Interpretable Neural Networks using EAGGA

Simon Stürzebecher SIMON.STUERZEBECHER@CAMPUS.LMU.DE

Abstract

Keywords: tabular data, multi-objective optimization, interpretability, deep learning

1 Introduction

Tabular Data - most common type of data - bad performance of deep learning models on tabular data, recent research suggests that heavy regularisation is necessary

regularisation - LM: ridge (L2) + lasso (L1) * ridge: overcome multi-collinearity by pulling coeffs close to 0 * lasso: variable selection by setting coeffs = 0 - NN: * L2 regularisation implicitly included in SGD when using weight decay (pull params towards 0) -¿ does this combat multi-collinearity as well or serve another purpose? * L1 + key claim of DL: feature selection obsolete, network finds important ones itself + still, feature usually always has some impact (back this up), no explicit feature selection + explicit feature selection simply by not inluding a feature * another classical NN regularisation technique: dropout against co-adaptation * early stopping against overfitting when using iterative method -¿ all of those require specifying Hyperparameters in advance

2 Background and Related Works

2.1 Interpretability

As there is no clear definition for interpretability, we will consider it as "the ability to provide *explanations* in *understandable terms* to a human", where *explanations* are logical decision rules and *understandable terms* relate to commonly used terms in the domain of the problem, as suggested by Zhang et al. (2021, chap. 1). Further, we use the term "explanability" in an exchangeable manner with "interpretability", as is commonly done.

(Why Interpretability is desirable?) - in many ways desirable, as per Zach (2019, pp. 3-4), e.g. * **gaining trust**: most notable use case, e.g. medical diagnosis: predictions need to be validated by doctors (Antamis et al., 2024, p. 1) * **model debugging** - model is optimised w.r.t. loss and judged on accuracy - might have unexpected performance drops in certain situations, makes model unreliable (Zhang et al., 2021, 1B) - having an interpretable model + model induction process can help find ways to improve accuracy + reliability * **scientific understanding**: when only models can make sense of increasingly complex data anymore, interpretability enables extraction of learnt knowledge encoded in the model to make it accessible + reliable to humans (also mentioned here (Antamis et al., 2024, p. 1)) * **subconcious bias**, e.g. loan approval: must ensure that decision is not discriminatory * **regulatory** - such as EU's "Right to Explanation" warranted by the GDPR (Antamis et al., 2024, p. 1), (Zhang et al., 2021, 1B) or - drug approval processes in situations where a machine learning model discovered a new drug, process needs to be transparent for the regulator to approve (Zhang et al., 2021, 1B)

Commonly, methods for model interpretation are divided into intrinsic methods, where the search space only comprises models with a structure simple enough to be considered "explainable" (such as tree-based or simple linear models), and post-hoc methods, where interpretation methods are applied after model training. Amongst post-hoc techniques, we can further divide the space into model-specific (such as analysing GLM coefficients) and model-agnostic (e.g. partial depence plots, ALE) methods. Molnar (2022, chap. 3.2)

(Taxonomy) Zhang et al. (2021, chap. 2) extend this distinction to a 3-dimensional taxonomy, allowing for better categorisation of neural networks (NN), a model class that in its fully-connected feedforwad form is inherently non-interpretable. - **Passive vs Active Approaches** * passive: post-hoc * active: changing architecture or training process to make model interpretable - **Type of Explanations** * examples: providing examples of what leads to desired output * attribution: attributing effect on output for a specific feature * hidden semantics: examine what type of inputs specific neurons / layers pick up on * rules: logical rules, e.g. if-then, trees - **Local vs Global Interpretability** * local: explanation based on individual samples * semi-local: explanation based on set of samples, e.g. grouped together by some criterion * global: explaining network as a whole

(Evaluation) - objective evaluation difficult, as no clear definition of interpretability - Doshi-Velez and Kim (2017, 3) propose a taxonomy to categorise possible evaluation methods * **application-grounded evaluation** - most rigorous method, also most expensive + time consuming - idea: evaluate "interpretability model" directly w.r.t. the task -¿ human experts evaluate the outcome - e.g.: model performing medical diagnosis on patients would be evaluated by based on doctors doing the same * **human-grounded metrics** - similar to application-grounded, but tries to simplify task (while preserving its essence) so that laypersons can do it - cheaper due to larger, less qualified subject pool - especially suitable if only general concepts of the tasks need to be validated - examples for evaluation set-up * binary forced choice: human evaluator chooses which of two explanations he finds better * forward simulation: evaluator must correctly simulate model output when presented model input + explanation * **functionally-grounded evaluation** - assess explanation quality via some formally defined proxy for interpretability - no human time nor cost required beyond initial formulation of proxy - e.g. if we already have an interpretable model class (e.g. identified via human-grounded evaluation), can then rank different models of that class based on proxy - main challenge: what proxies

2.2 Hyperparameter Optimization (HPO)

- in contrast to model parameters, which are optimised during training, hyperparameters are parameters describing the ML algo + are specified before training - still often have great impact on final (trained) model performance -¿ can be optimised too - formal definition * dataset $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y}$ consisting of n tuples $(\boldsymbol{x}^{(i)}, y^{(i)}) \overset{i.i.d.}{\sim} \mathbb{P}_{\boldsymbol{xy}}$ (data-generating distribution), i.e. $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$ * ML algo $\mathcal{I}(\cdot, \boldsymbol{\lambda})$ with HP config $\boldsymbol{\lambda} \in \Lambda$ maps dataset to fitted model $f : \mathcal{X} \to \mathbb{R}^g \subseteq \mathcal{Y}$ (where $g \in \mathbb{N}$ is # of classes of target), i.e. $\mathcal{I} : (\mathbb{D} \times \Lambda) \to \mathcal{H}$, with $\mathbb{D}$ space of all finite datasets and $\mathcal{H}$ hypothesis space (space of all models f) * denote $\mathcal{I}_{\boldsymbol{\lambda}}$ as (untrained) model with fixed HP config $\boldsymbol{\lambda}$ * loss function L * goal of model training:

for fixed $\boldsymbol{\lambda}$, have $\mathcal{I}$ find model f minimising expected generalisation error $GE(\mathcal{I}, \boldsymbol{\lambda}, \mathcal{D}, L) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathbb{P}_{\boldsymbol{x}y}}[L(y, \mathcal{I}_{\boldsymbol{\lambda}}(\mathcal{D})(\boldsymbol{x}))]$ * goal of HP optim: find $\boldsymbol{\lambda}$ minimising expected generalisation error, i.e. $\arg\min_{\boldsymbol{\lambda}\in\Lambda} GE(\mathcal{I}, \boldsymbol{\lambda}, \mathcal{D}, L)$ - (Karl et al., 2023, p. 3) - HPO = black-box optim problem -¿ we can apply any black box algo to tune

[model free] (grid + random search) - most basic strategy: grid search, for all HPs define range of interest, then evaluate cartesian product of those -¿ scales poorly in # of HP dimensions + # of query points per HP range - random search * randomly sample value for each HP until it runs out of budget * explorative: for budget B, each HP will (likely) be queried with B different values, whereas grid search only $B^{1/N}$ for N HPs * thus better than grid in case of some HPs being non-informative (almost always the case), as it doesn't waste time specifically exploring these (as does grid) * usually useful as baseline, as no assumptions about model, easy to parallelise, high exploration, and in expectation will "achieve performance arbitrarily close to the optimum"

(evolutionary algorithms) - another model-free class of optimisation algorithms, inspired by naturally occuring evolution - based on a population, which iteratively (iteration = generation) generates λ offspring via 3 operators (all sub-bullets from Goldberg (1989, pp. 10-)) * **reproduction**: pick an individual to reproduce with p proportional to its fitness * **crossover** - select 2 "parents" from pool of reproducing individuals - for a position k in their HP configs, with certain p, swap their values after k, yielding 2 children * **mutation**: with certain p, change values of an individual, e.g. by adding Gaussian noise (Gaussian mutation) to real values, or flipping bit for binary values - selection at end of each genertion: keep μ best individuals (based on some fitness function), either from only offspring ("(μ, λ)-selection") or (more commonly) from population + offspring ("$(\mu + \lambda)$-selection", guaranteed to keep best individual) - pro: conceptually simple + can handle even complex parameter spaces (continuous, discrete, hierarchical, etc.) given appropriate implementation of operators all above starting at HPO from (Feurer and Hutter, 2019, chap. 1.3) - prominent examples * CMA-ES - "Covariance Matrix Adapation Evolution Strategy" - offspring generation exclusively via multivariate normal (Hansen, 2023, p. 8) * mean = weighted average of previous generation * covariance = weighted covariance of previous generation, with weighting as for mean - weighing scheme done to sample in a way as to reproduce previously successful (i.e. selected) steps (Hansen, 2023, p. 11) * differential evolution (Storn and Price, 1997, -) - init population randomly so that entire param space is covered * D-dim vectors $\boldsymbol{x}_{i,G}$ with $i = 1, 2, ..., n$ for gen G - mutation * for each $\boldsymbol{x}_{i,G}$, generate **mutant vector** $\boldsymbol{v}_{i,G+1} = \boldsymbol{x}_{r_1,G} + F \cdot (\boldsymbol{x}_{r_2,G} - \boldsymbol{x}_{r_3,G})$ * $r_1, r_2, r_3 \in \{1, 2, ..., n\}$ mutually different random idx + different from i * constant $F \in [0, 2]$ - crossover * **trial vector** $\boldsymbol{u}_{i,G+1} = (u^{(1)}_{i,G+1}, u^{(2)}_{i,G+1}, ..., u^{(D)}_{i,G+1})$ - chose random index $R \in \{1, 2, ..., D\}$ - sample $p \sim U[0, 1]$ - define crossover constant CR $\in [0, 1]$ - $u_{i,G+1} = \begin{cases} v^{(j)}_{i,G+1} & \text{if } p \leq \text{CR or } j = R \\ x^{(j)}_{i,G} & \text{else} \end{cases}$ - selection: greedy, compare fitness of $\boldsymbol{u}_{i,G+1}$ with $\boldsymbol{x}_{i,G}$, pick better

[model based] - contrast to model free, fit a surrogate model on blackbox problem + optimise this (Bayesian optimisation) - iterative algo comprising of * surrogate model for black box problem, needs to be able to model mean + variance * acquisition function, to decide which point to query next - in each iteration * fit surrogate model on all data points (posterior distribution) * get highest utility point from acquisition function for newly fitted surrogate model -¿ add to data points -¿ hence Bayesian, get new query point given already previously fitted points * acquisition function trades off exploration + exploitation of surrogate * BO up to here from (Feurer and Hutter, 2019, chap. 1.3.2) + (Frazier, 2018, pp. 2-3) - no optimising of model directly, instead iteratively optimise acquisition function - common choices for surrogate models * **gaussian processes** - pros * fully specified by mean + covariance - covariance function (aka kernel) solely determines quality of GP - kernel as function of two points from search space, yields their covariance - usual property of kernels: the closer points are in search space, the more strongly their correlation (Frazier, 2018, p. 5) * well-calibrated uncertainty estimates * closed-form computability - con: neither scales well in # of data points nor in # of HP dimensions BUT workaround: sparse GPs, approx full GP with small subset of original dataset * **random forests** - can handle complex search spaces (high-dim, categorical, hierarchical), unlike GPs - computational complexity scales far better than GPs * GPs $O(n^3)$ fitting, $O(n^2)$ predicting * RFs $O(n \log n)$ fitting, $O(\log n)$ predicting - common choices for acquisition functions * **expected improvement** * **Thompson sampling** - all BO stuff if not specified differently comes from (Feurer and Hutter, 2019, chap. 1.3.2)

2.3 Neural Architecture Search (NAS)

- approaches specifically for NN HPO due to flexible structure of NNs and thus very large search space (each layer can have different # nodes, different activations, for CNN different pooling, etc. operations) - we don't employ NAS for our extension as research focusses on the NLP and image domains, whose datasets exhibit strong correlation amongst features (i.e. tokens or pixels, respectively), an effect that is much weaker for tabular data (Borisov et al., 2024, p. 1) - at least want to give a little overview over most notable approaches * **cell search space** - modularise NN architecture into cells, NN as chain-structure of those - cell = basic building block, fixed component of an NN * e.g. linear layer with specific # of neurons in feedforward * or convolutional / pooling / etc layer for CNN - then optimise sequential placement of the cells as HPO problem * e.g. via random search or BO * Zoph and Le (2017, p. 3) + Zoph et al. (2018, pp. 2-4) use RNN to recursively optimise HPs of a cell given already determined previous cells (+ HPs of current cell), which is trained via RL with the different HPs the RNN predicts being the actions and performance on held-out data being reward - (Elsken et al., 2019, chap. 3.2) * **one-shot model** - trains one overall "fabric" comprising all architectures of search space - visualisation as DAG, nodes are nodes, edges in-between are operations on a node - each path through DAG (from input to output node) represents one architecture of search space -¿ train entire DAG, then pick optimal path (architecture) - pro: very efficient training, individual architectures share operations along edges they share, training one-shot model trains all subsumed models (more expensive than training a single model but less expensive than trying out all configurations included in the fabric) - (Saxena and Verbeek, 2017, pp. 1-2, p.8)

2.4 Multi-Objective Optimization

- in most practical use cases one doesn't just want to optimise for performance but e.g. also for interpretability using some proxy metrics - formal definition

2.4.1 Pareto-optimality

2.4.2 A-priori

- requires specifying trade-off between objectives a-priori - simplest approach: different versions of **scalarization** * weighted sum of objective functions - $\min_{x \in \mathcal{X}} w_i f_i(x)$ with $\sum_{i=1}^{k} w_i = 1$ and $w_i > 0, \forall i = 1, ..., k$ - (Karl et al., 2023, p. 11) * Chebyshev: provably converges to Pareto front

2.4.3 A-posteriori

- problem a priori * either restrict search space to "enforce our will" (e.g. only use 50% of features), optimise only prediction performance + use this * or leave search space unrestricted but adjust loss function to incorporate multiple objectives + take optimum from there * in either case: no knowledge of interplay between HP config + performance on all objectives * a-posteriori evaluates configs just as a-priori, but keeps track of multiple "best" (non-dominated) solutions -¿ makes relationship between HP config + objectives visible - NSGA-II (MOEA) * non-dominated sorting * crowding distance -¿ give intuition why non-dom-sort (asc) + crowding dist (desc) makes sense for selecting individuals

3 EAGGA

- combines all previously mentioned topics: interpretability, HPO, MOO - for our problem (interpretability) we cannot use regular EA algorithms as we are also interested in reducing pairwise interactions + non-monotone feature effects - this creates high-dim search space as shown in EAGGA paper -¿ they provide a framework to efficiently traverse through a space that is reduced to only sensible HP configuration (e.g. interaction of A,B with A monot incr and B monot decr -¿ couldn't be guaranteed -¿ don't look at it in first place) - NSGA-II inspired evolutionary / genetic algorithm - introduces group structure for more efficient optimization over the entire search space

4 Extension to neural networks

- why extend it to NNs? * NNs notoriously uninterpretable due to complex transformation of the feature space * due to (lin alg) non-linear activation functions, interactions can be modelled * no monotonicity guarantees - our approach (general algorithm) * EAGGA algorithm implemented just as described in the paper * HP init - NN total layers $\in \{3, ..., 10\}$, i.e. hidden layers $\in \{1, ..., 8\}$, init from trunc geom with p=0.5 - NN nodes per hidden layer (for each group) $\in \{3, ..., 20\}$, init from trunc geom with p=0.5 - NN dropout % $\in [0, 1]$, init from trunc gamma with shape=2, scale=0.15 * group structure init - feature detector * as described in paper * but instead of fitting 10 trees + taking rel. # of features used as p for trunc geom, we simply use 0.5 (sklearn dectree examination not straightforward) * also in preliminary experiments we found the sampled # of features to be used occassionally to be ¿ # of non-0 values in normalised information gain filter (e.g. if former is = total # of features and one feature is indep. from target) in these edge cases we only use the features with non-0 filter values - interaction detector * as described in paper * same as for feat detect: use p=0.5 to sample # of interactions used instead of 10 dec trees * also for FAST algorithm don't use RSS (lin reg metric) but mean accuracy, as we fit log reg - monotonicity detector * as described in paper * use default HPs (max depth 30, minsplie = 20) of mlr3 classification tree implementation, as this is the library the paper uses -¿ both interaction + monotonicity detectors fit their models on 80% of train split from holdout + eval on remaining 20% (implementation details) * group structures + datasets - group structures implemented as described in the paper with additional list encoding sign of monotonicity of the individual features as detected by the monotonicity detector - included features are passed to dataset, dataset outputs included features, multiplied by the features' individual monotonicity (-1 / 1) - entire group structure is passed to NN, where architecture is built accordingly * neural network - "instead of XGB as in the original paper, we apply the method to NNs to examine whether this type of regularisation can achieve interpretability on NNs while outperforming EAGGA on XGB (original paper), ..., this requires special architecture, etc. pp." - NN * comprises of "sub-NNs", one for each equivalence relation -¿ basically non-fully connected NN * hidden layers use ReLU activation and dropout afterwards, then a shared output layer with concatenated sub-NNs' outputs as input and sigmoid activation (implicit in the loss function for better numerical stability, NN itself outputs "logits", which refers to pre-activation output in pytorch) - feature sparsity thus achieved by only training on included feature groups -¿ goes somewhat against of deep learning where model is supposed to decide itself, which feature to "use" / put importance on -¿ ELABORATE - feature interaction achieved by grouping, max-operation in ReLU induces interaction effect (different kind of interaction than e.g. in LM with multiplication) -¿ equation why the interacting features need to be grouped together when using ReLU, cf. photos - monotonicty constraint achieved by clipping weights to [0, infty) for restricted equivalence relations (monotonic decrease achieved via dataset object multiplying features with their individual signs), bias clipping not necessary (constant additive term) -¿ equation how monotonicity is enforced with this * evaluation, holdout, cv, early stopping - evaluation via dominated hypervolume along AUC-ROC, NF, NI, NNM as defined in original paper * NF simply rel. # of included features * NI = sum of all possible pairwise interactions in each group over all possible pairwise interactions among all features = $\frac{\sum_g^G \binom{p_g}{2}}{\binom{p}{2}}$ * NNM = rel. # unconstrained features - outer holdout split: 2/3 train, 1/3 test (as in paper) * run EAGGA on holdout train split, i.e. train + select best $\mu = 100$ individuals based on non-dom-sorting (ascending) + crowding distance (descending) as tie breaker - inner CV split on outer train portion: 5-fold (as in paper) * in each fold fit model on 80% of CV-train portion, use remaining 20% of CV-train for early stopping - early stopping criterion * for each fold, always train for min 200 epochs, keep track of model with lowest loss * after that, use patience of 100: if current model's loss is ¿ mean of last 100 epochs' losses * if no early stopping, train each fold for max 10 minutes * then stop training, return model with lowest loss -¿ add graphic of entire dataset split - then evaluate on last fold from CV + keep best μ individuals + generate $\lambda = 10$ offspring for new generation + repeat * hardware: training on Sagemaker Notebook instance - initial experiments on ml.g4dn.xlarge instance (2 vCPUs, 16GiB RAM, 1 NVIDIA T4, cf. https://aws.amazon.com/de/ec2/instance-types/g4/) using cuda not much faster than on ml.t3.medium (2 Intel Xeon 8000 vCPUs, 4GiB RAM, cf. https://aws.amazon.com/de/ec2/instance-types/t3/) - thus decided for more economic + ressourcen-schonend t3.medium * known bugs - in rare cases (anecdotally once every

5-10 datasets), gga_mutate seems to be generating -1 as monotonicity attribute, despite np.random.randint(low=0, high=2, size=1) * loop crashes at group_structure creation, for these cases subtract previous runtime from 8hrs + load from last generation in output via load_population * so far only happened for madeline in after gen-8.json after 5hrs 45mins (at start of gen-9, which wasn't exported yet) -¿ loaded this and ran for another 2hrs 15mins

5 Experimental Results and Discussion

6 Conclusion and Future Outlook

- MO BO on group structure space possible? - here is a citation Schneider et al. (2023).

Appendix A. Software used

for implementation we used openml Vanschoren et al. (2014), Feurer et al. (2021), numpy Harris et al. (2020), pandas pandas development team (2023), Wes McKinney (2010), pytorch Ansel et al. (2024), scikit-learn Pedregosa et al. (2011), scipy Virtanen et al. (2020), pymoo Blank and Deb (2020), and tqdm da Costa-Luis et al. (2024)

Appendix B.

In this appendix we prove the following theorem from Section 6.2:
Theorem *Let u, v, w be discrete variables such that v, w do not co-occur with u (i.e., $u \neq 0 \Rightarrow v = w = 0$ in a given dataset $\mathcal{D}$). Let N_{v0}, N_{w0} be the number of data points for which $v = 0, w = 0$ respectively, and let I_{uv}, I_{uw} be the respective empirical mutual information values based on the sample $\mathcal{D}$. Then*

$$N_{v0} \; > \; N_{w0} \;\; \Rightarrow \;\; I_{uv} \; \leq \; I_{uw}$$

with equality only if u is identically 0. ■

Appendix C.

Proof. We use the notation:

$$P_v(i) \; = \; \frac{N_v^i}{N}, \quad i \neq 0; \quad P_{v0} \; \equiv \; P_v(0) \; = \; 1 - \sum_{i \neq 0} P_v(i).$$

These values represent the (empirical) probabilities of v taking value $i \neq 0$ and 0 respectively. Entropies will be denoted by H. We aim to show that $\frac{\partial I_{uv}}{\partial P_{v0}} < 0$....

Remainder omitted in this sample. See http://www.jmlr.org/papers/ for full paper.

References

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 929–947, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703850. doi: 10.1145/3620665.3640366. URL `https://doi.org/10.1145/3620665.3640366`.

Thanasis Antamis, Anastasis Drosou, Thanasis Vafeiadis, Alexandros Nizamis, Dimosthenis Ioannidis, and Dimitrios Tzovaras. Interpretability of deep neural networks: A review of methods, classification and hardware. *Neurocomputing*, 601:128204, 2024. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2024.128204. URL `https://www.sciencedirect.com/science/article/pii/S0925231224009755`.

J. Blank and K. Deb. pymoo: Multi-objective optimization in python. *IEEE Access*, 8: 89497–89509, 2020.

Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(6):7499–7519, June 2024. ISSN 2162-2388. doi: 10.1109/tnnls.2022.3229161. URL `http://dx.doi.org/10.1109/TNNLS.2022.3229161`.

Casper da Costa-Luis, Stephen Karl Larroque, Kyle Altendorf, Hadrien Mary, richardsheridan, Mikhail Korobov, Noam Yorav-Raphael, Ivan Ivanov, Marcel Bargull, Nishant Rodrigues, Shawn, Mikhail Dektyarev, Michał Górny, mjstevens777, Matthew D. Pagel, Martin Zugnoni, JC, CrazyPython, Charles Newey, Antony Lee, pgajdos, Todd, Staffan Malmgren, redbug312, Orivej Desh, Nikolay Nechaev, Mike Boyle, Max Nordlund, MapleCCC, and Jack McCracken. tqdm: A fast, extensible progress bar for python and cli, November 2024. URL `https://doi.org/10.5281/zenodo.14231923`.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv: Machine Learning*, 2017. URL `https://api.semanticscholar.org/CorpusID:11319376`.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search. In Hutter et al. (2019), pages 69–86.

Matthias Feurer and Frank Hutter. Hyperparameter optimization. In Hutter et al. (2019), pages 3–38.

Matthias Feurer, Jan N. van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas Müller, Joaquin Vanschoren, and Frank Hutter. Openml-python: an extensible python api for openml. *Journal of Machine Learning Research*, 22(100):1–5, 2021. URL `http://jmlr.org/papers/v22/19-920.html`.

Peter I. Frazier. A tutorial on bayesian optimization, 2018. URL `https://arxiv.org/abs/1807.02811`.

David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989. ISBN 0201157675.

Nikolaus Hansen. The cma evolution strategy: A tutorial, 2023. URL `https://arxiv.org/abs/1604.00772`.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J.

Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automatic Machine Learning: Methods, Systems, Challenges.* Springer, 2019.

Florian Karl, Tobias Pielok, Julia Moosbauer, Florian Pfisterer, Stefan Coors, Martin Binder, Lennart Schneider, Janek Thomas, Jakob Richter, Michel Lang, Eduardo C. Garrido-Merchán, Juergen Branke, and Bernd Bischl. Multi-objective hyperparameter optimization in machine learning—an overview. *ACM Trans. Evol. Learn. Optim.*, 3(4), December 2023. doi: 10.1145/3610536. URL `https://doi.org/10.1145/3610536`.

Christoph Molnar. *Interpretable Machine Learning.* 2 edition, 2022. URL `https://christophm.github.io/interpretable-ml-book`.

The pandas development team. pandas-dev/pandas: Pandas, January 2023. URL `https://doi.org/10.5281/zenodo.7549438`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics, 2017. URL `https://arxiv.org/abs/1606.02492`.

Lennart Schneider, Bernd Bischl, and Janek Thomas. Multi-objective optimization of performance and interpretability of tabular supervised machine learning models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '23, page 538–547, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701191. doi: 10.1145/3583131.3590380. URL `https://doi.org/10.1145/3583131.3590380`.

Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4): 341–359, Dec 1997. ISSN 1573-2916. doi: 10.1023/A:1008202821328. URL `https://doi.org/10.1023/A:1008202821328`.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *SIGKDD Explor. Newsl.*, 15(2):49–60, June 2014. ISSN 1931-0145. doi: 10.1145/2641190.2641198. URL `https://doi.org/10.1145/2641190.2641198`.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

J. Zach. Interpretability of deep neural networks. 2019. URL `https://api.semanticscholar.org/CorpusID:198979458`.

Yu Zhang, Peter Tiňo, Aleš Leonardis, and Ke Tang. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5): 726–742, 2021. doi: 10.1109/TETCI.2021.3100641.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning, 2017. URL `https://arxiv.org/abs/1611.01578`.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.