DYNAMIC POST-HOC NEURAL ENSEMBLERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Ensemble methods are known for enhancing the accuracy and robustness of machine learning models by combining multiple base learners. However, standard approaches like greedy or random ensembles often fall short, as they assume a constant weight across samples for the ensemble members. This can limit expressiveness and hinder performance when aggregating the ensemble predictions. In this study, we explore employing neural networks as ensemble methods, emphasizing the significance of dynamic ensembling to leverage diverse model predictions adaptively. Motivated by the risk of learning low-diversity ensembles, we propose regularizing the model by randomly dropping base model predictions during the training. We demonstrate this approach lower bounds the diversity within the ensemble, reducing overfitting and improving generalization capabilities. Our experiments showcase that the dynamic neural ensemblers yield competitive results compared to strong baselines in computer vision, natural language processing, and tabular data.

1 INTRODUCTION

Ensembling machine learning models is a well-established practice among practitioners and researchers, primarily due to its enhanced predictive performance over single-model predictions. Ensembles are favored for their superior accuracy and ability to provide calibrated uncertainty estimates and increased robustness against covariate shifts (Lakshminarayanan et al., 2017). Combined with their relative simplicity, these properties make ensembling the method of choice for many applications, such as medical imaging and autonomous driving, where reliability is paramount.

Despite these advantages, the process of selecting post-hoc models that are both accurate and diverse remains a challenging combinatorial problem, especially as the pool of candidate models grows. Commonly used heuristics, particularly in the context of tabular data, such as greedy selection (Caruana et al., 2004) and various weighting schemes, attempt to optimize ensemble performance based on metrics evaluated on a held-out validation set or through cross-validation. However, these methods face significant limitations. Specifically, the selection of models to include in the ensemble and the determination of optimal ensembling strategies (e.g., stacking weights) are critical decisions that, if not carefully managed, can lead to overfitting on the validation data. Although neural networks are good candidates for generating ensembling weights, few studies rely on them as a post-hoc ensembling approach. We believe this happens primarily due to a lack of ensemble-related inductive biases that provide regularization.

In this work, we introduce a novel approach to post-hoc ensembling using neural networks. Our proposed *Neural Ensembler* dynamically generates the weights for each base model in the ensemble on a per-instance basis, a.k.a dynamical ensemble selection (Ko et al., 2008). To mitigate the risk of overfitting the validation set, we introduce a regularization technique inspired by the inductive biases inherent to the ensembling task. Specifically, we propose randomly dropping base models during training, inspired by previous work on DropOut in Deep Learning (Srivastava et al., 2014).

In summary, our contributions are as follows:

1. We propose a simple yet effective post-hoc ensembling method based on a neural network that dynamically ensembles base models.
2. To prevent the formation of low-diversity ensembles, we introduce a regularization technique that involves randomly dropping base model predictions during training. We demonstrate theo-

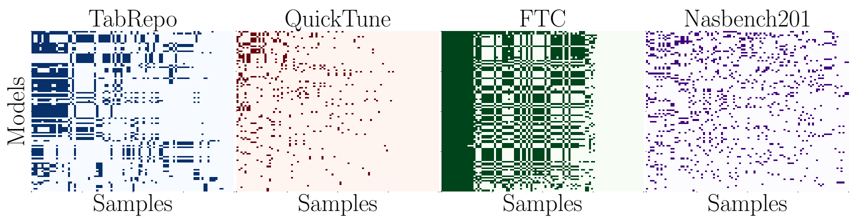


Figure 1: **Wrong Models Per Samples Across Meta-Datasets.** Every dark cell represents data instances where a model's prediction is wrong. Different models fail on different instances, therefore, only instance-specific dynamic ensembles are optimal.

retically that this lower bounds the diversity of the generated ensemble. Additionally, we ablate its effect through various experiments.

3. Through extensive experiments, we show that Neural Ensemblers consistently select competitive ensembles across a wide range of data modalities, including tabular data (for both classification and regression), computer vision, and natural language processing.

To promote reproducibility, we have made our code publicly available in the following anonymous repository [1]. We hope that our codebase, along with the diverse set of benchmarks used in our experiments, will serve as a valuable resource for the development and evaluation of future post-hoc ensembling methods.

2 Background and Motivation

Post-Hoc ensembling uses set of fitted base models $\{z_1, ..., z_M\}$ such that every model outputs predictions $z_m(x) : \mathbb{R}^D \to \mathbb{R}$. These outputs are combined by a stacking ensembler $f(z(x); \theta) := f(z_1(x), ..., z_M(x); \theta) : \mathbb{R}^M \to \mathbb{R}$, where $z(x) = [z_1(x), ..., z_M(x)]$ is the concatenation of the base models predictions. While the base models are estimated using a training set $\mathcal{D}_{\text{Train}}$, the ensembler's parameters θ are typically obtained by minimizing a loss function on a validation set $\mathcal{D}_{\text{Val}}$ such that:

$$\theta \in \arg\min_{\theta} \sum_{(x,y)\in\mathcal{D}_{\text{Val}}} \mathcal{L}(f(z(x); \theta), y). \tag{1}$$

In the general case, this objective function can be optimized using gradient-free optimization method such as evolutionary algorithms (Purucker & Beel, 2023b) or greedy search (Caruana et al., 2004) together with a linear combination $\theta \in \mathbb{R}^M$ of the model outputs:

$$f(z(x); \theta) = \sum_m \theta_m z_m(x). \tag{2}$$

Additionally, if we constraint the ensembler weights such that $\forall_i \theta_i \in \mathbb{R}_+$ and $\sum_i \theta_i = 1$ and assume probabilistic base models $z_m(x) = p(y|x, m)$, then we can interpret Equation 2 as:

$$p(y|x) = \sum_i p(y|x, m)p(m), \tag{3}$$

which is referred to as Bayesian Model Average, and uses $\theta_m = p(m)$. In the general case, the probabilistic ensembler $p(y|x) = p(y|z_1(x), ..., z_M(x), \beta)$ is a stacker model parametrized by β.

2.1 Motivating Dynamic Ensembling

We motivate in this Section the need for dynamic ensembling by analyzing base models' predictions in real data taken from our experimental metadatasets. In reality, Equation 3 does not specify the distribution $p(m)$. Generally, it is safe to assume that the performance associated with an ensembler $f(z_m(x), \theta)$ is optimal if we *dynamically* select the optimal aggregation $\theta_m(x) = p(m|m)$ on a per-data point basis, instead of a static θ. To motivate this observation, we selected four datasets

[1]https://anonymous.4open.science/r/NeuralEnsemblers

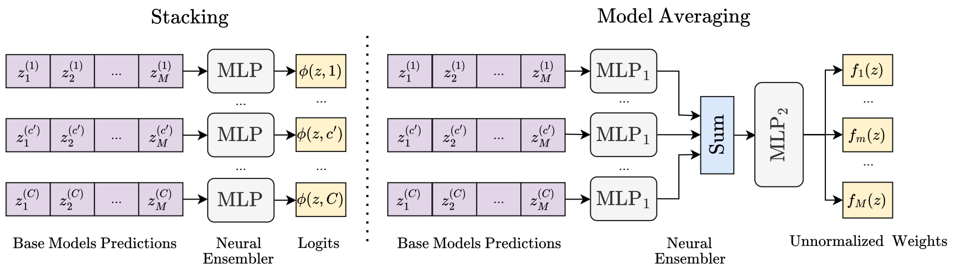


Figure 2: **Architecture of Neural Ensemblers (classification)**. The stacking mode uses a single MLP shared across base model class predictions. It outputs the logit per class, used for computing the final probability via SoftMax. In Model Averaging mode, it generates the unnormalized weights for every model, which are normalized with SoftMax.

from different modalities: *TabRepo* (Tabular data (Salinas & Erickson, 2023)), *QuickTune* (Computer Vision (Arango et al., 2024)), *FTC* (NLP-Section D.1) and *NasBench 201* (NAS for Computer Vision (Dong & Yang, 2020)). Then, we compute the per-sample error for 100 models in 100 samples. We report the results in Figure 1, indicating failed predictions with dark colors. We observe that models make different errors across samples, demonstrating the lack of optimality for static ensembling weights.

3 Neural Ensemblers

We use neural networks as ensemblers by training on the base model predictions $z(x) = [z_1(x); ...; z_M(x)]$ as input. For regression, $z(x) : \mathbb{R}^D \rightarrow \mathbb{R}^M$ outputs the base models' point predictions given by $x \in \mathbb{R}^D$, while for classification $z(x;c) : \mathbb{R}^D \rightarrow [0,1]^M$ returns the probabilities predicted by the base models for class c. In our discussion we consider two functional modes for the ensemblers: as network outputting weights for model averaging or as a direct stacking model that outputs the prediction. In **stacking** mode for regression, we aggregate the base model point predictions using a neural network to estimate the final prediction $\hat{y} = \phi(z;\beta)$, where β are the network parameters. In the **model-averaging** mode, the Neural Ensembler outputs the weights $\theta_m(z;\beta)$ to combine the model predictions as in Equation 4.

$$\hat{y} = \sum_m \theta_m(z;\beta) \cdot z_m(x). \tag{4}$$

Regardless of the functional mode, the Neural Ensembler has a different output $\hat{y}$ for regression and classification. In regression, the output $\hat{y}$ is a point estimation of the mean for a normal distribution such that $p(y|x;\beta) = \mathcal{N}(\hat{y}, \sigma)$. For classification, the input is the probabilistic prediction of the base models per class $z_m(x;c) = p(y = c|x, m)$, while the output is a categorical distribution $\hat{y} = p(y = c|z(x), \beta)$. We optimize β by minimizing the negative log-likelihood over the validation dataset $\mathcal{D}_{\text{Val}}$ as:

$$\min_{\beta} \mathcal{L}(\beta; D_{\text{Val}}) = \min_{\beta} \sum_{(x,y)\in\mathcal{D}_{\text{Val}}} -\log p(y|x;\beta). \tag{5}$$

3.1 Architecture

We discuss the architectural implementation of the Neural Ensembler for the classification case, which we show in Figure 2. For the **stacking** mode, we use an MLP that outputs the logit $\phi(z(x;c);\beta) : \mathbb{R}^M \rightarrow [0,1]$ of each class c. Specifically, the network receives as inputs the base models' predictions $z(x,c)$ for the class c and outputs the corresponding predicted logit for this class, i.e. $\phi(z;c)$. The model predictions per class are fed independently, enabling sharing the network parameters β. Subsequently, we compute the probability $p(y = c|x) = \frac{\exp^{\phi(z;c)}}{\sum_{c'} \exp^{\phi(z;c')}}$, with $\text{MLP}(z(x;c);\beta)$. In regression, the final prediction is the output $\phi(z(x);\beta)$.

For **model averaging** mode, we use a novel architecture based on a Deep Set (Zaheer et al., 2017) embedding of the base models predictions. We compute the dynamic weights $\theta_m(z;\beta) = \frac{\exp f_m(z;\beta)}{\sum_{m'} f_{m'}(z;\beta)}$, where the unnormalized weight per model $f_m(z;\beta)$ is determined via two MLPs. The first one $\text{MLP}_1 : \mathbb{R}^M \to \mathbb{R}^H$ embeds the predictions per class $z(x,c')$ into a latent dimension of size H, whereas the second network $\text{MLP}_2 : \mathbb{R}^M \to \mathbb{R}^H$ aggregates the embeddings and outputs the unnormalized weights, as shown in Equation 6. Notice that the Neural Ensemblers' input dimension and number of parameters are independent of the number of classes, due to our proposed parameter-sharing mechanism.

$$f(z;m,\beta) = \text{MLP}_2\left(\sum_{c'} \text{MLP}_1\left(z(x;c');\beta_1\right);\beta_2\right). \tag{6}$$

3.2 The Risk of Diversity Collapse

In this section, we elaborate on one risk that might arise during learning ensembles, as a base model might be highly correlated with the target in the validation set. We dub this model a *Preferred Base Model*.

Definition 1 (Preferred Base Model). *Consider a target variable $y \in R$ and a set of uncorrelated base models predictions $\mathcal{Z} = \{z_m | z_m \in \text{R}, m = 1, ..., M\}$. z_p is the Preferred Base Model if it has the highest sample correlation to the target, i.e. $\rho_{z_p,y} \in [0,1], \rho_{z_p,y} > \rho_{z_m,y}, \forall z_m \in \mathcal{Z}/\{z_p\}$.*

Proposition 1. *If the correlation of the preferred model $\rho_{z_p,y} \to 1$, then $\rho_{z_i,y} \to 0, \forall z_i \in \mathcal{Z}/\{z_p\}$, given the conditions of Definition 1.*

Sketch of Proof. Given that the base models predictions are uncorrelated, their correlations $\rho_{z_m,z_p} = 0, \forall m \neq p$, then it holds that $-\sqrt{1-\rho^2_{z_p,y}} \leq \rho_{z_m,y} \leq \sqrt{1-\rho^2_{z_p,y}}, \forall z_m \in \mathcal{F}/\{z_p\}$. We provide more details in the Appendix A. □

On the other hand, an important aspect when building ensembles is guaranteeing diversity among the models (Wood et al., 2023; Jeffares et al., 2024). This has motivated some approaches to explicitly account for diversity when searching the ensemble configuration (Shen et al., 2022; Purucker et al., 2023). A common way to measure diversity is the ambiguity (Krogh & Vedelsby, 1994), which can be derived after decomposing the loss function (Jeffares et al., 2024). Unfortunately, even simple ensembles risk obtaining low diversity although their base models are uncorrelated. As we will shortly show, this happens especially when there is a *preferred base model*.

Proposition 2 (Diversity Collapse). *If the correlation of the preferred model is $\rho_{z_p,y} \to 1$ in an ensemble with prediction $\bar{z} = \sum_m \theta_m z_m$, then the ensemble diversity $\alpha \to 0$, where $\alpha := \mathbb{E}\left[\sum_m \theta_m (z_m - \bar{z})^2\right]$, i.e. $\lim_{\rho_{z_p,y} \to 1} \alpha = 0$.*

We give a detailed proof in the Appendix A. However, we can build the intuition after noticing that given Proposition 1, $\lim_{\rho_{z_p,y} \to 1} \bar{z} = z_p$.

3.3 Base Models' DropOut

Using Neural Ensemblers tackles the need for dynamical ensembling. Moreover, it gives additional expressivity associated with neural networks. However, there is also a risk of overfitting and diversity collapse. As we showed in Section 3.2, this happens when there is a preferred model on which the ensembler mainly relies while neglecting other base model predictions. Although it might effectively decrease the validation loss (Equation 5), it does not necessarily generalize to test samples. Inspired by previous work (Srivastava et al., 2014), we propose to drop some base models during training forward passes. Intuitively, this forces the ensembler to rely on different base models to perform the predictions, instead of merely using the preferred base model(s).

```
Algorithm 1: Training Algorithm for Neural Ensemblers with Base Models' DropOut
Input: Base model predictions $\{z_1(x), ..., z_M(x)\}$, validation data $\mathcal{D}_{\text{Val}}$, probability of
       retaining $\gamma$, mode $\in \{\text{Stacking}, \text{Averaging}\}$.
Output: Neural Ensembler's parameters $\beta$
1  Initialize randomly parameters $\beta$ ;
2  while $done$ do
3      Sample masking vector $r \in \mathbb{R}^M, r_i \sim \text{Ber}(\gamma)$;
4      Mask base models predictions $z_{\text{drop}}(x) = r \odot z(x)$ ;
5      if $mode = \text{Stacking}$ then
6          Compute predictions $\hat{y} = \phi\left(\frac{1}{\gamma} z_{\text{drop}}; \beta\right)$ ;
7      else
8          Compute weights $\theta\left(\frac{1}{\gamma} z_{\text{drop}}; m, \beta, r\right)$ using Equation 7 ;
9          Compute predictions $\hat{y}$ using Equation 4 ;
10     end
11     Update neural ensembles parameters $\beta$ using $\nabla \mathcal{L}(\beta; \mathcal{D}_{\text{Val}})$
12 end
13 return $\beta$;
```

Formally, we mask the inputs such as $r_m \cdot z_m(x)$, where $r_m \sim \text{Ber}(\gamma)$, where $\text{Ber}(\gamma)$ is the Bernoulli distribution with parameter γ with represents the probability of keeping the base model, while $\delta = 1 - \gamma$ represents the DropOut rate. We also mask the weights when using model averaging:

$$\theta(z; m, \beta, r) = \frac{r_m \cdot \exp f_m(z; \beta)}{\sum_{m'} r_{m'} \cdot \exp f_{m'}(z; \beta)}. \tag{7}$$

As *DropOut* changes the scale of the inputs, the *weight scaling rule* should be applied during inference by multiplying the dropped variables by the retention probability γ. Alternatively, we can scale the variables during training by $\frac{1}{\gamma}$. In Algorithm 1, we detail how to train the Neural Ensemblers by dropping base model predictions. It has two modes, acting as a direct stacker or as a model averaging ensembler. We demonstrate that base models' DropOut avoids *diversity collapse* by lower bounding the diversity as stated in Proposition 6, even for the simplest ensembling case.

Proposition 3 (Avoiding Diversity Collapse). *As the correlation of the preferred model $\rho_{p_m,y} \to 1$, the diversity $\alpha \to 1 - \gamma$, when using Base Models' DropOut with probability of retaining γ.*

Sketch of Proof. We want to compute $\lim_{\rho_{z_p,y} \to 1} \alpha = \lim_{\rho_{z_p,y} \to 1} \mathbb{E}\left[\sum_m \theta_m (z_m - \bar{z})^2\right]$. By using $\bar{z} = \sum_m r_m \, \theta_m z_m$, and assuming, without loss of generality, that the predictions are standardized, we obtain $\mathbb{V}(r \cdot z_m) = \gamma$. This lead as to $\lim_{\rho_{z_p,y} \to 1} \alpha = 1 - \gamma$, after following a procedure similar to Proposition 2. We provide the complete proof in Appendix A.

4 Experiments and Results

4.1 Proof-of-Concept: Ensemble of Quadratic Functions

To further highlight the importance of the dynamical ensemble with diverse base models, we propose a simple regression problem with a third-degree polynomial with the ground truth function $y_{\text{true}} = 1.33x^3 - 0.77x^2 - 0.31x - 1$, where $x \in [-1, 1]$. We fit three second-degree polynomials on different subsets of the training dataset, to obtain three base functions $z_1(x), z_2(x), z_3(x)$. As ensemblers we consider *i)* a static model average $\sum_i \theta_m z_m(x)$, *ii)* a dynamic model stacker $\phi(z(x); \beta)$ and *iii)* model average $\sum_m \theta_m(z(x); \beta) \cdot z_m(x)$. In this experiment, $\phi(\cdot)$ and $\theta_m(\cdot)$ are two-layer MLPs with 10 hidden neurons with parameters β and trained on the validation data. Figure 3 shows the specific data, base functions, and learned ensemblers. The rightmost plot shows that the dynamic model averaging (MSE=0.0101) and stacker (MSE=0.0055), by expressing more complex functions, better model the ground truth than the static ensemble (MSE=0.1662).

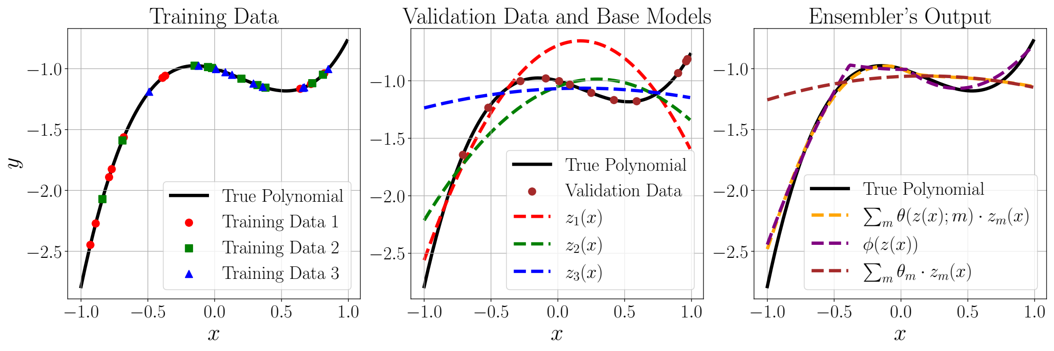


Figure 3: **Intuition Example for Dynamical Ensembling**. When considering three base models $z_1(x), z_2(x), z_3(x)$, the dynamical ensemblers $\phi(\cdot)$ and $\theta_m(\cdot)$ depending on x achieve better performance than a static model average, as observable in the right-most figure.

Table 1: Metadatasets Information.

Meta-Dataset	Modality	Task Information	No. Datasets	Avg. Samples for Validation	Avg. Samples for Test	Avg. Models per Dataset	Avg. Classes per Dataset
Nasbench (100)	Vision	NAS, Classification (Dong & Yang, 2020)	3	11000	6000	100	76.6
Nasbench (1K)	Vision	NAS, Classification (Dong & Yang, 2020)	3	11000	6000	1K	76.6
QuickTune (Micro)	Vision	Finetuning, Classification (Arango et al., 2024)	30	160	160	255	20.
QuickTune (Mini)	Vision	Finetuning, Classification (Arango et al., 2024)	30	1088	1088	203	136.
FTC	Language	Finetuning, Classification, Section D.1	6	39751	29957	105	4.6
TabRepo Clas.	Tabular	Classification (Salinas & Erickson, 2023)	83	1134	126	1530	3.4
TabRepo Reg.	Tabular	Regression (Salinas & Erickson, 2023)	17	3054	3397	1530	-
Sk-Learn Pipelines.	Tabular	Classification, Section D.2	69	1514	1514	500	5.08

4.2 Experimental Setup

Meta-Datasets. In our experiments, we utilize four meta-datasets with pre-computed predictions, which allows us to simulate ensembles without the need to fit models. These meta-datasets cover diverse data modalities, including Computer Vision, Tabular Data, and Natural Language Processing. Additionally, we evaluate the method on datasets without pre-computed predictions to assess the performance of ensembling methods that require model fitting. Table 1 reports the main information related to these datasets. Particularly for *Nasbench*, we created 2 versions by subsampling 100 and 1000 models. The metadataset for *Finetuning Text Classifiers (FTC)* was generated by ourselves to evaluate ensembling techniques on text classification tasks by finetuning language models such as GPT2 (Radford et al., 2019), Bert (Devlin et al., 2018) and Bart (Lewis et al., 2019). We also generate a set of fitted *Scikit-Learn Pipelines* on classification datasets. In this case, we stored the pipeline in memory, allowing us to evaluate our method in practical scenarios where the user has fitted models instead of predictions. We detailed information about the creation of these two meta-datasets in Appendix D. Information about each dataset lies in the respective referred work under the column *Task Information* in Table 1.

Baselines. We compare the **Neural Ensemblers (NE)** with other common and competitive ensemble approaches. 1) **Single best** selects the best model according to the validation metric; 2) **Random** chooses randomly $N = 50$ models to ensemble, 3) **Top-N** ensembles the best N models according to the validation metric; 4) **Greedy** creates an ensemble with $N = 50$ models by iterative selecting the one that improves the metric as proposed by previous work (Caruana et al., 2004); 5) **Quick** builds the ensemble with 50 models by adding model subsequently only if they strictly improve the metric; 6) **CMAES** (Purucker & Beel, 2023b) uses an evolutive strategy with a post-processing method for ensembling, 7) **Model Average (MA)** computed the sum of the predictions with constant weights as in Equation 3. We also compare to methods that perform ensemble search iteratively via Bayesian Optimization such as 8) **DivBO** (Shen et al., 2022), and 9) **Ensemble Optimization (EO)** (Levesque et al., 2016). Finally, we report results by using common ML models as stackers, such as 10) **SVM**, 11) **Random Forest**, 12) **Gradient Boosting**, and 13) **Logistic/Linear Regression**. We used the default configurations provided by Sckit-learn (Pedregosa et al., 2011) for these stackers. The input to the models is the concatenation of all the base models'

Table 2: Average Normalized Error.

	FTC	NB (100)	NB (1000)	QT-Micro	QT-Mini	TR-Class	TR-Class (AUC)
Single-Best	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$
Random	1.3377$_{\pm 0.2771}$	0.7283$_{\pm 0.0752}$	0.7491$_{\pm 0.2480}$	6.6791$_{\pm 3.4638}$	4.7284$_{\pm 2.9463}$	1.4917$_{\pm 1.6980}$	1.7301$_{\pm 1.8127}$
Top5	0.9511$_{\pm 0.0364}$	0.6979$_{\pm 0.0375}$	0.6296$_{\pm 0.1382}$	*0.6828*$_{\pm 0.3450}$	0.8030$_{\pm 0.2909}$	0.9998$_{\pm 0.1233}$	0.9271$_{\pm 0.2160}$
Top50	1.1012$_{\pm 0.1722}$	0.6347$_{\pm 0.0395}$	0.5650$_{\pm 0.1587}$	1.0662$_{\pm 0.9342}$	1.0721$_{\pm 0.4671}$	0.9800$_{\pm 0.1773}$	0.9297$_{\pm 0.2272}$
Quick	0.9494$_{\pm 0.0371}$	0.6524$_{\pm 0.0436}$	0.5787$_{\pm 0.1510}$	0.7575$_{\pm 0.2924}$	*0.7879*$_{\pm 0.2623}$	0.9869$_{\pm 0.1667}$	*0.9054*$_{\pm 0.2232}$
Greedy	0.9494$_{\pm 0.0374}$	0.7400$_{\pm 0.1131}$	1.0000$_{\pm 0.0000}$	0.9863$_{\pm 0.4286}$	0.9297$_{\pm 0.1435}$	0.9891$_{\pm 0.1693}$	0.9090$_{\pm 0.2197}$
CMAES	*0.9489*$_{\pm 0.0392}$	0.6401$_{\pm 0.0343}$	0.5797$_{\pm 0.1575}$	1.0319$_{\pm 0.5000}$	0.9086$_{\pm 0.1121}$	0.9935$_{\pm 0.1953}$	1.1878$_{\pm 1.1457}$
Random Forest	0.9513$_{\pm 0.0359}$	0.6649$_{\pm 0.0427}$	0.6891$_{\pm 0.3039}$	1.4738$_{\pm 1.3510}$	1.2530$_{\pm 0.4875}$	1.0041$_{\pm 0.2330}$	1.0924$_{\pm 0.6284}$
Gradient Boosting	1.0097$_{\pm 0.1033}$	1.2941$_{\pm 0.5094}$	1.2037$_{\pm 0.3528}$	0.8514$_{\pm 0.5003}$	1.6121$_{\pm 1.7023}$	1.0452$_{\pm 0.3808}$	1.0663$_{\pm 0.4884}$
SVM	**0.9453**$_{\pm 0.0383}$	0.6571$_{\pm 0.0483}$	0.7015$_{\pm 0.3067}$	1.1921$_{\pm 0.8266}$	1.4579$_{\pm 0.6233}$	**0.9585**$_{\pm 0.2160}$	1.4701$_{\pm 1.3486}$
Linear	0.9609$_{\pm 0.0347}$	0.7891$_{\pm 0.1978}$	0.7782$_{\pm 0.1941}$	0.7333$_{\pm 0.4457}$	0.9291$_{\pm 0.3580}$	0.9776$_{\pm 0.2844}$	1.0329$_{\pm 0.4022}$
MA	1.0709$_{\pm 0.0845}$	0.6381$_{\pm 0.0349}$	**0.5610**$_{\pm 0.1490}$	1.1548$_{\pm 0.8465}$	1.2173$_{\pm 0.6107}$	1.0917$_{\pm 1.0135}$	0.9977$_{\pm 0.2278}$
DivBO	1.0155$_{\pm 0.1452}$	0.6915$_{\pm 0.0536}$	0.9120$_{\pm 0.1524}$	1.3935$_{\pm 1.4316}$	1.0635$_{\pm 0.7587}$	1.0908$_{\pm 1.0104}$	1.0899$_{\pm 1.0297}$
EO	1.0208$_{\pm 0.1159}$	0.6365$_{\pm 0.0445}$	0.5704$_{\pm 0.1619}$	1.0185$_{\pm 0.6464}$	1.0367$_{\pm 0.4394}$	1.0851$_{\pm 1.0136}$	0.9377$_{\pm 0.2310}$
NE-Stack (Ours)	0.9491$_{\pm 0.0451}$	*0.6331*$_{\pm 0.0378}$	0.5836$_{\pm 0.1592}$	**0.6104**$_{\pm 0.3656}$	**0.7545**$_{\pm 0.2960}$	1.0440$_{\pm 0.3309}$	1.0035$_{\pm 0.5295}$
NE-MA (Ours)	0.9527$_{\pm 0.0402}$	**0.6307**$_{\pm 0.0363}$	*0.5621*$_{\pm 0.1483}$	0.8297$_{\pm 0.4974}$	0.8236$_{\pm 0.2240}$	*0.9592*$_{\pm 0.2144}$	**0.9028**$_{\pm 0.2157}$

predictions. We concatenated the probabilistic predictions from all the classes in the classification tasks. This sometimes produced a large dimensional input space. Finally, we include models from Dynamic Ensemble Search (DES) literature that are in the *DESlib* library (Cruz et al., 2020) such as **KNOP** (Cavalin et al., 2013), **KNORAE** (Ko et al., 2008) and **MetaDES** (Cruz et al., 2015).

Neural Ensemblers' Setup. We train the neural networks for 10000 update steps, with a batch size of 2048. If the GPU memory is not enough for fitting the network because of the number of base models, or the number of classes, we decreased the batch size to 256. Additionally, we used the Adam optimizer and a network with four layers, 32 neurons, and a probability of keeping base models $\gamma = 0.25$, or alternatively a DropOut rate $\delta = 0.75$. Notice that the architecture of the ensemblers slightly varies depending on the mode (*Stacking* or *Model Average*). For the Stacking mode, we use an MLP with four layers and 32 neurons with ReLU activations. For MA mode, we use two MLPs as in Equation 6: 1) MLP$_1$ has 3 layers with 32 neurons, while 2) MLP$_2$ has one layer with the same number of neurons. Although changing some of these hyperparameters might improve the performance, we keep this setup constant for all the experiments, after checking that the Neural Ensemblers perform well in a subset of the Quick-Tune metadataset (*extended version*).

4.3 Research Questions and Associated Experiments

RQ 1: Can Neural Ensemblers outperform other common and competitive ensembling methods across data modalities?

Experimental Protocol. To answer this question, we compare the neural ensembles in stacking and averaging mode to the baselines across all the meta-datasets. We run every ensembling method three times for every dataset. In all the methods we use the validation data for fitting the ensemble, while we report the results on the test split. Specifically, we report the average across datasets of two metrics: negative log-likelihood (NLL) and error. For the tabular classification, we compute the ROC-AUC. As these metrics vary for every dataset, we normalize metrics by dividing them by the *single-best* metric. Therefore, a method with a normalized metric below one is improving on top of using the single best base model. We report the standard deviation across the experiments per dataset and highlight in bold the best method.

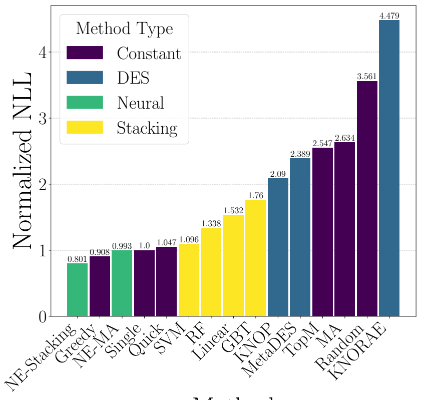


Figure 4: Results on Scikit-Learn Pipelines.

Results. The results reported in Table 2 and Table 3 show that our proposed neural networks **are competitive post-hoc ensemblers**. In general, we observe that the Neural Ensemblers variants obtain either the best (in bold) or second best (italic) performance across almost all meta-datasets

Table 3: Average Normalized NLL.

	FTC	NB (100)	NB (1000)	QT-Micro	QT-Mini	TR-Class	TR-Reg
Single-Best	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$
Random	1.5450$_{\pm 0.5289}$	0.6591$_{\pm 0.2480}$	0.7570$_{\pm 0.2900}$	6.8911$_{\pm 3.1781}$	5.8577$_{\pm 3.2546}$	1.7225$_{\pm 1.9645}$	1.8319$_{\pm 2.1395}$
Top5	0.8406$_{\pm 0.0723}$	0.6659$_{\pm 0.1726}$	0.6789$_{\pm 0.3049}$	1.5449$_{\pm 1.8358}$	1.1496$_{\pm 0.3684}$	1.0307$_{\pm 0.5732}$	$\underline{\textit{0.9939}}_{\pm 0.0517}$
Top50	0.8250$_{\pm 0.1139}$	0.5849$_{\pm 0.2039}$	$\underline{\textit{0.6487}}_{\pm 0.3152}$	3.3068$_{\pm 2.6197}$	3.0618$_{\pm 2.2960}$	1.0929$_{\pm 1.0198}$	1.0327$_{\pm 0.2032}$
Quick	0.7273$_{\pm 0.0765}$	0.5957$_{\pm 0.1940}$	0.6497$_{\pm 0.3030}$	1.1976$_{\pm 1.1032}$	0.9747$_{\pm 0.2082}$	$\underline{\textit{0.9860}}_{\pm 0.2201}$	1.0211$_{\pm 0.1405}$
Greedy	**0.6943**$_{\pm 0.0732}$	$\underline{\textit{0.5785}}_{\pm 0.1972}$	1.0000$_{\pm 0.0000}$	$\underline{\textit{0.9025}}_{\pm 0.2378}$	$\underline{\textit{0.9093}}_{\pm 0.1017}$	**0.9665**$_{\pm 0.0926}$	1.0149$_{\pm 0.1140}$
CMAES	1.2356$_{\pm 0.5295}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	4.1728$_{\pm 2.8724}$	4.6474$_{\pm 3.0180}$	1.3487$_{\pm 1.3390}$	1.0281$_{\pm 0.1977}$
Random Forest	0.7496$_{\pm 0.0940}$	0.8961$_{\pm 0.3159}$	0.9340$_{\pm 0.4262}$	3.7033$_{\pm 2.8145}$	2.2938$_{\pm 2.2068}$	1.2655$_{\pm 0.4692}$	1.0030$_{\pm 0.0871}$
Gradient Boosting	0.7159$_{\pm 0.1529}$	1.7288$_{\pm 1.2623}$	1.2764$_{\pm 0.4787}$	1.9373$_{\pm 1.2839}$	2.6193$_{\pm 2.3159}$	1.4288$_{\pm 1.2083}$	1.0498$_{\pm 0.2128}$
SVM	0.7990$_{\pm 0.0909}$	0.7744$_{\pm 0.2967}$	0.9358$_{\pm 0.5706}$	5.4377$_{\pm 3.3807}$	4.0019$_{\pm 3.6601}$	1.3884$_{\pm 1.4276}$	2.7975$_{\pm 3.0219}$
Linear	0.7555$_{\pm 0.0898}$	0.7400$_{\pm 0.2827}$	0.8071$_{\pm 0.2206}$	1.3960$_{\pm 1.2334}$	1.1031$_{\pm 0.7038}$	1.1976$_{\pm 1.1024}$	3.1488$_{\pm 3.2813}$
MA	0.9067$_{\pm 0.1809}$	0.5970$_{\pm 0.2034}$	0.6530$_{\pm 0.3028}$	4.7921$_{\pm 3.0780}$	4.0168$_{\pm 2.8560}$	1.4724$_{\pm 1.9401}$	1.3342$_{\pm 1.3515}$
DivBO	0.7695$_{\pm 0.1195}$	0.7307$_{\pm 0.3061}$	0.7125$_{\pm 0.3982}$	1.2251$_{\pm 1.0293}$	0.9430$_{\pm 0.2036}$	1.0023$_{\pm 0.3411}$	1.0247$_{\pm 0.1473}$
EO	0.7535$_{\pm 0.1156}$	0.5801$_{\pm 0.2051}$	0.6911$_{\pm 0.2875}$	1.3702$_{\pm 1.6389}$	0.9649$_{\pm 0.2980}$	1.0979$_{\pm 1.0289}$	1.0183$_{\pm 0.0993}$
NE-Stack (Ours)	0.7562$_{\pm 0.1836}$	**0.5278**$_{\pm 0.2127}$	**0.6336**$_{\pm 0.3456}$	**0.7486**$_{\pm 0.6831}$	**0.6769**$_{\pm 0.2612}$	1.3268$_{\pm 0.7498}$	1.2379$_{\pm 0.4083}$
NE-MA (Ours)	$\underline{\textit{0.6952}}_{\pm 0.0730}$	0.5822$_{\pm 0.2147}$	0.6522$_{\pm 0.3131}$	1.0177$_{\pm 0.5151}$	0.9166$_{\pm 0.0936}$	1.0515$_{\pm 1.0003}$	**0.9579**$_{\pm 0.0777}$

and metrics. Noteworthily, the greedy approach is very competitive, especially for the *FTC* and *TR-Class*. This is coherent with previous work supporting greedy ensembling as a robust method for tabular data (Erickson et al., 2020). We hypothesize that dynamic ensembling contributes partially to the strong results for the Neural Ensemblers. However, the expressivity gained is not enough, because it can lead to overfitting. To understand this, we compare to Dynamic Ensemble Selection (DES) methods. Specifically, we use *KNOP*, *MetaDES*, and *KNORAE*, and evaluate all methods in *Scikit-learn Pipelines* metadataset, as we can easily access the fitted models. We report the results of the test split in Figure 4, where we distinguish among four types of models to facilitate the reading: *Neural, DES, Stacking* and *Constant*. We can see that Neural Ensemblers are the most competitive approaches, especially on *stacking* mode. Additionally, we report the metrics on the validation split in Figure 8 (Appendix E), where we observe that some dynamic ensemble approaches such as *Gradient Boosting (GBT)*, *Random Forest (RF)* and *KNORAE* exhibit overfitting, while Neural Enemblers are more robust in this sense.

RQ 2: Do Neural Ensemblers need a strong group of base models, i.e. found using Bayesian Optimization?

Experimental Protocol. Practitioners use some methods such as greedy ensembling as post-hoc ensemblers, i.e., they consider a set of models selected by a search algorithm such as Bayesian Optimization as base learners. *DivBO* enhances the Bayesian Optimization by accounting for the diversity in the ensemble in the acquisition function. We run experiments to understand whether the Neural Ensemblers' performance depends on a strong subset of 50 base models selected by *DivBO*, and whether it can help other methods. We conduct additional experiments by randomly selecting 50 models to understand the impact and significance of merely using a smaller set of base models. We normalize the base of the metric on the *single-best* base model from the complete set contained in the respective dataset.

Results. We report in Table 4 the results with the two selection methods (random and *DivBO*) using a subset of common baselines, where we normalize using the metric of the *single-best* from the whole set of models. We limit the number of baselines for brevity, and the extended version includes more baselines in Table 15. We also include results in separate tables for the two selection methods (Appendix E). We can compare directly with the results in Table 3. We observe that reducing the number of base models with *DivBO* negatively affects the performance of the Neural Ensemblers. Surprisingly, randomly selecting the subset of base models improves the results in two metadatasets (*TR-Class* and *NB-1000*). We hypothesize that decreasing the number of base models is beneficial for these metadatasets. With over 1000 base models available, the likelihood of identifying a preferred model and overfitting the validation data increases in these metadatasets. Naturally, decreasing the number of base models can also be detrimental for the Neural Ensemblers, as this happens for some metadatasets such as *TR-Reg* and *QT-Micro*. In contrast to the Neural Ensemblers, selecting a subset of strong models with *DivBO* improves the performance for some baselines such as Model Averaging (MA) or *TopK* ($K = 25$). In other words, it works as a preprocessing method for these ensembling approaches. Overall, the results in Tables 4 and 15 demonstrate that **Neural Ensemblers do not need a strong group of base models to achieve competitive results**.

Table 4: Average Normalized NLL with a Subset of Base Models.

	Selector	FTC	NB (100)	NB (1000)	QT-Micro	QT-Mini	TR-Class	TR-Reg
Single	-	$1.0000_{\pm 0.0000}$	$1.0000_{\pm 0.0000}$	$1.0000_{\pm 0.0000}$	*1.0000*$_{\pm 0.0000}$	$1.0000_{\pm 0.0000}$	$1.0000_{\pm 0.0000}$	**1.0000**$_{\pm 0.0000}$
Single	DivBO	$1.0000_{\pm 0.0000}$	$1.0000_{\pm 0.0000}$	$0.8707_{\pm 0.3094}$	$1.7584_{\pm 2.0556}$	$1.1846_{\pm 0.2507}$	$1.1033_{\pm 0.9951}$	$1.0039_{\pm 0.0424}$
Random	DivBO	$0.9305_{\pm 0.3286}$	$0.6538_{\pm 0.2123}$	$0.9724_{\pm 0.0478}$	$1.1962_{\pm 1.0189}$	$0.9717_{\pm 0.1919}$	$1.0107_{\pm 0.3431}$	$1.0302_{\pm 0.1250}$
Top25	DivBO	$0.7617_{\pm 0.1136}$	**0.5564**$_{\pm 0.1961}$	$0.9762_{\pm 0.0413}$	$1.1631_{\pm 0.9823}$	$0.9431_{\pm 0.2035}$	$1.0023_{\pm 0.3411}$	$1.0247_{\pm 0.1473}$
Quick	DivBO	$0.7235_{\pm 0.0782}$	$0.6137_{\pm 0.1945}$	$0.9646_{\pm 0.0614}$	$1.2427_{\pm 1.1130}$	$0.9544_{\pm 0.2050}$	$1.0014_{\pm 0.3423}$	$1.0400_{\pm 0.1949}$
Greedy	DivBO	*0.7024*$_{\pm 0.0720}$	$0.6839_{\pm 0.3003}$	$0.9762_{\pm 0.0413}$	$1.1659_{\pm 0.9789}$	$0.9435_{\pm 0.2029}$	$1.0024_{\pm 0.3410}$	$1.0271_{\pm 0.1531}$
MA	DivBO	$0.7245_{\pm 0.0788}$	$0.5712_{\pm 0.2185}$	$0.9678_{\pm 0.0558}$	$1.0559_{\pm 0.7452}$	$0.9501_{\pm 0.1617}$	$1.0068_{\pm 0.4141}$	$1.0237_{\pm 0.1502}$
Single	Random	$1.0067_{\pm 0.0164}$	$1.0000_{\pm 0.0000}$	$0.9240_{\pm 0.3504}$	$1.2915_{\pm 0.9952}$	$1.1261_{\pm 0.3134}$	$1.0225_{\pm 0.3353}$	$1.1378_{\pm 0.4641}$
Top25	Random	$0.8397_{\pm 0.1000}$	$0.5848_{\pm 0.1980}$	*0.6526*$_{\pm 0.3019}$	$3.6553_{\pm 2.7053}$	$3.0436_{\pm 2.1378}$	$1.2599_{\pm 1.5015}$	$1.0611_{\pm 0.2799}$
Quick	Random	$0.7305_{\pm 0.0764}$	$0.5958_{\pm 0.1917}$	$0.6656_{\pm 0.2968}$	$1.7769_{\pm 2.1443}$	$1.1646_{\pm 0.3728}$	$1.0797_{\pm 1.0007}$	$1.0151_{\pm 0.1546}$
Greedy	Random	*0.7024*$_{\pm 0.0720}$	$0.5783_{\pm 0.1857}$	$0.6617_{\pm 0.2839}$	$1.6723_{\pm 2.1446}$	$0.9961_{\pm 0.1290}$	$1.0725_{\pm 0.9978}$	*1.0023*$_{\pm 0.0961}$
MA	Random	$0.9069_{\pm 0.1812}$	$0.8677_{\pm 0.2292}$	$0.6698_{\pm 0.2898}$	$4.8593_{\pm 3.1360}$	$3.4575_{\pm 2.6490}$	$1.4759_{\pm 1.9396}$	$1.4286_{\pm 1.7242}$
NE-Stack (Ours)	DivBO	$0.7715_{\pm 0.2141}$	$0.6204_{\pm 0.2234}$	$1.0000_{\pm 0.0000}$	$1.5040_{\pm 1.9442}$	*0.8329*$_{\pm 0.2659}$	*0.9729*$_{\pm 0.3952}$	$6.9453_{\pm 3.4749}$
NE-MA (Ours)	DivBO	$0.7036_{\pm 0.0698}$	*0.5704*$_{\pm 0.2345}$	$1.0000_{\pm 0.0000}$	$1.1237_{\pm 0.9964}$	$0.9200_{\pm 0.1966}$	$1.0016_{\pm 0.3407}$	$1.0070_{\pm 0.0977}$
NE-Stack (Ours)	Random	$0.7709_{\pm 0.2204}$	$0.7551_{\pm 0.2493}$	**0.6187**$_{\pm 0.2950}$	**0.8292**$_{\pm 0.5466}$	**0.8160**$_{\pm 0.3852}$	**0.9540**$_{\pm 0.5077}$	$4.2183_{\pm 3.4808}$
NE-MA (Ours)	Random	**0.6972**$_{\pm 0.0712}$	$0.7911_{\pm 0.2147}$	$0.6650_{\pm 0.2750}$	$1.6877_{\pm 2.1535}$	$1.0903_{\pm 0.2578}$	$1.0674_{\pm 0.9998}$	$1.0277_{\pm 0.1994}$

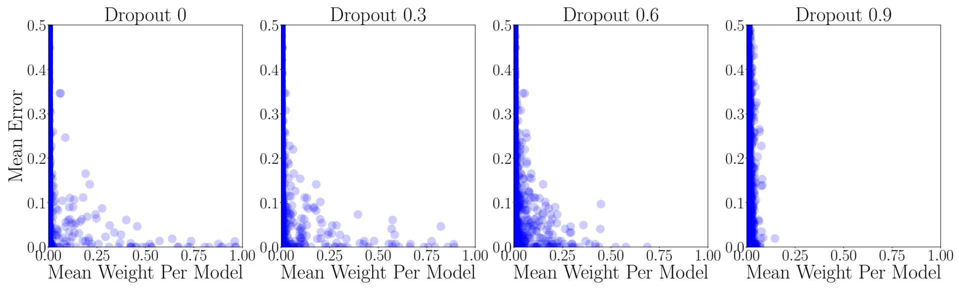


Figure 5: Mean weights assigned to the base models decrease with DropOut rate. Every data point is a model. The errors and weights are the mean values across many datasets for every model.

RQ 3: What is the impact of the DropOut regularization scheme?

Experimental Protocol. Firstly, we do a small experiment to observe how the DropOut rate affects the weights when using Neural Ensemblers in MA mode. For this, we use *QT-Micro* datasets, and save each model's weights θ_m and error after training the neural network. Subsequently, to understand how much the base learners DropOut helps the Neural Ensemblers, we run an ablation by trying the following values for the DropOut rate $\delta \in \{0.0, 0.1, \dots, 0.9\}$. We compute the average NLL for three seeds per dataset and divide this value by the one obtained for $\delta = 0.0$ in the same dataset. Therefore, we realize that a specific DropOut rate is improving over the default network without regularization if the normalized NLL is below 1.

Results. We show how the mean of the weights per model is related to the mean error of the models for different DropOut rates in Figure 5. When there is no DropOut some weights are close to one, i.e. they are preferred models. As we increase the value, many models with high weights decrease. If the rate is very high (e.g. 0.9), we will have many models contributing to the ensemble, with weights different from zero. Our ablation study demonstrates that non-existing or high DropOut are detrimental to the Neural Ensembler performance in general. As shown in Figure 6, this behavior is consistent in all datasets and both modes, but *TR-Reg* metadataset on *Stacking* mode. In general, we observe that **Neural Ensemblers obtain better performance when using base models' DropOut**.

5 Related Work

Ensembles for Tabular Data. For tabular data, ensembles are known to perform better than individual models (Sagi & Rokach, 2018; Salinas & Erickson, 2023).Therefore, ensembles are often used in real-world applications (Dong et al., 2020), to win competitions (Koren, 2009; Kaggle, 2024), and by automated machine learning (AutoML) systems as a modeling strategy (Purucker & Beel, 2023b; Purucker et al., 2023). Methods like Bagging (Breiman, 1996) or Boosting (Freund et al., 1996) are often used to boost the performance of individual models. In contrast, post-hoc ensembling (Shen et al., 2022; Purucker & Beel, 2023a) aggregates the predictions of an arbitrary set of

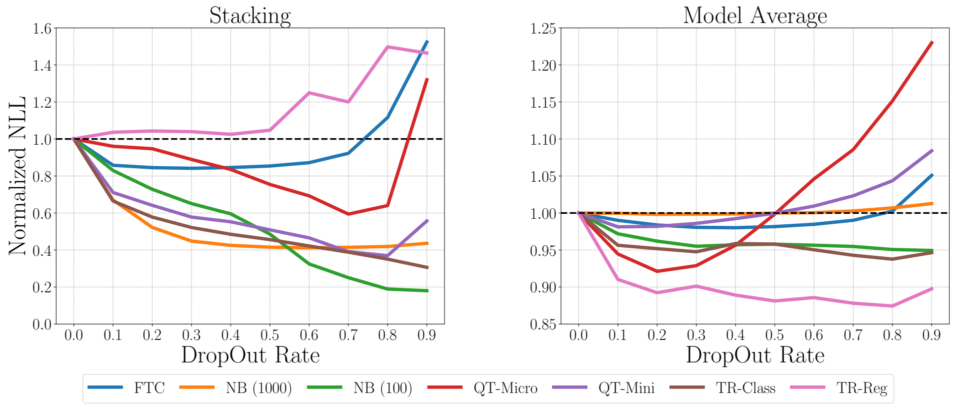


Figure 6: Ablation of the DropOut rate.

fitted base models. Post-hoc ensembles are build by stacking (Wolpert, 1992; Van der Laan et al., 2007), ensemble selection (a.k.a. pruning) (Caruana et al., 2004; Tsoumakas et al., 2009), dynamic ensemble selection (Ko et al., 2008; Britto Jr et al., 2014), or through a systematic search for an optimal ensemble (Levesque et al., 2016; Shen et al., 2022; Poduval et al., 2024).

Ensembles for Deep Learning. Ensembles of neural networks (Hansen & Salamon, 1990; Krogh & Vedelsby, 1994; Dietterich, 2000) have gained significant attention in deep learning research, both for their performance-boosting capabilities and their effectiveness in uncertainty estimation. Various strategies for building ensembles exist, with deep ensembles (Lakshminarayanan et al., 2017) being the most popular one, which involve independently training multiple initializations of the same network. Extensive empirical studies (Ovadia et al., 2019; Gustafsson et al., 2020) have shown that deep ensembles outperform other approaches for uncertainty estimation, such as Bayesian neural networks (Blundell et al., 2015; Gal & Ghahramani, 2016; Welling & Teh, 2011).

Mixture-of-Experts. Our idea of generating ensemble base model weights is closely connected to the mixture-of-experts (MoE) (Jacobs et al., 1991; Jordan & Jacobs, 1993; Shazeer et al., 2017), where one network is trained with specialized sub-modules that are activated based on the input data. Batch ensembles (Wen et al., 2020; Wenzel et al., 2020) are also closely related, as they aim to capture diverse model behaviors by simultaneously training multiple models with shared weights but different random projections. Alternatively, we could include a layer, aggregating predictions by encouraging diversity (Zhang et al., 2020). In contrast to these approaches, our Neural Ensemblers can ensemble any (black-box) model and are not restricted to gradient-based approaches.

Dynamic Ensemble Selection. Our Neural Ensembler is highly related to dynamic ensemble selection. Both dynamically aggregate the predictions of base models per instance (Cavalin et al., 2013; Ko et al., 2008). Traditional dynamic ensemble selection methods aggregate the most competent base models by paring heuristics to measure competence with clustering, nearest-neighbor-based, or traditional tabular algorithms (like naive Bayes) as meta-models (Cruz et al., 2018; 2020). In contrast, we use an end-to-end trained neural network to select *and weight* the base models per instance.

6 Conclusions

In this work, we tackled the challenge of post-hoc ensemble selection and the associated risk of overfitting on the validation set. We introduced the *Neural Ensembler*, a neural network that dynamically assigns weights to base models on a per-instance basis. To reduce overfitting, we proposed a regularization technique that randomly drops base models during training, which we theoretically showed enhances ensemble diversity. Our empirical results demonstrated that Neural Ensemblers consistently form competitive ensembles across diverse data modalities, including tabular data (classification and regression), computer vision, and natural language processing. In the future, we aim to explore in-context learning (Brown et al., 2020), where a pretrained Neural Ensembler could generate base model weights at test time, using their predictions as contextual input. We discuss broader impact and limitations in Appendix B.

Reproducibility Statement

All code and datasets used in this paper are publicly available at `https://anonymous.4open.science/r/NeuralEnsemblers`. We provide the code for running all the experiments, including baselines on all meta-datasets. We also provide instructions for the environment setup to ensure reproducibility. We will deanonymize the link upon acceptance. We further provide all proofs in Appendix A. Moreover, we ensured reproducibility by testing our approach on many meta-datasets and across several domains, for which details can be found in Appendix D. We also compared our method to many baselines commonly used in ensembling, described in Section 4.3. Lastly, we detail additional results and extended data tables in Appendix E.

References

Sebastian Pineda Arango, Fabio Ferreira, Arlind Kadra, Frank Hutter, and Josif Grabocka. Quick-tune: Quickly learning which pretrained model to finetune and how. In *The Twelfth International Conference on Learning Representations*, 2024.

Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren. Openml benchmarking suites. *arXiv:1708.03731v2 [stat.ML]*, 2019.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1613–1622, Lille, France, 07–09 Jul 2015. PMLR.

Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

Alceu S Britto Jr, Robert Sabourin, and Luiz ES Oliveira. Dynamic selection of classifiers—a comprehensive review. *Pattern recognition*, 47(11):3665–3680, 2014.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*. ACM, 2004.

Paulo R Cavalin, Robert Sabourin, and Ching Y Suen. Dynamic selection approaches for multiple classifier systems. *Neural computing and applications*, 22:673–688, 2013.

Rafael MO Cruz, Robert Sabourin, George DC Cavalcanti, and Tsang Ing Ren. Meta-des: A dynamic ensemble selection framework using meta-learning. *Pattern recognition*, 48(5):1925–1935, 2015.

Rafael MO Cruz, Robert Sabourin, and George DC Cavalcanti. Dynamic classifier selection: Recent advances and perspectives. *Information Fusion*, 41:195–216, 2018.

Rafael MO Cruz, Luiz G Hafemann, Robert Sabourin, and George DC Cavalcanti. Deslib: A dynamic ensemble selection library in python. *Journal of Machine Learning Research*, 21(8): 1–5, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

Thomas G. Dietterich. Ensemble Methods in Machine Learning. In *Multiple Classifier Systems*, pp. 1–15, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-45014-6.

Xibin Dong, Zhiwen Yu, Wenming Cao, Yifan Shi, and Qianli Ma. A survey on ensemble learning. *Frontiers Comput. Sci.*, 14(2):241–258, 2020. doi: 10.1007/S11704-019-8208-Z.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020.

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *icml*, volume 96, pp. 148–156. Citeseer, 1996.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR.

Fredrik K Gustafsson, Martin Danelljan, and Thomas B Schön. Evaluating Scalable Bayesian Deep Learning Methods for Robust Computer Vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

Lars K. Hansen and Peter Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

Alan Jeffares, Tennison Liu, Jonathan Crabbé, and Mihaela van der Schaar. Joint training of deep ensembles fails due to learner collusion. *Advances in Neural Information Processing Systems*, 36, 2024.

Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6:181–214, 1993.

Kaggle. Write-ups from the 2024 automl grand prix. `https://www.kaggle.com/automl-grand-prix`, 2024. (accessed: 14.09.2024).

Albert HR Ko, Robert Sabourin, and Alceu Souza Britto Jr. From dynamic classifier selection to dynamic ensemble selection. *Pattern recognition*, 41(5):1718–1731, 2008.

Yehuda Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81 (2009):1–10, 2009.

Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems*, volume 7. MIT Press, 1994.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942, 2019.

Julien-Charles Levesque, Christian Gagné, and Robert Sabourin. Bayesian hyperparameter optimization for ensemble learning. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence, UAI 2016, June 25-29, 2016, New York City, NY, USA*. AUAI Press, 2016.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

Wei Chen Maggie, Phil Culliton. Tweet sentiment extraction, 2020. URL `https://kaggle.com/competitions/tweet-sentiment-extraction`.

Jason D McEwen, Christopher GR Wallis, Matthew A Price, and Alessio Spurio Mancini. Machine learning assisted bayesian model comparison: learnt harmonic mean estimator. *arXiv preprint arXiv:2111.12720*, 2021.

Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pp. 485–492, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4206-3. doi: 10.1145/2908812.2908918.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems 32*, pp. 13991–14002. Curran Associates, Inc., 2019.

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.

Pranav Poduval, Sanjay Kumar Patnala, Gaurav Oberoi, Nitish Srivasatava, and Siddhartha Asthana. Cash via optimal diversity for ensemble learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pp. 2411–2419, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901.

Lennart Purucker and Joeran Beel. Assembled-openml: Creating efficient benchmarks for ensembles in automl with openml. *arXiv preprint arXiv:2307.00285*, 2023a.

Lennart Oswald Purucker and Joeran Beel. Cma-es for post hoc ensembling in automl: A great success and salvageable failure. In *International Conference on Automated Machine Learning*, pp. 1–1. PMLR, 2023b.

Lennart Oswald Purucker, Lennart Schneider, Marie Anastacio, Joeran Beel, Bernd Bischl, and Holger Hoos. Q(d)o-es: Population-based quality (diversity) optimisation for post hoc ensemble selection in automl. In *International Conference on Automated Machine Learning*, pp. 10–1. PMLR, 2023.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Omer Sagi and Lior Rokach. Ensemble learning: A survey. *WIREs Data Mining Knowl. Discov.*, 8 (4), 2018. doi: 10.1002/WIDM.1249.

David Salinas and Nick Erickson. Tabrepo: A large scale repository of tabular model evaluations and its automl applications, 2023.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Yu Shen, Yupeng Lu, Yang Li, Yaofeng Tu, Wentao Zhang, and Bin Cui. Divbo: diversity-aware cash for ensemble learning. *Advances in Neural Information Processing Systems*, 35:2958–2971, 2022.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Grigorios Tsoumakas, Ioannis Partalas, and Ioannis Vlahavas. An ensemble pruning primer. *Applications of supervised and unsupervised ensemble methods*, pp. 1–13, 2009.

Lewis Tunstall, Oren Pereg, Luke Bates, Moshe Wasserblat, Unso Eun, Daniel Korat, Nils Reimers, and Tome Aarsen. Setfit-mnli, 2021. URL `https://huggingface.co/datasets/SetFit/mnli`.

Mark J Van der Laan, Eric C Polley, and Alan E Hubbard. Super learner. *Statistical applications in genetics and molecular biology*, 6(1), 2007.

Eric-Jan Wagenmakers and Simon Farrell. Aic model selection using akaike weights. *Psychonomic bulletin & review*, 11:192–196, 2004.

Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pp. 681–688, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In *International Conference on Learning Representations*, 2020.

Florian Wenzel, Jasper Snoek, Dustin Tran, and Rodolphe Jenatton. Hyperparameter ensembles for robustness and uncertainty quantification. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6514–6527. Curran Associates, Inc., 2020.

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

Danny Wood, Tingting Mu, Andrew M Webb, Henry WJ Reeve, Mikel Lujan, and Gavin Brown. A unified theory of diversity in ensemble learning. *Journal of Machine Learning Research*, 24 (359):1–49, 2023.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Sheheryar Zaidi, Arber Zela, Thomas Elsken, Chris C Holmes, Frank Hutter, and Yee Teh. Neural ensemble search for uncertainty estimation and dataset shift. *Advances in Neural Information Processing Systems*, 34:7898–7911, 2021.

Shaofeng Zhang, Meng Liu, and Junchi Yan. The diversified ensemble neural network. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 16001–16011. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/b86e8d03fe992d1b0e19656875ee557c-Paper.pdf`.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

APPENDIX

We want to summarize here all the appendix sections:

- Section A presents the proofs of propositions in the main paper.
- Section B discusses the limitations of our proposed method and its broader impacts.
- Section C further details research similar to our work.
- Section D explains the process of gathering and preparing the *FTC* and *Scikit-learn Pipelines* metadatasets used in our experiments.
- Section E includes supplementary tables and figures, such as average ranks and detailed performance metrics, that support and expand upon the main experimental results reported in the paper.
- Section F includes he computational cost associated with our method compared to baseline approaches, including runtime evaluations and discussions on efficiency.
- Section G includes the results of a proof-of-concept experiment using overparameterized base models (e.g., 10th-degree polynomials), demonstrating the effectiveness of our Neural Ensembler even when base models have high capacity.
- Section H includes the Critical Difference diagrams corresponding to our main results, illustrating the statistical significance of performance differences among methods and how to interpret these diagrams.
- Section I includes results from experiments with additional baseline methods such as *CatBoost* and *XGBoost*.
- Section J includes a sensitivity analysis of the Neural Ensembler's hyperparameters, exploring how variations in network size (number of layers and neurons) affect performance across different datasets.
- Section K explores the impact of using different amounts of validation data to train the Neural Ensembler, assessing its sample efficiency and how performance scales with varying data sizes.
- Section L explores the effect of merging the training and validation datasets on the performance of both base models and ensemblers.
- Section M explores an alternative formulation of the Neural Ensembler that operates on the original input space rather than on the base model predictions, including experimental results and discussions on its effectiveness

A PROOFS

Definition 2 (Preferred Base Model)**.** *Consider a target variable $y \in R$ and a set of uncorrelated base models predictions $\mathcal{Z} = \{z_m | z_m \in \mathrm{R}, m = 1, ..., M\}$. z_p is the Preferred Base Model if it has the highest sample correlation with respect the target, i.e. $\rho_{z_p,y} \in [0,1], \rho_{z_p,y} > \rho_{z_m,y}, \forall z_m \in \mathcal{Z}/\{z_p\}$.*

Proposition 4. *If the correlation of the preferred model $\rho_{z_p,y} \to 1$, then $\rho_{z_i,y} \to 0, \forall z_i \in \mathcal{Z}/\{z_k\}$, given the conditions of Definition 1.*

Proof. Let $\rho_{z_p,z_m}, \rho_{z_m,y}, \rho_{z_p,y}$ be the correlations among the preferred model z_p, a non-preferred model $z_m \neq z_m$ and the target y, then the correlation matrix is given by,

$$C = \begin{pmatrix} 1 & \rho_{z_p,z_m} & \rho_{z_p,y} \\ \rho_{z_p,z_m} & 1 & \rho_{z_m,y} \\ \rho_{z_p,y} & \rho_{z_m,y} & 1 \end{pmatrix} \tag{8}$$

The determinant of the correlation matrix is given by,

$$\mathrm{Det}(C) = 1 - \rho^2_{z_p,z_m} - \rho^2_{z_p,y} - \rho^2_{z_m,y} + 2\rho_{z_p,z_m}\rho_{z_p,y}\rho_{z_m,y} \tag{9}$$

$$\mathrm{Det}(C) = 1 - \rho^2_{z_p,y} - \rho^2_{z_m,y} \tag{10}$$

$$\mathrm{Det}(C) = 1 - \rho^2_{z_p,y} - \rho^2_{z_m,y} \geq 0 \tag{11}$$

$$\rho_{z_m,y} \leq \sqrt{1 - \rho^2_{z_p,y}}, \tag{12}$$

where we used the fact that $\rho_{z_p,z_m} = 0$, in Equation 10 and that the correlation matrix should be semidefinite in Equation 11, thus its determinant should hold $\mathrm{Det}(C) \geq 0$. This means that $\epsilon = \sqrt{1 - \rho^2_{z_p,y}}$ is an upper bound for the absolute value of the correlation $\rho_{z_m,y}$. The limit of the upper bound ϵ as $\rho_{z_p,y} \to 1$ is 0 and thus $\rho_{z_m,y} = 0$,

$$\lim_{\rho_{z_p,y} \to 1} \epsilon = \lim_{\rho_{z_p,y} \to 1} \sqrt{1 - \rho^2_{z_p,y}} = 0 \tag{13}$$

$$\lim_{\rho_{z_p,y} \to 1} \rho_{z_m,y} = 0. \tag{14}$$

Since $\rho_{z_m,y}$ is bounded by $-\epsilon \leq \rho_{z_m,y} \leq \epsilon$.

□

Proposition 5 (Diversity Collapse)**.** *If the correlation of the preferred model is $\rho_{z_p,y} \to 1$ in an ensemble with prediction $\bar{z} = \sum_m \theta_m z_m$, then the ensemble diversity $\alpha \to 0$, where $\alpha := \mathbb{E}\left[\sum_m \theta_m (z_m - \bar{z})^2\right]$.*

Proof. Firstly, we show that under conditions of Definition 1, $\theta_m = \hat{\rho}_{z_m,y}$, where $\hat{\rho}_{z_m,y}$ is the *sample* correlation. Assume that we have a matrix of evaluations $X \in \mathrm{R}^{N \times M}$, with components $x_{n,m}$ indicating the prediction of the m-th model for the n-th instance. If $Y \in \mathbb{R}^{N \times 1}$ is the ground-truth, then the ensemble weights $\Theta \in \mathbb{R}^{M \times 1}$ can be computed with closed form solution of the objective $\min_\Theta ||Y - X\Theta||^2$ as: $\Theta = (X^T X)^{-1} X^T Y$.

Without loss of generality, we assume that the random variables are standardized. Given that every row of X have samples from uncorrelated random variables, then $X^T X \approx I$, denoting I as the identity matrix, thus

$$\Theta = X^T Y \tag{15}$$

Every component of Θ can expressed as summatorias $\theta_m = \sum_n x_{n,m} y_n$. Note that $x_{n,m}$ is a standardized sample (i.e. mean 0, variance 1) from the random variable z_m, thus it equals the sample correlation $\theta_m = \sum_n x_{n,m} y_n = \hat{\rho}_{z_m,y}$.

Now we develop the formal demonstration of Proposition 2. We note to the right side a hint to the criteria that was applied to derive every equation.

$$\lim_{\rho_{z_p,y}\to 1} \alpha = \lim_{\rho_{z_p,y}\to 1} \mathbb{E}\left[\sum_m \theta_m (z_m - \bar{z})^2\right] \qquad \text{Definition of } \alpha. \tag{16}$$

$$= \lim_{\rho_{z_p,y}\to 1} \sum_m \theta_m \mathbb{E}\left[(z_m - \bar{z})^2\right] \qquad \text{Properties of expectations.} \tag{17}$$

$$= \lim_{\rho_{z_p,y}\to 1} \sum_m \theta_m \left(\mathbb{E}[z_m - \bar{z}]^2 + \mathbb{V}[z_m - \bar{z}]\right) \qquad \text{Expectation of a squared variable.} \tag{18}$$

$$= \lim_{\rho_{z_p,y}\to 1} \sum_m \theta_m \mathbb{V}[z_m - \bar{z}] \qquad \text{Means of } z_m \text{ and } \bar{z} \text{ are 0.} \tag{19}$$

$$= \lim_{\rho_{z_p,y}\to 1} \sum_m \theta_m \left(\mathbb{V}[z_m] - \mathbb{V}[\bar{z}]\right) \qquad \text{Property of sum of variances.} \tag{20}$$

$$= \lim_{\rho_{z_p,y}\to 1} \sum_m \theta_m \left(1 - \mathbb{V}[\bar{z}]\right) \qquad \text{Variance of } z_m \text{ is 1.} \tag{21}$$

$$= \lim_{\rho_{z_p,y}\to 1} \sum_m \theta_m \left(1 - \mathbb{V}\left[\sum_{m'} \theta_{m'} z_{m'}\right]\right) \qquad \text{Definition of } \bar{z}. \tag{22}$$

$$= \lim_{\rho_{z_p,y}\to 1} \sum_m \theta_m \left(1 - \sum_{m'} \theta_{m'}^2 \mathbb{V}\left[z_{m'}\right]\right) \qquad \text{Properties of variance.} \tag{23}$$

$$= \lim_{\rho_{z_p,y}\to 1} \sum_m \theta_m \left(1 - \sum_{m'} \theta_{m'}^2\right) \qquad \text{Variance of } z_m \text{ is 1.} \tag{24}$$

$$= \sum_m \lim_{\rho_{z_p,y}\to 1} \theta_m \left(1 - \sum_{m'} \theta_{m'}^2\right) \qquad \text{Property of sum of limits.} \tag{25}$$

$$= \sum_m \lim_{\rho_{z_p,y}\to 1} \theta_m \cdot \lim_{\rho_{z_p,y}\to 1} \left(1 - \sum_{m'} \theta_{m'}^2\right) \qquad \text{Product-sum property of limits.} \tag{26}$$

$$= \left[\sum_m \lim_{\rho_{z_p,y}\to 1} \rho_{z_m,y}\right] \cdot \left[\lim_{\rho_{z_p,y}\to 1} \left(1 - \sum_{m'} \rho_{z_{m'},y}^2\right)\right] \qquad \text{Given that } \theta_m = \hat{\rho}_{z_m,y} \approx \rho_{z_m,y}. \tag{27}$$

$$= 1 \cdot \left[\lim_{\rho_{z_p,y}\to 1} \left(1 - \sum_{m'} \rho_{z_{m'},y}^2\right)\right] \qquad \text{Given Proposition 1.} \tag{28}$$

$$= 1 - \sum_{m'} \lim_{\rho_{z_p,y}\to 1} \rho_{z_{m'},y}^2 \qquad \text{Properties of limits.} \tag{29}$$

$$\lim_{\rho_{z_p,y}\to 1} \alpha = 0 \qquad \text{Given Proposition 1.} \tag{30}$$

□

Lemma 1. *When using dropout, it still holds that* $\theta_m = \hat{\rho}_{z_m,y}$.

Proof.

$$\min_\Theta \mathbb{E}_{R\sim \mathrm{Ber}(\gamma)} ||Y - (R * X)\Theta||^2 \tag{31}$$

$$= \min_\Theta ||Y - \gamma X\Theta||^2 + \gamma(1-\gamma)||\Gamma\Theta||^2, \tag{32}$$

where $\Gamma = (\mathrm{diag}(\mathrm{X}^{\mathrm{T}}\mathrm{X}))^{1/2}$. If we use the uncorrelation of the base models, we obtain:

$$= \min_\Theta ||Y - \gamma X\Theta||^2 + \gamma(1-\gamma)||\Theta||^2 \tag{33}$$

$$= \min_\Theta ||Y - \tilde{X}\Theta||^2 + \gamma(1-\gamma)||\Theta||^2, \tag{34}$$

which is equivalent to ridge regression, with closed form solution:

$$\begin{aligned}\Theta =& \left(\tilde{X}^T\tilde{X} + \gamma(1-\gamma)I\right)^{-1}\tilde{X}^TY && (35)\\ =& \left([\gamma^2 + \gamma(1-\gamma)]I\right)^{-1}\tilde{X}^TY && (36)\\ =& \left(\gamma I\right)^{-1}\tilde{X}^TY && (37)\\ =& \frac{1}{\gamma}\tilde{X}^TY && (38)\\ =& X^TY. && (39)\end{aligned}$$

□

Proposition 6 (Avoiding Diversity Collapse). *As the correlation of the preferred model $\rho_{p_m,y} \to 1$, the diversity $\alpha \to 1-\gamma$, when using Base Models' DropOut with probability of retaining γ.*

Proof. We follow a similar procedure as for Proposition 2, by considering $\bar{z} = \sum_m r_m\ \theta_m z_m$. We demonstrate that $\mathbb{V}(r \cdot z_m) = \gamma$, given that $r \sim \text{Bernoulli}(\gamma)$.

$$\begin{aligned}&\mathbb{V}(r_m \cdot z_m) = \mathbb{V}(r_m)\cdot\mathbb{V}(z_m) + \mathbb{V}(z_m)\cdot\mathbb{E}(r_m)^2 + \mathbb{V}(r_m)\cdot\mathbb{E}(z_m)^2 && (40)\\ &\mathbb{V}(r_m \cdot z_m) = \mathbb{V}(r_m) + \mathbb{E}(r_m)^2 && (41)\\ &\mathbb{V}(r_m \cdot z_m) = \gamma(1-\gamma) + \gamma^2 && (42)\\ &\mathbb{V}(r_m \cdot z_m) = \gamma. && (43)\end{aligned}$$

Then, we evaluate the variance of the ensemble using DropOut $\mathbb{V}(\bar{z})$:

$$\begin{aligned}\mathbb{V}(\bar{z}) &= \mathbb{V}\left(\sum_m r_m\cdot\theta_m\cdot z_m\right) && (44)\\ \mathbb{V}(\bar{z}) &= \sum_m \mathbb{V}(r_m\cdot\theta_m\cdot z_m) && (45)\\ \mathbb{V}(\bar{z}) &= \sum_m \theta_m^2\mathbb{V}(r_m\cdot z_m) && (46)\\ \mathbb{V}(\bar{z}) &= \gamma\sum_m\theta_m^2. && (47)\end{aligned}$$

Applying Equation 47 into Equation 22, we obtain:

$$\begin{aligned}\lim_{\rho_{z_p,y}\to 1}\alpha &= \lim_{\rho_{z_p,y}\to 1}\mathbb{E}\left[\sum_m \gamma_m\cdot\theta_m(z_m-\bar{z})^2\right] && (48)\\ &= \lim_{\rho_{z_p,y}\to 1}\sum_m\theta_m\left(1-\gamma\sum_m\theta_m^2\right) && (49)\\ &= \lim_{\rho_{z_p,y}\to 1}\left(1-\gamma\sum_{m'}\rho^2_{z_{m'},y}\right) && (50)\\ &= 1-\gamma\cdot\lim_{\rho_{z_p,y}\to 1}\left(\sum_{m'}\rho^2_{z_{m'},y}\right) && (51)\\ \lim_{\rho_{z_p,y}\to 1}\alpha &= 1-\gamma. && (52)\end{aligned}$$

□

B Limitations and Broader Impact.

While our proposed method offers several advantages for post-hoc ensemble selection, it is important to recognize its limitations. Unlike simpler ensembling heuristics, our approach requires tuning

Table 5: Metadatasets Information

Meta-Dataset	Modality	Task Information	No. Datasets	Avg. Samples for Validation	Avg. Samples for Test	Avg. Models per Dataset	Avg. Classes per Dataset
Nasbench (100)	Vision	NAS, Classification (Dong & Yang, 2020)	3	11000	6000	100	76.6
Nasbench (1K)	Vision	NAS, Classification (Dong & Yang, 2020)	3	11000	6000	1K	76.6
QuickTune (Micro)	Vision	Finetuning, Classification (Arango et al., 2024)	30	160	160	255	20.
QuickTune (Mini)	Vision	Finetuning, Classification (Arango et al., 2024)	30	1088	1088	203	136.
FTC	Language	Finetuning, Classification, Section D.1	6	39751	29957	105	4.6
TabRepo Clas.	Tabular	Classification (Salinas & Erickson, 2023)	83	1134	126	1530	3.4
TabRepo Reg.	Tabular	Regression (Salinas & Erickson, 2023)	17	3054	3397	1530	-
Sk-Learn Pipelines.	Tabular	Classification, Section D.2	69	1514	1514	500	5.08

Table 6: Search Space.

Hyperparameter	**Values**
Model	GPT2, Bert-Large, Albert-Large, Bart-Large, T5-Large
Learning Rate	0.00001, 0.0001, 0.0005, 0.001, 0.005
LoRA Rank	8, 16, 32, 64, 128

multiple training and architectural hyperparameters. Although we employed a fixed set of hyperparameters across all modalities and tasks in our experiments, this robustness may not generalize to all new tasks. In such cases, hyperparameter optimization may be necessary to achieve optimal performance. However, this could also enhance the results presented in this paper. Additionally, some bayesian approaches (McEwen et al., 2021) could further increase the robustness to the validation dataset size. Our approach is highly versatile and can be seamlessly integrated into a wide variety of ensemble-based learning systems, significantly enhancing their predictive capabilities. Because our method is agnostic to both the task and modality, we do not expect any inherent negative societal impacts. Instead, its effects will largely depend on how it is applied within different contexts and domains, making its societal implications contingent on the specific use case.

C Related Work Addendum

Ensemble Search via Bayesian Optimization. Ensembles of models with different hyperparameters can be built using Bayesian optimization by iteratively swapping a model inside an ensemble with another one that maximizes the expected improvement (Levesque et al., 2016). DivBO (Shen et al., 2022) and subsequent work (Poduval et al., 2024) combine the ensemble's performance and diversity as a measure for expected improvement. Besides Bayesian Optimization, an evolutionary search can find robust ensembles of deep learning models (Zaidi et al., 2021). Although these approaches find optimal ensembles, they can overfit the validation data used for fitting if run for many iterations.

D Details On Metadatasets

We provide general information about the datasets in Table 5.

D.1 Finetuning Text Classifiers (FTC) Metadataset

Search Space It comprises three hyperparameters: model, learning rate and LoRA rank (Hu et al., 2022). We consider consider five models: 1) GPT2, 124M parameters; (Radford et al., 2019); 2) Bert-Large, 336M ; (Devlin et al., 2018); 3) Bart-Large 400 M, parameters (Lewis et al., 2019); 4) Albert-Large, 17M parameters (Lan et al., 2019); and 5) T5-Large, 770 M parameters (Raffel et al., 2020). For the other two hyperparameters we also consider five different values as specified in Table 6.

Datasets The metadataset contains predictions of models finetuned in five metadatasets for text classification: 1) IMDB (Maas et al., 2011); 2) Tweet (Maggie, 2020), 3) News (Zhang et al., 2015), 4) DBpedia (Zhang et al., 2015), 5) SST2 (Socher et al., 2013) and 6) SetFit (Tunstall et al., 2021). For every dataset, we create two versions. The first one is trained with the complete training data, while the second version is only with a subset of 10 % of the samples. All the datasets are for text

Table 7: Metadataset Information

Dataset	# Classes	# Train Samples	# Val. Samples	# Test Samples	# Confs (100%)	# Confs. (10%)
IMDB (Maas et al., 2011)	2	20,000	5,000	25,000	125	125
Tweet (Maggie, 2020)	3	27,485	5,497	3,534	100	100
News (Zhang et al., 2015)	4	96,000	24,000	7,600	99	120
DBpedia (Zhang et al., 2015)	14	448,000	112,000	70,000	25	65
SST-2 (Socher et al., 2013)	2	43,103	13,470	10,776	125	125
SetFit (Tunstall et al., 2021)	3	393,116	78,541	62,833	25	100

classification beween 2 to 14 classes, including diverse domains such as movies, reviews, news, tweets, and text entailment data. We provide further information about the datasets in Table 7.

Metadataset Creation and Composition We created the dataset by finetuning every model to the train split and, subsequently, saving their predictions on the validation and test split. The validation split corresponds to 20 % of the available train data. As for some datasets (*SST-2* and *SetFit*) the test data is not completely provided by the creators, or it has hidden labels, we obtain it by using 20 % of the remaining test data. Specifically The models are fitted up to 5 epochs, using 1 GPU A100 with batch size equal to 2 and no LoRA dropout. We vary only the model, learning rate, and LoRA rank, while keeping the default hyperparameters in the TRAINER object from the *Transformers Library* (version 4.41.0). Although we evaluate the models in a grid, some runs yielded out-of-memory errors for some configurations. In total, the metadataset contains 1134 evaluated configurations, representing around 3800 GPU hours. Additionally, we report information about the metadataset in Table 7.

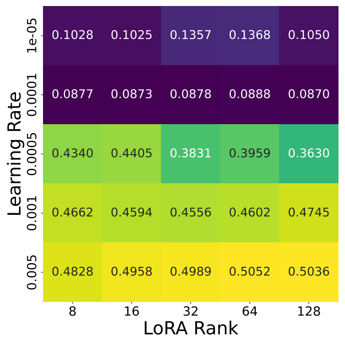


Figure 7: Mean error across datasets.

Hyperparameter Importance We explore the importance of two hyperparameters, learning rate, and LoRA rank, by plotting the mean error as a heatmap in Figure 7. The error corresponds to the average across different models and datasets. The learning rate is an important hyperparameter, while increasing the LoRA rank does not affect the performance significantly in low learning rates. This behaviour is interesting, as it showcases that a small rank is enough for a successful finetuning in this context.

D.2 SCIKIT LEARN PIPELINES

Search Space Our primary motivation is to investigate the ensembling of automated machine learning pipelines to enhance performance across various classification tasks. To effectively study ensembling methods and benchmark different strategies, we require a diverse set of pipelines. Therefore, we construct a comprehensive search space inspired by the TPOT library (Olson et al., 2016), encompassing a wide range of preprocessors, feature selectors, and classifiers. The pipelines are structured in three stages—preprocessor, feature selector, and classifier—which allows us to systematically explore numerous configurations. This extensive and diverse search space enables us to examine the impact of ensembling on a variety of models and serves as a robust benchmark for evaluating different ensembling techniques. Detailed descriptions of the components and their hyperparameters are provided in Tables 8, 9, and 10.

Table 8: Classifiers and their hyperparameters used in the TPOT search space.

Classifier	**Hyperparameters**
`sklearn.naive_bayes.GaussianNB`	None
`sklearn.naive_bayes.BernoulliNB`	`alpha` (float, [1e-3, 100.0], default=50.0) `fit_prior` (categorical, {True, False})
`sklearn.naive_bayes.MultinomialNB`	`alpha` (float, [1e-3, 100.0], default=50.0) `fit_prior` (categorical, {True, False})
`sklearn.tree.DecisionTreeClassifier`	`criterion` (categorical, {'gini', 'entropy'}) `max_depth` (int, [1, 10], default=5) `min_samples_split` (int, [2, 20], default=11) `min_samples_leaf` (int, [1, 20], default=11)
`sklearn.ensemble.ExtraTreesClassifier`	`n_estimators` (constant, 100) `criterion` (categorical, {'gini', 'entropy'}) `max_features` (float, [0.05, 1.0], default=0.525) `min_samples_split` (int, [2, 20], default=11) `min_samples_leaf` (int, [1, 20], default=11) `bootstrap` (categorical, {True, False})
`sklearn.ensemble.RandomForestClassifier`	`n_estimators` (constant, 100) `criterion` (categorical, {'gini', 'entropy'}) `max_features` (float, [0.05, 1.0], default=0.525) `min_samples_split` (int, [2, 20], default=11) `min_samples_leaf` (int, [1, 20], default=11) `bootstrap` (categorical, {True, False})
`sklearn.ensemble.GradientBoostingClassifier`	`n_estimators` (constant, 100) `learning_rate` (float, [1e-3, 1.0], default=0.5) `max_depth` (int, [1, 10], default=5) `min_samples_split` (int, [2, 20], default=11) `min_samples_leaf` (int, [1, 20], default=11) `subsample` (float, [0.05, 1.0], default=0.525) `max_features` (float, [0.05, 1.0], default=0.525)
`sklearn.neighbors.KNeighborsClassifier`	`n_neighbors` (int, [1, 100], default=50) `weights` (categorical, {'uniform', 'distance'}) `p` (categorical, {1, 2})
`sklearn.linear_model.LogisticRegression`	`penalty` (categorical, {'l1', 'l2'}) `C` (float, [1e-4, 25.0], default=12.525) `dual` (categorical, {True, False}) `solver` (constant, 'liblinear')
`xgboost.XGBClassifier`	`n_estimators` (constant, 100) `max_depth` (int, [1, 10], default=5) `learning_rate` (float, [1e-3, 1.0], default=0.5) `subsample` (float, [0.05, 1.0], default=0.525) `min_child_weight` (int, [1, 20], default=11) `n_jobs` (constant, 1) `verbosity` (constant, 0)
`sklearn.linear_model.SGDClassifier`	`loss` (categorical, {'log_loss', 'modified_huber'}) `penalty` (categorical, {'elasticnet'}) `alpha` (float, [0.0, 0.01], default=0.005) `learning_rate` (categorical, {'invscaling', 'constant'}) `fit_intercept` (categorical, {True, False}) `l1_ratio` (float, [0.0, 1.0], default=0.5) `eta0` (float, [0.01, 1.0], default=0.505) `power_t` (float, [0.0, 100.0], default=50.0)
`sklearn.neural_network.MLPClassifier`	`alpha` (float, [1e-4, 0.1], default=0.05) `learning_rate_init` (float, [0.0, 1.0], default=0.5)

Table 9: Preprocessors and their hyperparameters used in the TPOT search space.

Preprocessor	**Hyperparameters**
`None`	None
`sklearn.preprocessing.Binarizer`	`threshold` (float, [0.0, 1.0], default=0.5)
`sklearn.decomposition.FastICA`	`tol` (float, [0.0, 1.0], default=0.0)
`sklearn.cluster.FeatureAgglomeration`	`linkage` (categorical, {'ward', 'complete', 'average'}) `metric` (categorical, {'euclidean', 'l1', 'l2', 'manhattan', 'cosine'})
`sklearn.preprocessing.MaxAbsScaler`	None
`sklearn.preprocessing.MinMaxScaler`	None
`sklearn.preprocessing.Normalizer`	`norm` (categorical, {'l1', 'l2', 'max'})
`sklearn.kernel_approximation.Nystroem`	`kernel` (categorical, {'rbf', 'cosine', 'chi2', 'laplacian', 'polynomial', 'poly', 'linear', 'additive_chi2', 'sigmoid'}) `gamma` (float, [0.0, 1.0], default=0.5) `n_components` (int, [1, 10], default=5)
`sklearn.decomposition.PCA`	`svd_solver` (categorical, {'randomized'}) `iterated_power` (int, [1, 10], default=5)
`sklearn.preprocessing.PolynomialFeatures`	`degree` (constant, 2) `include_bias` (categorical, {False}) `interaction_only` (categorical, {False})
`sklearn.kernel_approximation.RBFSampler`	`gamma` (float, [0.0, 1.0], default=0.5)
`sklearn.preprocessing.RobustScaler`	None
`sklearn.preprocessing.StandardScaler`	None
`tpot.builtins.ZeroCount`	None
`tpot.builtins.OneHotEncoder`	`minimum_fraction` (float, [0.05, 0.25], default=0.15) `sparse` (categorical, {False}) `threshold` (constant, 10)

Table 10: Feature selectors and their hyperparameters used in the TPOT search space.

Selector	**Hyperparameters**
`None`	None
`sklearn.feature_selection.SelectFwe`	`alpha` (float, [0.0, 0.05], default=0.025)
`sklearn.feature_selection.SelectPercentile`	`percentile` (int, [1, 100], default=50)
`sklearn.feature_selection.VarianceThreshold`	`threshold` (float, [0.0001, 0.2], default=0.1)
`sklearn.feature_selection.RFE`	`step` (float, [0.05, 1.0], default=0.525) `estimator` (categorical, {'sklearn.ensemble.ExtraTreesClassifier'}) **Estimator Hyperparameters:** `n_estimators` (constant, 100) `criterion` (categorical, {'gini', 'entropy'}) `max_features` (float, [0.05, 1.0], default=0.525)
`sklearn.feature_selection.SelectFromModel`	`threshold` (float, [0.0, 1.0], default=0.5) `estimator` (categorical, {'sklearn.ensemble.ExtraTreesClassifier'}) **Estimator Hyperparameters:** `n_estimators` (constant, 100) `criterion` (categorical, {'gini', 'entropy'}) `max_features` (float, [0.05, 1.0], default=0.525)

Datasets We utilized the OpenML Curated Classification benchmark suite 2018 (OpenML-CC18) (Bischl et al., 2019) as the foundation for our meta-dataset. OpenML-CC18 comprises 72 diverse classification datasets carefully selected to represent a wide spectrum of real-world problems, varying in size, dimensionality, number of classes, and domains. This selection ensures comprehensive coverage across various types of classification tasks, providing a robust platform for evaluating the performance and generalizability of different ensembling approaches.

Metadataset Creation To construct our meta-dataset, we randomly selected 500 pipeline configurations for each dataset from our comprehensive search space. Each pipeline execution was constrained to a maximum runtime of 15 minutes. During this process, we had to exclude three datasets (*connect-4, Devnagari-Script, Internet-Advertisements*) due to excessive computational demands that exceeded our runtime constraints. For data preprocessing, we standardized the datasets by removing missing values and encoding categorical features. We intentionally left other preprocessing tasks to be handled autonomously by the pipelines themselves, allowing them to adapt to the specific characteristics of each dataset. This approach ensures that the pipelines can perform necessary transformations such as scaling, normalization, or feature engineering based on their internal configurations, which aligns with our objective of evaluating automated machine learning pipelines in a realistic setting.

Table 11: Average Ranked NLL

	FTC	NB-Micro	NB-Mini	QT-Micro	QT-Mini	TR-Class	TR-Reg
Single-Best	14.0000$_{\pm 0.8944}$	14.0000$_{\pm 1.0000}$	12.8333$_{\pm 1.0408}$	6.8833$_{\pm 2.3219}$	7.3667$_{\pm 2.5049}$	7.5361$_{\pm 3.8531}$	7.7647$_{\pm 3.9096}$
Random	16.0000$_{\pm 0.0000}$	7.6667$_{\pm 4.0415}$	9.3333$_{\pm 0.5774}$	15.2500$_{\pm 0.8068}$	15.2333$_{\pm 0.7739}$	12.0000$_{\pm 3.9962}$	13.6176$_{\pm 1.8669}$
Top5	11.3333$_{\pm 2.5033}$	9.6667$_{\pm 1.1547}$	7.0000$_{\pm 1.0000}$	6.9000$_{\pm 1.9360}$	8.3000$_{\pm 1.9678}$	6.2470$_{\pm 3.3951}$	7.2353$_{\pm 4.2907}$
Top50	10.8333$_{\pm 1.7224}$	4.6667$_{\pm 1.1547}$	4.3333$_{\pm 0.5774}$	11.4000$_{\pm 1.2959}$	11.5333$_{\pm 1.5025}$	6.8554$_{\pm 3.4006}$	6.8824$_{\pm 3.7397}$
Quick	6.1667$_{\pm 1.6021}$	6.3333$_{\pm 3.0551}$	*3.6667*$_{\pm 2.5166}$	4.7333$_{\pm 2.0331}$	6.0333$_{\pm 2.5661}$	6.2048$_{\pm 2.8531}$	*5.9412*$_{\pm 3.2107}$
Greedy	3.5000$_{\pm 0.8367}$	*4.0000*$_{\pm 2.6458}$	12.8333$_{\pm 1.0408}$	*3.5167*$_{\pm 1.7786}$	*4.5667*$_{\pm 2.4238}$	*5.9036*$_{\pm 3.0268}$	6.5882$_{\pm 3.2607}$
CMAES	12.1667$_{\pm 4.4460}$	14.0000$_{\pm 1.0000}$	12.8333$_{\pm 1.0408}$	12.9667$_{\pm 2.6811}$	13.9667$_{\pm 1.3322}$	9.9277$_{\pm 3.8880}$	6.2353$_{\pm 2.4630}$
Random Forest	7.0000$_{\pm 2.7568}$	13.0000$_{\pm 2.6458}$	14.0000$_{\pm 2.6458}$	11.9833$_{\pm 1.2964}$	9.3000$_{\pm 3.8699}$	11.4217$_{\pm 5.1493}$	8.2353$_{\pm 4.8157}$
Gradient Boosting	**3.1667**$_{\pm 4.8339}$	14.6667$_{\pm 1.5275}$	13.5000$_{\pm 2.1794}$	9.7000$_{\pm 5.2203}$	10.1167$_{\pm 5.1890}$	11.4217$_{\pm 4.9463}$	8.3529$_{\pm 4.7558}$
SVM	10.6667$_{\pm 1.3663}$	10.0000$_{\pm 7.0000}$	11.3333$_{\pm 7.2342}$	13.9167$_{\pm 1.2532}$	11.6833$_{\pm 5.0198}$	10.8434$_{\pm 4.2641}$	11.7941$_{\pm 5.5707}$
Linear	7.6667$_{\pm 2.0656}$	11.0000$_{\pm 1.0000}$	10.0000$_{\pm 2.0000}$	6.8000$_{\pm 2.8816}$	6.5833$_{\pm 3.3479}$	10.3855$_{\pm 5.2587}$	14.3529$_{\pm 3.4989}$
MA	11.1667$_{\pm 2.8577}$	6.6667$_{\pm 1.1547}$	4.3333$_{\pm 2.3094}$	13.2167$_{\pm 1.0059}$	12.8167$_{\pm 1.6000}$	9.0241$_{\pm 4.0567}$	8.2941$_{\pm 5.1813}$
DivBO	7.1667$_{\pm 3.5449}$	9.0000$_{\pm 4.5826}$	5.3333$_{\pm 5.7735}$	4.9167$_{\pm 2.5123}$	5.1000$_{\pm 2.7114}$	6.6145$_{\pm 2.9377}$	6.7647$_{\pm 2.9054}$
EO	5.8333$_{\pm 3.7103}$	5.3333$_{\pm 1.5275}$	7.3333$_{\pm 1.5275}$	5.8833$_{\pm 2.2194}$	5.5000$_{\pm 2.5052}$	6.2169$_{\pm 2.6689}$	7.0000$_{\pm 2.5739}$
NE-Stack (Ours)	6.0000$_{\pm 4.7329}$	**1.0000**$_{\pm 0.0000}$	**3.3333**$_{\pm 4.0415}$	**3.2000**$_{\pm 3.4381}$	**2.3333**$_{\pm 2.0899}$	10.0241$_{\pm 5.8351}$	12.2941$_{\pm 3.5314}$
NE-MA (Ours)	*3.3333*$_{\pm 2.5033}$	5.0000$_{\pm 3.0000}$	4.0000$_{\pm 1.7321}$	4.7333$_{\pm 2.0331}$	5.5667$_{\pm 1.9945}$	**5.3735**$_{\pm 3.2674}$	**4.6471**$_{\pm 2.3702}$

Table 12: Average Ranked Error

	FTC	NB-Micro	NB-Mini	QT-Micro	QT-Mini	TR-Class	TR-Class (AUC)
Single-Best	12.1667$_{\pm 2.5626}$	15.0000$_{\pm 0.8660}$	13.8333$_{\pm 0.7638}$	9.8167$_{\pm 3.7908}$	9.1167$_{\pm 3.5591}$	8.9157$_{\pm 4.1480}$	9.3086$_{\pm 4.1030}$
Random	15.8333$_{\pm 0.4082}$	13.0000$_{\pm 0.0000}$	12.0000$_{\pm 1.0000}$	15.9667$_{\pm 0.1826}$	15.9667$_{\pm 0.1826}$	11.3253$_{\pm 5.0723}$	12.2716$_{\pm 4.3747}$
Top5	6.0833$_{\pm 4.2710}$	11.6667$_{\pm 2.0817}$	8.3333$_{\pm 2.0817}$	*4.4000*$_{\pm 2.8780}$	4.6167$_{\pm 2.9204}$	8.7048$_{\pm 3.6325}$	7.2099$_{\pm 4.0519}$
Top50	13.5000$_{\pm 2.2583}$	*3.3333*$_{\pm 1.5275}$	**2.0000**$_{\pm 1.0000}$	8.3000$_{\pm 4.1618}$	9.7333$_{\pm 2.6546}$	7.9880$_{\pm 4.1548}$	7.2222$_{\pm 3.6929}$
Quick	6.3333$_{\pm 4.1433}$	7.6667$_{\pm 1.1547}$	6.3333$_{\pm 2.3094}$	5.5833$_{\pm 3.5186}$	*3.5500*$_{\pm 2.3575}$	8.3253$_{\pm 3.1771}$	*6.8148*$_{\pm 3.8033}$
Greedy	5.0833$_{\pm 3.8264}$	11.3333$_{\pm 3.0551}$	13.8333$_{\pm 0.7638}$	9.0000$_{\pm 3.5012}$	6.9833$_{\pm 3.1961}$	8.0843$_{\pm 3.3467}$	6.9321$_{\pm 3.2984}$
CMAES	*4.5833*$_{\pm 2.2004}$	5.0000$_{\pm 3.6056}$	6.3333$_{\pm 2.0817}$	8.5833$_{\pm 3.1103}$	6.3167$_{\pm 2.1794}$	8.8313$_{\pm 3.7776}$	9.6605$_{\pm 4.5537}$
Random Forest	6.6667$_{\pm 4.4121}$	8.3333$_{\pm 4.6188}$	9.5000$_{\pm 4.4441}$	11.7000$_{\pm 3.6237}$	12.5000$_{\pm 1.9343}$	8.8313$_{\pm 4.0817}$	8.7778$_{\pm 4.5600}$
Gradient Boosting	10.5000$_{\pm 3.2711}$	15.6667$_{\pm 0.2887}$	14.5000$_{\pm 1.5000}$	8.4667$_{\pm 4.2729}$	12.4000$_{\pm 3.1139}$	9.1024$_{\pm 4.9840}$	9.0123$_{\pm 5.0812}$
SVM	**3.5000**$_{\pm 3.3317}$	8.0000$_{\pm 1.7321}$	8.6667$_{\pm 5.1316}$	10.1167$_{\pm 4.8382}$	13.3667$_{\pm 2.6682}$	**7.0602**$_{\pm 3.9296}$	10.5185$_{\pm 5.4906}$
Linear	7.8333$_{\pm 2.6394}$	11.6667$_{\pm 3.5119}$	11.3333$_{\pm 2.0817}$	5.7500$_{\pm 3.2450}$	7.5167$_{\pm 2.9870}$	7.7892$_{\pm 5.2179}$	9.0185$_{\pm 5.1705}$
MA	12.8333$_{\pm 2.4014}$	4.6667$_{\pm 0.5774}$	*2.3333*$_{\pm 1.5275}$	10.0667$_{\pm 3.5349}$	11.3667$_{\pm 2.9271}$	8.4277$_{\pm 4.2949}$	8.7531$_{\pm 3.7200}$
DivBO	9.4167$_{\pm 3.3229}$	10.6667$_{\pm 1.1547}$	12.6667$_{\pm 1.5275}$	10.0833$_{\pm 2.9916}$	7.3833$_{\pm 3.2262}$	8.1506$_{\pm 3.6599}$	8.4938$_{\pm 3.4228}$
EO	10.0000$_{\pm 4.6476}$	4.6667$_{\pm 3.2146}$	4.0000$_{\pm 2.0000}$	9.3667$_{\pm 2.8555}$	8.5000$_{\pm 2.6425}$	8.1988$_{\pm 4.2677}$	7.6852$_{\pm 2.8410}$
NE-Stack (Ours)	4.6667$_{\pm 3.0768}$	*3.3333*$_{\pm 1.1547}$	7.3333$_{\pm 0.5774}$	**3.4500**$_{\pm 2.3095}$	**2.7167**$_{\pm 2.2232}$	8.9639$_{\pm 5.4352}$	7.6173$_{\pm 5.5437}$
NE-MA (Ours)	7.0000$_{\pm 1.7889}$	**2.0000**$_{\pm 1.0000}$	3.0000$_{\pm 2.0000}$	5.3500$_{\pm 3.7374}$	3.9667$_{\pm 2.2967}$	*7.3012*$_{\pm 3.9900}$	**6.7037**$_{\pm 3.7855}$

E ADDITIONAL RESULTS

In this Section we report additional results from our experiments:

- Average Ranking of baselines for the Negative Log-likelihood (Table 11) and Classification Errors (Table 12).
- Average NLL in *Scikit-learn Pipelines* metadataset (Figure 8).
- Average NLL using a subset of base models selected via *DivBO* (Table 13), random (Table 14) or both (Table 15).

Table 13: Average NLL for Subset of Base Models with DivBO

	Selector	FTC	NB (100)	NB (1000)	QT-Micro	QT-Mini	TR-Class	TR-Reg
Single	-	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	**1.0000**$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	*1.0000*$_{\pm 0.0000}$	**1.0000**$_{\pm 0.0000}$
Single	DivBO	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	**0.8707**$_{\pm 0.3094}$	1.7584$_{\pm 2.0556}$	1.1846$_{\pm 0.2507}$	1.1033$_{\pm 0.9951}$	*1.0039*$_{\pm 0.0424}$
Random	DivBO	0.9305$_{\pm 0.3286}$	0.6538$_{\pm 0.2123}$	0.9724$_{\pm 0.0478}$	1.1962$_{\pm 1.0189}$	0.9717$_{\pm 0.1919}$	1.0107$_{\pm 0.3431}$	1.0302$_{\pm 0.1250}$
Top25	DivBO	0.7617$_{\pm 0.1136}$	**0.5564**$_{\pm 0.1961}$	0.9762$_{\pm 0.0413}$	1.1631$_{\pm 0.9823}$	0.9431$_{\pm 0.2035}$	1.0023$_{\pm 0.3411}$	1.0247$_{\pm 0.1473}$
Quick	DivBO	0.7235$_{\pm 0.0782}$	0.6137$_{\pm 0.1945}$	*0.9646*$_{\pm 0.0614}$	1.2427$_{\pm 1.1130}$	0.9544$_{\pm 0.2050}$	1.0014$_{\pm 0.3423}$	1.0400$_{\pm 0.1949}$
Greedy	DivBO	**0.7024**$_{\pm 0.0720}$	0.6839$_{\pm 0.3003}$	0.9762$_{\pm 0.0413}$	1.1659$_{\pm 0.9789}$	0.9435$_{\pm 0.2029}$	1.0024$_{\pm 0.3410}$	1.0271$_{\pm 0.1531}$
MA	DivBO	0.7245$_{\pm 0.0788}$	0.5712$_{\pm 0.2185}$	0.9678$_{\pm 0.0558}$	*1.0559*$_{\pm 0.7452}$	0.9501$_{\pm 0.1617}$	1.0068$_{\pm 0.4141}$	1.0237$_{\pm 0.1502}$
NE-Stack (Ours)	DivBO	0.7715$_{\pm 0.2141}$	0.6204$_{\pm 0.2234}$	1.0000$_{\pm 0.0000}$	1.5040$_{\pm 1.9442}$	**0.8329**$_{\pm 0.2659}$	**0.9729**$_{\pm 0.3952}$	6.9453$_{\pm 3.4749}$
NE-MA (Ours)	DivBO	*0.7036*$_{\pm 0.0698}$	*0.5704*$_{\pm 0.2345}$	1.0000$_{\pm 0.0000}$	1.1237$_{\pm 0.9964}$	*0.9200*$_{\pm 0.1966}$	1.0016$_{\pm 0.3407}$	1.0070$_{\pm 0.0977}$

Table 14: Average NLL for Subset of Base Models randomly

	Seletor	FTC	NB (100)	NB (1000)	QT-Micro	QT-Mini	TR-Class	TR-Reg
Single	-	1.0000$_{\pm0.0000}$	1.0000$_{\pm0.0000}$	1.0000$_{\pm0.0000}$	*1.0000*$_{\pm0.0000}$	1.0000$_{\pm0.0000}$	*1.0000*$_{\pm0.0000}$	**1.0000**$_{\pm0.0000}$
Single	Random	1.0067$_{\pm0.0164}$	1.0000$_{\pm0.0000}$	0.9240$_{\pm0.3504}$	1.2915$_{\pm0.9952}$	1.1261$_{\pm0.3134}$	1.0225$_{\pm0.3353}$	1.1378$_{\pm0.4641}$
Top25	Random	0.8397$_{\pm0.1000}$	*0.5848*$_{\pm0.1980}$	*0.6526*$_{\pm0.3019}$	3.6553$_{\pm2.7053}$	3.0436$_{\pm2.1378}$	1.2599$_{\pm1.5015}$	1.0611$_{\pm0.2799}$
Quick	Random	0.7305$_{\pm0.0764}$	0.5958$_{\pm0.1917}$	0.6656$_{\pm0.2968}$	1.7769$_{\pm2.1443}$	1.1646$_{\pm0.3728}$	1.0797$_{\pm1.0007}$	1.0151$_{\pm0.1546}$
Greedy	Random	*0.7024*$_{\pm0.0720}$	**0.5783**$_{\pm0.1857}$	0.6617$_{\pm0.2839}$	1.6723$_{\pm2.1446}$	*0.9961*$_{\pm0.1290}$	1.0725$_{\pm0.9978}$	*1.0023*$_{\pm0.0961}$
MA	Random	0.9069$_{\pm0.1812}$	0.8677$_{\pm0.2292}$	0.6698$_{\pm0.2898}$	4.8593$_{\pm3.1360}$	3.4575$_{\pm2.6490}$	1.4759$_{\pm1.9396}$	1.4286$_{\pm1.7242}$
NE-Stack	Random	0.7709$_{\pm0.2204}$	0.7551$_{\pm0.2493}$	**0.6187**$_{\pm0.2950}$	**0.8292**$_{\pm0.5466}$	**0.8160**$_{\pm0.3852}$	**0.9540**$_{\pm0.5077}$	4.2183$_{\pm3.4808}$
NE-MA	Random	**0.6972**$_{\pm0.0712}$	0.7911$_{\pm0.2147}$	0.6650$_{\pm0.2750}$	1.6877$_{\pm2.1535}$	1.0903$_{\pm0.2578}$	1.0674$_{\pm0.9998}$	1.0277$_{\pm0.1994}$

Table 15: Average NLL for Subset of Base Models (more baselines)

	Selector	FTC	NB (100)	NB (1000)	QT-Micro	QT-Mini	TR-Class	TR-Reg
Single	DivBO	1.0000$_{\pm0.0000}$	1.0000$_{\pm0.0000}$	1.0000$_{\pm0.0000}$	1.0000$_{\pm0.0000}$	1.0000$_{\pm0.0000}$	1.0000$_{\pm0.0000}$	*1.0000*$_{\pm0.0000}$
Random	DivBO	0.9305$_{\pm0.3286}$	0.6538$_{\pm0.2123}$	0.9767$_{\pm0.0404}$	0.8220$_{\pm0.3507}$	0.8814$_{\pm0.2062}$	0.9748$_{\pm0.1415}$	1.0287$_{\pm0.1365}$
Top25	DivBO	0.7617$_{\pm0.1136}$	**0.5564**$_{\pm0.1961}$	0.9805$_{\pm0.0339}$	0.7916$_{\pm0.3245}$	0.8569$_{\pm0.2178}$	0.9672$_{\pm0.1381}$	1.0232$_{\pm0.1550}$
Quick	DivBO	0.7235$_{\pm0.0782}$	0.6137$_{\pm0.1945}$	0.9687$_{\pm0.0542}$	0.8048$_{\pm0.3271}$	0.8672$_{\pm0.2195}$	0.9652$_{\pm0.1353}$	1.0378$_{\pm0.1959}$
Greedy	DivBO	*0.7024*$_{\pm0.0720}$	0.6839$_{\pm0.3003}$	0.9805$_{\pm0.0339}$	0.7910$_{\pm0.3209}$	0.8572$_{\pm0.2174}$	0.9673$_{\pm0.1384}$	1.0256$_{\pm0.1601}$
CMAES	DivBO	0.8915$_{\pm0.1759}$	1.0000$_{\pm0.0000}$	1.0000$_{\pm0.0000}$	1.0000$_{\pm0.0000}$	1.0000$_{\pm0.0000}$	0.9671$_{\pm0.1179}$	1.0250$_{\pm0.1595}$
Random Forest	DivBO	0.7932$_{\pm0.1194}$	0.9338$_{\pm0.3436}$	1.1678$_{\pm0.2906}$	1.6807$_{\pm1.1160}$	1.5507$_{\pm0.8449}$	1.3195$_{\pm0.6428}$	1.0534$_{\pm0.1277}$
Gradient Boosting	DivBO	0.7908$_{\pm0.1848}$	1.4011$_{\pm0.5359}$	1.0000$_{\pm0.0000}$	2.4553$_{\pm1.1191}$	2.7220$_{\pm1.3007}$	1.2106$_{\pm0.5937}$	1.1246$_{\pm0.4052}$
Linear	DivBO	0.7433$_{\pm0.0870}$	0.6471$_{\pm0.2272}$	1.0248$_{\pm0.0430}$	0.9690$_{\pm0.4209}$	1.1132$_{\pm0.4525}$	1.0814$_{\pm1.0056}$	1.0316$_{\pm0.1403}$
SVM	DivBO	0.8312$_{\pm0.0943}$	0.7406$_{\pm0.2612}$	1.0786$_{\pm0.1362}$	5.2463$_{\pm3.4452}$	4.8194$_{\pm3.5169}$	1.5156$_{\pm1.5597}$	2.7777$_{\pm2.9458}$
MA	DivBO	0.7245$_{\pm0.0788}$	0.5712$_{\pm0.2185}$	0.9719$_{\pm0.0486}$	*0.7603*$_{\pm0.2876}$	0.8802$_{\pm0.1741}$	*0.9621*$_{\pm0.1306}$	1.0222$_{\pm0.1570}$
Single	Random	1.0067$_{\pm0.0164}$	1.0000$_{\pm0.0000}$	1.0540$_{\pm0.0936}$	0.8958$_{\pm0.3297}$	0.9786$_{\pm0.2444}$	0.9902$_{\pm0.1529}$	1.1314$_{\pm0.4475}$
Top25	Random	0.8397$_{\pm0.1000}$	0.5848$_{\pm0.1980}$	*0.7283*$_{\pm0.1534}$	2.7131$_{\pm1.5510}$	2.5961$_{\pm1.9153}$	1.2873$_{\pm1.5773}$	1.0588$_{\pm0.2762}$
Quick	Random	0.7305$_{\pm0.0764}$	0.5958$_{\pm0.1917}$	0.7467$_{\pm0.1420}$	0.9847$_{\pm0.1836}$	0.9737$_{\pm0.1340}$	1.0043$_{\pm0.2497}$	1.0123$_{\pm0.1529}$
Greedy	Random	*0.7024*$_{\pm0.0720}$	0.5783$_{\pm0.1857}$	0.7478$_{\pm0.1352}$	0.9171$_{\pm0.0932}$	0.8960$_{\pm0.0795}$	0.9762$_{\pm0.1082}$	1.0002$_{\pm0.1038}$
CMAES	Random	1.2164$_{\pm0.3851}$	1.0000$_{\pm0.0000}$	1.0000$_{\pm0.0000}$	1.0000$_{\pm0.0000}$	1.0000$_{\pm0.0000}$	1.1544$_{\pm0.9890}$	**0.9985**$_{\pm0.0954}$
RF	Random	0.7505$_{\pm0.0915}$	0.8325$_{\pm0.2255}$	0.9850$_{\pm0.0175}$	2.9838$_{\pm1.7970}$	2.6001$_{\pm1.8694}$	1.2006$_{\pm0.4367}$	1.0058$_{\pm0.0898}$
GBT	Random	0.7235$_{\pm0.1605}$	1.8377$_{\pm1.4510}$	1.2177$_{\pm0.3771}$	1.9521$_{\pm1.2076}$	2.8683$_{\pm1.9763}$	1.2329$_{\pm0.6934}$	1.0470$_{\pm0.1913}$
Linear	Random	0.7541$_{\pm0.0897}$	0.7400$_{\pm0.2827}$	0.9103$_{\pm0.1384}$	1.1776$_{\pm0.3187}$	1.5498$_{\pm0.8446}$	1.0417$_{\pm1.0146}$	1.0408$_{\pm0.1296}$
SVM	Random	0.8010$_{\pm0.0903}$	0.7767$_{\pm0.3006}$	0.9821$_{\pm0.3676}$	5.1993$_{\pm3.4269}$	5.1080$_{\pm3.2902}$	1.4479$_{\pm1.4990}$	2.8328$_{\pm3.0214}$
MA	Random	0.9069$_{\pm0.1812}$	0.8677$_{\pm0.2292}$	0.7543$_{\pm0.1312}$	4.1694$_{\pm2.5844}$	3.0315$_{\pm2.4578}$	1.5059$_{\pm1.9654}$	1.4185$_{\pm1.6833}$
NE-Stack	Random	0.7709$_{\pm0.2204}$	0.7551$_{\pm0.2493}$	**0.6882**$_{\pm0.1602}$	**0.6925**$_{\pm0.4510}$	**0.6854**$_{\pm0.2801}$	**0.9387**$_{\pm0.4628}$	4.2065$_{\pm3.4678}$
NE-MA	Random	**0.6972**$_{\pm0.0712}$	0.7911$_{\pm0.2147}$	0.7535$_{\pm0.1164}$	0.9291$_{\pm0.0999}$	0.9191$_{\pm0.0624}$	0.9922$_{\pm0.2525}$	1.0259$_{\pm0.2002}$
NE-Stack	DivBO	0.7715$_{\pm0.2141}$	0.6204$_{\pm0.2234}$	1.0000$_{\pm0.0000}$	1.0442$_{\pm0.8691}$	*0.7482*$_{\pm0.2254}$	0.9671$_{\pm0.3746}$	6.9574$_{\pm3.4368}$
NE-MA	DivBO	0.7036$_{\pm0.0698}$	*0.5704*$_{\pm0.2345}$	1.0000$_{\pm0.0000}$	0.7632$_{\pm0.3319}$	0.8349$_{\pm0.2075}$	0.9677$_{\pm0.1455}$	1.0057$_{\pm0.1121}$

E.1 UNNORMALIZED RESULTS

We report the results for the research question 1 in Tables 16 and 17, omitting the normalization. This helps us to understand how much the normalization is changing the results. However, as the different datasets have metrics with different scales, it is important to normalize the results to get a better picture of the relative performances. This is especially problematic in the regression tasks, as it depends on the target scale, therefore we omit it. Even without the normalization, our proposed approach achieves the best results across different datasets. The ranking results remain the same independently of the normalization.

Table 16: Average Unnormalized NLL.

	FTC	NB-Micro	NB-Mini	QT-Micro	QT-Mini	TR-Class
Single-Best	0.3412$_{\pm0.3043}$	1.9175$_{\pm1.2203}$	1.5696$_{\pm0.8119}$	0.4499$_{\pm0.5107}$	0.6339$_{\pm0.5276}$	0.3647$_{\pm0.3300}$
Random	0.4521$_{\pm0.2982}$	1.2202$_{\pm1.0069}$	1.3005$_{\pm0.9988}$	2.1473$_{\pm0.6480}$	2.7139$_{\pm1.0686}$	0.4344$_{\pm0.3341}$
Top5	0.2975$_{\pm0.2799}$	1.2940$_{\pm0.9934}$	1.1911$_{\pm0.9833}$	0.4234$_{\pm0.4473}$	0.6702$_{\pm0.5330}$	0.3576$_{\pm0.3306}$
Top50	0.2819$_{\pm0.2631}$	1.1515$_{\pm0.9713}$	*1.1496*$_{\pm0.9797}$	0.7813$_{\pm0.5492}$	1.2587$_{\pm0.8605}$	*0.3561*$_{\pm0.3331}$
Quick	0.2443$_{\pm0.2128}$	1.1679$_{\pm0.9614}$	**1.1455**$_{\pm0.9608}$	0.3952$_{\pm0.4441}$	0.6030$_{\pm0.4869}$	0.3562$_{\pm0.3326}$
Greedy	*0.2366*$_{\pm0.2107}$	1.1498$_{\pm0.9652}$	1.5696$_{\pm0.8119}$	*0.3895*$_{\pm0.4453}$	*0.5817*$_{\pm0.4870}$	0.3566$_{\pm0.3317}$
CMAES	0.3552$_{\pm0.2130}$	1.9175$_{\pm1.2203}$	1.5696$_{\pm0.8119}$	1.1665$_{\pm0.7614}$	1.8487$_{\pm0.6070}$	0.3862$_{\pm0.3325}$
Random Forest	0.2434$_{\pm0.2033}$	1.7354$_{\pm1.4580}$	1.6589$_{\pm1.4265}$	0.8846$_{\pm0.6550}$	0.9870$_{\pm0.8090}$	0.4452$_{\pm0.3734}$
Gradient Boosting	**0.2262**$_{\pm0.1915}$	2.2979$_{\pm0.5750}$	1.8206$_{\pm0.6327}$	1.2480$_{\pm1.6045}$	1.7043$_{\pm1.9723}$	0.4708$_{\pm0.4264}$
SVM	0.2626$_{\pm0.2241}$	1.6698$_{\pm1.4024}$	1.6590$_{\pm1.4274}$	1.3608$_{\pm0.3953}$	1.8931$_{\pm1.2176}$	0.3951$_{\pm0.3356}$
Linear	0.2496$_{\pm0.2130}$	1.3021$_{\pm0.9815}$	1.3626$_{\pm0.9836}$	0.4701$_{\pm0.5151}$	0.6582$_{\pm0.6088}$	0.4095$_{\pm0.3914}$
MA	0.2804$_{\pm0.2074}$	1.1612$_{\pm0.9664}$	1.1530$_{\pm0.9720}$	1.0974$_{\pm0.5102}$	1.5240$_{\pm0.8090}$	0.3788$_{\pm0.3342}$
DivBO	0.2643$_{\pm0.2526}$	1.2355$_{\pm0.8746}$	1.1659$_{\pm0.9729}$	0.4065$_{\pm0.4489}$	0.5906$_{\pm0.4825}$	0.3588$_{\pm0.3339}$
EO	0.2599$_{\pm0.2481}$	1.1532$_{\pm0.9801}$	1.2024$_{\pm0.9656}$	0.4144$_{\pm0.4542}$	0.5937$_{\pm0.4804}$	0.3572$_{\pm0.3329}$
NE-Stack	0.2375$_{\pm0.2031}$	**1.0706**$_{\pm0.9554}$	1.1543$_{\pm1.0696}$	**0.3747**$_{\pm0.4494}$	**0.4923**$_{\pm0.4658}$	0.4587$_{\pm0.4479}$
NE-MA	0.2399$_{\pm0.2176}$	*1.1486*$_{\pm0.9892}$	1.1596$_{\pm0.9991}$	0.3998$_{\pm0.4474}$	0.5832$_{\pm0.4900}$	**0.3536**$_{\pm0.3311}$

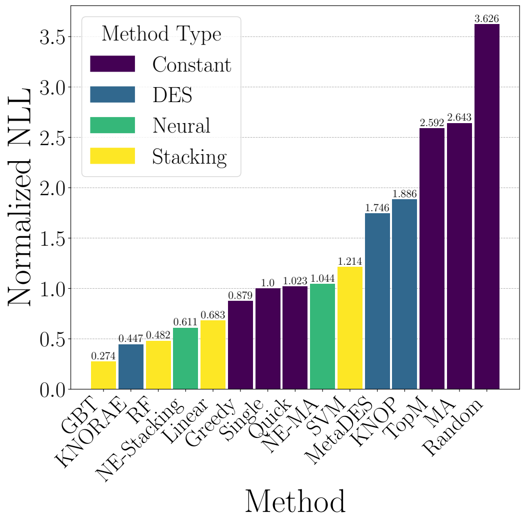


Figure 8: Average Normalized NLL in Scikit-learn pipelines (Validation).

Table 17: Average Unnormalized Error.

	FTC	**NB-Micro**	**NB-Mini**	**QT-Micro**	**QT-Mini**	**TR-Class**	**TR-Class (AUC)**
Single-Best	0.0868$_{\pm 0.0853}$	0.4320$_{\pm 0.3331}$	0.4509$_{\pm 0.3019}$	0.1285$_{\pm 0.1499}$	0.1667$_{\pm 0.1477}$	0.1604$_{\pm 0.1545}$	0.1169$_{\pm 0.1164}$
Random	0.1012$_{\pm 0.0910}$	0.3092$_{\pm 0.2444}$	0.3502$_{\pm 0.2417}$	0.5677$_{\pm 0.2487}$	0.5295$_{\pm 0.2306}$	0.1840$_{\pm 0.1632}$	0.1422$_{\pm 0.1356}$
Top5	0.0831$_{\pm 0.0825}$	0.3005$_{\pm 0.2374}$	0.3007$_{\pm 0.2251}$	$\underline{\mathit{0.1079}}_{\pm 0.1375}$	0.1502$_{\pm 0.1359}$	0.1594$_{\pm 0.1527}$	**0.1125**$_{\pm 0.1142}$
Top50	0.0881$_{\pm 0.0829}$	$\underline{\mathit{0.2778}}_{\pm 0.2266}$	0.2788$_{\pm 0.2262}$	0.1334$_{\pm 0.1495}$	0.1816$_{\pm 0.1555}$	0.1584$_{\pm 0.1524}$	$\underline{\mathit{0.1139}}_{\pm 0.1160}$
Quick	0.0831$_{\pm 0.0824}$	0.2840$_{\pm 0.2304}$	0.2842$_{\pm 0.2289}$	0.1121$_{\pm 0.1409}$	0.1504$_{\pm 0.1425}$	0.1583$_{\pm 0.1517}$	0.1145$_{\pm 0.1192}$
Greedy	0.0824$_{\pm 0.0811}$	0.3253$_{\pm 0.2827}$	0.4509$_{\pm 0.3019}$	0.1251$_{\pm 0.1487}$	0.1596$_{\pm 0.1450}$	0.1562$_{\pm 0.1523}$	0.1140$_{\pm 0.1186}$
CMAES	$\underline{\mathit{0.0822}}_{\pm 0.0810}$	0.2788$_{\pm 0.2247}$	0.2869$_{\pm 0.2358}$	0.1234$_{\pm 0.1478}$	0.1579$_{\pm 0.1456}$	0.1589$_{\pm 0.1541}$	0.1269$_{\pm 0.1311}$
Random Forest	0.0833$_{\pm 0.0829}$	0.2958$_{\pm 0.2444}$	0.3706$_{\pm 0.3698}$	0.1666$_{\pm 0.1955}$	0.2122$_{\pm 0.1874}$	0.1591$_{\pm 0.1528}$	0.1198$_{\pm 0.1210}$
Gradient Boosting	0.0838$_{\pm 0.0809}$	0.4682$_{\pm 0.2852}$	0.4920$_{\pm 0.2559}$	0.1425$_{\pm 0.1773}$	0.2092$_{\pm 0.1855}$	0.1620$_{\pm 0.1592}$	0.1240$_{\pm 0.1293}$
SVM	**0.0819**$_{\pm 0.0806}$	0.2917$_{\pm 0.2431}$	0.3356$_{\pm 0.2444}$	0.1627$_{\pm 0.2031}$	0.2341$_{\pm 0.2032}$	$\underline{\mathit{0.1553}}_{\pm 0.1550}$	0.1407$_{\pm 0.1358}$
Linear	0.0830$_{\pm 0.0816}$	0.3098$_{\pm 0.2200}$	0.3469$_{\pm 0.2097}$	0.1191$_{\pm 0.1503}$	0.1670$_{\pm 0.1547}$	0.1577$_{\pm 0.1568}$	0.1195$_{\pm 0.1203}$
MA	0.0874$_{\pm 0.0803}$	0.2795$_{\pm 0.2271}$	**0.2765**$_{\pm 0.2248}$	0.1435$_{\pm 0.1618}$	0.1971$_{\pm 0.1651}$	0.1609$_{\pm 0.1580}$	0.1186$_{\pm 0.1262}$
DivBO	0.0846$_{\pm 0.0832}$	0.3024$_{\pm 0.2489}$	0.3817$_{\pm 0.1893}$	0.1295$_{\pm 0.1431}$	0.1616$_{\pm 0.1458}$	0.1573$_{\pm 0.1540}$	0.1170$_{\pm 0.1200}$
EO	0.0851$_{\pm 0.0829}$	0.2786$_{\pm 0.2283}$	0.2813$_{\pm 0.2274}$	0.1330$_{\pm 0.1561}$	0.1659$_{\pm 0.1471}$	0.1589$_{\pm 0.1577}$	0.1177$_{\pm 0.1237}$
NE-Stack	0.0824$_{\pm 0.0817}$	0.2781$_{\pm 0.2275}$	0.2879$_{\pm 0.2339}$	**0.1033**$_{\pm 0.1348}$	**0.1424**$_{\pm 0.1314}$	0.1607$_{\pm 0.1511}$	0.1152$_{\pm 0.1159}$
NE-MA	0.0828$_{\pm 0.0818}$	**0.2772**$_{\pm 0.2264}$	$\underline{\mathit{0.2773}}_{\pm 0.2262}$	0.1093$_{\pm 0.1344}$	$\underline{\mathit{0.1490}}_{\pm 0.1383}$	**0.1551**$_{\pm 0.1515}$	0.1149$_{\pm 0.1211}$

F Analysis of Computational Cost

To better understand the trade-off between performance and computational cost, we analyze the average normalized Negative Log-Likelihood (NLL) and the runtime of various ensembling techniques. Our focus is on post-hoc ensembling methods, assuming that all base models are pre-trained. This allows us to isolate and compare the efficiency of the ensembling processes themselves, in contrast to methods like DivBO, which sequentially train models during the search, leading to higher computational demands.

As shown in Figure 9, the Neural Ensemblers (**NE-MA** and **NE-Stack**) achieve the best average performance while maintaining a competitive runtime. Notably, our method has a shorter runtime than the *Greedy* ensembling method and surpasses it in terms of performance. Additionally, the Neural Ensemblers are faster than traditional machine learning models such as *Gradient Boosting*, *Support Vector Machines (SVM)*, *Random Forests*, and optimization algorithms like *CMAES*. While simpler methods like *Top5* and *Top50* exhibit faster runtimes, they do so at the expense of reduced accuracy.

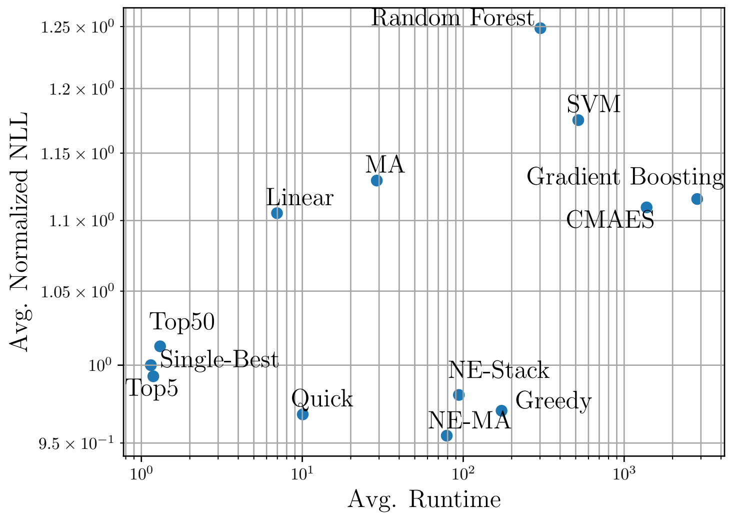


Figure 9: Trade-off between performance and computational cost for different ensembling methods. The Neural Ensemblers achieve the best performance with competitive runtime, outperforming other methods in both accuracy and efficiency.

G Proof-of-Concept with Overparameterized Base Models

A key question is whether dynamic ensemblers like our Neural Ensembler (NE) offer benefits when base models are overparameterized and potentially overfit the data. Specifically, does the advantage of the NE persist in scenarios where the base models have high capacity?

To explore this, we extend our Proof-of-Concept experiment by ensembling 10th-degree polynomials instead of 2nd-degree ones, thereby increasing the complexity of the base models and introducing the risk of overfitting. We follow the same protocol as in Section 4.1, comparing the performance of the NE with the fixed-weight ensemble method.

As illustrated in Figure 10, even with overparameterized base models that tend to overfit the training data, the NE (specifically the weighted Model-Averaging version) achieves the best performance on unseen data. This improvement occurs because the NE dynamically adjusts the ensemble weights based on the input. Thus, dynamic ensembling is advantageous not only when base models underfit but also when they overfit the data. In contrast, the fixed-weight approach lacks this adaptability and cannot compensate for the overfitting behavior of the base models.

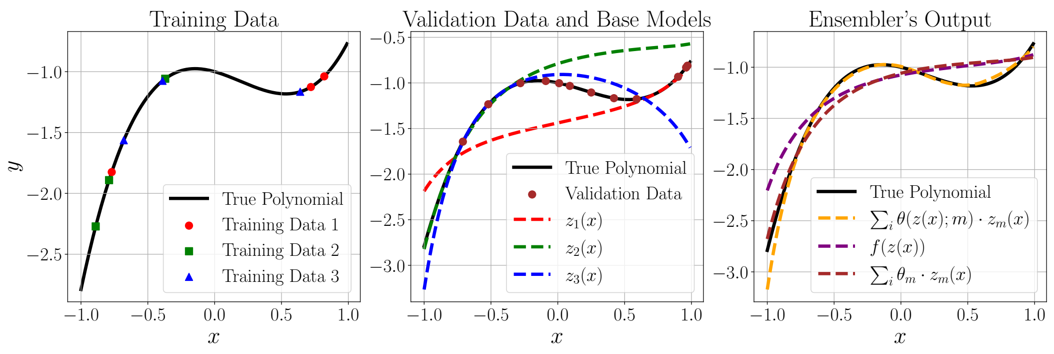


Figure 10: Proof-of-Concept experiment with overparameterized base models (10th-degree polynomials). The Neural Ensembler outperforms fixed-weight ensemble method by mitigating overfitting through dynamic, input-dependent weighting.

H Critical Difference Diagrams

We want to gain deeper insights into the difference between our proposed method and the baselines.

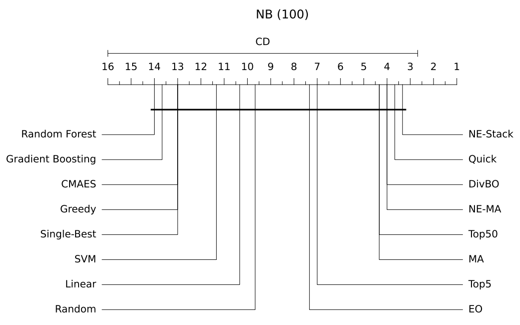
Critical Difference Diagrams To this end, we present critical difference (CD) diagrams to visualize and statistically analyze the performance of different methods across multiple datasets. The CD diagrams are generated using the `autorank` library[2], which automates the statistical comparison by employing non-parametric tests like Friedman test followed by the Nemenyi post-hoc test. These diagrams show the average ranks of the methods along the horizontal axis, where methods are positioned based on their performance across datasets. The critical difference value, which depends on the number of datasets, is represented by a horizontal bar above the ranks. Methods not connected by this bar exhibit statistically significant differences in performance.

Evaluation We computed CD diagrams across all datasets in all metadatasets (Figures 11a-11g) and an aggregated diagram across all datasets in all metadatasets (Figure 11h). We observe that although the performance difference is not substantial compared to other top-performing post-hoc ensembles like *Greedy* and *Quick*, our Neural Ensembler (**NE-MA**) consistently achieves the best performance in the aggregated results (Figure 11h). We also highlight that the **NE** versions were the top-performing approaches across all metadatasets except FTC, even though we did not modify the method's hyperparameters.

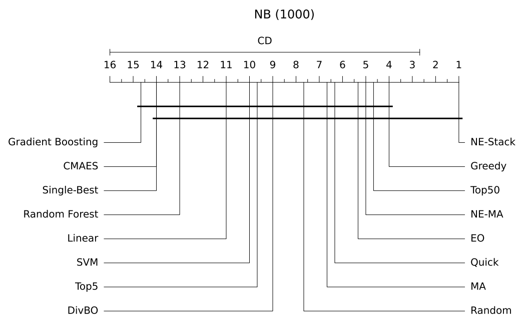
I Additional Baselines

We extended our comparison to include advanced methods like *CatBoost*, *XGBoost* and *LightGBM* which are widely used and high-performing. We also include an ensembler using Akaike weighting (Wagenmakers & Farrell, 2004) or pseudo bayesian model averaging. As shown in Table 18, *CatBoost* surpasses our Neural Ensembler (**NE**) only in the *FTC* metadataset. In all other metadatasets, the **NE** method achieves better performance. The rankings in Table 19 and the aggregated Critical Difference diagram in Figure 12 confirm that the **NE** consistently outperforms these baselines across most metadatasets.

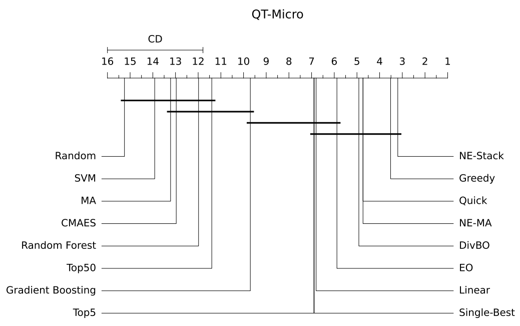
[2]`https://github.com/sherbold/autorank`



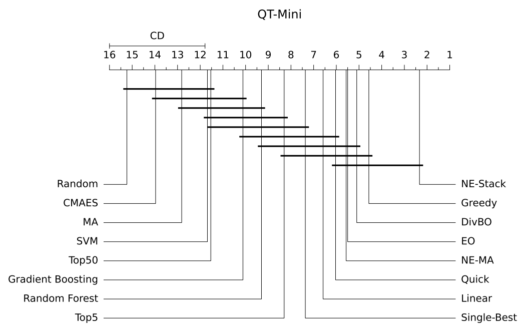
(a)



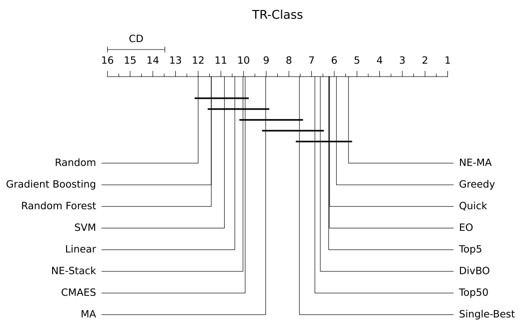
(b)



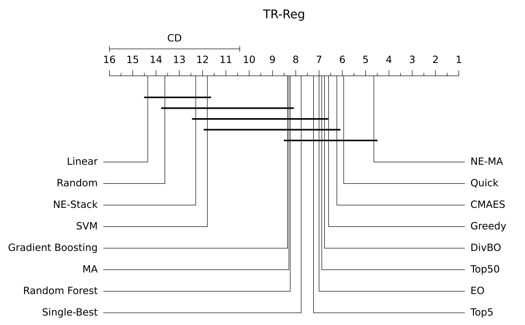
(c)



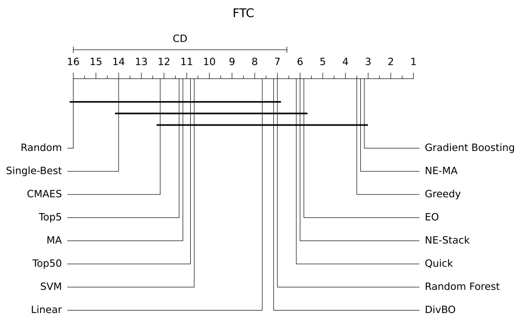
(d)



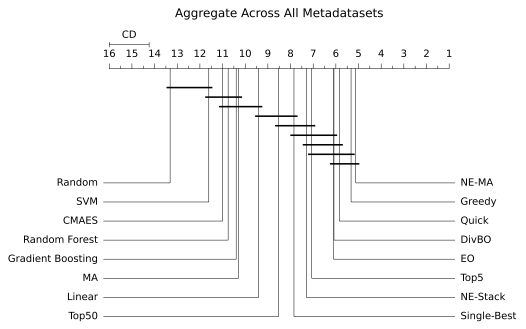
(e)



(f)



(g)



(h)

Figure 11: (a - g) Critical Difference (CD) diagrams aggregating datasets across metadatasets, (h) CD diagram aggregating the performance across all metadatasets.

Table 18: Extended Average Negative Log Likelihood (NLL) for all methods including additional baselines. A lower NLL indicates better performance.

	FTC	**NB-Micro**	**NB-Mini**	**QT-Micro**	**QT-Mini**	**TR-Class**	**TR-Reg**
Single-Best	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$
Random	1.5450$_{\pm 0.5289}$	0.6591$_{\pm 0.2480}$	0.7570$_{\pm 0.2900}$	6.8911$_{\pm 3.1781}$	5.8577$_{\pm 3.2546}$	1.7225$_{\pm 1.9645}$	1.8319$_{\pm 2.1395}$
Top5	0.8406$_{\pm 0.0723}$	0.6659$_{\pm 0.1726}$	0.6789$_{\pm 0.3049}$	1.5449$_{\pm 1.8358}$	1.1496$_{\pm 0.3684}$	1.0307$_{\pm 0.5732}$	*0.9939*$_{\pm 0.0517}$
Top50	0.8250$_{\pm 0.1139}$	0.5849$_{\pm 0.2039}$	0.6487$_{\pm 0.3152}$	3.3068$_{\pm 2.6197}$	3.0618$_{\pm 2.2960}$	1.0929$_{\pm 1.0198}$	1.0327$_{\pm 0.2032}$
Quick	0.7273$_{\pm 0.0765}$	0.5957$_{\pm 0.1940}$	0.6497$_{\pm 0.3030}$	1.1976$_{\pm 1.1032}$	0.9747$_{\pm 0.2082}$	*0.9860*$_{\pm 0.2201}$	1.0211$_{\pm 0.1405}$
Greedy	*0.6943*$_{\pm 0.0732}$	*0.5785*$_{\pm 0.1972}$	1.0000$_{\pm 0.0000}$	*0.9025*$_{\pm 0.2378}$	*0.9093*$_{\pm 0.1017}$	**0.9665**$_{\pm 0.0926}$	1.0149$_{\pm 0.1140}$
CMAES	1.2356$_{\pm 0.5295}$	1.0000$_{\pm 0.0000}$	1.0000$_{\pm 0.0000}$	4.1728$_{\pm 2.8724}$	4.6474$_{\pm 3.0180}$	1.3487$_{\pm 1.3390}$	1.0281$_{\pm 0.1977}$
Random Forest	0.7496$_{\pm 0.0940}$	0.8961$_{\pm 0.3159}$	0.9340$_{\pm 0.4262}$	3.7033$_{\pm 2.8145}$	2.2938$_{\pm 2.2068}$	1.2655$_{\pm 0.4692}$	1.0030$_{\pm 0.0871}$
Gradient Boosting	0.7159$_{\pm 0.1529}$	1.7288$_{\pm 1.2623}$	1.2575$_{\pm 0.4460}$	1.9373$_{\pm 1.2839}$	2.6193$_{\pm 2.3159}$	1.4288$_{\pm 1.2083}$	1.0498$_{\pm 0.2128}$
SVM	0.7990$_{\pm 0.0909}$	0.7744$_{\pm 0.2967}$	0.9358$_{\pm 0.5706}$	5.4377$_{\pm 3.3807}$	4.0019$_{\pm 3.6601}$	1.3884$_{\pm 1.4276}$	2.7975$_{\pm 3.0219}$
Linear	0.7555$_{\pm 0.0898}$	0.7400$_{\pm 0.2827}$	0.8071$_{\pm 0.2206}$	1.3960$_{\pm 1.2334}$	1.1031$_{\pm 0.7038}$	1.1976$_{\pm 1.1024}$	3.1488$_{\pm 3.2813}$
XGBoost	0.8292$_{\pm 0.1434}$	0.7389$_{\pm 0.2326}$	0.9092$_{\pm 0.5304}$	3.7822$_{\pm 3.1194}$	2.6119$_{\pm 2.3911}$	1.7697$_{\pm 1.4672}$	1.2580$_{\pm 0.4875}$
CatBoost	**0.6887**$_{\pm 0.0953}$	0.8092$_{\pm 0.2513}$	0.9512$_{\pm 0.5083}$	2.6262$_{\pm 2.6482}$	2.4145$_{\pm 1.8989}$	1.2570$_{\pm 1.2859}$	1.0454$_{\pm 0.1550}$
LightGBM	0.7973$_{\pm 0.1946}$	3.6004$_{\pm 2.5822}$	5.3943$_{\pm 4.7980}$	3.0378$_{\pm 2.7945}$	3.6860$_{\pm 3.2856}$	1.8298$_{\pm 1.1596}$	1.6250$_{\pm 2.1651}$
Akaike	0.8526$_{\pm 0.1403}$	0.5838$_{\pm 0.2031}$	*0.6485*$_{\pm 0.3166}$	3.1574$_{\pm 2.5898}$	2.6888$_{\pm 2.0620}$	1.0930$_{\pm 1.0203}$	1.0221$_{\pm 0.1793}$
MA	0.9067$_{\pm 0.1809}$	0.5970$_{\pm 0.2034}$	0.6530$_{\pm 0.3028}$	4.7921$_{\pm 3.0780}$	4.0168$_{\pm 2.8560}$	1.4724$_{\pm 1.9401}$	1.3342$_{\pm 1.3515}$
DivBO	0.7695$_{\pm 0.1195}$	0.7307$_{\pm 0.3061}$	0.6628$_{\pm 0.3435}$	1.2251$_{\pm 1.0293}$	0.9430$_{\pm 0.2036}$	1.0023$_{\pm 0.3411}$	1.0247$_{\pm 0.1473}$
EO	0.7535$_{\pm 0.1156}$	0.5801$_{\pm 0.2051}$	0.6911$_{\pm 0.2875}$	1.3702$_{\pm 1.6389}$	0.9649$_{\pm 0.2980}$	1.0979$_{\pm 1.0289}$	1.0183$_{\pm 0.0993}$
NE-Stack	0.7562$_{\pm 0.1836}$	**0.5278**$_{\pm 0.2127}$	**0.6336**$_{\pm 0.3456}$	**0.7486**$_{\pm 0.6831}$	**0.6769**$_{\pm 0.2612}$	1.3268$_{\pm 0.7498}$	1.2379$_{\pm 0.4083}$
NE-MA	0.6952$_{\pm 0.0730}$	0.5822$_{\pm 0.2147}$	0.6522$_{\pm 0.3131}$	1.0177$_{\pm 0.5151}$	0.9166$_{\pm 0.0936}$	1.0515$_{\pm 1.0003}$	**0.9579**$_{\pm 0.0777}$

Table 19: Extended rankings based on NLL for all methods including additional baselines. A lower rank indicates better performance.

	FTC	**NB-Micro**	**NB-Mini**	**QT-Micro**	**QT-Mini**	**TR-Class**	**TR-Reg**
Single-Best	17.6667$_{\pm 0.9832}$	16.6667$_{\pm 2.3094}$	15.0000$_{\pm 2.5981}$	7.0167$_{\pm 2.5103}$	7.5333$_{\pm 2.5221}$	8.6325$_{\pm 4.6487}$	9.0294$_{\pm 4.5705}$
Random	20.0000$_{\pm 0.0000}$	9.3333$_{\pm 5.6862}$	11.6667$_{\pm 2.0817}$	19.0333$_{\pm 1.3060}$	19.0167$_{\pm 0.9143}$	14.0301$_{\pm 5.0166}$	16.4118$_{\pm 2.9803}$
Top5	13.6667$_{\pm 3.2660}$	11.6667$_{\pm 1.5275}$	8.3333$_{\pm 1.5275}$	6.9000$_{\pm 1.9360}$	8.5667$_{\pm 2.3146}$	7.0542$_{\pm 4.1083}$	8.4118$_{\pm 5.1455}$
Top50	13.3333$_{\pm 1.7512}$	5.3333$_{\pm 1.5275}$	5.0000$_{\pm 1.0000}$	13.9667$_{\pm 2.0212}$	13.9500$_{\pm 2.0776}$	7.6747$_{\pm 3.6729}$	8.1176$_{\pm 4.4984}$
Quick	7.6667$_{\pm 1.9664}$	7.3333$_{\pm 3.7859}$	*4.3333*$_{\pm 3.5119}$	4.7333$_{\pm 2.0331}$	6.1333$_{\pm 2.7131}$	6.9036$_{\pm 3.4273}$	*7.0000*$_{\pm 4.1231}$
Greedy	4.5000$_{\pm 1.3784}$	*4.3333*$_{\pm 3.2146}$	15.0000$_{\pm 2.5981}$	*3.5167*$_{\pm 1.7786}$	*4.7000*$_{\pm 2.5278}$	*6.6506*$_{\pm 3.6138}$	7.8824$_{\pm 4.1515}$
CMAES	15.3333$_{\pm 5.5737}$	16.6667$_{\pm 2.3094}$	15.0000$_{\pm 2.5981}$	15.8667$_{\pm 4.0809}$	17.4000$_{\pm 2.1066}$	11.4578$_{\pm 4.7094}$	7.3529$_{\pm 2.9356}$
Random Forest	8.6667$_{\pm 2.7325}$	15.6667$_{\pm 3.0551}$	16.6667$_{\pm 2.0817}$	15.0000$_{\pm 1.5702}$	11.1500$_{\pm 4.9674}$	13.3614$_{\pm 6.1159}$	9.6471$_{\pm 5.5895}$
Gradient Boosting	*4.0000*$_{\pm 5.4037}$	17.3333$_{\pm 3.0551}$	15.6667$_{\pm 3.7528}$	11.9667$_{\pm 6.8857}$	12.3000$_{\pm 6.8778}$	13.2771$_{\pm 5.8400}$	9.6471$_{\pm 5.4076}$
SVM	13.3333$_{\pm 1.6330}$	11.6667$_{\pm 8.5049}$	13.0000$_{\pm 8.6603}$	17.5333$_{\pm 1.5533}$	14.4500$_{\pm 6.5828}$	12.4639$_{\pm 5.0827}$	14.5882$_{\pm 6.9826}$
Linear	9.3333$_{\pm 2.1602}$	13.0000$_{\pm 2.6458}$	12.3333$_{\pm 3.2146}$	7.0000$_{\pm 3.1073}$	6.7833$_{\pm 3.4433}$	11.8434$_{\pm 6.0897}$	17.8824$_{\pm 4.4565}$
XGBoost	13.1667$_{\pm 5.4559}$	12.6667$_{\pm 4.1633}$	14.0000$_{\pm 6.2450}$	14.1333$_{\pm 2.1413}$	12.2500$_{\pm 3.0562}$	16.9819$_{\pm 3.4698}$	16.1765$_{\pm 3.7953}$
CatBoost	**3.5000**$_{\pm 3.3317}$	15.0000$_{\pm 2.6458}$	15.6667$_{\pm 2.3094}$	11.7000$_{\pm 1.8919}$	12.9667$_{\pm 3.5548}$	11.3012$_{\pm 4.8583}$	10.4118$_{\pm 4.8484}$
LightGBM	10.6667$_{\pm 6.1860}$	17.0000$_{\pm 5.1962}$	17.0000$_{\pm 5.1962}$	13.4333$_{\pm 2.8093}$	15.1333$_{\pm 2.8945}$	17.6867$_{\pm 3.7900}$	13.2353$_{\pm 5.2978}$
Akaike	13.5000$_{\pm 3.3166}$	5.0000$_{\pm 1.0000}$	4.6667$_{\pm 0.5774}$	12.5333$_{\pm 2.0083}$	12.6833$_{\pm 1.9761}$	7.9639$_{\pm 4.0166}$	7.3824$_{\pm 4.6788}$
MA	14.1667$_{\pm 3.8687}$	8.0000$_{\pm 1.7321}$	5.0000$_{\pm 3.4641}$	16.7667$_{\pm 1.4003}$	16.1000$_{\pm 2.1270}$	10.3916$_{\pm 4.9226}$	9.9412$_{\pm 6.5141}$
DivBO	8.5000$_{\pm 4.5497}$	11.0000$_{\pm 6.2450}$	*4.3333*$_{\pm 4.0415}$	4.9500$_{\pm 2.5876}$	5.2000$_{\pm 2.7687}$	7.4337$_{\pm 3.5585}$	7.8824$_{\pm 3.6552}$
EO	7.1667$_{\pm 4.5789}$	6.0000$_{\pm 2.0000}$	9.0000$_{\pm 2.6458}$	5.8833$_{\pm 2.2194}$	5.6167$_{\pm 2.5484}$	7.0241$_{\pm 3.4144}$	8.2353$_{\pm 3.4192}$
NE-Stack	7.6667$_{\pm 5.9889}$	**1.0000**$_{\pm 0.0000}$	**3.6667**$_{\pm 4.6188}$	**3.3000**$_{\pm 3.6874}$	**2.4000**$_{\pm 2.3282}$	11.7470$_{\pm 7.1326}$	15.2353$_{\pm 4.6169}$
NE-MA	4.1667$_{\pm 3.3116}$	5.3333$_{\pm 3.5119}$	4.6667$_{\pm 2.3094}$	4.7667$_{\pm 2.1445}$	5.6667$_{\pm 2.1389}$	**6.1205**$_{\pm 3.8553}$	**5.5294**$_{\pm 2.7184}$

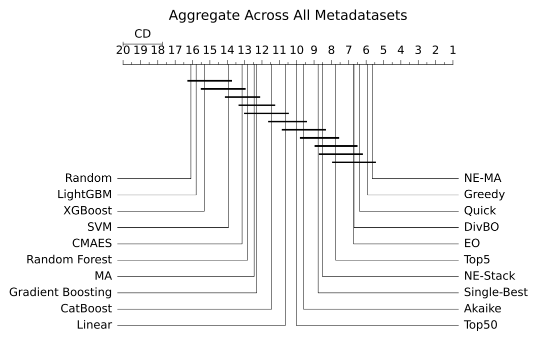
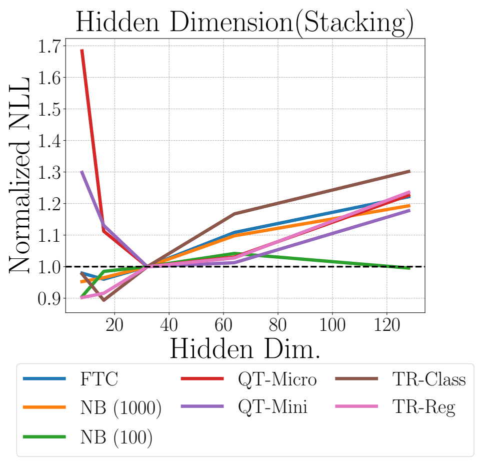
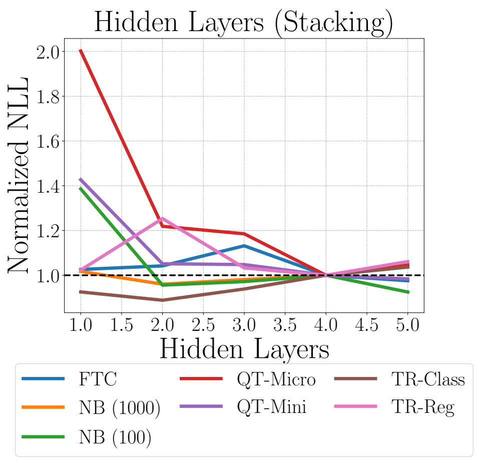


Figure 12: Critical Difference diagram including additional baselines.



(a)



(b)

Figure 13: Ablation study of Neural Ensembler hyperparameters: number of neurons per layers (a) and number of layers (b). Normalized NLL values below one indicate improved performance over the default setting ($L = 4$, $H = 32$).

J Sensitivity Analysis of Neural Ensembler Hyperparameters

While our proposed Neural Ensembler (NE) uses MLPs with 4 layers and 32 neurons per layer by default, it is important to understand how sensitive the NE's performance is to changes in its hyperparameters. To this end, we conduct an extensive ablation study by varying the number of layers $L \in \{1, 2, 3, 4, 5\}$ and the number of neurons per layer (hidden dimension) $H \in \{8, 16, 32, 64, 128\}$ in the stacking mode of the NE.

For each configuration, we compute the Negative Log Likelihood (NLL) on every task in the meta-datasets and normalize these values by the performance obtained with the default hyperparameters ($L = 4$, $H = 32$). This normalization allows us to compare performance changes across different tasks and metadatasets, accounting for differences in metric scales. A normalized NLL value below one indicates improved performance compared to the default setting.

The results of this ablation study are presented in Figures 13a and 13b, we observe that increasing the hidden dimension beyond $H = 32$ generally leads to worse performance across all metadatasets. Decreasing the hidden dimension below $H = 32$ yields slight performance improvements on several metadatasets, but also causes significant performance drops on a few others. Similarly, Figure 13b shows that increasing the number of layers beyond $L = 4$ provides minimal performance gains on a few metadatasets, while decreasing L below 4 results in improvements on only three out of seven metadatasets.

These findings indicate that there is no single hyperparameter configuration that is optimal across all datasets. However, our default configuration ($L = 4$, $H = 32$) strikes a balance, providing robust performance across diverse tasks without the need for extensive hyperparameter tuning, and can be effectively applied in various settings without the need for dataset-specific hyperparameter optimization.

K Ablation Study on Validation Data Size

To assess the Neural Ensembler's (NE) dependence on validation data size and its sample efficiency, we conducted an ablation study by varying the proportion of validation data used for training.

In this study, we evaluate the NE in stacking mode across all metadatasets, using different percentages of the available validation data: $1\%, 5\%, 10\%, 25\%, 50\%, 100\%$. For each configuration, we compute the Negative Log Likelihood (NLL) on the test set and normalize these values by the

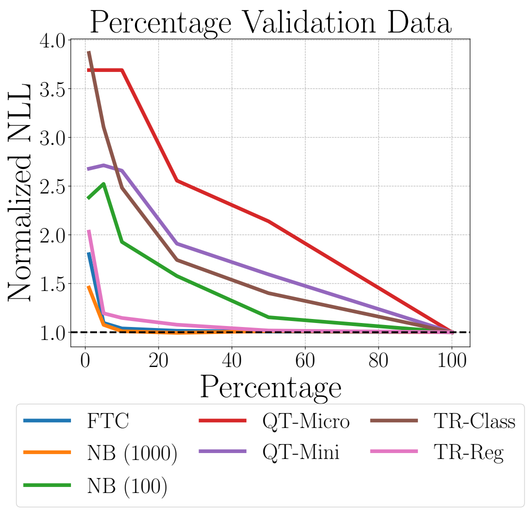


Figure 14: Ablation study on the percentage of validation data used for training the Neural Ensembler. The normalized NLL is plotted against the percentage of validation data, with values below 1 indicating performance better than or equal to using the full validation set.

performance achieved when using 100% of the validation data. This normalization allows us to compare performance changes across different tasks, accounting for differences in metric scales. A normalized NLL value below 1 indicates performance better than the baseline with full validation data, while a value above 1 indicates a performance drop.

The results are presented in Figure 14. We observe that reducing the amount of validation data used for training the NE leads to a relative degradation of the performance. However, the performance drops are relatively modest in three metadatasets. For these experiments we used the same dropout rate 0.75. We could improve the robustness in lower percentages of validation data by using a higher dropout rate. The DropOut mechanism prevents overfitting by randomly omitting base models during training, while parameter sharing reduces the number of parameters and promotes learning common representations.

L Effect of Merging Training and Validation Data

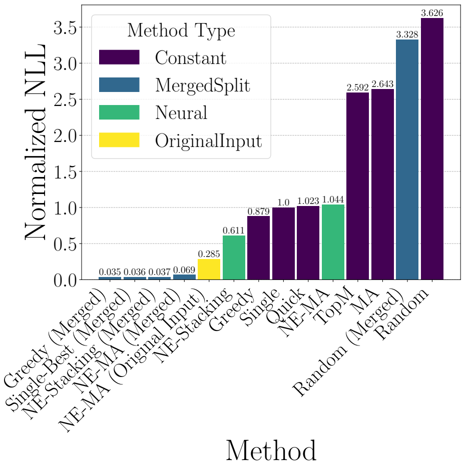
In our experimental setup (Section 4), we train the base models using the training split and the ensemblers using the validation split, then evaluate on the test split. An important question is whether merging the training and validation data could improve the performance of both the base models and the ensemblers. Specifically, we explore:

(a) Can baseline methods that do not require a validation split, such as *Random*, achieve better performance if the base models are trained on the merged dataset (training + validation)?

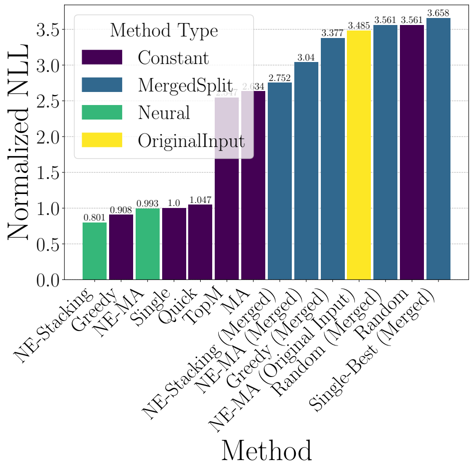
(b) Would training both the base models and the ensemblers on the merged dataset be beneficial, given that more data might enhance their learning?

To investigate these questions, we conducted experiments on two metadatasets: *Scikit-learn Pipelines* and *FTC*. We trained the base models on the merged dataset and also trained the ensemblers on this same data. Five representative baselines were compared: **NE-MA**, **NE-Stack**, *Greedy*, *Random* and *Single-best*.

Figure 15b presents the test set performance on the *Scikit-learn Pipelines* metadataset. Training on the merged dataset did not improve performance compared to training on the original splits. In fact, the results are similar to the *Random* ensembling method, indicating no significant gain. This suggests that training both the base models and ensemblers on the same (larger) dataset may lead to **overfitting**, hindering generalization to unseen data.



(a) Performance on merged dataset (training data) for *Scikit-learn Pipelines*.



(b) Test set performance on *Scikit-learn Pipelines* with merged training.

Figure 15: Impact of training on merged dataset for *Scikit-learn Pipelines*. Training on the merged dataset leads to overfitting, as evidenced by low NLL on training data but no improvement on test data.

Further evidence of overfitting is shown in Figure 15a, where we report the Negative Log Likelihood (NLL) on the merged dataset (effectively the training data). The NLL values are significantly lower for models trained on the merged dataset, confirming that they fit the training data well but do not generalize to the test set.

Similar observations were made on the *FTC* metadataset, as presented in Table 16a and Figure 16b. Although the *Random* method trained on the merged dataset shows a slight improvement, the overall performance gains are minimal.

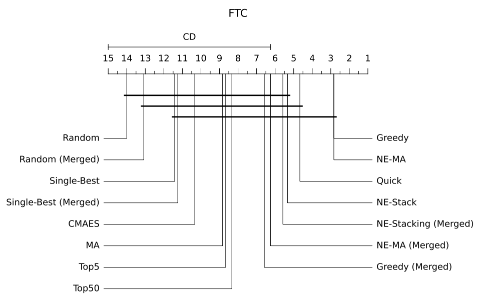
From these results, we conclude that:

(a) Training base models on the merged dataset does not significantly enhance the performance of baseline methods that do not require a validation split.

(b) Using the merged dataset to train both the base models and the ensemblers is not beneficial and may lead to overfitting, reducing generalization to the test set.

Algorithm	**FTC (Avg. Normalized NLL)**	**FTC (Avg. Rank)**
Single-Best	$1.0000_{\pm 0.0000}$	$11.4167_{\pm 1.5626}$
Single-Best (Merged)	$1.2816_{\pm 0.6710}$	$11.2500_{\pm 4.0466}$
Random	$1.5450_{\pm 0.5289}$	$14.0000_{\pm 0.8944}$
Random (Merged)	$1.5365_{\pm 0.7142}$	$13.0833_{\pm 2.2454}$
Top5	$0.8406_{\pm 0.0723}$	$8.6667_{\pm 2.9439}$
Top50	$0.8250_{\pm 0.1139}$	$8.3333_{\pm 1.9664}$
Quick	$0.7273_{\pm 0.0765}$	$4.6667_{\pm 1.7512}$
Greedy	$\mathbf{0.6943}_{\pm 0.0732}$	$\mathbf{2.8333}_{\pm 1.6021}$
Greedy (Merged)	$0.7537_{\pm 0.2652}$	$6.5833_{\pm 4.3637}$
CMAES	$1.2356_{\pm 0.5295}$	$10.3333_{\pm 3.4448}$
MA	$0.9067_{\pm 0.1809}$	$8.8333_{\pm 2.4014}$
NE-Stack	$0.7562_{\pm 0.1836}$	$5.3333_{\pm 4.6762}$
NE-Stacking (Merged)	$0.7428_{\pm 0.2208}$	$5.5833_{\pm 3.3529}$
NE-MA	$\underline{\mathit{0.6952}}_{\pm 0.0730}$	$\mathbf{2.8333}_{\pm 2.2286}$
NE-MA (Merged)	$0.7461_{\pm 0.2084}$	$6.2500_{\pm 2.6410}$

(a)



(b)

Figure 16: Merged-split baselines on the *FTC* metadataset: (a) Test set performance with merged training, (b) Critical Difference diagram.

M NEURAL ENSEMBLERS OPERATING ON THE ORIGINAL INPUT SPACE

In Section 3.1, we discussed that the Neural Ensembler in model-evaraging mode (**NE-MA**) computes weights $\theta_m(z;\beta)$ that rely solely on the base model predictions z for each instance. Specifically, the weights are defined as:

$$\theta_m(z;\beta) = \frac{\exp f_m(z;\beta)}{\sum_{m'} f_{m'}(z;\beta)} \tag{53}$$

where z represents the base model predictions for an instance $x \in \mathcal{X}$, and $\mathcal{X}$ denotes the original input space. As our experiments encompass different data modalities, x can be a vector of tabular descriptors, an image, or text.

An alternative formulation involves computing the ensemble weights directly from the original input instances instead of using the base model predictions. This approach modifies the weight computation to:

$$\hat{y} = \sum_m \theta_m(x;\beta) \cdot z(x;m) = \sum_m \frac{\exp f_m(x;\beta)}{\sum_{m'} \exp f_{m'}(x;\beta)} \cdot z(x;m) \tag{54}$$

where f_m now operates on the original input space $\mathcal{X}$ to produce unnormalized weights.

However, adopting this formulation introduces challenges in selecting an appropriate function f_m for different data modalities. For example, if the instances are images, f_m must be a network capable of processing images, such as a convolutional neural network. This requirement prevents us from using the same architecture across all modalities, limiting the generalizability of the approach.

To evaluate this idea, we tested this alternative neural ensembler on the *Scikit-learn Pipelines* meta-dataset, which consists of tabular data. We implemented f_m as a four-layer MLP. Our results, represented by the yellow bar in Figure 15b, indicate that this strategy does not outperform the original approach proposed in Section 3, which uses the base model predictions as input.

We hypothesize that computing ensemble weights directly from the original input space may be more susceptible to overfitting, especially when dealing with datasets that have noisy or high-dimensional features. Additionally, this strategy may require tuning the hyperparameters of the network f_m for each dataset to achieve optimal performance, reducing its effectiveness and generalizability across diverse datasets.

N SIGNIFICANCE ON CHALLENGING DATASETS

Given the high performance between *Greedy* and **NE-MA**, we wanted to understand when the second one would obtain strong significant results. We found that that **NE-MA** is particularly well performing in challenging datasets with a large number of classes. Given table 7, we can see that four meta-datasets have a high (> 10) number of classes, thus they have datasets with a lot of classes. We selected these metadatasets (*NB(100), NB(1000), QT-Micro, QT-Mini*), and plotted the significance compared to *Greedy* and *Random Search*. The results reported in Figure 17 demonstrate that our approach is significantly better than Greedy in these metadatasets.

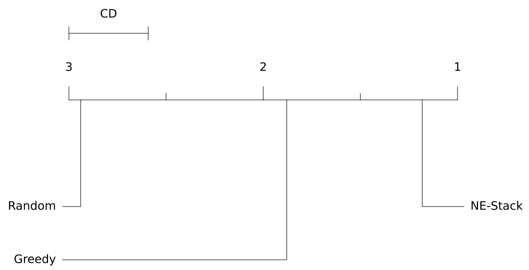


Figure 17: CD Diagram in datasets with a lot of classes.