Advances in Tabular Machine Learning (WS 2024/25) 1-18 Submitted 15.03.2025

# Interpretable Neural Networks using EAGGA

**Simon Stürzebecher** SIMON.STUERZEBECHER@CAMPUS.LMU.DE

## Abstract

- tabular data still difficult for NNs, where it's still outperformed by other ML model classes - research suggests NN performance benefits from heavy regularisation - using EAGGA, we regularise an NN and achieve both comparable performance as XGB on EAGGA as well as interpretability - for this, we propose a network architecture specifically suited for the EAGGA algorithm

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

## 1 Introduction

Tabular Data - most common type of data - still difficult for neural networks - (Borisov et al., 2024, p. 7499) - recent research suggests that strong regularisation is beneficial to NN performance on tab data (Kadra et al., 2021, 8)

- we know regularisation from linear models can come with improvements in interpretabiltiy * e.g. LM-LASSO (L1) regularisation as feature selection -¿ reduces # features used in model by setting some coeffs to 0 (Tibshirani, 2018, p. 267) - other forms of regularisation improve performance * e.g. LM-Ridge (L2) on multi-collinear data (Hoerl and Kennard, 1970, p. 55) * e.g. NN dropout, reduces co-adaptation (Hinton et al., 2012, p. 1) * e.g. NN early stopping, reduces overfitting on training data (Finnoff et al., 1993, p. 778f)

- we want to explore if we can use NN regularisation to tackle both interpretability and improved performance (i.e. "comparable" to XGBoost) on tab data - using EAGGA framework, which proved it can improve performance while keeping (already high) performance of XGBoost on tab data -¿ see if interpretabiltiy improvements translate to NN + performance can also be on par

## 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 $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ maps dataset to model $\hat{f} : \mathcal{X} \to \mathbb{R}^g \subseteq \mathcal{Y}$ (where $g \in \mathbb{N}$ is # of classes of target), i.e. $\mathcal{I} : (\mathbb{D} \times \boldsymbol{\Lambda}) \to \mathcal{H}$, with $\mathbb{D}$ space of all finite datasets and $\mathcal{H}$ hypothesis space (space of all models $\hat{f}$) * 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 $\hat{f}$ minimising expected generalisation error $GE(\mathcal{I}, \boldsymbol{\lambda}, \mathcal{D}, L) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathbb{P}_{\boldsymbol{xy}}}[L(y, \mathcal{I}_{\boldsymbol{\lambda}}(\mathcal{D})(\boldsymbol{x}))]$ * goal of HP optim: find $\boldsymbol{\lambda}$ minimising expected generalisation error, i.e. $\arg\min_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} GE(\mathcal{I}, \boldsymbol{\lambda}, \mathcal{D}, L)$ - (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 $\lambda$ 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 $\mu$ best individuals (based on some fitness function), either from only offspring ("$(\mu, \lambda)$-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** $\boldsymbol{u}_{i,G+1} = (u_{i,G+1}^{(1)}, u_{i,G+1}^{(2)}, ..., u_{i,G+1}^{(D)})$ - chose random index $R \in \{1, 2, ..., D\}$ - sample $p \sim U[0, 1]$ - define crossover constant $\mathrm{CR} \in [0, 1]$ - $u_{i,G+1} = \begin{cases} v_{i,G+1}^{(j)} & \text{if } p \leq \mathrm{CR} \text{ or } j = R \\ x_{i,G}^{(j)} & \text{else} \end{cases}$ - 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. 7499) - 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 * vector of objectives $c_1, c_2, ..., c_m = \boldsymbol{c} : \boldsymbol{\Lambda} \rightarrow \mathbb{R}^m$ * goal: minimise vector * (Karl et al., 2023, p. 11)

#### 2.4.1 Pareto-optimality

- problem: usually conflicting objectives, i.e. minimisation of $\boldsymbol{c}$ not possible along all dimensions - thus aim to find trade-off solutions of non-dominated points - we say a point dominates another * if there is no other point strictly better in at least one dimension and better or equal in the remaining ones * formally: $\boldsymbol{\lambda}$ dominates $\boldsymbol{\lambda}'$ ($\boldsymbol{\lambda} \prec \boldsymbol{\lambda}'$) if and only if $\forall i \in \{1, ..., m\} : c_i(\boldsymbol{\lambda}) \leq c_i(\boldsymbol{\lambda}') \wedge \exists j \in \{1, ..., m\} : c_j(\boldsymbol{\lambda}) < c_j(\boldsymbol{\lambda}')$ * (Karl et al., 2023, pp. 7f) + (Goldberg, 1989, pp. 198f) - Pareto set: set of nondominated points $\mathcal{P} := \{\boldsymbol{\lambda} \in \boldsymbol{\Lambda} | \nexists \boldsymbol{\lambda}' \in \boldsymbol{\Lambda} \text{ s.t. } \boldsymbol{\lambda}' \prec \boldsymbol{\lambda}\}$ - Pareto front: image of nondominated points - goal: find set of nondominated points $\hat{\mathcal{P}}$ approximating true Pareto set $\mathcal{P}$ well (evaluation) - if knowledge over true Pareto-front, evaluating set of individuals can be done based on distance of estimated to true Pareto front - if no knowledge over true Pareto-front, volume-based approaches are popular, which measure volume between Pareto front estim and some chosen reference point (usually worst point in objective space) - (Karl et al., 2023, pp. 8-10)

#### 2.4.2 A-priori

- requires specifying trade-off between objectives a-priori, will outline 2 popular approaches - e.g. different versions of **scalarization** * weighted sum of objective functions - $\arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} w_i c_i(\boldsymbol{\lambda})$ with $\sum_{i=1}^{k} w_i = 1$ and $w_i > 0, \forall i = 1, ..., k$ - drawbacks * solution sensitive to weights * different users might have different opinions on weights (Srinivas and Deb, 1994, chap. 3.1) - (Karl et al., 2023, p. 11) * $\epsilon$-constraint - translate all but one objective into constraint, then optimise remaining objective under subject to the constraints - w.l.o.g. $\arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} c_1(\boldsymbol{\lambda})$ s.t. $c_2(\boldsymbol{\lambda}) \leq \epsilon_2, ..., c_m(\boldsymbol{\lambda}) \leq \epsilon_m$ - conceptually similar to weighted sum: also sensitive to constraints, must be chosen sensibly - (Karl et al., 2023, p. 12) - **lexicographic method** * define priorisation of objectives * greedily optimise objectives in order of priority, constraint to the solutions of the already optimised (higher-priority) objectives * again very dependent on user-defined priorisation * (Riera et al., 2023, p. 13749)

#### 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 * in practical applications, it is oftentimes useful to make the decision of which point of the Pareto set to use a-posteriori, e.g. if only a slight decrease in one objective translates to a significant improvement in another that would have been missed if the problem was optimised using a-priori methods * a-posteriori evaluates configs just as a-priori, but keeps track of multiple "best" (non-dominated) solutions -¿ makes relationship between HP config + objectives visible - aside from usualy baselines grid and random seach there are multi-objective BO adaptations, mainly using either of two approaches (1) fit single surrogate model on scalarised objectives, e.g. ParEGO, which employs the augemented Tchebycheff function as scalarisation to ensure the Pareto front is explored sufficiently (Karl et al., 2023, pp. 15f) + (Knowles, 2006, pp. 54-56) (2) fit one surrogate per objective, then - either one acquisition function per objective -¿ return set of promising next candidates - or one overall acquisition function aggregating surrogates, e.g. EHI, maximises expected improvement of hypervolume (Karl et al., 2023, p. 16) + (Emmerich et al., 2006, pp. 8f) - lastly, there is also multi-objective EA (MOEA) algorithms to explore Pareto front, one of which is NSGA-II (Nondominated Sorting Genetic Algorithm) * improves upon predecessor NSGA (Srinivas and Deb, 1994) by making it parameterless, ensuring elitism, and reducing computational complexity of ranking of individuals (Deb et al., 2002, p. 182) * uses all the regular operators, i.e. reproduction, mutation, crossover can be used from single-objective EA (SOEA) * difference to SOEA: ranking of individuals (SOEA uses scalar fitness, MOEA mutliple objectives) - ranking mechanism based on 2 parts * non-dominated sorting, ensures elitism - iterative procedure to rank individuals in population by their fronts (1) determine pareto front, assign rank 0 (2) remove pareto front from population (3) if individuals left, repeat from 1, increment rank # - (Goldberg, 1989, p. 201) + (Deb et al., 2002, pp. 183f) * crowding distance, ensures sufficient diversity, i.e. exploration of pareto front - assigns score depending on how crowded area around individual in objective space is - crowding distance of an individual = mean side length of cuboid spanned by its nearest neighbours as vertices in objective space - individuals without two neighbour in any dimension are assigned infinitely high distance value - (Deb et al., 2002, p. 185) -¿ the less the crowding distance, the more "crowded" an area is by other individuals -¿ rank individuals by nds front rank (ascending) + crowding distance (descending) as tie breaker * ranking used for selecting $\mu$ best individuals to keep for next generation + for reproduction: via binary tournament solution: select two random individuals, pick best w.r.t. nds + cd for offspring creation (mutation / crossover) * using nds for ranking / fitness makes intuitive sense, but why cd? - goal of MOO: not simply optimise HV, but approx true Pareto front well - only having individuals representing one area in objective space makes Pareto front estimate very "unstable": removing this area (i.e. solutions form that area) would "collapse" entire front - instead having multiple areas be represented makes the estimate "stable": removing one area wouldn't impact Pareto front estimate a lot - (Goldberg, 1989, p. 185, pp. 189-192)

## 3 EAGGA

- NSGA-II inspired evolutionary + genetic algorithm - uses AUC-ROC as performance metric and 3 interpretation metrics * NF: rel. # of features used in model * NI: rel. # of pairwise feature interaction effects * NNM: rel. # of non-monotone feature effects - aims to find high performing models (AUC) with low NF, low NI, and NNM -¿ ensures

better interpretability than simply optimising for performance but creates very large objective space $\check{\boldsymbol{\Lambda}}$ to be optimised over with tuples $(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{I}_s, \boldsymbol{m}_{\boldsymbol{I}_s}) = \check{\boldsymbol{\lambda}} \in \check{\boldsymbol{\Lambda}}$ with * $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ "regular" model HP config * $\boldsymbol{s} \in \{0,1\}^p$ vector denoting feature usage * $\boldsymbol{I}_s \in \{0,1\}^{p \times p}$ matrix denoting pairwise interactions * $\boldsymbol{m}_{\boldsymbol{I}_s} \in \{-1,0,1\}^p$ vector denoting monotonicity constraint of each feature (-1 decreasing, 0 none, 1 increasing) - thus would require to solve $\arg\min_{\check{\boldsymbol{\lambda}} \in \check{\boldsymbol{\Lambda}}}(GE(\mathcal{I}_{\check{\boldsymbol{\lambda}}}, \mathcal{D}), NF(\hat{f}_{\mathcal{D},\check{\boldsymbol{\lambda}}}), NI(\hat{f}_{\mathcal{D},\check{\boldsymbol{\lambda}}}), NNM(\hat{f}_{\mathcal{D},\check{\boldsymbol{\lambda}}}))$ - (Schneider et al., 2023, pp. 540f) - introduces equivalence relation $R$ "allowed to interact" on features that significantly reduces search space for more efficient optimization - thus: augmented search space $\tilde{\boldsymbol{\Lambda}} = \boldsymbol{\Lambda} \times \mathcal{G}$ comprising model HP space and group structure space * each group structure $\boldsymbol{G} \in \mathcal{G}$ consists of $g$ groups - $G_1$ set of excluded features - $\forall k = 2, ..., g : G_k = (E_k, M_{E_k})$ tuple * $E_k$ set of features allowed to interact with each other * $M_{E_k}$ monotonicity constraint of entire group $k$ - EA on model hyperparameters $\boldsymbol{\Lambda}$ - GGA on group structure space $\mathcal{G}$ - (Schneider et al., 2023, pp. 541f) - futher introduces special initialisation of group structures to increase sample-efficiency of EAGGA algorithm compared to random init - (Schneider et al., 2023, pp. 542f)

## 4 Extension to neural networks

- original EAGGA applied to XGBoost model proved to vastly outperform union of competitor models w.r.t. dominated hypervolume and comparable or better than ParEGO on extended search space $\check{\boldsymbol{\Lambda}}$ (Schneider et al., 2023, pp. 543-545) - 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 with (sigmoid implicit in the loss function for better numerical stability, NN itself outputs "logits", which refers to pre-activation output in pytorch) * loss is binary cross entropy loss, optimizer is AdamW with default params - 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 remaining from CV + keep best $\mu$ individuals + generate $\lambda = 10$ offspring for new generation + repeat - final evaluation of individuals of pareto set * for each individual, train model on training set of hold-out for max # of epochs taken during CV-training * decided on max # epochs instead of mean after looking at loss graphs in preliminary experiments, also refer Figure 2 * those show that there is no uptick in losses on the early stopping portion (disjunct from training portion) of the set, hence max reasonable * 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 * did not train on philippine and gina datasets (308, 970 features, respectively), as they ran out of memory when computing interaction detectors * 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

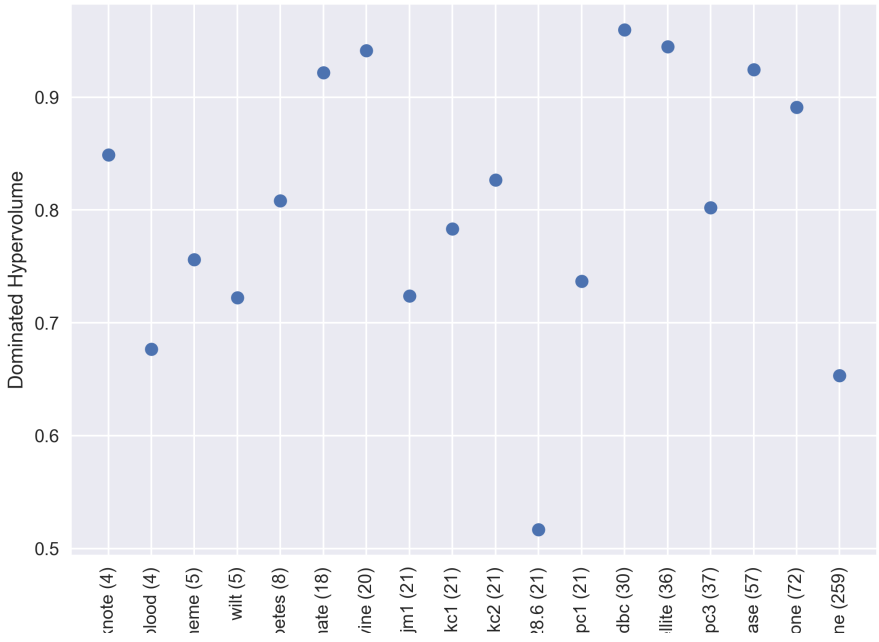


Figure 1: caption

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

- preliminary * our implementation of feature + interaction detectors likely tend to include - more features + interactions for high p datasets - less for low p datasets - remember: original paper samples # of features included from truncated geometric distribution, what is not mentioned is probability of this distribution is determined from fitting 10 trees + looking at relative # of features used (cf. their github repo /R/TunerEAGGA.R function get_n_selected_rpart()) -¿ mlr3 default decision tree max depth 30 * i.e. datasets with p ¡ 30 likely use all features, which translates to trunc geom prob = 1 * vice-versa for p ¿¿ 30 datasets relative # features ¡ 0.5 (our trunc geom prob) * similar reasoning for pairwise interactions * individuals not in Pareto sets occassionally exhibit AUC ¡ 0.5 - this is not by mistake, usually AUC on binary target can simply be inverted (simply predict the opposite class) - this cannot easily be inverted due to monotonicity constraints (weights cannot simply be multiplied by -1 if constraint) - could happen e.g. due to AdamW and weight clipping (employed to enforce monotonicity) * AdamW uses momentum * possible scenario: large momentum would yield negative weights, but after epoch they're clipped to [0,infty) + training stops due to early stopping * weight clipping very imperfect way of enforcing monotonicity, but currently in pytorch unfortunately only way to implement

this - Figure 1 suggests comparable performance to XGBoost EAGGA * unfortunately no comparison to unrestricted NN * would not be sensible, as NF, NI, NNM would be 1, hence hypervolume = 0, as values along 3 dimensions would be at reference point - overview over pareto sets can be accessed on our github repo at /code/export/*.csv * NN HPs - total layers mostly 3-4, most commonly 3, goes as high as 7 (Satellite (36), diabetes (8)) - nodes per hidden layer mostly 3-6, goes as high as 12 (diabetes (8)) - p dropout goes as high as 0.7 (climate (18), spambase (57)), but mostly in the 0.1 to 0.3 range * group structures - great diversity in NF across datasets * phoneme (5) up to 1 * blood (4) up to 1 * banknote (4) up to 0.75 * diabetes (8) up to 0.5 * climate (18) up to 0.67 -¿ low p datasets in the benchmark tend to have higher NF - possible consequence of shorter evaluation time - low p datasets have much higher # generations - group structure space much more likely to be exhausted, i.e. more exploration of NN hps - NI, NNM consequently (bounded by NF, can never be more than max # for respective # included features) rather low low * dhv contributions - measures contribution of an individual to the hypervolume, i.e. difference between hypervolume of entire pareto front vs hypervolume of pareto front without individual $\boldsymbol{\lambda}$: $\mathrm{CON}_{\mathcal{P}}(\boldsymbol{\lambda}) = \mathrm{HYP}(\mathcal{P}) - \mathrm{HYP}(\mathcal{P} \setminus \{\boldsymbol{\lambda}\})$, where $\mathrm{HYP}(\mathcal{P})$ denotes the hypervolume induced by the pareto set $\mathcal{P}$ (Bringmann and Friedrich, 2010, p. 384) - predominately low for fitted models - mostly highest for featureless learner predicting majority class -¿ sign of good exploration of pareto front + stable estimate, refer Section 2.4.3 - loss graphs Figure 2 * suggests models could have benefitted from longer training on some datasets, as some loss curves haven't converged when stopping criterion hit * crit was likely triggered by short-term spike, thus could potentially be resolved by comparing average of last k loss against average of ¿patience¡ losses prior to that to not be as exposed to short-term spikes in loss * on other datasets, graphs suggest earlier stop would have been totally fine as the networks have long converged, but longer training likely not an issue as loss graphs come from early stopping dataset portion, which is disjunct from training set - dhv over generations Figure 3 * compute on val sett, i.e. had 5 folds per individual - NF, NI, NNM always the same for each fold - but 5 different AUCs -¿ hypervolume of (mean(AUC 1, AUC 2, ..., AUC 5), NF, NI, NNM) * artifacts / drops along y likely due to inconsistent computation of Pareto front by third-party library - preliminary experiments on dummy pareto fronts with 10x4 metrics: noticed nds function returning different rankings for same front - returned ranking switched back and forth between two only slightly different options (only 1 or 2 indices were swapped) - not sure what caused this, as made same observation using another library that was planned as alternative - thus unfortunately not fixable for me * all but 3 datasets (2 of which only trained for 1 generation, aynway) show improvement of dominated hypervolume over generations * BUT: absolute as well as relative improvement almost negligible -¿ no large final dhv decrease would we just have evaluated the models gotten from the detectors - this was also the reason we didn't run EAGGA on philippine and gina simply without the detectors (as that's where they crashed) -¿ evidence suggested that detectors are vital to initial performance - original paper supports this assumption (Schneider et al., 2023, Fig. 4, p. 545)

## 6 Conclusion and Future Outlook

(Conclusion) - tabular data still difficult discipline for neural networks - research suggests heavy regularisation to improve performance on tabular data - EAGGA proved to be successful in making XGBoost more interpretable while keeping performance on par with unregularised XGBoost - we extend EAGGA to neural networks to see if we can utilise the regularisation induced by it to make NNs both interpretable and performant on tabular data - as consequence propose new architecture allowing to model equivalence relations of EAGGA - found overall performance comparable to that of XGBoost fitted using EAGGA, which is a plus, but no outperformance

(Future Outlook) - MO BO on group structure space possible?

## 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. Plots

## Appendix C.

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 D.

**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

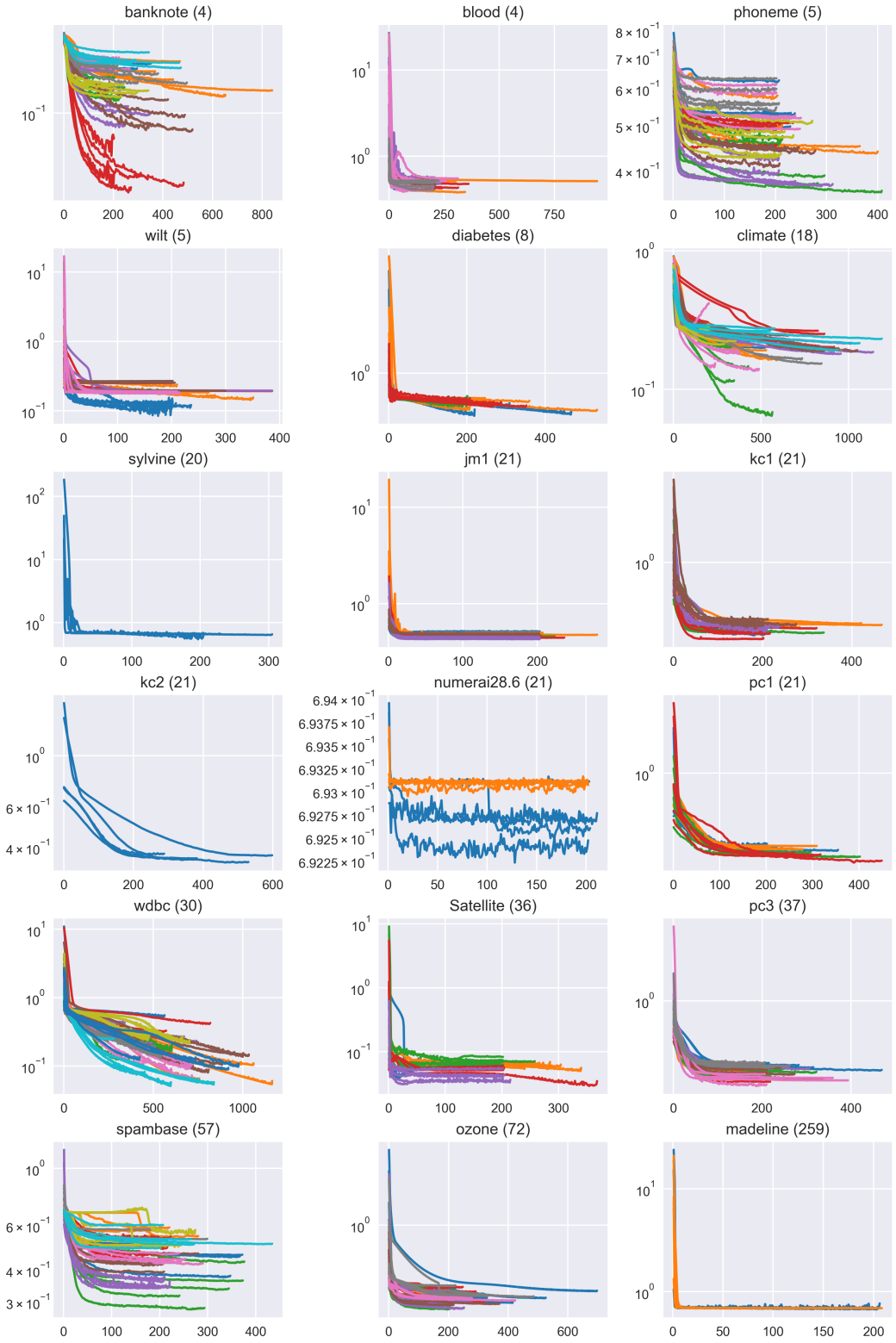


Figure 2: Pareto set loss of all datasets, evaluated on early stopping set. Same colours denote losses coming from folds of the same individual. x-axis portrait epochs, y-axis binary cross entropy loss.

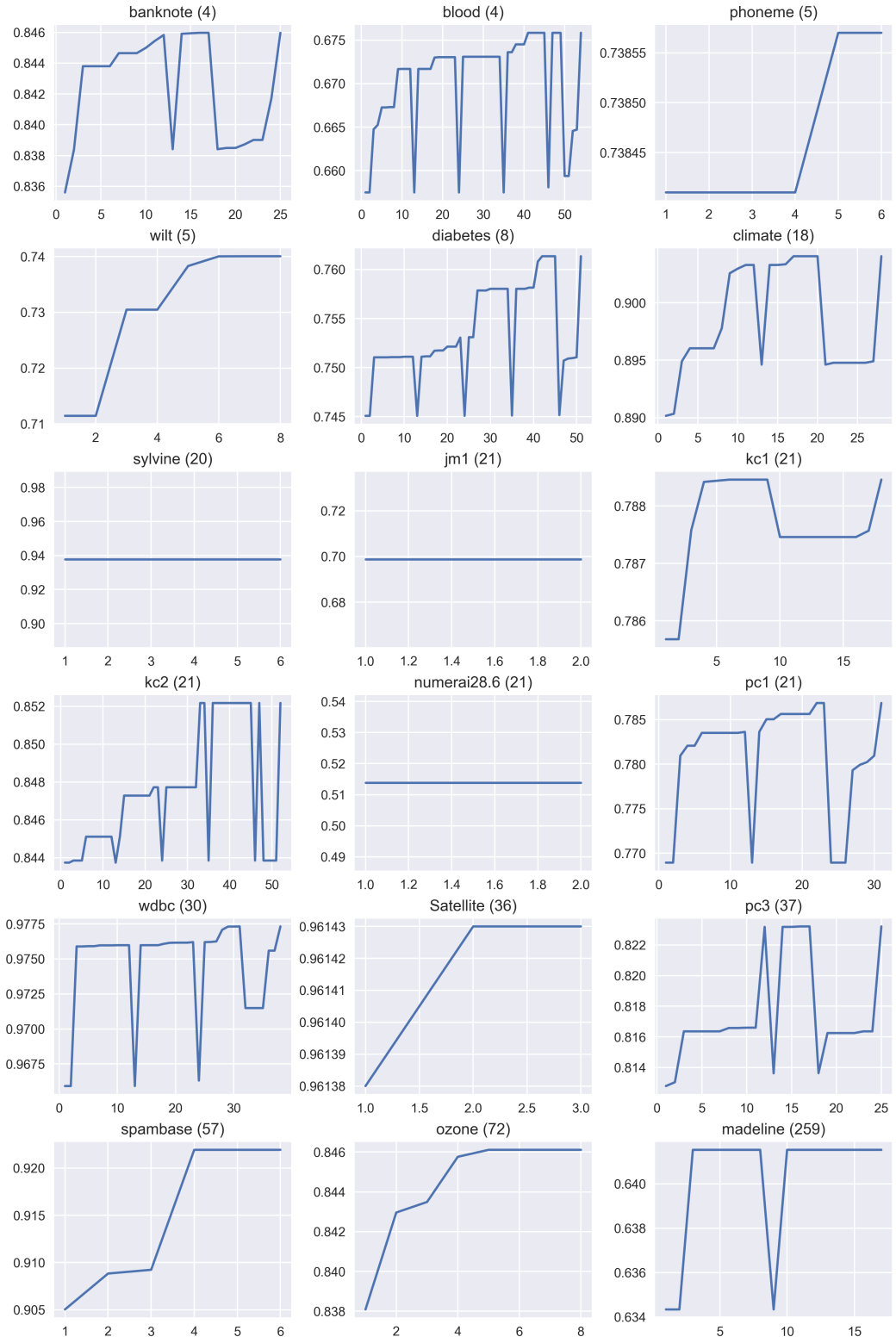


Figure 3: Dominated hypervolume over generations, evaluated on validation set. x-axis portrait generations, y-axis dominated hypervolume using mean AUC over folds.

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.

Karl Bringmann and Tobias Friedrich. An efficient algorithm for computing hypervolume contributions**. *Evol. Comput.*, 18(3):383–402, September 2010. ISSN 1063-6560.

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.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi: 10.1109/4235.996017.

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.

M.T.M. Emmerich, K.C. Giannakoglou, and B. Naujoks. Single- and multiobjective evolutionary optimization assisted by gaussian random field metamodels. *IEEE Transactions on Evolutionary Computation*, 10(4):421–439, 2006. doi: 10.1109/TEVC.2005.859463.

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.

William Finnoff, Ferdinand Hergert, and Hans Georg Zimmermann. Improving model selection by nonconvergent methods. *Neural Networks*, 6(6):771–783, 1993. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80122-4. URL https://www.sciencedirect.com/science/article/pii/S0893608005801224.

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.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012. URL https://arxiv.org/abs/1207.0580.

Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970. ISSN 00401706. URL http://www.jstor.org/stable/1267351.

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

Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 23928–23941. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/c902b497eb972281fb5b4e206db38ee6-Paper.pdf.

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.

J. Knowles. Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006. doi: 10.1109/TEVC.2005.851274.

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.

Jefferson A. Riera, Ricardo M. Lima, and Omar M. Knio. A review of hydrogen production and supply chain modeling and optimization. *International Journal of Hydrogen Energy*, 48(37):13731–13755, 2023. ISSN 0360-3199. doi: https://doi.org/10.1016/j.ijhydene.2022.12.242. URL https://www.sciencedirect.com/science/article/pii/S0360319922060505.

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.

N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evol. Comput.*, 2(3):221–248, September 1994. ISSN 1063-6560. doi: 10.1162/evco.1994.2.3.221. URL https://doi.org/10.1162/evco.1994.2.3.221.

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.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 12 2018. ISSN 0035-9246. doi: 10.1111/j.2517-6161.1996.tb02080.x. URL https://doi.org/10.1111/j.2517-6161.1996.tb02080.x.

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.