Q(D)O-ES: Population-based Quality (Diversity) Optimisation for Post Hoc Ensemble Selection in AutoML

Anonymous[1]

[1]Anonymous Institution

Abstract Automated machine learning (AutoML) systems commonly ensemble models post hoc to improve predictive performance, typically via greedy ensemble selection with replacement (GES), as proposed by Caruana et al. (2004). However, we believe that GES may not always be optimal, as it performs a simple deterministic greedy search. In this work, we introduce two novel population-based ensemble selection methods, QO-ES and QDO-ES, and compare them to GES. While QO-ES optimizes solely for predictive performance, QDO-ES also considers the diversity of ensembles within the population, maintaining a diverse set of well-performing ensembles during optimization based on ideas of quality diversity optimisation. The methods are evaluated using 71 classification datasets from the AutoML benchmark, demonstrating that QO-ES and QDO-ES often outperform GES. Our results further suggest that diversity can be beneficial for post hoc ensembling but also increases the risk of overfitting.

1 Introduction

Many automated machine learning (AutoML) systems do not return a single model but rather an ensemble of models. Following the taxonomy of ensembles from Cruz et al. (2018), AutoML systems perform three ensemble steps: *Generation*, the system generates *base models* while searching for an optimal configuration (*e.g.*, Auto-Sklearn (Feurer et al., 2015, 2020)) or cross-validating predetermined configurations (*e.g.*, AutoGluon (Erickson et al., 2020)); *Selection*, they select a subset of the base models generated in the first step, *e.g.* the top 50 models; *Aggregation*, they employ *post hoc ensembling* to aggregate the prediction of all selected base models.

Looking at ten prominent open-source AutoML systems, 60% rely on *post hoc* ensembling: Auto-Sklearn 1 (Feurer et al., 2015), Auto-Sklearn 2 (Feurer et al., 2020), AutoGluon (Erickson et al., 2020), Auto-PyTorch (Mendoza et al., 2018; Zimmer et al., 2021), H2O AutoML (LeDell and Poirier, 2020), and MLJAR (Płońska and Płoński, 2021). Among them, only H2O AutoML uses *stacking* (Wolpert, 1992) with a linear model, while all others rely on *greedy ensemble selection with replacement* (GES) (Caruana et al., 2004, 2006). The four systems that do not use *post hoc* ensembling by default – TPOT (Olson et al., 2016), GAMA (Gijsbers and Vanschoren, 2020), FLAML (Wang et al., 2021), and LightAutoML (Vakhrushev et al., 2021) – return the single best model; nevertheless, all of them but TPOT offer *post hoc* ensembling as an option.

The ten systems never studied the potential of *post hoc* ensembling nor evaluated different methods in their publications. The most frequently used method, GES, searches in a greedy manner for an optimal weight vector to linearly aggregate the predictions of base models. We believe, as GES uses a simple deterministic greedy search, there is potential to improve upon GES.

To do so, we focused on population-based optimisation methods as they have already shown success in optimisation (Kennedy and Eberhart, 1995; Das and Suganthan, 2010; Hansen and Auger, 2014; Kochenderfer and Wheeler, 2019). In particular, such methods allow us to add stochasticity while building the ensemble, *e.g.*, through crossover or random sampling. In contrast, GES performs greedy improvement at each search step and thus cannot take advantage of interactions between base models that do not immediately improve performance; which might lead into local optima.

 © 2023 the authors, released under CC BY 4.0

This motivated us to explore whether population-based ensemble selection with replacement improves over GES for AutoML.

Likewise, it is generally accepted in the literature that the diversity of an ensemble, from now on called *ensemble diversity*, can improve its performance (Dietterich, 2000b; Kuncheva and Whitaker, 2003; Sagi and Rokach, 2018). Prior work on ensemble selection *without* replacement, where one searches for an optimal subset of base models instead of a weight vector, incorporated ensemble diversity in the search (Partridge and Yates, 1996; Banfield et al., 2005; Partalas et al., 2010; Li et al., 2012). This motivated us to also include ensemble diversity in our optimisation method.

In the context of machine learning, ensemble diversity can be understood as a measure of the degree to which base models make different errors (Hansen and Salamon, 1990; Dietterich, 2000a; Banfield et al., 2005; Kumar and Kumar, 2012; Wood et al., 2023). For a given objective function or task, however, there often seems to be a trade-off between ensemble diversity and performance (Tang et al., 2006; Ahmed et al., 2017; Wood et al., 2023). This suggests that approaches such as that by Li et al. (2012), which jointly optimise ensemble diversity and performance, may be ineffective. For this reason, in our work, we focus on quality diversity optimisation (QDO).

QDO (Cully et al., 2015; Mouret and Clune, 2015; Chatzilygeroudis et al., 2020) is a recent trend in population-based optimisation. To avoid confusion, we note that the term *diversity* in QDO does not refer to ensemble diversity, but rather to *behavioural diversity*, i.e., the variability in behaviour between members of a given population. QDO maximises a single-objective function while maintaining a behaviourally diverse population. In robotics, for example, we could be interested in constructing bipedal robots with varying behaviour, e.g., tall and heavy or small and lightweight ones, that all should be able to walk fast in a straight line (Mouret and Clune, 2015).

With QDO for ensemble selection, we propose to maximise ensemble performance while maintaining a population of ensembles that is behaviourally diverse w.r.t. ensemble diversity. In other words, we maintain a population of top-performing ensembles with varying amounts of ensemble diversity. In contrast, a traditional population-based search keeps only the best-performing ensembles in the population – regardless of their ensemble diversity. As a behaviourally diverse population may be beneficial during optimisation (Mouret and Clune, 2015; Nguyen et al., 2015; Lehman and Stanley, 2011; Chatzilygeroudis et al., 2020), and ensemble diversity may be beneficial for performance (Sagi and Rokach, 2018), we see considerable promise in also considering QDO in our exploration of population-based ensemble selection.

Our contribution is the empirical performance comparison of GES against population-based search with and without diversity (QDO-ES and QO-ES, where QO stands for quality optimisation) for *post hoc* ensemble selection in AutoML. We show that population-based methods often perform better than GES on 71 classification datasets from the AutoML benchmark (Gijsbers et al., 2022). Moreover, we found that QDO-ES and QO-ES statistically significantly outperform GES on validation data, although the significance does not generalise to test data.

Our code is available on GitHub: `https://anon-github.automl.cc/r/8279`.

2 Background

The goal of *post hoc ensembling* for AutoML is to aggregate a pool $P = \{p_1, ..., p_m\}$ of base models consisting of all models that are trained and validated during model selection or a subset thereof. For example, Auto-Sklearn sets P to the 50 best models according to the validation score. The ensemble is trained on the models' predictions generated during validation, *i.e.*, the predictions on hold-out validation data or the out-of-fold predictions of k-fold cross-validation. Thereby, *post hoc* ensembling methods optimize a *user-defined* objective function. For the rest of this paper, we focus on classification, but the concept explained and our method can be extended to regression.

Formally, ensemble selection with replacement minimises an *ensemble loss function* $L(E)$ on the validation data, where E is a multiset of base models, *i.e.*, a set that allows repetition, which can be written as $E = (P, r)$, and $r(i)$ denotes the number of times base model p_i is repeated in E. Given

an ensemble E, we can compute its weight vector

$$w_E = \left[\frac{r(i)}{\sum_{j=1}^{m} r(j)} \;\middle|\; i \in [1 \dots m] \right]. \tag{1}$$

The prediction of E is the w_E-weighted arithmetic mean of the prediction probabilities of all base models. The class with the highest prediction probability is predicted.

Greedy ensemble selection with replacement (GES) (Caruana et al., 2004, 2006) iteratively builds E by means of a greedy deterministic search process that adds one base model at each iteration. GES is an ensemble selection method, also called *ensemble pruning* (Tsoumakas et al., 2009). Hence, it produces a sparse w_E by design. Zero-weighted base models can be removed from the ensemble and increase neither inference time nor ensemble size; making both smaller than in non-sparse *ensemble weighting* methods, such as stacking. Algorithm 1 corresponds to the original definition of GES and has been implemented as such in AutoGluon. Auto-Sklearn omitted line 8, which means that the resulting ensemble might not be the one with the best performance *on the validation data*. The approach to break ties at lines 5 and 8 is also specific to the implementation.

Algorithm 1 Greedy Ensemble Selection with Replacement

Input: Pool of base models P, ensemble loss function L, number of iterations I
Output: Weight vector w of length $|P|$

1: $r \leftarrow [0 \cdots 0]$ ▷ Initialize the empty ensemble $E = (P, r)$.
2: $r_{history} \leftarrow \{r\}$
3: **for** $1 \dots I$ **do**
4: $\quad R \leftarrow \{r' \mid r' = r$ with one element incremented by $1\}$ ▷ All possible repetitions of base model for this iteration.
5: $\quad r \leftarrow pickOne(\arg\min_{r' \in R} L((P, r')))$ ▷ *Select* the repetition(s) minimising the loss and pick one (to break ties).
6: $\quad r_{history} \leftarrow r_{history} \cup \{r\}$
7: **end for**
8: $r^* \leftarrow pickOne(\arg\min_{r' \in r_{history}} L((P, r')))$ ▷ Take the best seen ensemble as final ensemble.
9: **return** w computed with Equation 1 using $E = (P, r^*)$.

3 Related Work

GES was first introduced to AutoML by Auto-Sklearn 1 (Feurer et al., 2015). The authors *stated* that GES outperforms alternatives such as stacking or gradient-free numerical optimisation. Recently, Purucker and Beel (2022) have shown that GES can perform better than other *post hoc* ensembling methods for data from OpenML (Vanschoren et al., 2013). Otherwise, to the best of our knowledge, GES was never compared to any other *post hoc* ensembling method.

Following Auto-Sklearn, most AutoML systems chose to employ GES as well; Auto-PyTorch and AutoGluon even based their initial implementation on the one from Auto-Sklearn. Aside from minor variations (*e.g.* in tie-breaking or computing the final weights), in all these cases, the underlying algorithm follows GES as defined by Caruana et al. (2004).

Population-based *alternatives* to GES have been explored in other fields, *e.g.*, to find a subset of base models for an ensemble (Partridge and Yates, 1996; Zhou et al., 2002; Zhou and Tang, 2003; Cavalcanti et al., 2016; Onan et al., 2017). QDO has been used previously to build ensembles (Boisvert and Sheppard, 2021; Nickerson and Hu, 2021; Cardoso et al., 2021b,a, 2022; Ferigo et al., 2022), but always with a focus on maintaining a behaviourally diverse *population of base models*, from which to build an ensemble after optimisation. In contrast, we focus on the behavioural diversity of a *population of ensembles*. Our QDO approach relates more closely to multi-objective optimisation for ensemble selection with quality and diversity as objectives (Partridge and Yates, 1996; Martınez-Munoz and Suárez, 2004; Banfield et al., 2005; Partalas et al., 2010; Li et al., 2012; Cavalcanti et al., 2016). However, multi-objective approaches optimise both objectives, while QDO optimises only performance, benefiting from a behaviourally diverse population during optimisation.

4 Methods: Population-based Quality (Diversity) Optimization for Ensemble Selection

Our methods maintain a population of ensembles to perform a stochastic search for an optimal weight vector. Following the concepts of ensemble selection *with replacement*, as introduced by GES, the final weight vector is sparse by design, and we express an ensemble as a multiset of base models E. We distinguish between quality optimisation for ensemble selection (QO-ES) and quality diversity optimisation for ensemble selection (QDO-ES) based on how the population is maintained.

Following QDO terminology, we store the population in an *archive* A, a set of size a. We build on pyribs (Tjanaka et al., 2021), a Python library for QDO, to implement archives. In QO-ES, we maintain the population by simply storing the a observed solutions with the lowest loss in A. In contrast, to maintain a population for QDO-ES we additionally require a notion of behavioural diversity and a storage mechanism for A that considers behavioural diversity.

In the following, we detail first how we maintain behavioural diversity in QDO-ES, and then the ways in which stochastic decisions are used in our approaches.

4.1 Maintaining Populations with Behavioural Diversity

QDO-ES requires a behaviour space $\mathcal{B}$ for *ensemble diversity*, such that we can determine the behavioural diversity w.r.t. ensemble diversity, and an archive that considers behavioural diversity.

4.1.1 A Behaviour Space for Ensemble Diversity

We map an ensemble E to a behaviour space $\mathcal{B}$ according to two ensemble diversity metrics:

Average loss correlation (ALC) measures the *explicit* ensemble diversity following previous work on correlation-based ensemble diversity metrics (Tumer and Ghosh, 1999; Brown et al., 2005). Our implementation measures the average Pearson correlation between loss vectors over all pairs of non-zero weighted base models in E. A loss vector contains the difference between 1 and the prediction probability of the correct class for each instance.

Configuration space similarity (CSS) measures the *implicit* ensemble diversity by measuring the average pairwise similarity of configurations of base models included in E using the Gower similarity (Gower, 1971); it is implicit, since different configurations do not guarantee different predictions. To the best of our knowledge, the similarity of configurations has never been used in an ensembling context, but implicitly measuring the behavioural diversity has been used successfully in QDO for reinforcement learning (Ferigo et al., 2022). Moreover, CSS does not require a potentially biased definition of ensemble diversity, because it measures an existing variation in the *input space of algorithms* that produce the base models; in comparison, ALC measures the ensemble diversity in the *output space of the base models produced by these algorithms*.

We choose a two-dimensional behaviour space, because these are known to work well when behavioural diversity is not aligned with performance (Pugh et al., 2016). As pointed out in the introduction, there appears to be a trade-off between ensemble diversity and performance. Thus, the alignment of behavioural diversity and performance of an ensemble is generally unknown.

4.1.2 An Archive for QDO-ES

In a typical QDO application, we divide the behaviour space *once* into a niches (sometimes called partitions), represented by bins. The niches are computed based on the *theoretical range* of $\mathcal{B}$'s dimensions such that the niches' boundaries are uniformly distributed across $\mathcal{B}$. During optimization, the best-observed solutions for each niche are then stored in the archive A. In detail, solutions are stored in the niche corresponding to their behaviour values if their loss is smaller than the current solution in the bin (or if the bin is empty). Thus, A in QDO enforces local competition between behaviourally similar solutions (Chatzilygeroudis et al., 2020). We used a *sliding boundaries archive* (Fontaine et al., 2019) for QDO-ES, motivated in Appendix D.1.

Note that QDO practitioners are interested in the best solutions for all niches, while we, in AutoML, are only interested in the single best ensemble. Our application of QDO is atypical; nevertheless, having access to a diverse population during optimisation can improve performance, even if one is only interested in a single best solution (Nguyen et al., 2015; Mouret and Clune, 2015).

4.2 Stochasticity during Optimisation

We implemented three ways to include stochasticity during optimisation: sampling of parents, crossover, and mutation. We sample two solutions to apply crossover (potentially followed by mutation) on them; alternatively, if no crossover is applied, we sample only one and mutate it.

4.2.1 Sampling. We implemented three approaches to sample parents from an archive: *deterministic sampling*, which returns the best solution(s) from the archive; a non-deterministic variant of *tournament selection* (Miller and Goldberg, 1995), described in Appendix D.2; and *dynamic sampling*, an adaptive approach that either uses deterministic or random sampling.

In dynamic sampling, the initial probability of random sampling is 50% such that both choices are equally likely in the first iteration. The probability of deterministic *vs.* random sampling is updated after each iteration, based on the ratio between the average performance of both sampling approaches over a window of recent iterations, such that the sampling strategy with higher average performance is more likely to be used in the next iteration; see Appendix D.3 for details.

We opted for an adaptive sampling approach because we expect the optimal probability of deterministic *vs.* random sampling changes over time – like an exploration-exploitation tradeoff (Qin and Suganthan, 2005; Audibert et al., 2009; Li et al., 2013). Furthermore, we did not want to introduce an additional hyperparameter.

4.2.2 Mutation. To keep the resulting weight vector sparse and to adhere to the concepts of ensemble selection with replacement, we follow the idea of GES, in the sense that we adjust r during mutation by incrementing one of its elements. We introduce additional stochasticity by randomly choosing this element. We provide more details and pseudo-code on the mutation operator in Appendix D.4.

4.2.3 Crossover. We use two-point crossover (Jong and Spears, 1992) or average crossover (Li et al., 2013). We apply crossover to r' and r'' for two ensembles E' and E'' instead of $w_{E'}$ and $w_{E''}$; otherwise, crossover could produce a vector for which we cannot produce an r for the multiset E, *i.e.*, compute the reverse of Equation 1. To produce a valid multiset, we round the results of average crossover to the next higher integer. For two-point crossover, we observed that the offspring were often almost only zeros, because r' and r'' are sparse. To counteract this, we apply two-point crossover only on the elements that are non-zero in r' or r'', see Appendix D.5 for more details.

The probability of using crossover adapts through the run, similar to the adaptive sampling approach. Following the same mechanism, the offspring produced by crossover may mutate.

4.3 Putting Everything Together

The final realisation of Q(D)O-ES is described in Algorithm 2. First, $initArchive(A, P)$ tries to insert an initial set of ensembles into the archive (Algorithm 2, line 1). We implemented three approaches to the initial set: the ensembles consisting only of a base model ($\{(P, r) \mid \exists! i : r(i) = 1\}$), all ensembles of size 2 including the best base model, or m-many random ensembles of size 2.

Second, a batch of solutions is built in each iteration (Lines 4-11) using $sample()$, $crossover()$, and $mutate()$. The inner workings of each of those functions depend on their hyperparameters and current adaptive probability. In the end, a new solution is created by only crossover, only mutation, or crossover and mutation. Finally, we evaluate the proposed solutions contained in the batch and try to insert them into the archive (Lines 12).

Due to randomness, proposed solutions may equal previously evaluated ones. Hence, we introduced rejection sampling (Algorithm 2, line 8) such that previously evaluated ensembles are not added to the batch. To avoid endless rejection sampling in observed edge cases, we added an emergency brake to stop the loop, see Appendix D.6 for details.

Finally, we compute r^* using $bestSolution(A)$, which returns the best r according to the validation loss among: the single best model, the average crossover of all solutions in A, and the best solution in A.

Algorithm 2 Population-based Quality (Diversity) Optimization for Ensemble Selection

Input: Pool of base models P, ensemble loss function L, quality (diversity) archive A, number of iterations I, batch size B
Output: Weight vector w of length $|P|$

```
A ← initArchive(A, P)                                  ▷ Fill the archive with a set of initial ensembles.
for 1 ... I do
    S ← ∅
    while |S| < B do                                   ▷ Build the batch S.
        r, r' ← sample(A)                              ▷ If crossover is deactivated, r = r'.
        r_sol ← crossover(r, r')
        r_sol ← mutate((P, r_sol))                     ▷ If mutate is deactivated, r_sol = mutate((P, r_sol)).
        if unknown(r_sol) then                         ▷ Rejection sampling.
            S ← S ∪ r_sol
        end if
    end while
    A ← insertIntoArchive(A, S)                        ▷ Add solutions to the archive and update the boundaries for QDO-ES.
end for
r* ← getBestSolution(A)
return w computed with Equation 1 using E = (P, r*).
```

5 Experiments

We compare GES, QO-ES, and QDO-ES. Moreover, we used the single best model as a baseline.

The datasets used in our experiments and the final evaluation follow the evaluation procedure described and discussed in the AutoML benchmark (AMLB) (Gijsbers et al., 2022). We follow the AMLB because, in our experiments, the output of an ensemble method simulates the output of an AutoML system as if it would have used the evaluated method for *post hoc* ensembling.

More precisely, we evaluated the methods w.r.t. ROC AUC using 10-fold cross-validation on the 71 OpenML classification datasets used in the AMLB. For multi-class, we use macro average one-vs-rest ROC AUC. Since ROC AUC requires prediction probabilities and is independent of a decision threshold, we complement it by also evaluating w.r.t. balanced accuracy, which requires predicted labels and is dependent on a threshold (rather than another threshold-independent metric such as log loss). In accordance to the AMLB procedure, we distinguish between binary and multi-class. We refer to a combination of a classification setting with an evaluation metric as a *scenario* (e.g., ROC AUC for multi-class).

Base Model Generation. To simulate *post hoc* ensembling methods for AutoML, we require a pool of base models P as generated by an AutoML system. Therefore, we ran Auto-Sklearn 1 on each fold of each dataset, twice – once for each metric. For each fold, we stored the set of all models validated during model selection. Afterwards, we pruned this set per fold to 50. As pruning is a preprocessing step before *post hoc* ensembling, we decided to include two pruning strategies in our experiments, denoted as *TopN* and *SiloTopN*, such that we obtain two *Ps* per fold. For *TopN*, we pruned to the top 50 best-performing models according to the validation score following Auto-Sklearn's default behaviour. For *SiloTopN*, we pruned to 50, such that as many top-performing models of each algorithm family are kept following AutoGluon's approach.

Following the AMLB, we gave Auto-Sklearn a budget of 4 hours, 32 GB of memory, and 8 cores per fold. In all our experiments, we used AMD EPYC 7452 CPUs. We increased the memory to 64 or 128 GB, to prevent Auto-Sklearn from running out of memory for 11 datasets. As a result, Auto-Sklearn generated at least 50 models for all but 5 datasets, see Appendix E.1 for an overview.

Hyperparameter settings. Because the behaviour of QO-ES and QDO-ES depends heavily on their hyperparameters, and to make sure that all methods are compared fairly, we defined a grid of hyperparameter settings and exhaustively evaluated it. This led to the evaluation of 219 distinct configurations: 1 for the single best model; 2 for GES, one for AutoGluon's and one Auto-Sklearn's variant (see Section 2); 108 for QD-ES; and 108 for QDO-ES. We also treat the pruning strategy as a

hyperparameter, which doubles our configuration space to 437 (the single best is identical for both pruning methods). The list of hyperparameter settings for Q(D)O-ES is available in Appendix E.2.

To compute the final evaluation score, the average over the folds, we ran each configuration for every fold of each dataset for both metrics – that is, in total, we evaluated $437 * 71 * 10 * 2 = 620540$ ensemble runs. We set the number of iterations to 50 for GES following Auto-Sklearn's default and guaranteed that Q(D)O-ES used the same number of total function evaluations, i.e., $50 * m$, by adjusting the number of iterations in Algorithm 2 depending on the batch size and having one remainder batch if needed. We implemented multiprocessing for GES and Q(D)O-ES and ran all configurations with the same hardware and resources used for Auto-Sklearn.

To select a configuration for each method on each dataset, we used leave-one-out cross-validation (LOO CV). For d datasets, the configuration with the highest *median normalised improvement* on $d - 1$ datasets is selected to represent the method on the out-of-fold dataset. A discussion on why we used the median normalised improvement and find it to be the only appropriate method for our experiments can be found in Appendix E.3.

Normalised Improvement. We use normalised improvement, following the AMLB. Thereby, we scale the scores per dataset, such that 0 is equal to the performance of the best configuration on this dataset and -1 is equal to the performance of the single best model. If their performance is equal, then we set everything as good as the single best model to -1 and penalize worse configurations to -10, see Appendix E.4 for details. For the LOO CV selection, we only normalise across configurations from the same method, while we normalise across all selected configurations for the final evaluation.

Statistical Tests. Again following the AMLB, we perform a Friedman test with a Nemenyi *post hoc* test ($\alpha = 0.05$) on the non-normalised scores of each method to obtain an absolute ranking of all methods and to test for statistically significant differences between them.

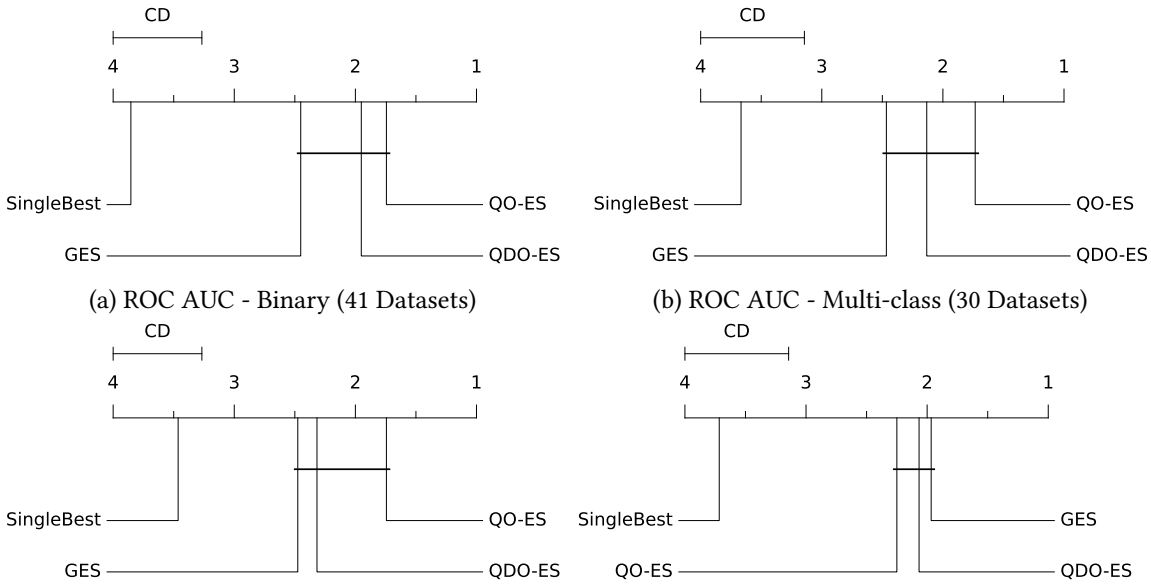
6 Results

For each scenario, we visualise the mean absolute rank and results of the statistical tests via critical differences in Figure 1. We observe that for all scenarios, *post hoc* ensemble selection is statistically significantly better than the single best model – we can always boost the performance on average. Q(D)O-ES ranks higher than GES in all scenarios, except for balanced accuracy multi-class.

To further investigate our results, we use boxplots of the normalised improvement to visualise the distribution of the relative performance for all methods over the datasets – see Figure 2. Even though the ensemble methods outperform the single best (indicated by a red line) on average, there are datasets on which this does not hold (as indicated by the numbers in the square brackets in Figure 2). Because all methods have access to the validation score of the single best and only propose an ensemble if it outperforms the single best, this phenomenon is directly linked to overfitting.

Overfitting. We observed that overfitting had a large influence on the final testing performance of our methods. We note that the difference between Q(D)O-ES and GES is statistically significant in all scenarios, when repeating our evaluation for validation instead of testing scores, and the normalised improvement gains for Q(D)O-ES are substantially larger than for GES, see Appendix F.1. Likewise, as expected, no *post hoc* ensembling method performs worse than the single best model on any dataset. Thus, the main challenge *post hoc* ensembling encounters is the lack of generalisation from validation to testing sets.

We believe that the validation data generated by Auto-Sklearn may be responsible for the large difference between validation and test performance. In our experiment and by default, Auto-Sklearn 1 uses a 33% hold-out split from the training data as validation data. In future work, it would be interesting to investigate the performance and generalisation of our methods for other AutoML systems that use more sophisticated validation procedures, *e.g.*, AutoGluon with n-repeated k-fold cross-validation.



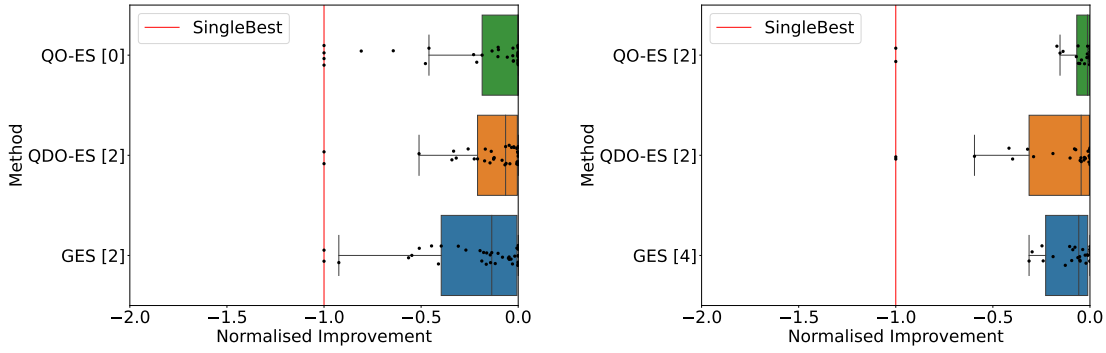
(a) ROC AUC - Binary (41 Datasets)

(b) ROC AUC - Multi-class (30 Datasets)

(c) Balanced Accuracy - Binary (41)

(d) Balanced Accuracy - Multi-class (30)

Figure 1: **Critical Difference Plots**: Mean rank of the methods (lower is better). Methods connected by a bar are not significantly different.



(a) ROC AUC - Binary (41)

(b) ROC AUC - Multi-class (30)

(c) Balanced Accuracy - Binary (41)

(d) Balanced Accuracy - Multi-class (30)

Figure 2: **Normalised Improvement Boxplots**: Higher is better. Each dot represents a dataset. The number in square brackets next to a method's name counts the outliers smaller than −2.

Regarding GES, we observe that based on the LOO CV procedure, AutoGluon's variant is always selected for balanced accuracy, while for ROC AUC, Auto-Sklearn's variant is selected for all multi-class classification datasets and ∼12% (5/41) binary classification datasets. For validation scores, AutoGluon's variant is always selected. We note that the difference in the implementations of AutoGluon and Auto-Sklearn (see Section 2) can be interpreted as differences in handling overfitting: the first variant is more prone to overfitting than the latter, due to relying more on the validation score for the selection of the final ensemble.

Ensemble Diversity. We note that for all methods, the preprocessing approach selected based on the LOO CV procedure was SiloTopN, rather than TopN. The diversity of base models fostered by including algorithms from each family seems to always be beneficial. We also observe this when analysing hyperparameter importance, see Appendix F.2; we found the preprocessing method to be the most important hyperparameter, followed by the sampling and initialisation method.

Regarding the use of ensemble diversity for optimisation, we see that QO-ES beats QDO-ES except for balanced accuracy multi-class on testing data. On validation data, QDO-ES always beats QO-ES, although the differences in rank are not significant. We hypothesise that adding ensemble diversity to the optimisation process can, in principle, be beneficial, but also increases the risk of overfitting, because it utilises the distribution of ensemble diversity, which might not generalise.

Efficiency. The size of an ensemble corresponds to how many base models must compute predictions and thus directly represents the inference efficiency of an ensemble. The average size of the generated ensembles is ∼8.5 for GES, ∼12 for QO-ES, and ∼12 for QDO-ES. Although Q(D)O-ES generates slightly bigger ensembles, their weight vectors are still sparse, considering that 50 base models were available in most cases.

Regarding the optimisation efficiency of the methods we studied, we looked at the average running time required for completing 50 iterations. GES needs on average ∼103 seconds. Our current implementation of QO-ES takes ∼165 seconds, and QDO-ES ∼196 seconds, but optimising our code would likely increase efficiency. We note that these times are insignificant compared to the time budget of 4 *hours* expended by Auto-Sklearn for finding the base models.

7 Conclusion

Post hoc ensembling is widely used in AutoML systems and can be crucial for obtaining peak predictive performance. One of the most popular methods is greedy ensemble selection with replacement (GES) (Caruana et al., 2004, 2006). In this work, we presented *QO-ES* and *QDO-ES*, two novel population-based optimisation algorithms for *post hoc* ensemble selection in AutoML. QO-ES maintains an archive of the best-performing ensembles and sequentially improves upon them, making use of mutation and crossover operators. QDO-ES builds upon QO-ES, but leverages concepts from quality diversity optimisation (Chatzilygeroudis et al., 2020), maintaining an archive of differently behaviourally diverse ensembles during optimisation. In extensive experiments, we demonstrated that 1) *post hoc* ensemble selection improves upon the single best model, 2) our population-based methods QO-ES and QDO-ES generally outperform GES (although the performance differences are not statistically significant on testing data), and 3) respecting the diversity of ensembles during optimization can be beneficial but also increases the risk of overfitting. Lastly, we want to highlight that overfitting is a serious challenge for *post hoc* ensembling in AutoML.

References

Ahmed, M. A. O., Didaci, L., Lavi, B., and Fumera, G. (2017). Using diversity for classifier ensemble pruning: an empirical investigation. *Theoretical and Applied Informatics*, vol. 29(No 1-2):25–39.

Audibert, J., Munos, R., and Szepesvári, C. (2009). Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theor. Comput. Sci.*, 410(19):1876–1902.

Banfield, R. E., Hall, L. O., Bowyer, K. W., and Kegelmeyer, W. P. (2005). Ensemble diversity measures and their application to thinning. *Inf. Fusion*, 6(1):49–62.

Boisvert, S. and Sheppard, J. W. (2021). Quality diversity genetic programming for learning decision tree ensembles. In Hu, T., Lourenço, N., and Medvet, E., editors, *Genetic Programming - 24th European Conference, EuroGP 2021, Held as Part of EvoStar 2021, Virtual Event, April 7-9, 2021, Proceedings*, volume 12691 of *Lecture Notes in Computer Science*, pages 3–18. Springer.

Brown, G., Wyatt, J. L., Harris, R., and Yao, X. (2005). Diversity creation methods: a survey and categorisation. *Inf. Fusion*, 6(1):5–20.

Cardoso, R. P., Hart, E., Kurka, D. B., and Pitt, J. (2021a). WILDA: wide learning of diverse architectures for classification of large datasets. In Castillo, P. A. and Laredo, J. L. J., editors, *Applications of Evolutionary Computation - 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, Virtual Event, April 7-9, 2021, Proceedings*, volume 12694 of *Lecture Notes in Computer Science*, pages 649–664. Springer.

Cardoso, R. P., Hart, E., Kurka, D. B., and Pitt, J. V. (2021b). Using novelty search to explicitly create diversity in ensembles of classifiers. In Chicano, F. and Krawiec, K., editors, *GECCO '21: Genetic and Evolutionary Computation Conference, Lille, France, July 10-14, 2021*, pages 849–857. ACM.

Cardoso, R. P., Hart, E., Kurka, D. B., and Pitt, J. V. (2022). The diversity-accuracy duality in ensembles of classifiers. In Fieldsend, J. E. and Wagner, M., editors, *GECCO '22: Genetic and Evolutionary Computation Conference, Companion Volume, Boston, Massachusetts, USA, July 9 - 13, 2022*, pages 627–630. ACM.

Caruana, R., Munson, A., and Niculescu-Mizil, A. (2006). Getting the most out of ensemble selection. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*, pages 828–833. IEEE Computer Society.

Caruana, R., Niculescu-Mizil, A., Crew, G., and Ksikes, A. (2004). Ensemble selection from libraries of models. In Brodley, C. E., editor, *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, volume 69 of *ACM International Conference Proceeding Series*. ACM.

Cavalcanti, G. D. C., Oliveira, L. S., Moura, T. J. M., and Carvalho, G. V. (2016). Combining diversity measures for ensemble pruning. *Pattern Recognit. Lett.*, 74:38–45.

Chatzilygeroudis, K. I., Cully, A., Vassiliades, V., and Mouret, J. (2020). Quality-diversity optimization: a novel branch of stochastic optimization. *CoRR*, abs/2012.04322.

Cruz, R. M. O., Sabourin, R., and Cavalcanti, G. D. C. (2018). Dynamic classifier selection: Recent advances and perspectives. *Inf. Fusion*, 41:195–216.

Cully, A., Clune, J., Tarapore, D., and Mouret, J. (2015). Robots that can adapt like animals. *Nat.*, 521(7553):503–507.

Das, S. and Suganthan, P. N. (2010). Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31.

Dietterich, T. G. (2000a). Ensemble methods in machine learning. In Kittler, J. and Roli, F., editors, *Multiple Classifier Systems, First International Workshop, MCS 2000, Cagliari, Italy, June 21-23, 2000, Proceedings*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer.

Dietterich, T. G. (2000b). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Mach. Learn.*, 40(2):139–157.

Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., and Smola, A. J. (2020). Autogluon-tabular: Robust and accurate automl for structured data. *CoRR*, abs/2003.06505.

Ferigo, A., Custode, L. L., and Iacca, G. (2022). Quality diversity evolutionary learning of decision trees. *CoRR*, abs/2208.12758.

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. (2020). Auto-sklearn 2.0: The next generation. *CoRR*, abs/2007.04074.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2962–2970.

Fontaine, M. C., Lee, S., Soros, L. B., de Mesentier Silva, F., Togelius, J., and Hoover, A. K. (2019). Mapping hearthstone deck spaces through map-elites with sliding boundaries. In Auger, A. and Stützle, T., editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, pages 161–169. ACM.

Gijsbers, P., Bueno, M. L. P., Coors, S., LeDell, E., Poirier, S., Thomas, J., Bischl, B., and Vanschoren, J. (2022). AMLB: an automl benchmark. *CoRR*, abs/2207.12560.

Gijsbers, P. and Vanschoren, J. (2020). GAMA: A general automated machine learning assistant. In Dong, Y., Ifrim, G., Mladenic, D., Saunders, C., and Hoecke, S. V., editors, *Machine Learning and Knowledge Discovery in Databases. Applied Data Science and Demo Track - European Conference, ECML PKDD 2020, Ghent, Belgium, September 14-18, 2020, Proceedings, Part V*, volume 12461 of *Lecture Notes in Computer Science*, pages 560–564. Springer.

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, 27(4):857–871.

Hansen, L. K. and Salamon, P. (1990). Neural network ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(10):993–1001.

Hansen, N. and Auger, A. (2014). Principled design of continuous stochastic search: From theory to practice. In Borenstein, Y. and Moraglio, A., editors, *Theory and Principled Methods for the Design of Metaheuristics*, Natural Computing Series, pages 145–180. Springer.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2014). An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 754–762. JMLR.org.

Jong, K. A. D. and Spears, W. M. (1992). A formal analysis of the role of multi-point crossover in genetic algorithms. *Ann. Math. Artif. Intell.*, 5(1):1–26.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE.

Kochenderfer, M. J. and Wheeler, T. A. (2019). *Algorithms for optimization, Chapter 8, Stochastic Methods*. Mit Press.

Kumar, G. and Kumar, K. (2012). The use of artificial-intelligence-based ensembles for intrusion detection: A review. *Appl. Comput. Intell. Soft Comput.*, 2012:850160:1–850160:20.

Kuncheva, L. I. and Whitaker, C. J. (2003). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Mach. Learn.*, 51(2):181–207.

LeDell, E. and Poirier, S. (2020). H2O AutoML: Scalable automatic machine learning. *7th ICML Workshop on Automated Machine Learning (AutoML)*.

Lehman, J. and Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223.

Li, N., Yu, Y., and Zhou, Z. (2012). Diversity regularized ensemble pruning. In Flach, P. A., Bie, T. D., and Cristianini, N., editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2012, Bristol, UK, September 24-28, 2012. Proceedings, Part I*, volume 7523 of *Lecture Notes in Computer Science*, pages 330–345. Springer.

Li, R., Emmerich, M. T. M., Eggermont, J., Bäck, T., Schütz, M., Dijkstra, J., and Reiber, J. H. C. (2013). Mixed integer evolution strategies for parameter optimization. *Evol. Comput.*, 21(1):29–64.

Lindauer, M., van Rijn, J. N., and Kotthoff, L. (2019). The algorithm selection competitions 2015 and 2017. *Artif. Intell.*, 272:86–100.

Martınez-Munoz, G. and Suárez, A. (2004). Aggregation ordering in bagging. In *Proc. of the IASTED International Conference on Artificial Intelligence and Applications*, pages 258–263. Citeseer.

Mendoza, H., Klein, A., Feurer, M., Springenberg, J. T., Urban, M., Burkart, M., Dippel, M., Lindauer, M., and Hutter, F. (2018). Towards automatically-tuned deep neural networks. In Hutter, F., Kotthoff, L., and Vanschoren, J., editors, *AutoML: Methods, Sytems, Challenges*, chapter 7, pages 141–156. Springer.

Miller, B. L. and Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Syst.*, 9(3).

Mouret, J. and Clune, J. (2015). Illuminating search spaces by mapping elites. *CoRR*, abs/1504.04909.

Nguyen, A. M., Yosinski, J., and Clune, J. (2015). Innovation engines: Automated creativity and improved stochastic optimization via deep learning. In Silva, S. and Esparcia-Alcázar, A. I., editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015*, pages 959–966. ACM.

Nickerson, K. L. and Hu, T. (2021). Principled quality diversity for ensemble classifiers using map-elites. In Krawiec, K., editor, *GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021*, pages 259–260. ACM.

Olson, R. S., Bartley, N., Urbanowicz, R. J., and Moore, J. H. (2016). Evaluation of a tree-based pipeline optimization tool for automating data science. In Friedrich, T., Neumann, F., and Sutton, A. M., editors, *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*, pages 485–492. ACM.

Onan, A., Korukoglu, S., and Bulut, H. (2017). A hybrid ensemble pruning approach based on consensus clustering and multi-objective evolutionary algorithm for sentiment classification. *Inf. Process. Manag.*, 53(4):814–833.

Partalas, I., Tsoumakas, G., and Vlahavas, I. P. (2010). An ensemble uncertainty aware measure for directed hill climbing ensemble pruning. *Mach. Learn.*, 81(3):257–282.

Partridge, D. and Yates, W. B. (1996). Engineering multiversion neural-net systems. *Neural Comput.*, 8(4):869–893.

Płońska, A. and Płoński, P. (2021). Mljar: State-of-the-art automated machine learning framework for tabular data. version 0.10.3.

Pugh, J. K., Soros, L. B., and Stanley, K. O. (2016). Searching for quality diversity when diversity is unaligned with quality. In Handl, J., Hart, E., Lewis, P. R., López-Ibáñez, M., Ochoa, G., and Paechter, B., editors, *Parallel Problem Solving from Nature - PPSN XIV - 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings*, volume 9921 of *Lecture Notes in Computer Science*, pages 880–889. Springer.

Purucker, L. and Beel, J. (2022). Assembled-openML: Creating efficient benchmarks for ensembles in autoML with openML. In *First Conference on Automated Machine Learning (Late-Breaking Workshop)*.

Qin, A. K. and Suganthan, P. N. (2005). Self-adaptive differential evolution algorithm for numerical optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005, 2-4 September 2005, Edinburgh, UK*, pages 1785–1791. IEEE.

Sagi, O. and Rokach, L. (2018). Ensemble learning: A survey. *WIREs Data Mining Knowl. Discov.*, 8(4).

Tang, E. K., Suganthan, P. N., and Yao, X. (2006). An analysis of diversity measures. *Mach. Learn.*, 65(1):247–271.

Tjanaka, B., Fontaine, M. C., Lee, D. H., Zhang, Y., Vu, T. T. M., Sommerer, S., Dennler, N., and Nikolaidis, S. (2021). pyribs: A bare-bones python library for quality diversity optimization. `https://github.com/icaros-usc/pyribs`.

Tsoumakas, G., Partalas, I., and Vlahavas, I. P. (2009). An ensemble pruning primer. In Okun, O. and Valentini, G., editors, *Applications of Supervised and Unsupervised Ensemble Methods*, volume 245 of *Studies in Computational Intelligence*, pages 1–13. Springer.

Tumer, K. and Ghosh, J. (1999). Linear and order statistics combiners for pattern classification. *CoRR*, cs.NE/9905012.

Vakhrushev, A., Ryzhkov, A., Savchenko, M., Simakov, D., Damdinov, R., and Tuzhilin, A. (2021). Lightautoml: Automl solution for a large financial services ecosystem. *CoRR*, abs/2109.01528.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2013). Openml: networked science in machine learning. *SIGKDD Explor.*, 15(2):49–60.

Wang, C., Wu, Q., Weimer, M., and Zhu, E. (2021). FLAML: A fast and lightweight automl library. In Smola, A., Dimakis, A., and Stoica, I., editors, *Proceedings of Machine Learning and Systems 2021, MLSys 2021, virtual, April 5-9, 2021*. mlsys.org.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5(2):241–259.

Wood, D., Mu, T., Webb, A. M., Reeve, H. W. J., Luján, M., and Brown, G. (2023). A unified theory of diversity in ensemble learning. *CoRR*, abs/2301.03962.

Zhou, Z. and Tang, W. (2003). Selective ensemble of decision trees. In Wang, G., Liu, Q., Yao, Y., and Skowron, A., editors, *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing, 9th International Conference, RSFDGrC 2003, Chongqing, China, May 26-29, 2003, Proceedings*, volume 2639 of *Lecture Notes in Computer Science*, pages 476–483. Springer.

Zhou, Z., Wu, J., and Tang, W. (2002). Ensembling neural networks: Many could be better than all. *Artif. Intell.*, 137(1-2):239–263.

Zimmer, L., Lindauer, M., and Hutter, F. (2021). Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 3079 – 3090. also available under https://arxiv.org/abs/2006.13799.

A Submission Checklist

1. For all authors…

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We state in the abstract and introduction that we compare GES to QO-ES and QDO-ES, which is what we did in the paper.

   (b) Did you describe the limitations of your work? [Yes] In the Appendix, see B.

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In the Appendix, see C.

   (d) Have you read the ethics author's and review guidelines and ensured that your paper conforms to them? `https://2023.automl.cc/ethics/` [Yes] We believe our paper confirms to them.

2. If you are including theoretical results…

   (a) Did you state the full set of assumptions of all theoretical results? [N/A] We included no theoretical results.

   (b) Did you include complete proofs of all theoretical results? [N/A] We included no theoretical results.

3. If you ran experiments…

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] See our repository mentioned in the introduction for all details.

   (b) Did you include the raw results of running the given instructions on the given code and data? [Yes] See our repository mentioned in the introduction for the results.

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [Yes] See our repository mentioned in the introduction.

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes] We believe that our code quality and documentation are sufficient.

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] See the Section 5 and the Appendix for detail. Additionally, see our repository mentioned in the introduction.

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes] We ran all methods on the same data.

   (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] We ran our approach for a gird of hyperparameters and assessed their importance, see Appendix E.2 and F.2.

   (h) Did you use the same evaluation protocol for the methods being compared? [Yes] We ran all methods on the same data with the same evaluation protocol and code.

(i) Did you compare performance over time? [No] We compared performance for a specific point in time (after 50 iterations of GES, i.e., after $m * 50$ function evaluations). Performance over time was out of scope for our experiments. In our opinion, anytime performance is much less relevant for *post hoc* ensembling compared to other AutoML methods such as HPO or NAS.

(j) Did you perform multiple runs of your experiments and report random seeds? [Yes] Yes, we used 10-fold cross-validation for all our runs. The used random seeds can be found in our code.

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] We took the average over the 10 folds as a score following previous work and have not reported variance across folds. Nevertheless, boxplots allow for assessing the variance of performance with respect to different data sets.

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [N/A] Such benchmarks were not available for our use case.

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 5.

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [Yes] See Section 5.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets…

(a) If your work uses existing assets, did you cite the creators? [Yes] See Section 5 and Appendix G.

(b) Did you mention the license of the assets? [Yes] See Appendix G.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See our repository mentioned in the introduction.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We are only using publicly available data that was used before in benchmarks.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We believe that the data we are using does not contain personally identifiable information or offensive content.

5. If you used crowdsourcing or conducted research with human subjects…

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We did not use crowdsourcing or conducted research with human subjects.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] We did not use crowdsourcing or conducted research with human subjects.

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] We did not use crowdsourcing or conducted research with human subjects.

B Limitations

We note that our work is limited with respect to the following points: 1) we found that overfitting is a large problem but only were able to compare our methods for Auto-Sklearn 1 and not additional AutoML systems that have better validation data available, such as Auto-Sklearn 2 or AutoGluon; 2) we were only able to explore a subset of all possible hyperparameters and hyperparameter settings for our newly proposed methods; 3) we were only able to evaluate w.r.t. two metrics; and 4) we were not able to explore all potential variations of QDO that could have been used for QDO-ES.

C Broader Impact Statement

We believe that our work is mostly abstract and methodical. Thus, after careful reflection, we determined that this work presents no notable or new negative impacts to society or the environment that are not already present for existing state-of-the-art AutoML systems. We proposed to replace one component of an AutoML system such that the predictive performance improves while efficiency stays mostly the same. Therefore, we only see the potential positive impact that higher predictive performance might help to make better decisions using AutoML.

D Supplements for the Algorithm Description

D.1 Motivation to use a Sliding Boundaries Archive for QDO-ES

For QDO-ES, we implemented a *sliding boundaries archive* (Fontaine et al., 2019). A sliding boundaries archive regularly recomputes the niche boundaries based on the *observed range* of behaviour values after initialising the niches based on the theoretical range.

We use a sliding boundaries archive because we cannot guarantee that the *observable* values of a behaviour space for ensemble diversity are uniformly distributed, as is usually assumed for QDO (Chatzilygeroudis et al., 2020; Nickerson and Hu, 2021). Moreover, the observable range of ensemble diversity metrics differs between datasets since they depend, *e.g.*, on the base models' predictions. The sliding boundaries archive allows us to maintain a similar granularity of the partitions between datasets. The granularity will also be finer than the initially computed partitions and thus enable more local competition.

Additionally and more importantly, because we keep one ensemble per niche, the sliding boundaries archive aligns the population with the underlying distribution of the observed behaviour space, making random sampling more representative.

D.2 Used Variant of Tournament Selection

In our non-deterministic variant of tournament selection, we define the tournament size T based on the number of solutions we want to sample. If we want to sample one solution, we randomly sample 10 solutions from the archive. We sample 20 solutions if we want to sample two solutions. In the first few iterations, we might have less than T solutions in the archive. In such a case, we take all solutions in the archive and additionally mutate solutions, taken from the initial set of solutions proposed to the archive, with Algorithm 3.

Given T-many solutions, we assign each solution a probability of being sampled. The best-performing solution is assigned a probability of 0.8, the second best $0.8 * 0.2^1$, the third best $0.8 * 0.2^2$, and so on until $0.8 * 0.2^{(T-1)}$. Finally, we sample either one or two solutions with these probabilities. If we sample two solutions, we sample without replacement.

D.3 Implementation of an Adaptive Probability

We have several components in our algorithm where a probability defines whether these are activated or deactivated or how they are configured. In all cases, we used a self-adaptive probability

instead of a predetermined constant value or function over time. Thus, avoiding additional hyperparameters and the bias of selecting a constant value or function over time for all tasks. This includes: dynamic sampling; whether crossover is used; and whether mutation is used after crossover.

We can represent such a choice as a binary decision between using a or b. Initially, the probability of using a, $Pr(a)$, is set to 50% (consequently $Pr(b) = 1 - Pr(a)$). That is, we randomly select whether to use a or b. Then, after each iteration, the probability is adapted based on the observed performance of a and b over the last f iterations. f is a hyperparameter. We set $f = 10$ because we aimed for a window over the performance such that bad performance in earlier iterations can be forgotten in face of dynamic changes to the performance of a and b. We track the performance of solutions in relation to their origin, i.e., whether the solutions were produced with a or b. Moreover, we do not change the probability if less than 2 solutions were created with a or b.

For the update, we first compute the average performance over all solutions seen in the last 10 iterations for a and b, denoted as a_{avg} and b_{avg}. Using these values, we compute: $Pr(a) = 1 - \frac{a_{avg}}{a_{avg} + b_{avg}}$. Finally, we bound $Pr(a)$ in $[0.05, 0.95]$ for dynamic sampling and in $[0.1, 0.9]$ otherwise. We apply these bounds to avoid reaching a probability of 0 for a or b such that it would not be chosen anymore and consequently not have solutions for an update.

We have tighter bounds for mutation after crossover and crossover, since we observed that a probability of 0.05 behaved similarly to a probability of 0 – not enough solutions were produced such that future updates could change the probability. This happens only for mutation after crossover and crossover, as both are only used for a subset of all solutions. In contrast, dynamic sampling is always used as the initial step for creating a new solution (see Algorithm 2); assuming dynamic sampling is used and not another sampling method.

D.4 Mutation Operator

Our mutation operator follows the idea of GES (Algorithm 1), in the sense that we adjust r during mutation by incrementing one of its elements. We introduce additional stochasticity by randomly choosing this element. Consequently, we implemented a randomized version of GES's update as a mutation operator, see Algorithm 3. However, since E is not built iteratively in Q(D)O-ES, we need to track which base model we previously added to avoid producing the same ensemble twice (handled by lines 2, 5, and 6).

Algorithm 3 Iteration Independent Randomized With-replacement-like Mutation

Input: Ensemble $E = (P, r)$
Output: Ensemble $E_{mutated} = (P, r_{mutated})$

1: $R \leftarrow \{r' \mid r' = r \text{ with one element incremented by } 1\}$ ▷ Obtain all possible extensions of r
2: $R_{seen} \leftarrow lookUpSeenFor(r)$ ▷ Obtain previously produced extensions
3: $R_{potential} \leftarrow R \setminus R_{seen}$
4: $r_{mutated} \leftarrow randomSample(R_{potential})$
5: $R_{seen} \leftarrow R_{seen} \cup r_{mutated}$ ▷ Update previously produced extensions
6: $storeForLookUp(r, R_{seen}); storeForLookUp(r_{mutated}, \emptyset)$ ▷ Store previously produced extensions
7: **return** $(P, r_{mutated})$

D.5 Two-Point Crossover of Repetition Vectors

For two-point crossover of two repetition vectors r and r', we first select the subsets r_{sub} and r'_{sub} of elements that are non-zero in either r or r'. If this subset is smaller than three, we fall back to average crossover. We require a vector of at least length three to perform two-point crossover. Otherwise, only one possibility to perform two-point crossover would exist.

Next, we select two distinct points randomly and crossover r_{sub} and r'_{sub} with these points. Afterwards, we fill r / r' with the elements of r_{sub} / r'_{sub} creating r_{co} / r'_{co}. Finally, we verify that the elements of r_{co} / r'_{co} are not all zero and return r_{co} / r'_{co}. If neither r_{co} nor r'_{co} contain non-zero elements, then we again fall back to average crossover to produce offspring.

Two-point crossover can produce two unique children. If this happens, we add both (mutated) children to the batch in QO-ES and QDO-ES (Algorithm 2).

D.6 Emergency Brake for Rejection Sampling

We added an emergency brake that temporarily changes the increment during mutations (Algorithm 3, Line 1) and/or probability of crossover when more than 50 solutions are rejected in one iteration.

We were motivated to do this because we observed an edge case where Algorithm 2 was not able to propose any unseen solutions. Thus, the algorithm ran into an endless loop of rejection sampling. We observed this in the first few iterations of the algorithm after the archive was initialised with ensembles consisting only of a base model, or when the probability of crossover was very high in the first few iterations.

Our overall emergency brake consists of two parts, one brake for rejections resulting from mutation and one for rejections resulting from crossover. In one iteration, both brakes can be activated (multiple) times at the same time if the algorithm still does not produce unseen solutions.

If mutation does not produce any new solutions and crossover is deactivated (or does not produce new solutions either), then the mutation emergency brake is activated after 50 rejections resulting from mutation. Once the break is activated, the increment of the repetition vectors during mutation (Algorithm 3, Line 1) is increased by 1 for the current iteration.

The rejections might also result from solutions created by crossover and not by mutation because crossover cannot produce unseen children, and the adaptive probability of mutation after crossover is too small to be called in the current iteration. In this case, we increase the probability of mutation after crossover to 100% for this iteration after 50 rejections stemming from crossover.

E Supplements for the Experiments

E.1 Dataset Overivew

Table 1 gives an overview of datasets and base models used in our experiments. Additionally, we show the average (over folds) for the number of base models and the number of distinct algorithms over the pools of base models for balanced accuracy $\mathcal{BA}$ and ROC AUC $\mathcal{R}$, respectively for the preprocessing methods *SiloTopN* and *TopN*.

E.2 Configuration Space

Table 2 shows all hyperparameter and their possible values for the configuration space of QO-ES and QDO-ES. We refer to the set of all ensembles consisting only of a single base model as *L1 Ensembles*, while we refer to all ensembles of size 2 that include the single best model, or m-many random ensembles of size 2 as *L2 Ensembles*.

E.3 Discussion on the Configuration Selection Approach

We use the median normalised improvement to select the most representative configuration during leave-one-out cross-validation because it reflects what we believe to be the distinguishing factor between different methods based on an evaluation following the AutoML benchmark (AMLB) (Gijsbers et al., 2022) – the highest median in the boxplots of the normalised improvement.

We are aware that alternative selection approaches could change our results or that one could, in principle, learn to select the best configuration. However, the only valid alternative in an evaluation following the AMLB would have been mean rank. As this is the only other distinguishing factor used in plots, specifically critical difference plots. However, as noted already in the AMLB and visible in its results, the absolute rank is not representative of a performance distribution and hides relative performance differences. In contrast, the median normalised improvement is based on a relative measure and relates to a performance distribution via the median.

Table 1: Supplementary information for all datasets and generated base models.

Dataset Name	OpenML Task ID	#Instances	#Features	#Classes	Memory (GB)	Mean #Base Models				Mean #Distinct Algorithms			
						$\mathcal{BA}_{TopN}$	$\mathcal{BA}_{SiloTopN}$	$\mathcal{R}_{TopN}$	$\mathcal{R}_{SiloTopN}$	$\mathcal{BA}_{TopN}$	$\mathcal{BA}_{SiloTopN}$	$\mathcal{R}_{TopN}$	$\mathcal{R}_{SiloTopN}$
yeast	2073	1484	9	10	32	50.0	50.0	50.0	50.0	1.7	16.0	2.1	16.0
KDDCup09_appetency	3945	50000	231	2	32	50.0	50.0	50.0	50.0	1.5	16.0	1.2	15.8
covertype	7593	581012	55	7	64	49.9	49.9	50.0	50.0	10.0	12.6	7.9	12.2
amazon-commerce-reviews	10090	1500	10001	50	32	50.0	50.0	50.0	50.0	1.4	15.8	1.8	15.9
Australian	146818	690	15	2	32	50.0	50.0	50.0	50.0	2.8	16.0	2.3	16.0
wilt	146820	4839	6	2	32	50.0	50.0	50.0	50.0	1.3	16.0	1.6	16.0
numerai28.6	167120	96320	22	2	32	50.0	50.0	50.0	50.0	5.3	15.8	5.0	15.7
phoneme	168350	5404	6	2	32	50.0	50.0	50.0	50.0	1.3	16.0	1.3	16.0
credit-g	168757	1000	21	2	32	50.0	50.0	50.0	50.0	1.5	16.0	2.6	16.0
steel-plates-fault	168784	1941	28	7	32	50.0	50.0	50.0	50.0	1.4	16.0	1.1	16.0
APSFailure	168868	76000	171	2	32	50.0	50.0	50.0	50.0	2.5	16.0	2.0	16.0
dilbert	168909	10000	2001	5	32	50.0	50.0	50.0	50.0	1.5	15.7	1.0	15.7
fabert	168910	8237	801	7	32	50.0	50.0	50.0	50.0	3.4	16.0	2.6	16.0
jasmine	168911	2984	145	2	32	50.0	50.0	50.0	50.0	3.0	15.8	1.9	15.9
airlines	189354	539383	8	2	64	50.0	50.0	50.0	50.0	5.8	14.5	5.9	14.4
dionis	189355	416188	61	355	128	20.8	20.8	28.2	28.2	7.4	7.4	7.9	7.9
albert	189356	425240	79	2	64	50.0	50.0	50.0	50.0	8.1	13.6	8.6	12.5
gina	189922	3153	971	2	32	50.0	50.0	50.0	50.0	1.0	16.0	1.0	16.0
ozone-level-8hr	190137	2534	73	2	32	50.0	50.0	50.0	50.0	1.0	16.0	1.8	15.9
vehicle	190146	846	19	4	32	50.0	50.0	50.0	50.0	1.6	16.0	1.6	16.0
madeline	190392	3140	260	2	32	50.0	50.0	50.0	50.0	1.2	16.0	1.3	16.0
philippine	190410	5832	309	2	32	50.0	50.0	50.0	50.0	1.0	16.0	1.2	16.0
ada	190411	4147	49	2	32	50.0	50.0	50.0	50.0	1.9	16.0	1.6	16.0
arcene	190412	100	10001	2	32	50.0	50.0	50.0	50.0	1.3	16.0	1.2	15.9
jannis	211979	83733	55	4	32	50.0	50.0	50.0	50.0	1.0	15.4	3.1	15.1
Diabetes130US	211986	101766	50	3	32	50.0	50.0	50.0	50.0	1.0	15.6	1.0	15.7
micro-mass	359953	571	1301	20	32	50.0	50.0	50.0	50.0	1.1	16.0	1.5	16.0
eucalyptus	359954	736	20	5	32	50.0	50.0	50.0	50.0	1.3	16.0	2.6	15.9
blood-transfusion-service-center	359955	748	5	2	32	50.0	50.0	50.0	50.0	1.9	16.0	4.1	16.0
qsar-biodeg	359956	1055	42	2	32	50.0	50.0	50.0	50.0	2.4	16.0	2.3	16.0
cnae-9	359957	1080	857	9	32	50.0	50.0	50.0	50.0	1.6	16.0	1.2	16.0
pc4	359958	1458	38	2	32	50.0	50.0	50.0	50.0	1.4	16.0	2.1	16.0
cmc	359959	1473	10	3	32	50.0	50.0	50.0	50.0	1.9	16.0	1.6	16.0
car	359960	1728	7	4	32	50.0	50.0	50.0	50.0	1.1	16.0	1.0	16.0
mfeat-factors	359961	2000	217	10	32	50.0	50.0	50.0	50.0	1.1	16.0	3.5	16.0
kc1	359962	2109	22	2	32	50.0	50.0	50.0	50.0	2.2	16.0	2.2	16.0
segment	359963	2310	20	7	32	50.0	50.0	50.0	50.0	1.4	16.0	1.4	16.0
dna	359964	3186	181	3	32	50.0	50.0	50.0	50.0	1.2	16.0	1.1	15.8
kr-vs-kp	359965	3196	37	2	32	50.0	50.0	50.0	50.0	1.8	16.0	1.1	15.8
Internet-Advertisements	359966	3279	1559	2	32	50.0	50.0	50.0	50.0	1.8	16.0	2.6	16.0
Bioresponse	359967	3751	1777	2	32	50.0	50.0	50.0	50.0	3.6	15.9	2.6	15.8
churn	359968	5000	21	2	32	50.0	50.0	50.0	50.0	1.3	16.0	1.2	16.0
first-order-theorem-proving	359969	6118	52	6	32	50.0	50.0	50.0	50.0	2.8	15.8	2.6	15.7
GesturePhaseSegmentationProcessed	359970	9873	33	5	32	50.0	50.0	50.0	50.0	1.1	15.8	2.0	15.9
PhishingWebsites	359971	11055	31	2	32	50.0	50.0	50.0	50.0	1.1	15.8	1.1	15.6
sylvine	359972	5124	21	2	32	50.0	50.0	50.0	50.0	1.3	16.0	1.1	16.0
christine	359973	5418	1637	2	32	50.0	50.0	50.0	50.0	3.5	15.8	2.9	15.9
wine-quality-white	359974	4898	12	7	32	50.0	50.0	50.0	50.0	1.0	16.0	1.7	16.0
Satellite	359975	5100	37	2	32	50.0	50.0	50.0	50.0	1.2	16.0	1.2	16.0
Fashion-MNIST	359976	70000	785	10	64	50.0	50.0	50.0	50.0	12.3	13.4	6.1	15.0
connect-4	359977	67557	43	3	32	50.0	50.0	50.0	50.0	1.0	16.0	1.0	15.7
Amazon_employee_access	359979	32769	10	2	32	50.0	50.0	50.0	50.0	2.5	15.9	1.7	16.0
nomao	359980	34465	119	2	32	50.0	50.0	50.0	50.0	1.0	15.9	1.0	15.9
jungle_chess_2pcs_raw_endgame_complete	359981	44819	7	3	32	50.0	50.0	50.0	50.0	1.0	16.0	1.0	16.0
bank-marketing	359982	45211	17	2	32	50.0	50.0	50.0	50.0	1.0	15.8	1.2	15.8
adult	359983	48842	15	2	32	50.0	50.0	50.0	50.0	1.3	16.0	1.0	16.0
helena	359984	65196	28	100	32	50.0	50.0	50.0	50.0	2.8	15.8	3.6	15.2
volkert	359985	58310	181	10	32	50.0	50.0	50.0	50.0	2.1	15.5	1.1	15.5
robert	359986	10000	7201	10	64	50.0	50.0	50.0	50.0	11.2	14.3	7.7	13.7
shuttle	359987	58000	10	7	32	50.0	50.0	50.0	50.0	1.2	16.0	2.1	16.0
guillermo	359988	20000	4297	2	32	50.0	50.0	50.0	50.0	3.1	15.5	2.9	15.6
riccardo	359989	20000	4297	2	32	50.0	50.0	50.0	50.0	1.5	15.6	1.1	15.5
MiniBooNE	359990	130064	51	2	32	50.0	50.0	50.0	50.0	1.4	15.7	1.1	15.4
kick	359991	72983	33	2	32	50.0	50.0	50.0	50.0	2.0	15.9	1.2	15.8
Click_prediction_small	359992	39948	12	2	32	50.0	50.0	50.0	50.0	1.0	16.0	1.0	15.9
okcupid-stem	359993	50789	20	3	32	50.0	50.0	50.0	50.0	1.1	15.9	1.0	15.5
sf-police-incidents	359994	2215023	9	2	64	49.5	49.5	50.0	50.0	10.5	12.9	6.1	13.1
KDDCup99	360112	4898431	42	23	128	13.5	13.5	29.2	29.2	5.1	5.1	9.2	9.2
porto-seguro	360113	595212	58	2	64	50.0	50.0	50.0	50.0	10.6	14.4	10.8	14.5
Higgs	360114	1000000	29	2	64	47.9	47.9	48.8	48.8	9.8	13.2	11.2	11.8
KDDCup09-Upselling	360975	50000	14892	2	128	50.0	50.0	49.0	49.0	4.9	13.4	6.0	11.7

Table 2: Q(D)O-ES Configuration Space.

Hyperparameter	Values
Preprocessing	SiloTopN, TopN
Batch Size	20, 40
Archive Size	16, 49
Archive Initialisation	L1 Ensembles, L2 of Single Best, Random L2
Sampling Method	Deterministic, Tournament, Dynamic
Crossover	two-point crossover, average crossover, no crossover

E.4 Normalised Improvement

Our implementation of normalised improvement follows the AutoML benchmark (Gijsbers et al., 2022). That is, we scale the scores for a dataset such that -1 is equal to the score of the single best model, and 0 is equal to the score of the best method on the dataset.

Formally, we normalise the score s_D of a method for a dataset D using:

$$\frac{s_D - s_D^b}{s_D^* - s_D^b} - 1, \tag{2}$$

with the score of the baseline s_D^b and the best-observed score for the dataset s_D^*. We assume that higher scores are always better.

We extend this definition for the edge cases where no method is better than the baseline, *i.e.*, $s_D^* - s_D^b = 0$. We assume that this edge case never happened in the AutoML benchmark. Otherwise, their definition and implementation would have been undefined / crashed. In our setting, such an edge case can happen due to overfitting such that the ensemble methods becomes worse than the single best model.

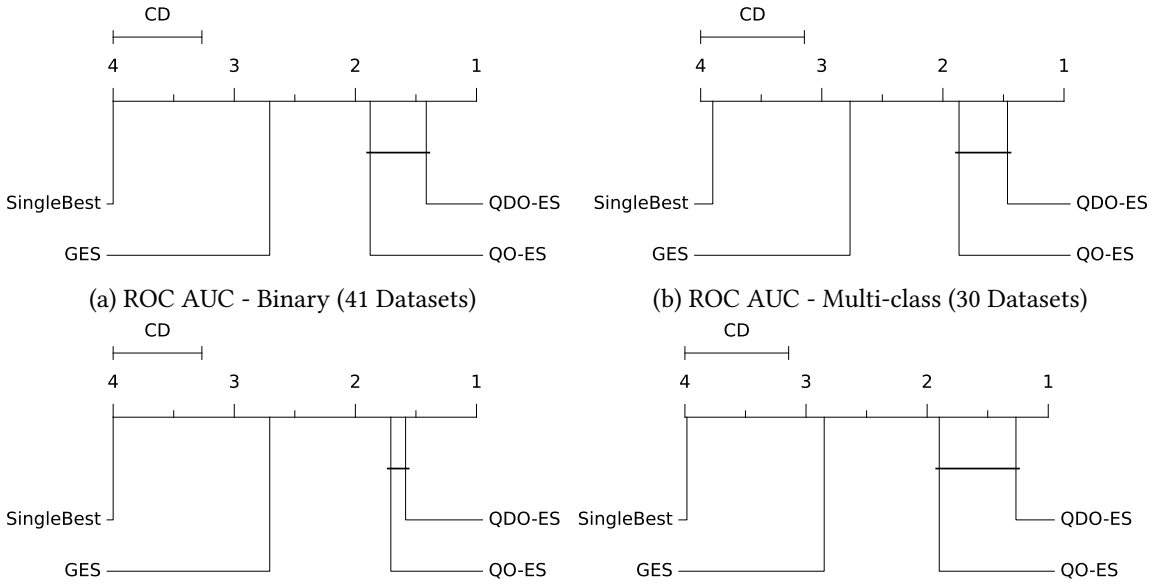
We generalize the definition of this edge case to the case $|s_D^* - s_D^b| <= \Delta$. We set Δ to 0.0001, because normalised improvement also becomes unrepresentative for a dataset if $s_D^* - s_D^b$ is very small (because we divide by $s_D^* - s_D^b$).

If the edge case happens, we set the score of all methods worse than the baseline to -10, following a penalization-like approach (*e.g.*, PAR10 from Algorithm Selection (Lindauer et al., 2019)). Methods for which $|s_D - s_D^b| <= \Delta$ holds are assigned a score of -1.

F Supplements for the Results

F.1 Results on Validation Data

Here, we repeat our evaluations described in Section 6 for the scores obtained on validation instead of testing data. For the results, see Figure 3 and Figure 4. We observe that Q(D)O-ES outperforms GES significantly in all scenarios, and QDO-ES always achieves the best mean rank. Moreover, the normalized improvement for QDO-ES is substantially larger than for GES and QO-ES.



(a) ROC AUC - Binary (41 Datasets)

(b) ROC AUC - Multi-class (30 Datasets)

(c) Balanced Accuracy - Binary (41)

(d) Balanced Accuracy - Multi-class (30)

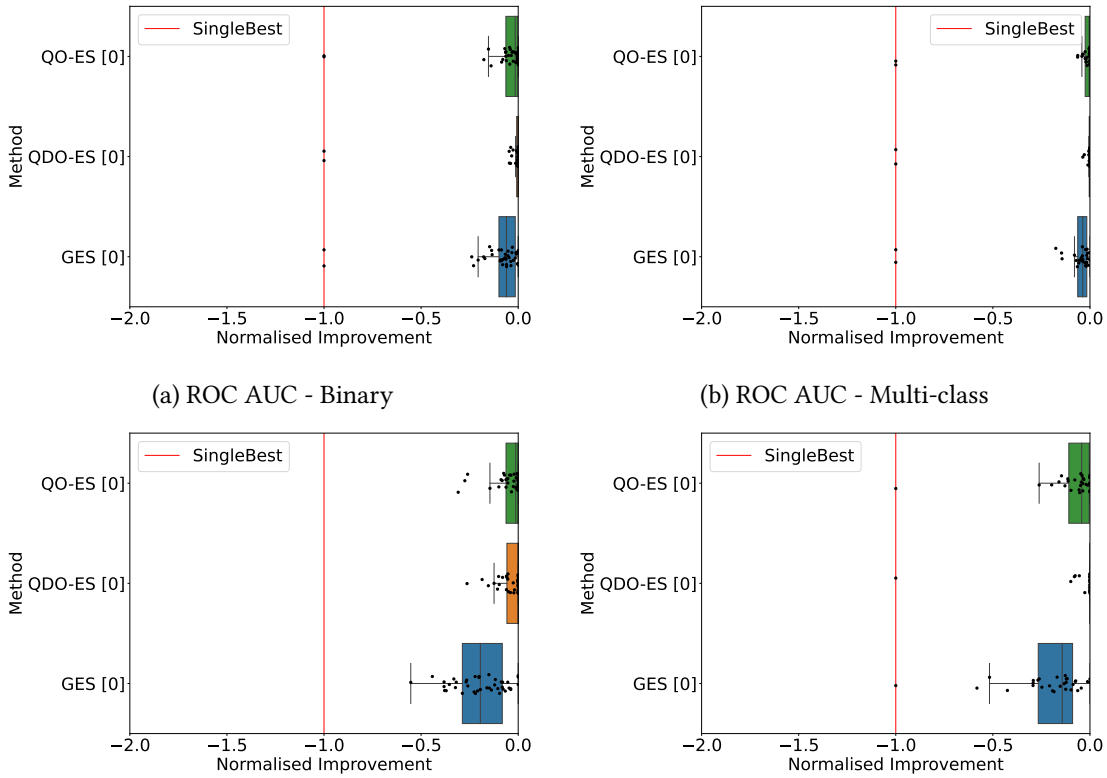
Figure 3: **Critical Difference Plots for Validation Scores**: Mean rank of the methods (lower is better). Methods connected by a bar are not significantly different.

F.2 Q(D)O-ES Hyperparameter Importance

We analyse the importance of hyperparameters using fANOVA (Hutter et al., 2014). Therefore, we compute the importance of the hyperparameters defined in Table 2 for each dataset for QO-ES and QDO-ES. Table 3 shows the mean importance (higher is more important) over all datasets split by metric and classification task for QDO-ES and QO-ES.

Table 3: Mean hyperparameter importance computed with fANOVA for QO-ES and QDO-ES.

	QDO-ES				QO-ES			
	Balanced Accuracy		ROC AUC		Balanced Accuracy		ROC AUC	
Hyperparameter	Binary	Multi-class	Binary	Multi-class	Binary	Multi-class	Binary	Multi-class
Preprocessing	0.57	0.52	0.68	0.51	0.60	0.75	0.69	0.73
Batch Size	0.03	0.03	0.02	0.03	0.03	0.01	0.02	0.02
Archive Size	0.02	0.02	0.01	0.03	0.02	0.01	0.01	0.01
Archive Initialisation	0.15	0.16	0.11	0.21	0.14	0.12	0.10	0.10
Sampling Method	0.15	0.18	0.12	0.16	0.14	0.06	0.11	0.07
Crossover	0.08	0.09	0.06	0.07	0.07	0.06	0.06	0.06



(a) ROC AUC - Binary

(b) ROC AUC - Multi-class

(c) Balanced Accuracy - Binary

(d) Balanced Accuracy - Multi-class

Figure 4: **Normalised Improvement Boxplots for Validation Scores**: Higher normalised improvement is better. Each black point represents the improvement for one dataset.

G Essential Python Frameworks for the Implementation and Experiments

The following frameworks were essential for our implementation and experiments:

- Auto-Sklearn 1.0 (Feurer et al., 2015), Version: 0.14.7, BSD-3-Clause License; We used Auto-Sklearn to generate base models and took the initial code for greedy ensemble selection from Auto-Sklearn's implementation.
- pyribs (Tjanaka et al., 2021), Version 0.4.0, MIT License; We used pyribs' framework to implement population-based search and the archives.
- Assembled (Purucker and Beel, 2022), Version 0.0.4, MIT License; We used Assembled to store the base models generated with Auto-Sklearn and to run our ensemble-related experiments.