Assembled-OpenML: Creating Efficient Benchmarks for Ensembles in AutoML with OpenML

Lennart Purucker[1] **Joeran Beel**[1]

[1]University of Siegen

Abstract Automated Machine Learning (AutoML) frameworks regularly use ensembles. Developers need to compare different ensemble techniques to select appropriate techniques for an AutoML framework from the many potential techniques. So far, the comparison of ensemble techniques is often computationally expensive, because many base models must be trained and evaluated one or multiple times. Therefore, we present Assembled-OpenML. Assembled-OpenML is a Python tool, which builds meta-datasets for ensembles using OpenML. A meta-dataset, called Metatask, consists of the data of an OpenML task, the task's dataset, and prediction data from model evaluations for the task. We can make the comparison of ensemble techniques computationally cheaper by using the predictions stored in a metatask instead of training and evaluating base models. To introduce Assembled-OpenML, we describe the first version of our tool. Moreover, we present an example of using Assembled-OpenML to compare a set of ensemble techniques. For this example comparison, we built a benchmark using Assembled-OpenML and implemented ensemble techniques expecting predictions instead of base models as input. In our example comparison, we gathered the prediction data of 1523 base models for 31 datasets. Obtaining the prediction data for all base models using Assembled-OpenML took ~1 hour in total. In comparison, obtaining the prediction data by training and evaluating just one base model on the most computationally expensive dataset took ~37 minutes.

1 Introduction

Combining the predictions of several models can produce better overall predictions (Dietterich, 2000; Kittler et al., 1998). Building an ensemble from a set of base models is a common practice in Machine Learning (ML) and Automated Machine Learning (AutoML). Real-world ML applications regularly use ensembles, see (Hakak et al., 2021; Gunturi and Sarkar, 2021; Chung et al., 2019). AutoML frameworks often have ensembles as just another algorithm in the search space during model selection (e.g., Random Forest, XGBoost, etc.). Alternatively, frameworks treat ensembles as a special part of the search space and focus on building one large ensemble. For example, AutoGluon (Erickson et al., 2020), Autostacker (Chen et al., 2018), and Automatic Frankensteining (Wistuba et al., 2017) use stacked generalization (Wolpert, 1992) and explicitly focus on building an ensemble. Lastly, ensembles can be built post hoc by using (a subset of) all models found during model selection. Here, AutoGluon and Auto-Sklearn (Feurer et al., 2015) use ensemble selection from libraries of models (Caruana et al., 2004). Post hoc ensembling can be formulated as its own optimization problem, and initial work in this direction has already been done (Zhao, 2022).

Many different ensemble techniques could be used in AutoML. Techniques from related fields such as dynamic ensemble or classifier selection could also be used (Ko et al., 2008; Giacinto and Roli, 2001). Hence, ensemble techniques must be compared to find the techniques that are best as part of the search space, as a special part of the search space, or for post hoc ensembling.

However, comparing ensemble techniques is computationally expensive, and there are no dedicated benchmarks for ensemble techniques to speed up the comparison. Usually, new and existing techniques are compared by evaluating them on ML datasets. Moreover, the ensembles are

© 2022 the authors, released under CC BY 4.0

built from a potentially large set of pre-selected base models. Such comparisons are unnecessarily computationally expensive, because base models are trained and evaluated one or multiple times for every comparison. To illustrate, base models are commonly trained and evaluated for every dataset once or each ensemble technique individually trains and evaluates the base models when needed; see (Cruz et al., 2018; Zhao et al., 2008; Allende-Cid et al., 2015; van Rijn et al., 2018; Álvarez et al., 2015).

We considered two solutions to enable less computationally expensive comparisons by avoiding the computational overhead of base models. First, compute and share trained base models for a benchmark set of datasets. By building ensembles from pre-trained base models, we can avoid the cost of training base models for every comparison. Second, compute and share the predictions of trained base models. Thus, allowing us to avoid training and evaluating base models by building ensembles from the prediction data instead of models. This is possible since we only require the base models' predictions to build and evaluate most ensemble techniques.

In this paper, we introduce a tool for the second solution, dubbed *Assembled-OpenML*. As the name suggested, we build upon the OpenML platform (Vanschoren et al., 2014) and its ecosystem of tools[1], which enable ML users to share and reuse machine learning (meta-)data. In detail, we present a Python tool that can automatically build a set of meta-datasets, called *Metatasks*, using data from OpenML. Assembled-OpenML selects a set of base models for an ML task from OpenML to create metatasks. A metatask contains data on the original task, the associated dataset, additional metadata, and each selected base model's predictions as well as confidences. All required (meta-)data for a metatask are fetched from OpenML.

Metatasks can be used to compare ensemble techniques while being less computationally expensive. The data stored in a metatasks allows us to simulate ensemble techniques, that is, execute ensemble techniques without having to train and evaluate base models. Thus, leaving only the computational overhead of the ensemble techniques. To illustrate, we were able to build 31 metatasks containing the prediction data of 1523 base models in ~1 hour using Assembled-OpenML. Training and evaluating just one base model to obtain its prediction data on the most computationally expensive dataset took ~37 minutes.

In this paper, we present the first version of Assembled-OpenML. Our contribution with Assembled-OpenML is the automation of creating metatasks and thus enabling efficient benchmarks for ensembles. Moreover, we show its use by simulating and comparing 5 ensemble techniques and 3 baselines on an example benchmark set of 31 metatasks. The framework Assembled-OpenML, example usage code, and data related to this paper can be found in our GitHub repository[2].

2 Related Work

In related research fields, like hyperparameter optimization and neural architecture search, the computational cost of comparing optimization techniques motivated the creation of surrogate and tabular benchmarks. Surrogate benchmarks provide a surrogate model that is used to predict the performance of a configuration such that the expensive evaluation of the configuration can be avoided (Eggensperger et al., 2015). Tabular benchmarks provide a look-up table to obtain the performance of configurations (Ying et al., 2019; Klein and Hutter, 2019).

Both types of benchmarks do not exist for ensemble techniques. Moreover, to the best of our knowledge, no previous work has tried to reduce the computational cost of comparing ensembles. We do not know of any work that systematically created, stored, and shared the predictions of base models. Likewise, we do not know of any appropriate repository of trained base models for ensembles. The closest to this would be the model zoo[3] from Caffe (Jia et al., 2014) or the

[1] See `https://github.com/openml` for OpenML's library of tools.

[2] `https://github.com/ISG-Siegen/assembled`

[3] `https://github.com/BVLC/caffe/wiki/Model-Zoo`

model garden[4] from TensorFlow (Yu et al., 2020) for transfer learning with pre-training. Both store models of deep neural networks for computer vision, natural language processing, or recommender systems. However, they are not appropriate because they do not store trained (traditional) models for tabular classification or regression tasks.

OpenML was frequently used to produce meta-datasets. So far, OpenML has been used mainly to produce meta-datasets consisting of meta-features in terms of complexity measures (like the number of instances of a dataset) and the performance of algorithms for a specific metric (Bilalli et al., 2017; Tornede et al., 2020; Olier et al., 2018; Kühn et al., 2018). Such undertakings also produced a set of basic tools to extract meta-data from OpenML[5]. Yet, none of these undertakings extracted predictions from OpenML.

Lastly, we are not aware of any exhaustive comparison or benchmark of ensemble techniques. The closest we have found to this is a GitHub repository with a Per Instance Algorithm Selection Benchmark for Multi-Classifier Systems and AutoML by Edward Bergman[6]. This benchmark could be reused or extended to compute and share trained models instead of predictions using OpenML.

3 Assembled-OpenML: a Meta-Dataset Collection Framework

Assembled-OpenML expects a task ID from OpenML as input to build a metatask. For reference, ~4100 classification and ~19000 regression tasks exist on OpenML at the time of writing. First, Assembled-OpenML fetches the original OpenML task. The OpenML task is used to collect the dataset and all information related to the dataset (e.g., a predefined validation split). Next, Assembled-OpenML fetches the set of top-n best performing configurations of the task according to a user-selected metric. Thereby, Assembled-OpenML ensures that the top-n set does not contain duplicated configurations. Lastly, Assembled-OpenML parses and collects the prediction data of each configuration in the top-n set. The prediction data includes the predictions and their confidences. To clarify a relevant edge case, we store the concatenated prediction data of each fold if an OpenML task used cross-validation. We use the Python extension for OpenML by Feurer et al. (2019) to fetch data from OpenML[7].

In this initial version, Assembled-OpenML still faces some limitations. Assembled-OpenML only supports classification tasks so far. The number of OpenML runs for a task represents the available amount of prediction data. Unfortunately, the number of runs available on OpenML for regression tasks is very small compared to the number of runs for classification tasks. To illustrate, the top 10 classification tasks have between ~417000 and ~159000 runs at the moment, while the top 10 regression tasks have between ~160 and 18 runs. Therefore, we focused on classification for the initial version. Similarly, we ignore evaluation repetitions because the vast majority of tasks do not include repetitions. Lastly, we encountered problems with corrupted prediction data[8].

Assembled-OpenML does not put any constraints on the top-n set besides ignoring duplicates. Additional constraints might not be appropriate, because we focus on the use case of AutoML. AutoML frameworks, like Auto-Sklearn, do not employ any additional constraints. Nevertheless, we want to support additional constraints in the future. For example, some ensemble techniques work best with a diverse set of base models (Banfield et al., 2005). Therefore, Assembled-OpenML with additional constraints could make interesting experimenters much cheaper computationally. For example, we could validate if it is a good idea to only store a diverse set of configurations during the run of an AutoML tool without having to expensively run AutoML tools[9].

[4] https://github.com/tensorflow/models

[5] For example: https://github.com/joaquinvanschoren/openml-metadata/ or https://github.com/bbilalli/MetadataFromOpenML

[6] https://github.com/eddiebergman/piasbenchmark

[7] See Appendix G for the references, versions, and licences of all required Python libraries.

[8] See Appendix D for more details on corrupted prediction data.

[9] We explored this in preliminary experiments, see Appendix E for more details.

4 Using Assembled-OpenML to Compare Ensemble Techniques

To provide an example of how to use Assembled-OpenML to compare ensemble techniques, we created a simple benchmark set of metatasks (Section 4.1). Furthermore, we implemented a set of ensemble techniques such that we can simulate their behavior by passing prediction data to the technique (Section 4.2).

4.1 Creating a Benchmark using Assembled-OpenML

For this example, we decided to use a list of OpenML task IDs from a curated benchmarking suite (Bischl et al., 2021) as input to Assembled-OpenML. We selected the benchmarking suite "OpenML-CC18"[10], which includes 72 tasks. The tasks in OpenML-CC18 adhere to a list of criteria appropriate for a benchmark, such as having to use 10-fold cross-validation. Thus, the first step is to run Assembled-OpenML for each task ID in OpenML-CC18. Here, we decided to use OpenML's Area Under ROC Curve (AUROC) metric to select the 50 best performing configurations for each task (if more than 50 exist). This took ~55 minutes without parallelization. The time it takes to build a metatask depends on the hardware, internet quality, and OpenML's response time[11].

The resulting 72 metatasks could already be used as a benchmark. However, we also want to detail the possibility of post-processing a set of metatasks. To do so, we created a script that filters metatasks and base models based on the following constraints.

To quantify the potential of ensemble techniques for a dataset, we used concepts from Dynamic/Algorithm Selection (Cruz et al., 2018; Kerschke et al., 2019): the Virtual Best Algorithm (VBA) and the Single Best Algorithm (SBA). The VBA represents an oracle-like perfect selector that always selects the prediction of the best base model for an instance. The SBA represents the average best selector and returns the predictions of the base model that is assumed to be the best on average over all instances (e.g., has the highest score on the validation data). We use the difference in performance between VBA and SBA (called the *VBA-SBA-Gap*) to filter metatasks. In other words, we assume for this benchmark that if a VBA-SBA-Gap exists, the task is more interesting for ensemble techniques. Having no gap is interpreted as one base model being as good as (naively) combining multiple base models. We required that metatasks have a VBA-SBA-Gap of 5% in performance to guarantee that there is a (theoretical) room for improvement over the SBA. Moreover, we removed worse-than-random base models and filtered base models with corrupted prediction data. Finally, we require that a valid metatask has at least 10 base models.

This post-processing took ~7 minutes without parallelization. 31 metatasks remained after post-processing. For details on these metatasks, refer to Table 1 in Appendix A. We provide code to re-build the 31 metatasks from an automatically created benchmark specification.

4.2 Simulating Ensemble Techniques

We execute an ensemble technique without having to train and evaluate the base models. Thus, leaving only the computational overhead of the ensemble technique. We deem this to be a simulation as any practical application of an ensemble technique would need to train and evaluate the base models. For example, scikit-learn's StackingClassifier[12] expects (untrained) base models as input.

To simulate an ensemble technique, we use the data stored in a metatask. We also use the original task's folds from OpenML, because the prediction data was created using cross-validation. For each fold, we split the fold's predictions on the test data in two halves with a ratio of 0.5, creating meta-train and meta-test predictions. The split is done in a stratified fashion. Next, the meta-train predictions are used to train or build the ensemble technique. Lastly, the meta-test

[10] See `https://www.openml.org/s/99` and `https://docs.openml.org/benchmark/`.

[11] All experiments in this paper are done on a workstation with an AMD Ryzen Threadripper PRO 3975WX CPU, SSD storage, 528 GB RAM, and a download speed of 1 Gbps. Moreover, we noticed no issues with OpenML's response time.

[12] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html`

predictions are used to evaluate the ensemble technique. As a result, we use a metatask to perform 10-fold cross-validation of the simulated ensemble techniques. Some ensemble techniques utilize the training data and not only the base models' predictions. We employed one-hot encoding and filled missing values with a static value to make the training data usable by such techniques.

We simulated the following techniques: Stacking (Wolpert, 1992); majority Voting; ensemble selection from libraries of models (Caruana et al., 2004); Dynamic Classifier Selection (DCS) (Giacinto and Roli, 2001); Dynamic Ensemble Selection (DES) (Ko et al., 2008); a SBA as well as a VBA versions of DCS (called DCS-SBA and DCS-VBA); and a novel oracle-like baseline called the Virtual Best Ensemble (VBE). See Appendix C for more details on the individual simulated ensemble techniques.

4.3 Results and Summary

Obtaining the prediction data used in our comparison took 1 hour and 2 minutes, which is the combined time of sequentially fetching and building the 72 original metatasks as well as reducing it to a benchmark set of 31 metatasks. The total prediction data of the 31 metatasks is equal to training and evaluating 1523 base models. In comparison, obtaining the predictions by training and evaluating one base model, a Histogram-based Gradient Boosting Classification Tree[13], on the most computationally expensive task/dataset, the dataset CIFAR_10 with OpenML task ID 167124, took ~37 minutes. The training of the model used parallelization on all 64 cores by default. This excludes the time it took to find the model and to build an environment for its execution.

We have no space for a detailed analysis of the example benchmark's results in this paper. Refer to Appendix B for more details on the results of running the simulated ensemble techniques on the benchmark created using Assembled-OpenML. Simulating all ensemble techniques for all datasets across all 10 folds took ~4 hours without parallelization.

5 Limitations and Broader Impact Statement

As the limitations mentioned in Section 3 hint at, the biggest limitation of Assembled-OpenML is its data source. OpenML does not have enough data on, for example, regression tasks. Moreover, data stored on OpenML is sometimes problematic. We were not able to reproduce/initialize some base models[14]. Additionally, we found unexplainable problems with the prediction data[15]. Still, we believe that OpenML is the best publicly available data source.

Assembled-OpenML enables less computationally expensive comparisons. Comparisons based on the data created by our tool could lead to re-evaluating the ensemble techniques used in existing (AutoML) applications. Additionally, our work could lead to initiatives that try to produce less computational expensive benchmarks for ensembles. Thus, reducing costs and environmental impact. A negative impact of Assembled-OpenML could be an increase in traffic and cost of OpenML. We tried to minimize the API calls made by Assembled-OpenML. Generally, an initiative to share prediction data would also be helpful to make comparisons less computationally expensive.

6 Conclusion

We presented the first version of Assembled-OpenML, a tool to generate metatasks using OpenML. Metatasks are meta-datasets that make it less computationally expensive to evaluate ensemble techniques. Additionally, we detailed an example of using Assembled-OpenML to compare ensemble techniques by building a benchmark set of metatasks and simulating ensemble techniques.

[13] Originally, we wanted to use the model with the highest AUROC for the runtime comparison. However, we were not able to initialize this model using OpenML's tools. We assume that this is a bug resulting from OpenML's development. See our code for more details. Consequently, we opted for the model with the next highest AUROC that we can initialize.

[14] Here, we are referring to the model with the highest AUROC on the dataset CIFAR_10 with OpenML task ID 167124.

[15] See Appendix D.

References

Allende-Cid, H., Allende, H., Monge, R., and Moraga, C. (2015). Discrete neighborhood representations and modified stacked generalization methods for distributed regression. *J. Univers. Comput. Sci.*, 21(6):842–855.

Álvarez, A., Sierra, B., Arruti, A., López-Gil, J.-M., and Garay-Vitoria, N. (2015). Classifier subset selection for the stacked generalization method applied to emotion recognition in speech. *Sensors*, 16(1):21.

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

Bilalli, B., Abelló Gamazo, A., and Aluja Banet, T. (2017). On the predictive power of meta-features in openml. *International Journal of Applied Mathematics and Computer Science*, 27(4):697–712.

Bischl, B., Casalicchio, G., Feurer, M., Gijsbers, P., Hutter, F., Lang, M., Mantovani, R. G., van Rijn, J. N., and Vanschoren, J. (2021). Openml benchmarking suites. In Vanschoren, J. and Yeung, S., editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual.*

Bridge, J. P., Holden, S. B., and Paulson, L. C. (2014). Machine learning for first-order theorem proving. *Journal of automated reasoning*, 53(2):141–172.

Bulloch, B. et al. (1991). Eucalyptus species selection for soil conservation in seasonally dry hill country-twelfth year assessment. *New Zealand journal of forestry science*, 21(1):10–31.

Caruana, R., Niculescu-Mizil, A., Crew, G., and Ksikes, A. (2004). Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18.

Chen, B., Wu, H., Mo, W., Chattopadhyay, I., and Lipson, H. (2018). Autostacker: a compositional evolutionary learning system. In Aguirre, H. E. and Takadama, K., editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*, pages 402–409. ACM.

Chung, Y.-W., Khaki, B., Li, T., Chu, C., and Gadh, R. (2019). Ensemble machine learning-based algorithm for electric vehicle user behavior prediction. *Applied Energy*, 254:113732.

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

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Eggensperger, K., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2015). Efficient benchmarking of hyperparameter optimizers via surrogates. In Bonet, B. and Koenig, S., editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 1114–1120. AAAI Press.

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

Evans, B. and Fisher, D. (1994). Overcoming process delays with decision tree induction. *IEEE expert*, 9(1):60–66.

Feurer, M., Klein, A., Eggensperger, Katharina Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28 (2015)*, pages 2962–2970.

Feurer, M., van Rijn, J. N., Kadra, A., Gijsbers, P., Mallik, N., Ravi, S., Müller, A., Vanschoren, J., and Hutter, F. (2019). Openml-python: an extensible python api for openml. *arXiv:1911.02490*.

Giacinto, G. and Roli, F. (2001). Dynamic classifier selection based on multiple classifier behaviour. *Pattern Recognition*, 34(9):1879–1882.

Gunturi, S. K. and Sarkar, D. (2021). Ensemble machine learning models for the detection of energy theft. *Electric Power Systems Research*, 192:106904.

Guyon, I., Gunn, S., Ben-Hur, A., and Dror, G. (2004). Result analysis of the nips 2003 feature selection challenge. *Advances in neural information processing systems*, 17.

Hakak, S., Alazab, M., Khan, S., Gadekallu, T. R., Maddikunta, P. K. R., and Khan, W. Z. (2021). An ensemble machine learning approach through effective feature extraction to classify fake news. *Future Generation Computer Systems*, 117:47–58.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Hutter, F., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods & evaluation. *Artif. Intell.*, 206:79–111.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.

Kerschke, P., Hoos, H. H., Neumann, F., and Trautmann, H. (2019). Automated algorithm selection: Survey and perspectives. *Evol. Comput.*, 27(1):3–45.

Kittler, J., Hatef, M., Duin, R. P., and Matas, J. (1998). On combining classifiers. *IEEE transactions on pattern analysis and machine intelligence*, 20(3):226–239.

Klein, A. and Hutter, F. (2019). Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*.

Ko, A. H., Sabourin, R., and Britto Jr, A. S. (2008). From dynamic classifier selection to dynamic ensemble selection. *Pattern recognition*, 41(5):1718–1731.

Kohavi, R. et al. (1996). Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Kdd*, volume 96, pages 202–207.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Kühn, D., Probst, P., Thomas, J., and Bischl, B. (2018). Automatic exploration of machine learning experiments on openml. *arXiv preprint arXiv:1806.10961*.

Lindauer, M., van Rijn, J. N., and Kotthoff, L. (2017). Open algorithm selection challenge 2017: Setup and scenarios. In *Proceedings of the Open Algorithm Selection Challenge 2017, Brussels, Belgium, September 11-12, 2017*, volume 79 of *Proceedings of Machine Learning Research*, pages 1–7. PMLR.

Lucas, D., Klein, R., Tannahill, J., Ivanova, D., Brandon, S., Domyancic, D., and Zhang, Y. (2013). Failure analysis of parameter-induced simulation crashes in climate models. *Geoscientific Model Development*, 6(4):1157–1171.

Madeo, R. C., Lima, C. A., and Peres, S. M. (2013). Gesture unit segmentation using support vector machines: segmenting gestures from rest positions. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 46–52.

Mansouri, K., Ringsted, T., Ballabio, D., Todeschini, R., and Consonni, V. (2013). Quantitative structure–activity relationship models for ready biodegradability of chemicals. *Journal of chemical information and modeling*, 53(4):867–878.

Moro, S., Cortez, P., and Rita, P. (2014). A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31.

Olier, I., Orhobor, O. I., Vanschoren, J., and King, R. D. (2018). Transformative machine learning. *arXiv preprint arXiv:1811.03392*.

pandas development team, T. (2020). pandas-dev/pandas: Pandas.

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

provided by Semeion, D. (2010). Research center of sciences of communication, via sersale 117, 00128, rome, italy.

Rice, J. R. (1976). The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier.

Sayyad Shirabad, J. and Menzies, T. (2005). The PROMISE repository of software engineering databases. School of Information Technology and Engineering, University of Ottawa, Canada.

Siebert, J. P. (1987). Vehicle recognition using rule based methods.

Simonoff, J. S. (2003). *Analyzing categorical data*, volume 496. Springer.

Tornede, A., Wever, M., and Hüllermeier, E. (2020). Extreme algorithm selection with dyadic feature representation. In *International Conference on Discovery Science*, pages 309–324. Springer.

van Rijn, J. N., Holmes, G., Pfahringer, B., and Vanschoren, J. (2018). The online performance estimation framework: heterogeneous ensemble learning for data streams. *Machine Learning*, 107(1):149–176.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2014). Openml: networked science in machine learning. *CoRR*, abs/1407.7722.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

Wistuba, M., Schilling, N., and Schmidt-Thieme, L. (2017). Automatic frankensteining: Creating complex ensembles autonomously. In Chawla, N. V. and Wang, W., editors, *Proceedings of the 2017 SIAM International Conference on Data Mining, Houston, Texas, USA, April 27-29, 2017*, pages 741–749. SIAM.

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

Yeh, I.-C., Yang, K.-J., and Ting, T.-M. (2009). Knowledge discovery on rfm model using bernoulli sequence. *Expert Systems with Applications*, 36(3):5866–5871.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. (2019). Nas-bench-101: Towards reproducible neural architecture search. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114. PMLR.

Yu, H., Chen, C., Du, X., Li, Y., Rashwan, A., Hou, L., Jin, P., Yang, F., Liu, F., Kim, J., et al. (2020). Tensorflow model garden. *GitHub*.

Zhang, K. and Fan, W. (2008). Forecasting skewed biased stochastic ozone days: analyses, solutions and beyond. *Knowledge and Information Systems*, 14(3):299–326.

Zhao, G., Shen, Z., Miao, C., and Gay, R. (2008). Enhanced extreme learning machine with stacked generalization. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1191–1198. IEEE.

Zhao, Y. (2022). Autodes: Automl pipeline generation of classification with dynamic ensemble strategy selection. *arXiv preprint arXiv:2201.00207*.

7 Reproducibility Checklist

1. For all authors…

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The abstract and introduction both claim that we present Assembled-OpenML, an example benchmark, and an example of simulating ensemble techniques. We do exactly that in Section 3, 4.1, and 4.2.

   (b) Did you describe the limitations of your work? [Yes] See Section 5.

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 5.

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

2. If you are including theoretical results…

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

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

3. If you ran experiments…

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] Code to re-create the data we have used exist in our GitHub repository[16]. Moreover, we include all code used to generate results for this paper. We documented our work with a README and the dependencies with a requirements file.

   (b) Did you include the raw results of running the given instructions on the given code and data? [Yes] The predictions of the simulated ensemble techniques for each metatask are stored in our GitHub repository.

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

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

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] See Section 4.2 and Appendix C. The details for the folds are omitted as they depend on OpenML and we did not create them.

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes] All simulated ensemble techniques use the same code/data to be executed.

[16] `https://github.com/ISG-Siegen/assembled`

(g) Did you run ablation studies to assess the impact of different components of your approach? [N/A] We did not represent a new approach for which ablation studies are appropriate.

(h) Did you use the same evaluation protocol for the methods being compared? [Yes] See Section 4.2.

(i) Did you compare performance over time? [N/A] We do not have a time component in our experiments.

(j) Did you perform multiple runs of your experiments and report random seeds? [Yes] As a result of our data, we used 10-fold cross validation to evaluate the simulated techniques variability. However, we do not have the random seeds used to generate the splits. As far as we know, OpenML does not store the seeds used to create the splits.

We have used a random state for our example benchmark to make our result reproducible across multiple executions. See our code for more details. However, we did not perform repetitions with different random states.

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Our visualization of the results contains the performance on all folds. But the used visualization does not include the correct mapping from fold number to performance. This can be found in the raw results if needed.

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [N/A] We do not have such benchmarks available. This is part of the motivation for our work.

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4.1, Section 4.2 and 4.3.

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [N/A] We did not tune hyperparameters. The simulated techniques are not sophisticated enough to include HPO yet. This is future work.

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

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

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

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Our tool Assembled-OpenML can be understood as a new asset. It is include per URL to our GitHub repository.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] See Section 4.1, we are using OpenML-CC18 and its data. We cited all data sources according to the guidelines of datasets on OpenML (and in OpenML-CC18).

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

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

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

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

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

A Details on Metatasks of the Example Benchmark

Table 1: **Metatasks of the Example Benchmark** The table contains details on metatasks that were obtained by building the example benchmark. Each OpenML task is associated to a dataset. The 31 tasks and datasets presented here were selected by post-processing/filtering the 72 metatasks obtained by running Assembled-OpenML for all tasks of the curated benchmark suite "OpenML-CC18".

Dataset	Task ID	Instances	Features	Classes	Base Models
numerai28.6[17]	167120	96 320	21	2	50
connect-4 (Dua and Graff, 2017)	146195	67 557	42	3	50
CIFAR_10 (Krizhevsky et al., 2009)	167124	60 000	3072	10	50
adult (Kohavi et al., 1996; Dua and Graff, 2017)	7592	48 842	14	2	50
bank-marketing (Moro et al., 2014)	14965	45 211	16	2	50
jm1 (Sayyad Shirabad and Menzies, 2005)	3904	10 885	21	2	50
GesturePhaseSegmentationProcessed (Madeo et al., 2013; Dua and Graff, 2017)	14969	9873	32	5	50
first-order-theorem-proving (Bridge et al., 2014)	9985	6118	51	6	50
phoneme[18]	9952	5404	5	2	50
Bioresponse[19]	9910	3751	1776	2	50
madelon (Guyon et al., 2004)	9976	2600	500	2	50
ozone-level-8hr (Zhang and Fan, 2008; Dua and Graff, 2017)	9978	2534	72	2	50
kc1 (Sayyad Shirabad and Menzies, 2005)	3917	2109	21	2	50
mfeat-morphological (Dua and Graff, 2017)	18	2000	6	10	50
steel-plates-fault (provided by Semeion, 2010)	146817	1941	27	7	50
pc3 (Sayyad Shirabad and Menzies, 2005)	3903	1563	37	2	50
cmc (Dua and Graff, 2017)	23	1473	9	3	47
pc4 (Sayyad Shirabad and Menzies, 2005)	3902	1458	37	2	50
pc1 (Sayyad Shirabad and Menzies, 2005)	3918	1109	21	2	49
qsar-biodeg (Mansouri et al., 2013)	9957	1055	41	2	50
credit-g (Dua and Graff, 2017)	31	1000	20	2	50
vehicle (Siebert, 1987)	53	846	18	4	50
analcatdata_dmft (Simonoff, 2003)	3560	797	4	6	50
diabetes (Dua and Graff, 2017)	37	768	8	2	50
blood-transfusion-service-center (Yeh et al., 2009)	10101	748	4	2	50
eucalyptus (Bulloch et al., 1991)	2079	736	19	5	50
credit-approval (Dua and Graff, 2017)	29	690	15	2	49
ilpd (Dua and Graff, 2017)	9971	583	10	2	50
cylinder-bands (Evans and Fisher, 1994; Dua and Graff, 2017)	14954	540	37	2	50
climate-model-simulation-crashes (Lucas et al., 2013)	146819	540	18	2	45
kc2 (Sayyad Shirabad and Menzies, 2005)	3913	522	21	2	33
MEAN	-	12 244	193.39	3.26	49.13

B Performance Results of the Example Benchmark

In the following analysis, we want to determine the average best ensemble technique for post hoc ensembling. The example benchmark provides us with data that simulates the use case of an AutoML framework after model selection, whereby the 50 best performing models have been pre-selected for ensembling.

In this use case, we are only interested in the performance w.r.t. OpenML's Area Under ROC Curve (AUROC) metric. That is, the metric that was used to select the 50 best performing models. Post hoc ensembling is an extension of the optimization process within the AutoML tool. Hence, to evaluate an ensemble technique for post hoc ensembling, it must be analyzed w.r.t. the metric that

[17] `https://www.kaggle.com/datasets/numerai/encrypted-stock-market-data-from-numerai`

[18] `https://sci2s.ugr.es/keel/dataset.php?cod=105`

[19] `https://www.kaggle.com/competitions/bioresponse/data`

is to be optimized during model selection. Extending this analysis to multiple metrics (and hence to multiple benchmarks) is left for future work.

Please refer to Figure 1 and 2 for a detailed presentation of the performance of all ensemble techniques across all datasets. Both figures show that the performance can drastically differ per fold. This is a problem especially for smaller datasets. Considering that we use 10-fold cross-validation and split the fold's prediction data with a fraction of 0.5, only 5% of the data are used for training by the ensemble technique and another 5% for evaluating the ensemble technique per fold. To illustrate, a dataset with 1000 instances would only have 50 instances for training and 50 more for evaluation per fold. Metatasks would require validation data to get more instances for the training phase of ensemble techniques. However, such data is not available on OpenML. As a result, evaluating with Assembled-OpenML might only be representative for larger datasets on OpenML. We see it as future work to explore data sources that include validation data.

To determine the average best, we use the closed gap metric following Lindauer et al. (2017). That is, we normalize the mean performance of each ensemble technique per dataset by using the mean performance of the VBA and SBA. We set the performance of the VBA equal to 1 and the performance of the SBA equal to 0. Any technique that has a higher performance than the SBA will get a positive value between 0 and 1 and a technique that performs worse than the SBA will get a negative value. The normalized value of an ensemble technique will show us how much it improved upon the SBA in relation to the VBA (or degraded performance for negative values). Finally, we take the mean over all datasets of all normalized values. To overcome the impact of too small datasets, we additionally evaluate it once only for datasets with at least 1900 samples in total. We select 1900 as the threshold based on the dataset "steel-plates-fault" (OpenML Task ID 146817) in Figure 1 and 2. Datasets with more than 1900 samples seem to vary less per fold performance. The results are shown in Table 2.

Table 2: **Ensemble Technique Results** Shown are the mean and standard deviation in parentheses of the closed gap normalized performances for each ensemble technique on all datasets and only on datasets with at least 1900 samples. The evaluated ensemble techniques are Dynamic Classifier Selection (DCS), Dynamic Ensemble Selection (DES), Stacking, Voting, Ensemble Selection (ES), Virtual Best Ensemble (VBE), and Virtual Best Algorithm (VBA).

Data	SBA	DCS	DES	Stacking	Voting	ES	VBE	VBA
All Datasets	0.0	-0.09 (±0.29)	-0.104 (±0.46)	-0.063 (±0.28)	-0.134 (±0.46)	-0.13 (±0.5)	0.307 (±0.43)	1.0 (±0.0)
Datasets $n \geq 1900$	0.0	-0.002 (±0.07)	0.005 (±0.1)	0.023 (±0.09)	-0.022 (±0.14)	0.041 (±0.11)	0.395 (±0.33)	1.0 (±0.0)

We can see from Table 2 that Ensemble selection and Stacking perform best. The average best ensemble technique including small datasets is Stacking. Excluding small datasets, Ensemble Selection is the average best. For larger datasets, both are able to improve upon the SBA on average. Likewise, for all ensemble techniques, the standard deviation is decreased and the mean performance is increased without smaller datasets. The high standard deviation and worse-than-SBA performance of all ensemble techniques while including small datasets further indicate that separate validation data per fold or larger datasets should be used to give the ensemble techniques enough data for training. In general, the subpar performance can also be explained by the missing diversity of base models in the benchmark. Selecting the top-n configurations can result in selecting models of only two different algorithms, e.g., for the OpenML Task 31 the top 100 configurations are from two different implementations of Random Forest[20]. This shows that more sophisticated methods to filter the configurations obtained from OpenML might be needed.

The performance of the VBE shows that the VBA is perhaps not the best oracle that we can use to compare ensemble techniques. As discussed in Caruana et al. (2004), we would like to set

[20] See `https://www.openml.org/search?type=task&sort=runs&id=31`

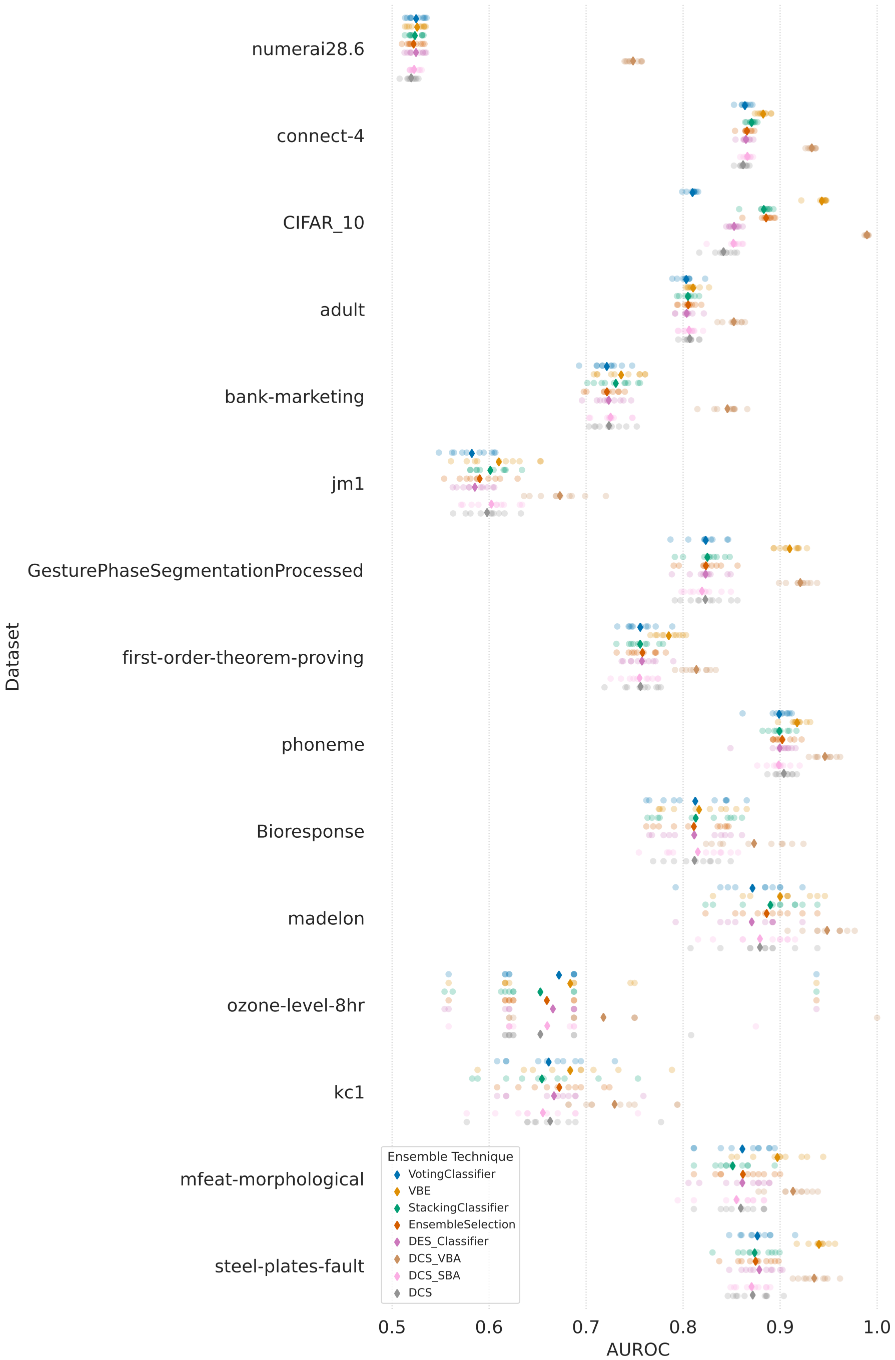

Figure 1: **Performance of Simulated Ensemble Techniques on the Example Benchmark - Part 1** The Area Under ROC Curve (AUROC) score of different ensemble techniques for all 10 folds per dataset and their mean (represented by the diamond).

the upper limit to the Bayes optimal performance for normalization if we knew it. The large gap between VBE and VBA could indicate that the VBA is not close to the Bayes optimal. Problems of the oracle for dynamic selection were already raised by Cruz et al. (2018).

In this initial version, Assembled-OpenML generated and stored the (meta-)data used for the comparisons. However, the usability of the evaluation can be improved upon. For the evaluation,

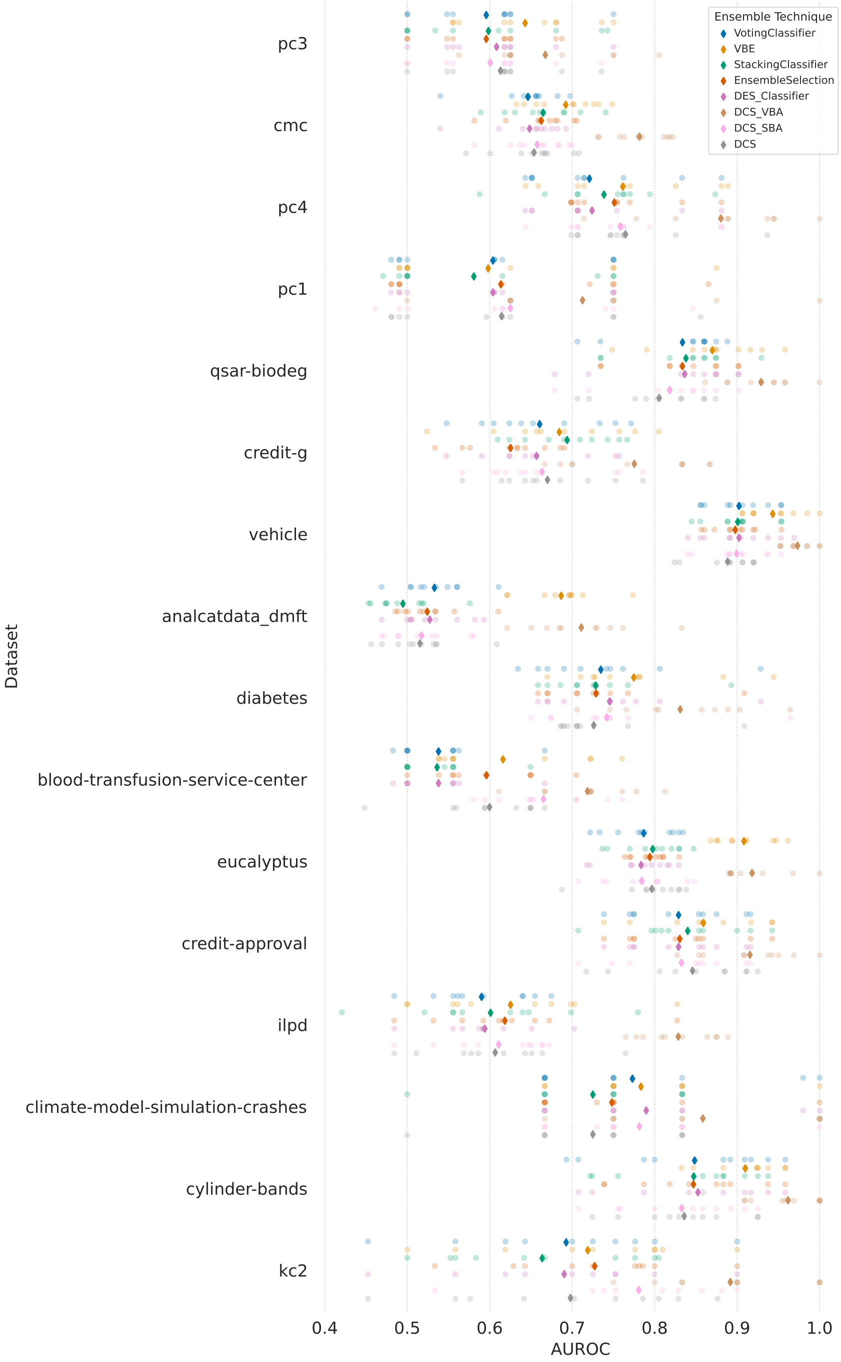

Figure 2: **Performance of Simulated Ensemble Techniques on the Example Benchmark - Part 2** The Area Under ROC Curve (AUROC) score of different ensemble techniques for all 10 folds per dataset and their mean (represented by the diamond).

we had to pass the prediction data to our own implementations of the ensemble techniques. In the future, we aim to automate passing the required data to any ensemble technique and evaluating the technique across all folds.

C Details on Simulated Ensemble Techniques

For this paper, we implemented the ensemble techniques ourselves. In the future, we want to reuse existing ensemble technique implementations and pass our data to the implementations. Assembled-OpenML shall support this by passing the data in the form of simulated/faked base models. Initial work on this has already begun. For this paper, the following ensemble techniques have been implemented:

- **Stacking Classifier**: We simulated an implementation of stacked generalization (Wolpert, 1992) (also called stacking). Thereby, a meta-learner learns to combine the predictions of the base models to predict the groundtruth. We use a default[21] LogisticRegression from scikit-learn (Pedregosa et al., 2011) as a meta-learner. We increased the number of maximum iterations to 1000. For our benchmark, we simulate stacking with the predictions' confidences. We are not using the so-called "passthrough" option, that is, training the meta-learner on the predictions and the original training data.

- **Voting Classifier**: We simulated a voting classifier (like sklearn's VotingClassifier). A voting classifier combines the predictions of base models through a majority voting rule. For our benchmark, we simulate voting with the predictions (i.e., "hard" voting).

- **Ensemble Selector** We simulate ensemble selection from libraries of models (Caruana et al., 2004). We implemented an ensemble selector similar to Auto-Sklearn's ensemble technique (Feurer et al., 2015). In ensemble selection, an ensemble is built in a greedy and iterative fashion such that the performance on a validation set is maximized by the non-weighted average of the selected base model's predictions. Thereby, base models can be selected multiple times. The frequency of selection is used as a weight for each base model's predictions at test time. We use an ensemble size (number of iterations) of 50 inspired by Auto-Sklearn's default value.

- **Dynamic Classifier Selector** We simulated an implementation of Dynamic Classifier Selection (DCS) (Giacinto and Roli, 2001). DCS tries to select the best classifier to classify each instance. This is related to *per instance algorithm selection* (Rice, 1976). For example, the classifier for a new instance can be selected based on the performance of the base models on (training) instances in the neighborhood of the new instance. For our benchmark, we simulate DCS using a default RandomForestRegressor from sklearn as an empirical performance model (EPM) of the prediction error (Hutter et al., 2014). We select the classifier for which the EPM predicts the lowest error. In this simulation, selecting a classifier means returning the classifier's predictions.

- **Dynamic Ensemble Selector**: We simulated an implementation of Dynamic Ensemble Selection (DES) (Ko et al., 2008). For each new instance, DES selects a subset of classifiers on the fly for which the predictions are aggregated. Selection techniques similar to DCS can be used. For our benchmark, we extended our Dynamic Classifier Selector implementation to return the combined predictions of a subset of classifiers. We also use a RandomForestRegressor as an EPM. We select the subset of classifiers by adding classifiers to the subset until the accumulated predicted error of the subset is greater than 50% of the total predicted error for an instance. We combine the predictions of a subset of classifiers using majority voting.

- **Dynamic Classifier Selector - SBA**: A simulated version of DCS that always returns the predictions of the best single classifier on the training data. In other words, this is the SBA for selection.

- **Dynamic Classifier Selector - VBA**: A simulated version of DCS that always returns the predictions of the classifier selected by the oracle-like VBA.

[21] Default values from scikit-learn version 1.0.2.

- **Virtual Best Ensemble - VBE**: We introduce a novel baseline called Virtual Best Ensemble (VBE). The VBE is a non-real oracle-like predictor to represent the case where a weighting ensemble technique (like stacking or ensemble selection) found the optimal set of weights for the test data on the training data. For simplicity, we assume that learning the weights on the test data finds an optimal set of weights for the test data. Thus, we use our Stacking Classifier implementation trained on the test data.

D Problems with the Prediction Data of OpenML

Obtaining predictions of a run is currently not integrated into the OpenML Python API[22]. We found that there are multiple prediction file formats, specifically different column names exist. While this is a minor problem, which we solved by checking all formats encountered so far, the bigger problem is a discrepancy between the predictions and the predictions' confidence values.

We have found many examples where the prediction was not equal to the class with the highest confidence value. This is an expected problem for some algorithms, for which the confidence values that are computed are not representative (for example, sklearn's SVM[23]). Moreover, we also sometimes expect this to be a problem of numerical precision. In both cases, we can fix the confidence values. However, we also found discrepancies that we cannot explain and, therefore, do not know how to fix them[24]. To handle this, we store information on the discrepancy and make it possible to filter runs with unexplainable discrepancies later on. In our examples, we always filtered unexplainable discrepancies.

E Fetching the Best Algorithms Instead of Configurations for More Diverse Base Models

Ensembles can perform better with a diverse set of base models (Banfield et al., 2005). Yet, by fetching the top-n configurations, Assembled-OpenML does not necessarily provide a diverse set of base models. We explored the possibly of fetching a more diverse set of base models in preliminary experiments. To do so, we tried fetching the best configuration of the top-n best performing algorithms/pipelines (called flows on OpenML) for a task instead of fetching the overall top-n best performing configurations. This can also be understood as an algorithm selection use case.

Although we can use the configurations of the top-n flows to produce a much more diverse set of base models, it drastically reduces the amount of usable data on OpenML. In total, only ~1600 flows exist compared to millions of runs. Furthermore, ensuring diversity in a collection of flows is problematic due to duplicated algorithms/pipelines. See Appendix F for more details on the duplicate problem of OpenML flows. Finally, we abandoned using the configurations of the top-n flows as base models because it seems to be too far away from the AutoML use case.

F Complications with OpenML Flow Duplicates

An OpenML flow can capture any algorithm/pipeline of supported ML frameworks. Yet, determining that two flows are duplicates of each other is complicated. OpenML stores the results of multiple ML frameworks. Hence, to ensure no duplicates across ML frameworks, we had to identify similar algorithms while being named differently across frameworks. To illustrate, scikit-learn calls a Random Forest Classifier "RandomForestClassifier" while mlr3 calls it "mlr.ranger". The two different implementations of the same algorithm are named differently. While a difference in implementation can affect performance, the difference might not be substantial enough. The two implementations might not be different enough to not be deemed duplicates for the sake of diversity in base models. Moreover, OpenML often stores multiple versions of algorithms. Hence, we had

[22] However, initial work on this topic exists: `https://github.com/openml/openml-python/pull/1128`

[23] `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`

[24] For more details and an example, see: `https://github.com/openml/openml-python/issues/1131`

to remove such version duplicates. Lastly, minor changes in a pipeline can already create a new flow object stored on OpenML. For example, we found minor differences such as "$sklearn.imputer$" vs. "$sklearn\_.imputer$". To automatically associate flows with such minor changes to each other, a sophisticated analysis of the flow object is needed. We solved this by comparing the similarity of two flows manually[25].

G Python Libraries used by Assembled-OpenML

The following Python libraries are used by Assembled-OpenML in its current version:

- OpenML-Python (Feurer et al., 2019), Version 0.12.2, BSD 3-Clause License;
- Pandas (pandas development team, 2020), Version 1.4.1, BSD 3-Clause License;
- Requests[26], Version 2.27.1, Apache License 2.0;
- SciPy (Virtanen et al., 2020), Version 1.8.0, BSD 3-Clause License;
- NumPy (Harris et al., 2020), Version 1.22.3, BSD 3-Clause License;
- python-Levenshtein[27], Version 0.12.2, GPL-2.0 License.

[25] To not be overwhelmed with too much manual comparison effort, we only manually check two flows if they appear to be (highly) similar based on the edit distance.

[26] `https://pypi.org/project/requests/`

[27] `https://pypi.org/project/python-Levenshtein/`