete control and can collect any data the feedback solution specifies vs when the pool of data available to the feedback system is fixed?

(3) How does our feedback solution compare against active learning solutions that also suggest new labeled data that can help improve the accuracy of ML models?

(4) How does our solution compare against other, traditional, data-scientific approaches?

Implementation. We use AutoSKlearn [18]. We implement our solution with ~ 500 lines of python code. Each training experiment runs AutoML for an hour. We repeat each experiment 10 times, and with 20 different test sets, to ensure statistical significance across runs. For our ALE-based solutions, we uniformly sample from the regions of the ALE-plot that exceed the variance threshold and add those points to the training set. This evaluation represents the lower-bound performance of our solution as users with domain knowledge can potentially do better than uniformly sampling.

Datasets. We use two seperate datasets:

Scream vs rest. This is our example from §2. We use the Pantheon emulator[54] to get the target performance (label) for a given network condition (feature variables).

Because we collect the data through emulation, we can easily collect any additional data the feedback solution specifies. We use 1161 for training the initial AutoML system and add an additional 280 points based on the feedback. We divide an additional 4850 data points into 20 (roughly) equally sized test sets at random (to measure statistical significance). We also collect 2000 data points uniformly at random as the candidate-set for our active learning benchmarks.

UCL dataset. To show the benefit of the interpretability of our solution we use the "Internet Firewall Data Data Set" from the UCL KDD archives [25]. We divide the data and use 40% for training (26212 trining samples), 20% for testing — which we further divide into 20 test sets for statistical significance, and the remaining 40% as our candidate feedback pool. To ensure we also measure statistical significance across different data splits, we also repeat this splitting 5 times and report the

Algorithm (X)	balanced accuracy	P(no feedback, X)	P(X, within ALE)	P(X, cross ALE)
Without feedback	68.7% ± 4.05%	NA	0.0009	3.33×10^{-6}
Within-ALE	71.2% ± 4.3%	0.0009	NA	1.66×10^{-5}
Cross-ALE	75.0% ± 4.4%	3.33×10^{-6}	0.99	NA
Uniform	64.1% ± 4.1%	0.99	4.02×10^{-5}	7.86×10^{-6}
Confidence based	67.1% ± 5.5%	0.99	2.38×10^{-7}	2.38×10^{-6}
Upsampling	76.7% ± 2.7%	2.38×10^{-7}	1	0.99
QBC	68.9% ± 5.1%	0.093	0.004	2.38×10^{-7}
Within-ALE-Pool (180 points)	67.4% ± 4.9%	0.99	7.86×10^{-6}	4.76×10^{-7}
Cross-ALE-Pool (91 points)	69.18% ± 3.9%	0.123	0.013	0.013

Table 1: Scream vs rest balanced accuracy. We use P(x,y) to show the p-values. The Within-ALE-Pool and Cross-ALE-Pool algorithms are the variants of the ALE approach. All the other algorithms add 280 points to the training set.

results across all these scenarios. This also is a multi-class classification problem with 4 classes.

Metrics. We compute *balanced accuracy* over each test set as our accuracy measure. We use this metric to avoid biases due to label imbalance. To measure statistical significance, we use the p-values reported by the one-sided wilcoxon signed ranked test [53]. We use an $\alpha = 5\%$.

Benchmarks. We use the following benchmarks:

Uniform. We uniformly sample from the feature space the same number of points that we use for the ALE feedback and add it to the training set as the simplest baseline.

Confidence-based feedback. We compare against one of the most commonly used active learning solution, confidence-based sampling [32, 50]. We use the prediction probability returned by AutoSKlearn as a measure of confidence and, from a uniformly sampled candidate pool (2000 uniformly sampled points for the Scream vs rest dataset), return the points with the least confidence as feedback. We fix the number of points we return to the user so that it is the same as the number of points returned by the ALE feedback.

QBC for AutoML. We also compare against QBC. We modify QBC so that it uses the models in the AutoML ensemble as the committee instead of creating a curated ensemble which is itself difficult [22, 45]. The main diffrence between this approach and ours is in using ALE-variance instead of entropy. Here, we use the same candidate pool as the one we use for the confidence-based feedback.

Upsampling. The first dataset suffers from label imbalance. In that instance we compare our solution to a standard data-science solution to label imbalance, upsampling [13].

4.1 Results over the Scream vs. rest dataset

We first describe our results over the scream vs rest dataset (Table 1). We also report the statistical significance (p-values) of the one sided Wilcoxon signed ranked test where the alternate hypothesis is that the non-ALE approach has less balanced accuracy compared to the ALE-based approach. We see our ALE based approaches outperform both active learning approaches with high statistical significance.

This type of feedback is what the ALE-based feedback is designed for — the user has the ability to gather more data based on the feedback and provide the necessary labels. Our intuition is that the ALE-based solutions outperform the other baselines because these baselines are limited to the unlabeled candidate data pool provided to them: they do not suggest new candidate points themselves but instead are only able to assign a score to any unlabeled data point in the given pool. In contrast, ALE suggests the entire subspace of samples which are likely to improve AutoML's accuracy.

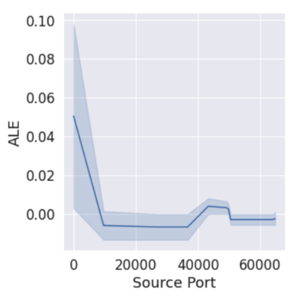
To check this hypothesis we also use the same candidate pool as the one used for the active learning baselines for the Cross-ALE and Within-ALE approaches (these are the Within-ALE-Pool and Cross-ALE-Pool algorithms in Table 1). Because we limit the algorithms to the candidate pool, we are not able to use the same number of points compared to the other benchmark active learning solutions, which significantly disadvantages our ALE-based approaches (we show the number of data points we add to the training set in parenthesis). The performance of the ALE-based algorithms drops significantly and becomes comparable to the other active learning solutions. Even in these scenarios, the ALE approach is comparable to the other active learning solutions and is preferred as a feedback solution to AutoML due to its interpretability.

Upsampling the training set outperforms all other approaches: upsampling deals with the fundamental underlying problem with the training set (label imbalance). Even in this case the Cross-ALE is on average within 1% of upsampling.

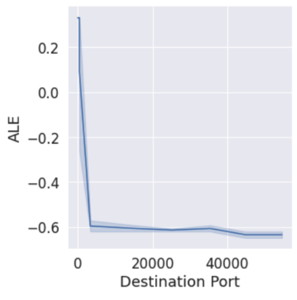
Finally, we see that, as expected, Cross-ALE outperforms Within-ALE. However, using Cross-ALE is more costly as the user will have to run the AutoML system multiple times.

4.2 Results over UCL dataset

We use this dataset to illustrate the interpretability of our ALE solution and summarize the overall accuracy numbers as follows: we find that ALE based feedback improves accuracy with statistical significance compared to the raw training data (P-value is 0.02 and 0.04 for Within-ALE and Cross-ALE respectively). Our baselines slightly (1-2% on average) outperform ALE, although without statistical significance (i.e., we cannot accept the alternate hypothesis that these approaches are better). However, we show next the solution provides significant benefit to operators by being interpret-able.



(a) Unpredictive feature



(b) Predictive feature

Figure 2: Example ALE plots from the UCL dataset 2

We illustrate the benefit of the interpretability of ALE in Figures 2a and 2b. As Figure 2a shows, the source port feature shows high variance especially around lower values. Based on domain knowledge, the operator knows source ports are typically assigned by the host kernel and though sometimes informative, are expected to be more noisy than other features. Alternatively, we see high variance across the destination port range 443-445 (Figure 2b). Port 443 is reserved for HTTPs which is one of the most common targets of distributed denial-of-service (DDoS) attacks [7]. Given these two insights, the operator can discard the bound on the source port and focus on collecting more data around the 443 destination port. In contrast, existing active learning solutions only choose a set of data points for labeling, and the user (1) has no idea why these points are needed; and (2) cannot leverage domain knowledge to determine the weaknesses of original data.

Setting the threshold. Our solution's only hyper-parameter is the threshold $\mathcal{T}$. In our experiments we used the median of the standard deviation across features: 0.02 for the scream vs rest and 0.01 for the two UCL datasets. Lower thresholds result in larger feature subspaces — a larger area for the user to sample — whereas larger thresholds result in smaller subspaces. It is important to consider the users' sampling budget to set the threshold. When the budget is high, it is better to set the threshold lower: larger feature subspaces are more likely to help prevent overfitting. In contrast, when the sampling budget is low, a higher threshold may be better as it helps focus sampling on feature subspaces that are more likely to fall on the decision boundary.

5 Discussion and Limitations

Our work presents an interpret-able feedback solution for AutoML where operators can *prioritize bounds containing features they know can influence the label* or tune the threshold they use for each feature based on their domain knowledge. Our approach also comes with the following limitations:

Bounded, at best, by the theoretical guarantees of QBC. Our feedback solution is based on QBC. While we do not provide a theoretical analysis of our solution, in principle, the theoretical analysis of QBC should apply here as well. However, it needs to be extended to account for our use of ALE instead of predictions as well as for the fact that we apply the feedback en-mass (as opposed to sample by sample).

Not applicable to AutoML systems that do not return an ensemble of models. Our solution takes advantage of AutoML systems such as AutoSKlearn [18] and TPoT [41] that return an ensemble of ML models. Some of the commercial AutoML offerings such as [9] also return an ensemble of models. Thus, there exist a sufficiently diverse set of AutoML systems where our solution is applicable. However, there are many AutoML systems where this is not the case e.g., [49]. In such cases, we would need an automated mechanism to find a sufficiently diverse set of ML models before we can return feedback to the user. This is a topic of future research.

6 Related Work

Active learning. Decades of research in active learning established many strategies to suggest data for labeling so that the learning algorithm can perform better with less training [45]. Popular active learning strategies include uncertainty sampling [32], QBC [47], expected model change [46], expected error reduction [44], and variance reduction [56]. Many works propose learning active learning strategies [10, 14, 17, 24, 30, 48]. However, existing active learning solutions have two key shortcomings that limit their usability for AutoML: (1) they do not provide any insight on why/how they suggest to label a particular set of data samples; and (2) most active learning solutions are based on the characteristics of a *single* and *fixed* ML model. Our algorithm is specifically designed for users of AutoML without much ML expertise. Hence, our algorithm differs from prior active learning work in that: (1) it provides interpretable data suggestions so that the users can understand the weaknesses of the training data; and (2) it naturally supports dynamic ML model and hyperparameter selection, which is the main focus of AutoML.

Interpretable machine learning. Interpretable ML enables humans to understand the cause of a decision by ML models [33, 37]. In contrast, our work (1) provides interpretable new data suggestions to AutoML users, and leverages a ML interpretation method to provide such suggestions; and (2) is the first that proposes using ALE to interpret AutoML pipelines.

7 Call to Further Research

We have found through our experience in both designing models for production networks; and in talking to our operators: a domain-customizable AutoML platform accompanied with interpretable feedback can make a huge difference in operators' ability to leverage ML. We took the first step and showed it is possible to create interpretable feedback solutions. We invite the community to help us further this research.

8 Acknowledgements

The authors would like to thank the anonymous reviewers and their shepherd Nick Feamster for their useful feedback on this work. We would also like to thank Srikanth Kandula for his feedback on the early drafts of this publication.

References

[1] The good, the bad, and the ugly of ml for networked systems. https://www.microsoft.com/en-us/research/video/the-good-the-bad-and-the-ugly-of-ml-for-networked-systems/.

[2] N. Agarwal, S. Jabin, S. Z. Hussain, et al. Analyzing real and fake users in facebook network based on emotions. In *2019 11th International Conference on Communication Systems & Networks (COMSNETS)*, pages 110–117. IEEE, 2019.

[3] D. Apley. Aleplot: Accumulated local effects (ALE) plots and partial dependence (PD) plots, 2017. *R package version*, 1, 2017.

[4] D. W. Apley and J. Zhu. Visualizing the effects of predictor variables in black box supervised learning models. *CoRR*, abs/1612.08468, 2019.

[5] B. Arzani, S. Ciraci, L. Chamon, Y. Zhu, H. H. Liu, J. Padhye, B. T. Loo, and G. Outhred. 007: Democratically finding the cause of packet drops. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 419–435, 2018.

[6] B. Arzani, S. Ciraci, B. T. Loo, A. Schuster, and G. Outhred. Taking the blame game out of data centers operations with netpoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 440–453. ACM, 2016.

[7] B. Arzani, S. Ciraci, S. Saroiu, A. Wolman, J. Stokes, G. Outhred, and L. Diwu. PrivateEye: Scalable and privacy-preserving compromise detection in the cloud. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[8] B. Arzani and B. Rouhani. Towards a domain-customized automated machine learning framework for networks and systems. *CoRR*, abs/2004.11931, 2020.

[9] https://azure.microsoft.com/en-us/services/machine-learning/automatedml/.

[10] Y. Baram, R. El-Yaniv, and K. Luz. Online choice of active learning algorithms. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2003.

[11] W. H. Beluch, T. Genewein, A. Nürnberger, and J. M. Köhler. The power of ensembles for active learning in image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9368–9377, 2018.

[12] L. Bottolo, S. Richardson, et al. Evolutionary stochastic search for bayesian model exploration. *Bayesian Analysis*, 5(3):583–618, 2010.

[13] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *J. Artif. Intell. Res.*, 16, 2002.

[14] H. Chu and H. Lin. Can active learning experience be transferred? In *Proceedings of IEEE International Conference on Data Mining (ICDM)*, 2016.

[15] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 153–167. ACM, 2017.

[16] B. Eriksson, P. Barford, J. Sommers, and R. Nowak. A learning-based approach for ip geolocation. In *International Conference on Passive and Active Network Measurement*, pages 171–180. Springer, 2010.

[17] M. Fang, Y. Li, and T. Cohn. Learning how to active learn: A deep reinforcement learning approach. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

[18] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. Auto-sklearn: efficient and robust automated machine learning. In *Automated Machine Learning*, pages 113–134. Springer, Cham, 2019.

[19] Y. Freund, H. S. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Machine learning*, 28(2), 1997.

[20] J. Gao, N. Yaseen, R. MacDavid, F. V. Frujeri, V. Liu, R. Bianchini, R. Aditya, X. Wang, H. Lee, D. A. Maltz, M. Yu, and B. Arzani. Scouts: Improving the diagnosis process through domain-customized incident routing. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM)*, 2020.

[21] Z. Ghahramani. The automatic statistician. 2014.

[22] R. Gilad-Bachrach, A. Navot, and N. Tishby. Query by committee made real. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2005.

[23] J. Holland, P. Schmitt, N. Feamster, and P. Mittal. New directions in automated traffic analysis. *arXiv preprint arXiv:2008.02695*, 2020.

[24] W. Hsu and H. Lin. Active learning by learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

[25] https://archive.ics.uci.edu/ml/datasets/Internet+Firewall+Data.

[26] N. Jay, N. H. Rotman, P. Godfrey, M. Schapira, and A. Tamar. Internet congestion control via deep reinforcement learning. *arXiv preprint arXiv:1810.03259*, 2018.

[27] I. Johansson and Z. Sarker. Self-clocked rate adaptation for multimedia. *RFC*, 8298:1–36, 2017.

[28] S. Kandula, D. Katabi, and J.-P. Vasseur. Shrink: A tool for failure diagnosis in ip networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 173–178, 2005.

[29] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed diagnosis in enterprise networks. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 243–254, 2009.

[30] K. Konyushkova, R. Sznitman, and P. Fua. Learning active learning from data. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[31] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[32] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 1994.

[33] Z. C. Lipton. The mythos of model interpretability. *Commun. ACM*, 61(10), 2018.

[34] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56. ACM, 2016.

[35] H. Mao, A. Narayan, P. Negi, H. Wang, J. Yang, H. Wang, M. Khani, S. He, R. Addanki, R. Marcus, et al. Park: An open platform for learning augmented computer systems. 2019.

[36] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210. ACM, 2017.

[37] T. Miller. Explanation in artificial intelligence: Insights from the social sciences. *CoRR*, abs/1706.07269, 2017.

[38] C. Molnar. *Interpretable Machine Learning*. Lulu. com, 2020.

[39] K. Murphy et al. The bayes net toolbox for matlab. *Computing science and statistics*, 33(2):1024–1034, 2001.

[40] I. J. Myung. Tutorial on maximum likelihood estimation. *Journal of mathematical Psychology*, 47(1):90–100, 2003.

[41] R. S. Olson and J. H. Moore. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Proceedings of the Workshop*

on Automatic Machine Learning (AutoML@ICML). PMLR, 2016.

[42] A. Roy, D. Bansal, D. Brumley, H. K. Chandrappa, P. Sharma, R. Tewari, B. Arzani, and A. C. Snoeren. Cloud datacenter sdn monitoring: Experiences and challenges. In *Proceedings of the Internet Measurement Conference 2018*, pages 464–470. ACM, 2018.

[43] A. Roy, H. Zeng, J. Bagga, and A. C. Snoeren. Passive realtime datacenter fault detection and localization. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 595–612, 2017.

[44] N. Roy and A. McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2001.

[45] B. Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[46] B. Settles, M. Craven, and S. Ray. Multiple-instance active learning. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2007.

[47] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Annual Conference on Computational Learning Theory (COLT)*, 1992.

[48] S. Sinha, S. Ebrahimi, and T. Darrell. Variational adversarial active learning. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2019.

[49] C. Steinruecken, E. Smith, D. Janz, J. R. Lloyd, and Z. Ghahramani. The automatic statistician. In *Automated Machine Learning - Methods, Systems, Challenges*. Springer, 2019.

[50] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2, 2001.

[51] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar. Learning to route. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 185–191. ACM, 2017.

[52] K. Winstein and H. Balakrishnan. Tcp ex machina: Computer-generated congestion control. 2013.

[53] R. Woolson. Wilcoxon signed-rank test. *Wiley encyclopedia of clinical trials*, pages 1–3, 2007.

[54] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein. Pantheon: the training ground for internet congestion-control research. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*.

[55] Q. Zhang, G. Yu, C. Guo, Y. Dang, N. Swanson, X. Yang, R. Yao, M. Chintalapati, A. Krishnamurthy, and T. Anderson. Deepview: Virtual disk failure diagnosis and pattern detection for azure. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 519–532, 2018.

[56] Z. E. Zheng and B. Padmanabhan. On active learning for data acquisition. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)*, 2002.

[57] D. Zhuo, M. Ghobadi, R. Mahajan, K.-T. Förster, A. Krishnamurthy, and T. Anderson. Understanding and mitigating packet corruption in data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 362–375. ACM, 2017.