Self-Training with AutoML: Empirical Exploration and Data-Driven Design Improvements

---

Martin Schumann

September 12, 2024
Version: Final

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Department of Mathematics,
Informatics and Statistics
Institute of Informatics

Artificial Intelligence and
Machine Learning

Master's Thesis

Self-Training with AutoML: Empirical Exploration and Data-Driven Design Improvements

Martin Schumann

Reviewer Eyke Hüllermeier
Institute of Informatics
LMU Munich

Supervisors Marcel Wever and Valentin Margraf

September 12, 2024

Martin Schumann

Self-Training with AutoML: Empirical Exploration and Data-Driven Design Improvements

Master's Thesis, September 12, 2024

Reviewer: Eyke Hüllermeier

Supervisors: Marcel Wever and Valentin Margraf

LMU Munich

Department of Mathematics, Informatics and Statistics

Institute of Informatics

Artificial Intelligence and Machine Learning (AIML)

Akademiestraße 7

80799 Munich

Abstract

With the global amount of data being estimated to be 175 zettabytes by 2025 [@SI20], the problem of labeling data is becoming more and more of a challenge. Many approaches have been suggested to allow for machine learning algorithms to utilize unlabeled data without the need for expensive and time-consuming human labeling. One of these approaches is semi-supervised learning, where a dataset contains a small amount of labeled data and pseudo-labels the rest of the dataset by predicting what labels for the unlabeled data will be. We achieve this using training two predictors, the first trained on only the labeled dataset, and the second trained on the labeled and pseudo-labeled dataset. This approach is different to co-ensembling and AutoSSL, which utilize the pseudo-labeled dataset to continue to train the original predictor. We also introduce a "safeguard system", which, with the help of meta-features, predicts whether or not a given dataset will benefit from pseudo-labeling. This allows for compute and time savings by not even having to train the second predictor. We investigate which meta-features make good candidates for this safeguard system and train an ML model which can be utilized to give insights into what makes a dataset a good candidate for semi-supervised learning. In this work, we evaluate the performance on well-known benchmark data sets using customized metrics. Furthermore, we explore which aspects of datasets lead to better or worse performance under which metrics and evaluate how this can lead to improvements in our safeguard system's design.

Acknowledgement

Contents

1 Introduction

One of the biggest problems in machine learning (ML) and artificial intelligence (AI) research is the availability of labeled data. What does this mean? Most data that exists and would be useful for training machine learning models do not have labels describing what the data represents, e.g., for images what objects they contain. These labels, however, are required to train many models, making labeled data one of the most important and costliest things in machine learning [Mur12; HNP09]. This is because labeling the data is most often still done by humans who add the labels, which is expensive, time-consuming, labor-intensive, and not easily scalable [Yao+18; Zho+17; Mur12]. It can also be error-prone.

For ML, a larger amount of correct data almost always leads to higher accuracy results of the model [HNP09]. This is not a problem when ML is done with unsupervised learning, (e.g., for clustering data) because it does not require labeled data, as the model categorizes and extracts certain inherent patterns or features of the data [Bar89; Mur12]. There are limitations to this approach however, including: the lack of "ground truth", i.e., assessing how well a model performs by comparing to correctly labeled answers and no easy way to provide feedback to influence the model. Furthermore, the unsupervised learning approach is sensitive to how the data is input and used (e.g., introducing biases) [Sch+22; CR18; SLH22; TE11].

Supervised learning, which relies on labeled data, and is used e.g., for prediction, does not have some of the limitations that unsupervised learning has. However, because of the smaller amount of labeled data, it has other limitations, including the aforementioned sensitivity to the quality of labels.

Because many ML algorithms rely on supervised learning, there is a tendency in machine learning research for training being executed on the same, much smaller subset of existing data sources, e.g., the IMDB dataset [Maa+11] or MNIST [LeC98; Pla98], as they are already labeled.

To deal with the problems arising from this lack of labeled training data, the concept of self-training was introduced [He+19; Scu65]. One common approach for

this is having an ML model which uses its learned "knowledge" to label unlabeled datapoints, and then utilizing these datapoints to train further, with the goal of improving the performance of the model [EH21]. A different approach works by having one ML model create labels for use in a second ML model. The first model, through supervised learning, is able to "self-label" unlabeled datasets, using the same "human" labels as the dataset it was trained on. These labels, known as "pseudo-labels" are then applied to an unlabeled data set, combined with existing "human" labels and fed into a separate supervised learning model. This (hopefully) eliminates the need for massive amounts of labeled data by generating some labels with the first model. To simplify selecting and tuning these models for each dataset, we utilize AutoML (Automated Machine Learning)-based libraries, which are able to automate many parts of an ML workflow [Feu+15]. This enables us to focus on higher-level tasks, rather than spending time, e.g., manually tuning hyperparameters or selecting which models to use.

Of course, the question is, if and under which circumstances adding more data in the form of automatically labeled data can lead to better performance in the second model compared to just using a smaller set of human labeled data. As there is no guarantee that "pseudo-labels" from the first model are accurate, adding more datapoints might lead to a worse performance for the second model. We therefore ask our main research question: can better performance of an AutoML trained ML model be achieved by utilizing previously unlabeled data, which a separately (AutoML) trained model has labeled?

1.1 Motivation and Problem Statement

For supervised learning, new data cannot be used until it is labeled. This means that canned benchmark datasets are used, which can contain incorrect [VEJ21; Ast+15], biased [Car19], or incomplete information [VEJ21; Ast+15]. Furthermore, older data might not correspond to changes in the real world [Din+21]. Two example datasets that fit these criteria and are widely used are the "Boston Housing" dataset [HR78; Del96; Car19] and the "Census Income" dataset [Din+21; BK96].

The goal of this work is to investigate if less reliance on canned benchmarks is possible with the help of semi-supervised learning. Although, as with most machine learning models, 100% accuracy can't be guaranteed while being sufficiently general [WM97; Mur12; Wol96], an improvement to the status quo is welcome. We

investigate whether it is possible to make sure that inaccuracies in the "pseudo-labeled" dataset do not lead to errors or performance loss, which can often occur [He+19; ZG09].

We investigate this accuracy problem and which factors could lead to worse performance, and try to eliminate these factors in order to achieve better performance at the end of the learning process. We also automate this process to eliminate a lot of expensive and time-consuming human labor [Zho+17; Yao+18].

Our results could also mean less costly and faster iteration of models with new unlabeled data, because newer data sources can be labeled and used to train an existing model. This is an ongoing and very important area of research, as many ML systems try to improve by having more up-to-date data [Dat].

Less reliance on a handful of datasets could lead to an improvement in performance and usefulness for ML models. This is particularly relevant in application areas where the amount of varied labeled datasets is slim [ZL18; And10; Lat+17].

Therefore, if being able to use semi-supervised models with similar or better performance compared to regular supervised learning models is possible, it would provide many benefits, such as reducing the reliance on canned labeled datasets, decreasing the cost of human labeling and increasing the speed of innovation in various domains.

1.2 Thesis Structure

Chapter 2

In this chapter, we introduce the state of the art in AutoML and Self-Supervised Learning research, and explore the foundational elements of meta-features.

Chapter 3

In this chapter, we describe the research that has been done to combine AutoML and Self-Supervised Learning.

Chapter 4

In this chapter, we describe the theory behind our approach, including the safeguard

system. This system can predict whether it is likely for the second classifier to perform better or worse. This makes it possible that the second classifier does not even need to be trained, which may be an expensive task.

Chapter 5

In this chapter, we present the experimental setup, including which datasets and performance metrics are used.

Chapter 6

In this chapter, we present the results of our experiments, including which meta-features can predict the performance impact of the classifiers, and how this information is used in the safeguard system.

Chapter 7

In this chapter, we discuss future work and our conclusions.

2 Foundations

In this chapter, we introduce foundational concepts related to our approach, including AutoML, semi-supervised learning and meta-features.

2.1 AutoML

AutoML (Automated Machine Learning) as a concept describes the process of automating ML tasks and workflows. There are many libraries which utilize AutoML algorithms to provide the ability to simplify a lot of work required for building ML systems [Feu+15]. This wide field of libraries includes ML-Plan [MWH18], Auto-WEKA [Tho+13], which is an extension to "WEKA", auto-sklearn [Feu+15], which is an extension to the popular scikit-learn [Ped+11] library, and AutoGluon [Eri+20], which is a newer AutoML framework. They accomplish simplification by automating a lot of the time and tedious process of setting up an ML pipeline. Among other things, they automate the data preprocessing and feature engineering steps, the model selection steps and the hyperparameter optimization [Feu+15; Tho+13; MWH18]. With the help of feature extraction and feature selection, the data is preprocessed and the relevant features are selected. For model selection, AutoML libraries try out different models to find the best one for a given dataset. Hyperparameter optimization ia another important task, which involves using different algorithms to select the optimal hyperparameters for a given model. Although most AutoML libraries support these features, they are not created equal, as we will see in the next section.

2.2 AutoGluon

AutoGluon is an AutoML library which promises to be very easy to use while having competitive performance. It can achieve this by "ensembling multiple models and

stacking them in multiple layers" [Eri+20], e.g., combining the output of multiple models for one prediction.

This framework was chosen for the quality of its models, its easy of use and its speed, especially compared to auto-sklearn. It is also one of the best performing AutoML libraries on current benchmarks [Gij+23].

2.3 Auto-Sklearn and Scikit-Learn

Another popular library, which has been used in a lot of AutoML research is auto-sklearn. By eliminating algorithm selection and hyperparameter tuning, it is able to make training ML models very easy. It is based on the widely used scikit-learn [Ped+11] library. Although we do not utilize auto-sklearn, we utilize scikit-learn [Ped+11]'s `RandomForestClassifier`. It is a classifier which combines multiple decision trees in order to improve performance [@Sha]. We use it for our safeguard system because of its ease of use, performance and being simple to tune and debug.

2.4 Semi-Supervised Learning

Semi-supervised learning is one way to deal with the problem of large amounts of unlabeled data existing. The basic concept is having a subset of the data being labeled and then use certain strategies to utilize on the rest of the unlabeled data. One of these strategies is known as "pseudo-labeling", which involves an algorithm labeling some or all of the unlabeled data. These "pseudo-labeled" data points are then used like normal data points. The combination of labeled and pseudo-labeled data can then be used to train with supervised learning algorithms [Lee+13].

2.5 Meta-Features

Meta-features are calculated measures of a dataset that describe aspects of the dataset [Alc+20]. These features can be used to predict the performance of ML

algorithms and can also be, e.g., utilized in AutoML for model selection and optimization [Kot+21].

Alcobaça et al., the authors of an implementation of meta features for python (pymfe [Alc+20]), provide a lot of different meta features to choose from. These meta-features can be split into multiple groups, including, but not limited to: model based, landmarking, and clustering [Alc+20; Riv+18]. Model-based meta features are "measures extracted from a model" [Riv+18], meaning that a simple model, most often a decision tree is trained and the characteristics of this model are the meta features. Examples include the number of leaves or the tree imbalance [@Ede]. Another meta-feature group is landmarking, which are "measures that use the performance of simple and fast learning algorithms to characterize datasets" [Riv+18], e.g., training simple classifiers and the extracting the characteristics of these classifiers. Examples include the nearest neighbor or naive Bayes classifiers [@Ede]. A third meta-feature group is clustering, which as the name implies, deals with the characteristics the clusters in a dataset have [Alc+20; PD19]. Examples include the Pearson correlation or the Dunn index [@Ede].

In this work we utilize meta-features for our safeguard system, to be able to predict if training with "pseudo-labeled" data will lead to an improvement of performance.

3 Related Work

3.1 AutoML and semi-supervised learning combined

There are two main papers that focus on unifying AutoML and semi-supervised learning. The first, by Engelen and Hoos introduces the term co-ensembling [EH21]. The paper introduces an approach for generating pseudo-labels: letting a group of K classifiers learn on a labeled dataset, and then having a subset of $K-1$ classifiers add artificial labels to an unlabeled portion of the dataset. This combined data is then used to train the K-th original classifier further. This classifier is then the final model. These classifiers are all trained with the help of auto-sklearn [Feu+15]. Selecting which data to label is done by having a certain threshold of minimum probability, and only selecting a set number of unlabeled data instances to label. Although the authors introduce this approach as being able to be repeated in a multistep process, their findings conclude that single-step is better, as multiple steps can lead to errors at the beginning being multiplied, leading to a much worse performance as the iteration continues. They therefore focus on single-step co-ensembling.

While this paper contains both binary and multi-class datasets, and utilizes the OpenML100 collection of datasets, its performance in binary datasets is still lacking, according to the authors.

The second paper, by Li et al., introduces the term AutoSSL (for auto semi-supervised learning) to describe unifying the two approaches (AutoML and SSL) from the view of semi-supervised learning (SSL). The authors use a clustering approach to characterize the data, and use meta-features of the data to inform the "automated learning" [Li+19] process. This "automated learning" [Li+19] process utilizes well-known SSL algorithms and combines it with AutoML techniques to tune the hyperparameters of these algorithms.

This approach is then compared to a normal auto-sklearn approach, which the authors claim has "highly competitive" [Li+19] performance compared to classic SSL techniques.

This paper [Li+19] focuses on small binary datasets, and the approach has not been extended to "multi-class problems or very large scale datasets" [Li+19], which are described as future work.

Li et al.'s paper is also referenced by [EH21] as having a different approach, although both papers focus on how to combine AutoML and semi-supervised learning. The differences lie in the angle from which the authors tackle the problem. Li et al. use semi-supervised learning algorithms and tune them using AutoML techniques, whereas Engelen and Hoos use AutoML tuned classifiers to try and improve classifier performance.

3.2 Autogluon's current semi-supervised support

Autogluon has experimental support for "unlabeled_data". However, it requires the experimental "FT_TRANSFORMER (Tabular Transformer, GPU is recommended.[...])" [@Aut, #L504] model, which "does not scale well to >100 features." [@Aut, #L504]. This translates to quite a few shortcomings: requiring a GPU and less than 100 features for good performance and only being able to be used with the Tabular Transformer model [@Aut, #L900-L911]. These shortcomings limit the usability compared to our approach.

There is also some support for "pseudo-labeling" [@Aut, #L1904], which labels data above a certain probability threshold and uses it as extra training data to refit the original model.

3.3 Summary

There is a real need for more research in the area of learning with large amounts of unlabeled data, as more and more the limits of labeled datasets are revealed [VEJ21; Sch+22]. The need to decrease the immense cost and effort required to label data

is also becoming more and more important. Our approach, which will be discussed in more detail in the next section, differs to current approaches in two important ways. First, they retrain a model instead of training another completely separate model from scratch, and they also lack our safeguard system.

4 Semi-Supervised AutoML Pipeline

As described in the previous chapters, there are multiple ways to combine semi-supervised learning with AutoML. These approaches use an existing model that was originally trained on labeled data. This model is then retrained on an updated dataset with new "pseudo-labeled" data (e.g., [EH21]).

This is different from our approach, where we train two independent models. The first model has been trained with the originally labeled data. The second model is then trained from scratch with a combination of data that has been "pseudo-labeled" by the first model, and originally labeled data. We call this method *linear-ensembling*, as the models are trained sequentially in a linear fashion. This is contrast to co-ensembling, where a single model is retrained, as defined in [EH21].

In the following sections, we describe our main pipeline (Section 4.1), which illustrates the concept behind our linear-ensembling approach. This pipeline produces the second trained model. However, sometimes the second model performs worse than the first model. We therefore introduce a safeguard system (Section 4.2), which decides which model should be returned.

4.1 AutoML Pipeline

In this section, we introduce our linear-ensembling pipeline (Figure 4.1), consisting of the two models M1 and M2 to be trained. Each model is a classifier trained using AutoGluon ("AutoML" in Figure 4.1). The models are trained one after the other and the pipeline returns M2 as the final model.

The pipeline utilizes three subsets of one dataset: the labeled data (d_l), the unlabeled data (d_u) and the test data (d_t, also labeled).

M1 is trained solely on labeled data. Once trained, it pseudo-labels all the unlabeled data. It only keeps those data points above a certain confidence threshold, which is a constant value (see Section 6.1). Pseudo-labeling involves a classifier (here: M1) predicting the label for a certain data point in d_u. In our case, the classifier predicts a probability distribution for the possible labels for each data point. Then, only if one label has a higher probability than the confidence threshold, the data point is used for further training. If no label has a high enough probability, the data point is not used for further training. This pseudo-labeling step is the "Adds labels" process in Figure 4.1 and is shown in more detail in Figure 4.2. All the newly pseudo-labeled data and labeled data are combined into a new dataset used to train the model M2.

To test our approach, the performance of M1 and M2 are tested on the test data and the performance is compared, according to the "accuracy" measurement (see Section 5.3).

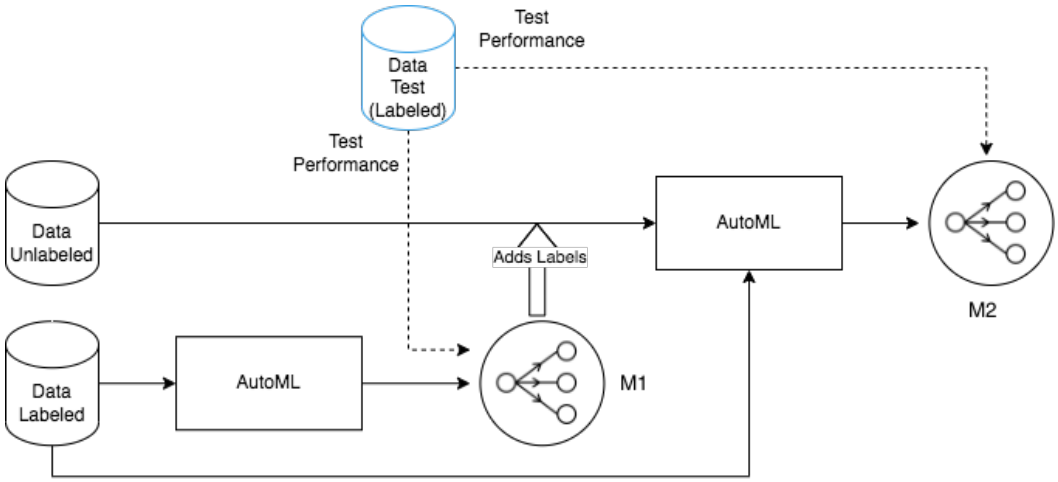

Fig. 4.1.: Graphical representation of the pipeline. M1 is the first model (labeler), M2 is the second model.

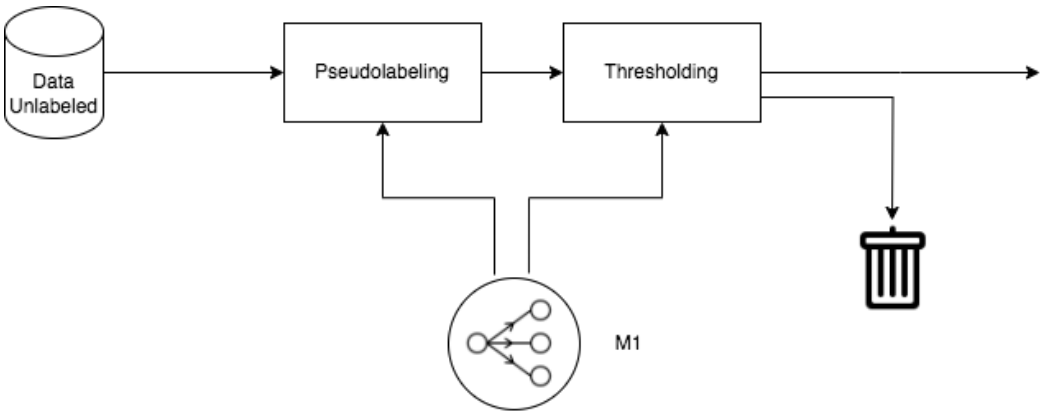

Fig. 4.2.: Graphical representation of the "Adds labels" process from Figure 4.1. Data with pseudo-labels that do not meet a certain threshold get "thrown away" (trashcan).

It is possible that M2 sometimes performs worse than M1. To overcome this problem, we introduce a safeguard system, described in Section 4.2.

4.2 Safeguard System

Sometimes, introducing more training data in the form of "pseudo-labeled" data can actually hurt the performance of a model. This effect, known as confirmation bias [Ara+20], means that, for our approach, M2 might perform worse than M1. This is because if the "pseudo-labels" of M1 are incorrect or the model is overly confident in its predictions, a lot of bad predictions are used to train M2. This means that M2 will learn from these bad predictions and labels, which might lead to it performing worse than M1, which was trained only on correctly labeled data. We therefore need to make a decision of whether to select M1 or M2 as our final model. Furthermore, if training M2 results in decreased performance, we would waste a lot of costly training time. We have decided to alleviate the problem of deteriorated performance and the high cost of training by introducing a safeguard system.

This system predicts whether a given M2 model will perform better or worse than the M1 model. Then, depending on settings, pseudo-labeling of data and the training of M2 will not occur. As can be seen in Figure 4.3, prediction is done by extracting meta-features from the initially labeled dataset, and then returning whether M2 would perform better. If M2 is predicted to perform better, the safeguard system returns "plus", otherwise it returns "minus".

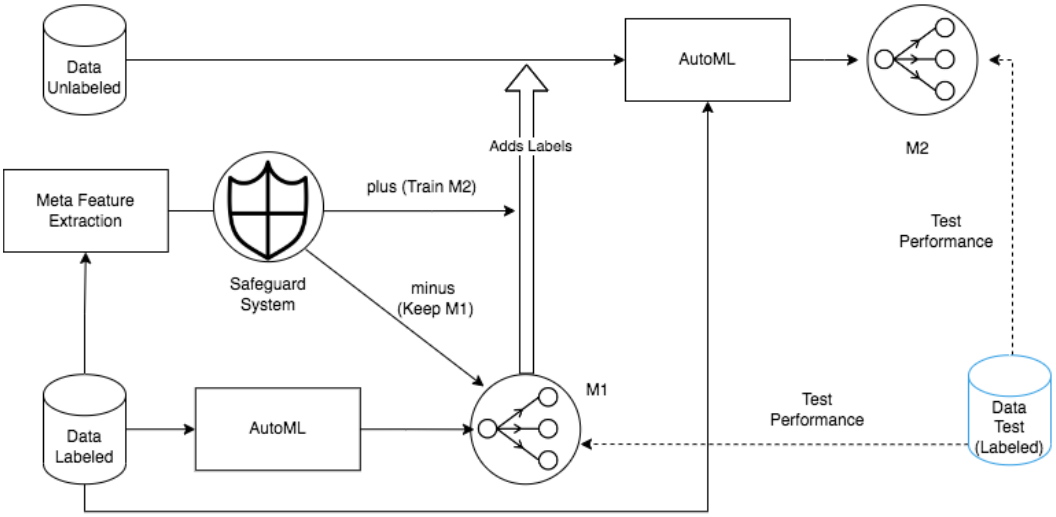

Fig. 4.3.: Pipeline with Safeguard System included. Pseudo-labeling and training of M2 only occurs if the safeguard system allows it. Otherwise M1 is kept as the main model.

This system provides two major benefits. First, it is able to give insight into when and how an ML model performs better or worse after being trained with pseudo-labeled data, and for which datasets this occurs. Second, it saves time and compute resources as it can be used to provide a quicker and more efficient insight into whether a dataset will perform worse with M2, and so M2 does not even need to be trained.

The safeguard system is implemented both using scikit-learn's `RandomForestClassifier` and AutoGluon. This is to make evaluating various performance differences easier, as is described in Section 6.4. Both systems are trained using the same procedure (see Section A.2).

5 Experiments

In this chapter, we explore how the experiments were run and what the criteria for performance and success are. We aim to answer our main research question: can better performance of an AutoML trained ML model be achieved by utilizing previously unlabeled data, which a separately (AutoML) trained model has labeled? To accomplish this, we split up our research into answering 5 research questions:

1. Which settings affect the performance of the model M2?

2. Does our approach perform favorably to AutoGluon's built in support for semi-supervised learning?

3. Is there a way to predict the performance of a certain dataset? Here we investigate the usage of meta-features.

4. Which meta-features perform the best for the safeguard system?

5. Which models and settings influence the performance of the safeguard system?

5.1 Experimental Setup

As described in Chapter 4, we run tests to compare whether utilizing pseudo-labeled data can improve the performance of a classifier. We also need to train and benchmark our safeguard system as described in Section 4.2.

We run both of these tasks on the LRZ's interactive and serial linux clusters [@Rec] (using `slurm`), which at the time of writing are running "Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz" CPUs. We do not utilize GPUs for our work, as they bring no huge performance benefit for our datasets (tabular data).

We utilize python 3.10 and anaconda version `anaconda3/2022.10` to run our experiments, as this is the latest version as part of `spack/23.1.0`, which is the most up-to-date software packaged for the linux clusters. The package versions utilized are also pinned in the `requirements.txt` files.

For AutoGluon's `fit` method (which trains a model), we keep the most of the default settings as described in [@Eri+24], except for increasing the time limit to 600 seconds and changing the quality parameter. As described in [@Eri+24], there are different qualities to choose from that define how fast the training and inference speed is and the quality and size of the models. We utilize two different quality settings in our testing: `medium quality` (the default) and `high quality`. For most of our tests, we use `medium quality` as it provides fast training and inference speeds, without a lot of overhead. We use the results obtained from the `high quality` setting to verify our insights gained from the `medium quality` results. We decided not to utilize `best quality` as it consumes a lot of resources. The results of datasets trained using the `medium quality` setting are used when training the safeguard system.

For pymfe, we also add a time limit of 600 seconds, as sometimes calculating the meta-features can take a long time. If the time limit is reached, our safeguard system method errors out and linear-ensembling continues with training M2.

5.2 Datasets

For the datasets, we utilize the OpenML100 [Bis+21] datasets and its successors, the OpenML-CC18 [Bis+21] datasets as a baseline for performance comparison. These datasets are widely used [Bis+21], and are suitable as a basis for benchmarking the performance of our approach and the safeguard system. For more variety and to see how the implementation functions on larger and more diverse datasets, we utilize some datasets which were compiled by Fusi et al. We put them in 3 dataset groups and named them CD1 [FSE18], CD2 [FSE18], and CD3 [FSE18] (see Section A.1).

For each individual dataset, we do no preprocessing or normalization, as AutoGluon takes care of these tasks. We split the dataset as follows: 80% is used for training, with 10% of that data being labeled, i.e., 8% of the total. The other 20% is used as test data to test the performance of our models. We utilize scikit-learn's `train_test_split` method with `random_state=42` to split our dataset.

5.3 Performance Metrics

There are a lot of different performance metrics which describe how well a classifier can classify. For binary classifiers, they include: accuracy, F_1, F_β, recall and precision [@Resa; @devb; @deva; @deve; @Resb]. All of these metrics rely on 4 values: TP (True Positives) and TN (True Negatives), which describe the number of classifications a binary classifier gets correct, and FP (False Positives) and FN (False Negatives), which are the number of classifications that are incorrect [@Resa]. The reason for splitting classifications into 4 categories is to distinguish between different types of misclassifications, e.g., for an MRI incorrectly not identifying cancer versus incorrectly identifying cancer. In binary classification, the positives correspond to one label, and negatives correspond to the other label. These metrics can also be used in multi-class, by having positives correspond to the correct label, and negatives correspond to all other incorrect labels. They are then also typically weighted by according to their class size.

Accuracy describes the ratio of correct predictions to the total number of predictions a classifier makes [@Resa]. It is the measurement we use to measure the performance of the classifiers on the datasets described in Section 5.2. The formula for binary classifiers is as follows:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall describes what percentage of positive classifications are actually correct [@deve].

$$recall = \frac{TP}{TP + FP}$$

[@Resb]

Precision is similar to recall, but it describes how well a classifier is able not to "label as positive a sample that is negative" [@devc].

$$precision = \frac{TP}{TP + FN}$$

[@Resb]

F_β describes the "**weighted** harmonic mean of precision and recall" [@devb], i.e., the ratio of the two including a β parameter to control the ratio.

$$F_\beta = \frac{(1+\beta^2) * TP}{(1+\beta^2) * TP + FP + \beta^2 * FN}$$

[@devb]

F_1 is the same formula as F_β with $\beta = 1$, i.e., an unweighted "harmonic mean" [@deva].

$$F_1 = \frac{2 * TP}{2 * TP + FP + 1 * FN}$$

[@deva]

We also implement a custom scorer, which weighs false negatives or false positives heavily. The general formula for it is:

$weighted_{loss} = (weight_{fn} * FN) + (weight_{fp} * FP) + (weight_{tn} * TN) + (weight_{tp} * TP)$

(based on [TGS10])

The weights for TN and TP are equal to 0.

This allows us to weight only false positives and/or false negatives heavily and so be able to influence the performance of the classifiers much more.

In our experiments, we use accuracy for the performance of the models M1 and M2, and we have utilized accuracy, F_1, F_β and recall to train the safeguard system's classifier.

5.4 Evaluation Metrics

In this section, we describe how we measure the performance of our approach, and how we quantify different subsets of our results to paint an overarching picture. To calculate the performance metrics, we utilize three python scripts. The first calculates the mean and graphs the safeguard performance: `safeguard_perf.py`. It ignores datasets where the safeguard system, i.e., pymfe errors out. The pymfe errors can occur for multiple reasons: a calculation time limit, a memory limit, or

another error. The second script (`graph_slurm_output.py`) graphs the before and after performance. The third script (`read_out_compare.py`) counts which linear-ensembling run performed better given 2 runs by counting which run won for each dataset. Its results are very close to the results for the mean in terms of efficacy.

Overall performance is evaluated using the "accuracy" measurement. We utilize the change in "accuracy" between M1 and M2 ($accuracy = accuracy_{M2} - accuracy_{M1}$) as the "general" performance metric. If $accuracy > 0$, then M2 has performed better, if $accuracy \leq 0$, then M1 has performed better. The "accuracy" measurement allows for an individual view of each dataset and also how much the safeguard system affects one dataset. The mathematical definition of "accuracy" is found in Section 5.3.

Another metric to evaluate the performance of our approach is the percentage of data ignored ($pd_{ignored}\%$). It describes, at a given threshold (see Section 6.1), what percentage of data is not used for training M2, and is useful for measuring the impact of thresholds. It is defined as follows:

$$pd_{ignored}\% = (100 * |d_{ignored}|)/|d_{pseudolabeled}|$$

with

- $d_{ignored}$ being a set of data points that are pseudo-labeled but have low confidence and are therefore not utilized.

- and $d_{pseudolabeled}$ being the total set of pseudo-labeled data points.

The third metric for evaluation is comparing two different runs (`run_1` and `run_2`) on the same dataset `d` and comparing which dataset "wins" in terms of "accuracy" for each run. These are then tallied up into a count. The pseudocode is as follows:

```
    wins_run_1 = 0
    wins_run_2 = 0

    for d in datasets:
        if accuracy(run_1, d) > accuracy(run_2, d):
            wins_run_1 = wins_run_1 + 1
        else:
            wins_run_2 = wins_run_2 + 1
```

To compare accuracy with and without the safeguard system, we define the following: Without the safeguard system (acc) describes the "accuracy" of M2 when the safeguard system is not applied ($acc = accuracy_{M2}$). With the safeguard system (acc_{ss}) either the "accuracy" of M1 when the safeguard system predicts "minus" ($acc_{ss} = accuracy_{M1}$) or M2 when the safeguard system predicts "plus" ($acc_{ss} = accuracy_{M2}$). Only datasets which do not hava a pymfe error are looked at. To evaluate how well the safeguard system performs, we count how often $acc_{ss} > acc$ for a suite of datasets.

A further metric to evaluate the performance of the safeguard system is comparing the mean accuracy with ($mean((acc_{ss}))$) and without ($mean((acc))$) the safeguard system. Let $(acc) = \{acc_1, \ldots, acc_n\}$ and $(acc_{ss}) = \{acc_{ss,1}, \ldots, acc_{ss,n}\}$, with each acc_i and $acc_{ss,i}$ being the accuracy of one dataset i.

$$\begin{aligned} mean((acc)) &= \frac{\sum_{i=1}^{|(acc)|} acc_i}{|(acc)|} \\ mean((acc_{ss})) &= \frac{\sum_{i=1}^{|(acc_{ss})|} acc_{ss,i}}{|(acc_{ss})|} \end{aligned} \tag{5.1}$$

Only datasets which do not have an error with the pymfe system are looked at. The mean gives a different view on the impact of the safeguard system by not looking at individual values but the overall performance impact on a suite of datasets.

We also utilize the MDI (mean decrease in impurity) graph, which shows the most significant features as the mean (blue bar) and the variation (standard deviation) around the "decrease in impurity" value (black bar) [@sci]. This graph allows us to show the importance of each feature and how much it contributes.

Additional measures are those created by the SHAP ("SHapley Additive exPlanations" [LL17]) library, which utilize the SHAP value (impact on model output) as a way to measure the performance of features of a dataset [LL17]. We describe them in more detail in Subsection 5.4.1.

5.4.1 SHAP

In this section we describe the SHAP library [LL17] and how we utilize it to evaluate and analyze our meta-features and safeguard system. `SHAP` is a python library that utilizes a "game theoretic approach" [@Lunc] to give insights into ML models. It does this by computing Shapley values, which, in their original form, calculate how to "fairly" distribute the reward of a game among players, who might be in a coalition. To calculate this, each subset of players and their contribution to the game is looked at. This approach is applied to an ML context by having the "reward" of the game be the prediction, and the players be the individual features [Mol20]. The "coalition" of features is then, e.g., a collection of pixels for an image based ML task [Mol20]. This approach makes SHAP able to calculate how features interact to influence a prediction.

We use two types of SHAP based graphs to explain our meta-feature selection and the impact of the meta-feature items on model output. The first is the bar plot [@Luna], which shows how much each meta-feature contributes to predictions [@Coo]. This feature importance is defined as the mean of the absolute Shapley values ($mean(|SHAPvalue|)$). The more a feature contributes towards a prediction, the higher the SHAP value. The second plot is the beeswarm plot [@Coo], which is showing two things. The first represents the feature's value ranging from low to high (blue to red), indicating the specific value of the feature for each data point. The second is the impact of each feature to the model output, as defined by the SHAP value.

6 Results

In this section, we present the results of our experiments by answering the research questions defined in Chapter 5.

6.1 Settings Affecting Performance (RQ1)

In this section, we answer the research question: which settings affect the performance of the model M2? There are 2 main aspects to consider here.

The first are the thresholds for keeping pseudo-labeled data. Each prediction from M1 comes with a probability distribution for each label which sums up to 100%, e.g. 80% for label 1 and 20% for label 2. When deciding which pseudo-labeled data to keep, we utilize the probability distribution and only keep the labels with at least one of the labels having a probability greater than the threshold. A high threshold means only a few predictions are used to train M2, the downside being that not a lot of pseudo-labeled data is used. Using very little pseudo-labeled data means there is litte possibility of M2 increasing its performance. A low threshold has the opposite problem, that a lot of possibly inaccurate pseudo-labels are used. The goal is to find an appropriate threshold which finds a good middle ground between these 2 problems.

We have found that a threshold of 0.8 achieves a balance between having too few new labels and having a lot of incorrect labels. We have also investigated the following thresholds: 0.5 (better than a coinflip for binary datasets), $1/|labels|$ (better than a coinflip for multi-class datasets), and 0.9. As can be seen in Figure 6.1 and Figure 6.2, a threshold of 0.9 leads to a worse performance of M2 compared to 0.8.

Both 0.5 and $1/|labels|$ had worse performance. $1/|labels|$ eliminated almost no datapoints at all (Figure 6.3), which makes it useless as a threshold. 0.5 eliminated al-

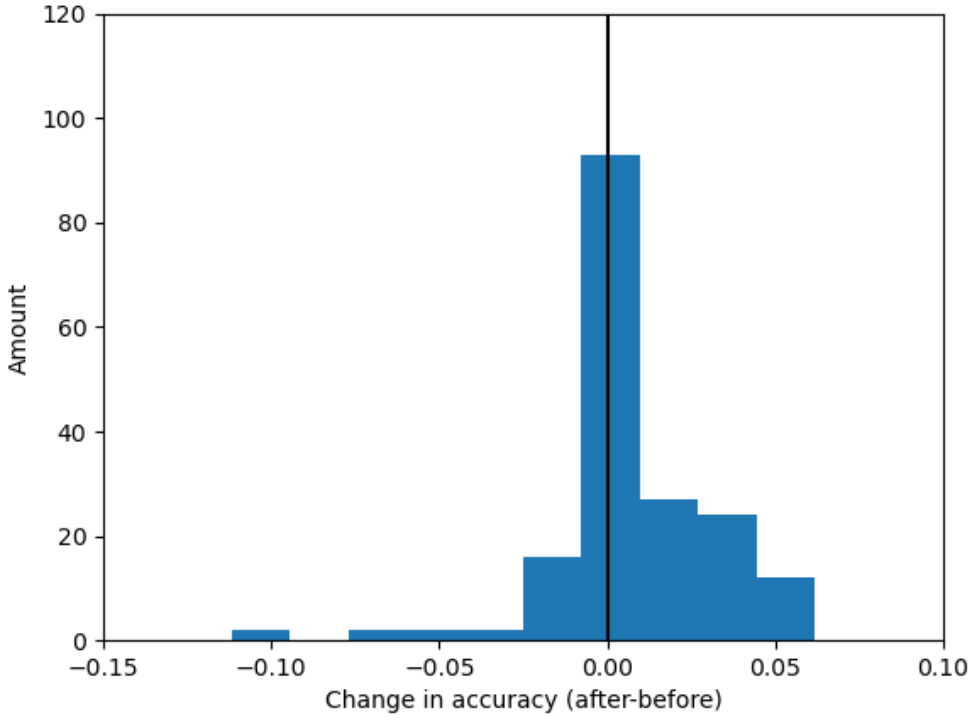

Fig. 6.1.: Histogram of a threshold of 80% and the change in performance.

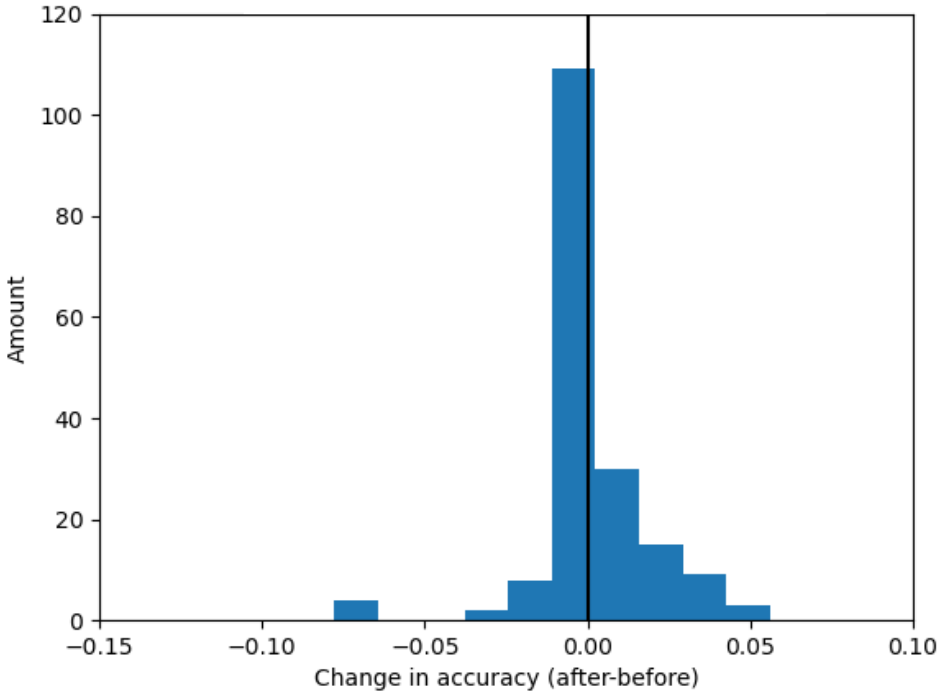

Fig. 6.2.: Histogram of a threshold of 90% and the change in performance.

most no datapoints for binary datasets, and while it worked for multi-class datasets (Figure 6.4), it still performed worse than the other thresholds (Table 6.1).

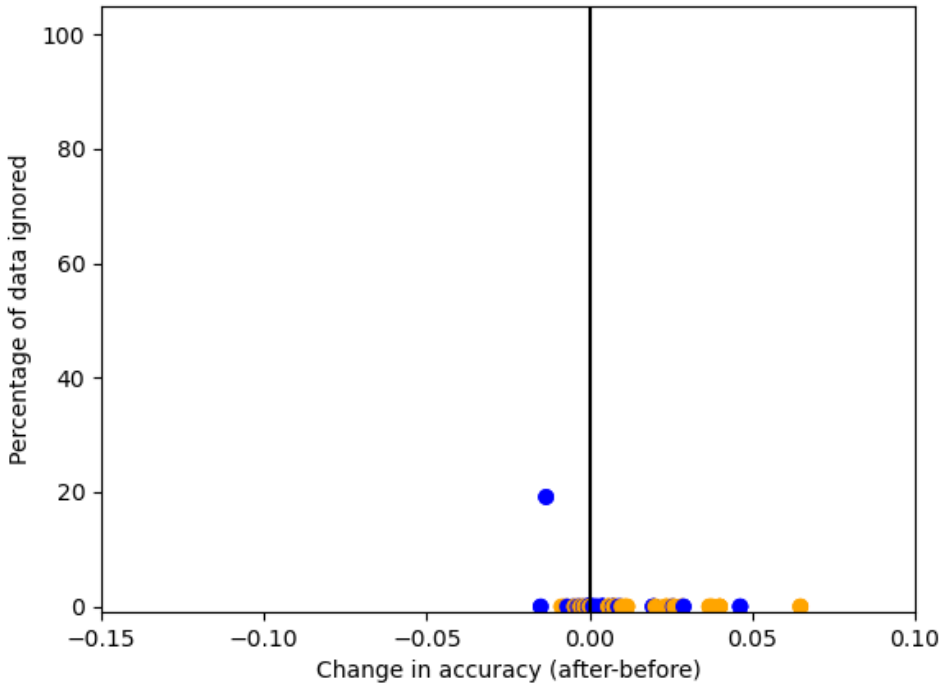

Fig. 6.3.: Percentage of data ignored vs change in accuracy for threshold of $1/|labels|$. Orange represents more than 2 labels in a class, whereas blue is a binary (2 labels) dataset.

When calculating the mean accuracy (see Section 5.4) using the safeguard system (Table 6.1), 0.8 also comes out ahead with the best value of an increase of 0.006 in the mean.

Threshold	Mean with Safeguard System	Mean	Is Better?
0.5	0.7924	0.7899	True
0.8	0.7967	0.7907	True
0.9	0.7812	0.7798	True
$1/\|labels\|$	0.7871	0.7882	False

Tab. 6.1.: Mean values for different thresholds, run on OpenML100. "Is Better?" being "True" means that the mean with safeguard system is larger than the mean without it.

The second aspect, as described in Section 5.1, is which quality settings we set for AutoGluon's `fit` method. We compare the default `medium quality` (MQ) with the `high quality` (HQ) output using the same safeguard system. The number of datasets compared is fewer than the total number of datasets because of time

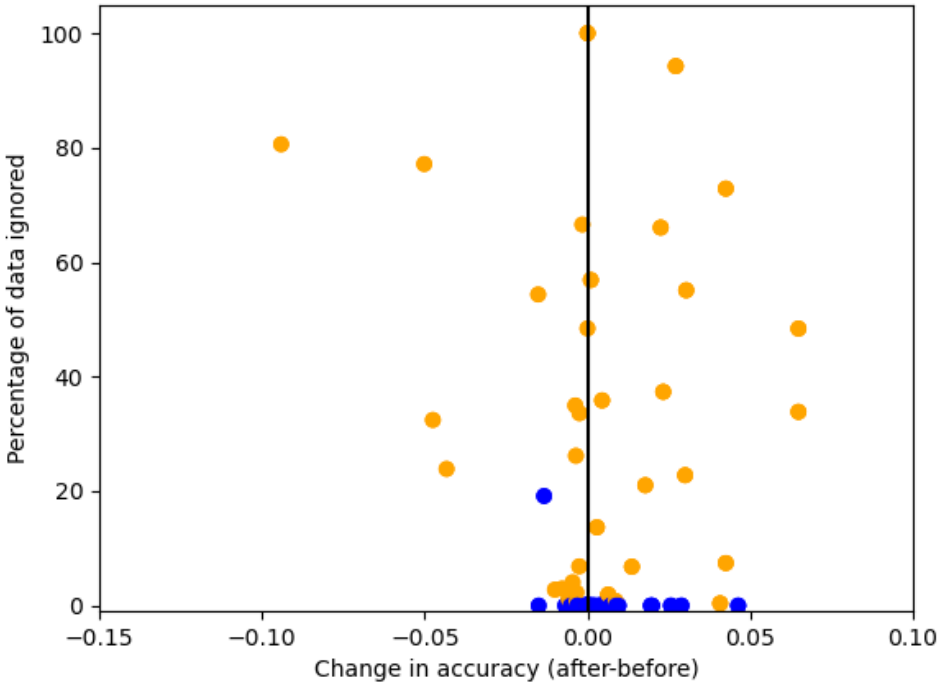

Fig. 6.4.: Percentage of data ignored vs change in accuracy for threshold of 0.5. Orange represents more than 2 labels in a class, whereas blue is a binary (2 labels) dataset.

limits and pymfe errors. We compare the performance using the "accuracy" measurement.

1. Compare M1 (HQ) vs M1 (MQ) on OpenML100
2. Compare M2 (HQ) vs M2 (MQ) on OpenML100
3. Compare M1 (HQ) vs M1 (MQ) on CD3
4. Compare M2 (HQ) vs M2 (MQ) on CD3

The results can be found in the following tables. We look at the results by comparing the count of datasets where HQ and MQ are better (Table 6.2 and Table 6.3) and the mean accuracies (Table 6.4).

In almost all scenarios, high quality is slightly better, as a count of individual better datasets (as opposed to the mean, described later). For our use-case, high quality is not that worth it to test the performance of linear-ensembling. This is because the

Datasets	$\|HQ\|$ better	$\|MQ\|$ better	$\|EqualPerf\|$	Winner
`OpenML100 HQ vs MQ (M1)`	40	36	7	HQ
`OpenML100 HQ vs MQ (M2)`	48	27	8	HQ

Tab. 6.2.: Comparing high quality vs medium quality on OpenML100.

Datasets	$\|HQ\|$ better	$\|MQ\|$ better	$\|EqualPerf\|$	Winner
`CD3 HQ vs MQ (M1)`	43	41	11	HQ
`CD3 HQ vs MQ (M2)`	40	41	14	MQ

Tab. 6.3.: Comparing high quality vs medium quality on CD3.

performance difference is minimal, especially when comparing performance after linear-ensembling.

In the following table (Table 6.4), we compare the mean accuracies for different quality selections.

Datasets	Mean with Safeguard System	Mean	Is Better?
CD3 HQ	0.6719	0.6733	False
CD3 MQ	0.7121	0.7019	True
OpenML100 HQ	0.7707	0.7690	True
OpenML100 MQ	0.7831	0.7764	True

Tab. 6.4.: Mean accuracies for different quality selections. HQ is high quality, MQ is medium quality. "Is Better?" being "True" means that the mean with safeguard system is larger than the mean without it.

Mean is somewhat better for medium quality, especially for CD3 HQ, which decreases its perf with the safeguard system. This shows that the linear-ensemble system overall works better on medium quality, including because the safeguard system is trained on medium quality datasets. This result can be seen in Figure 6.5, Figure 6.6, Figure 6.7 and Figure 6.8. This reason for this result, as can be seen, is that there are a lot more positive outliers for medium quality.

In the next sections, we will also look at the performance in terms of run-time and accuracy when utilizing the safeguard system.

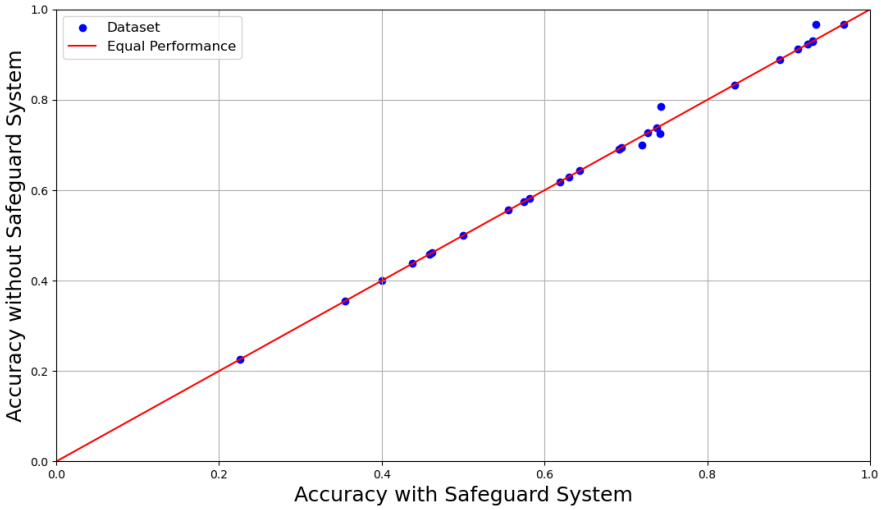

Fig. 6.5.: Performance Comparison of Safeguard vs Non-Safeguard Models: CD3 high quality

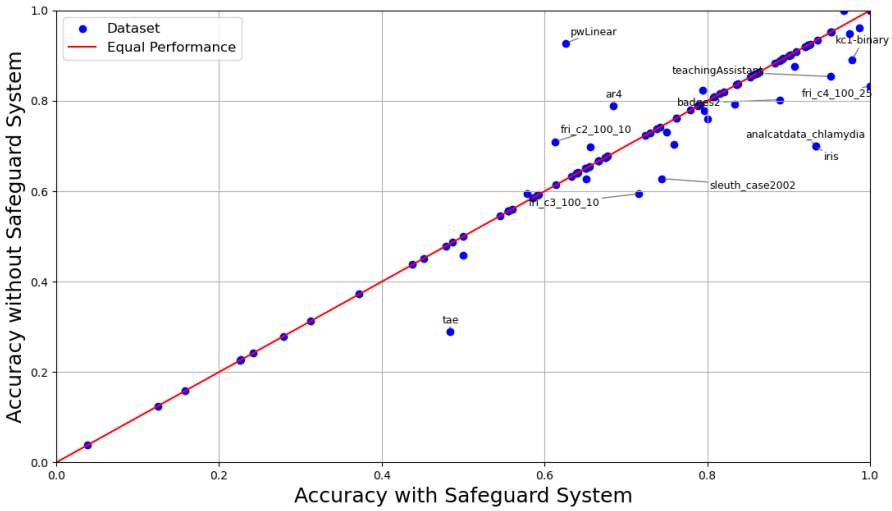

Fig. 6.6.: Performance Comparison of Safeguard vs Non-Safeguard Models: CD3 medium quality

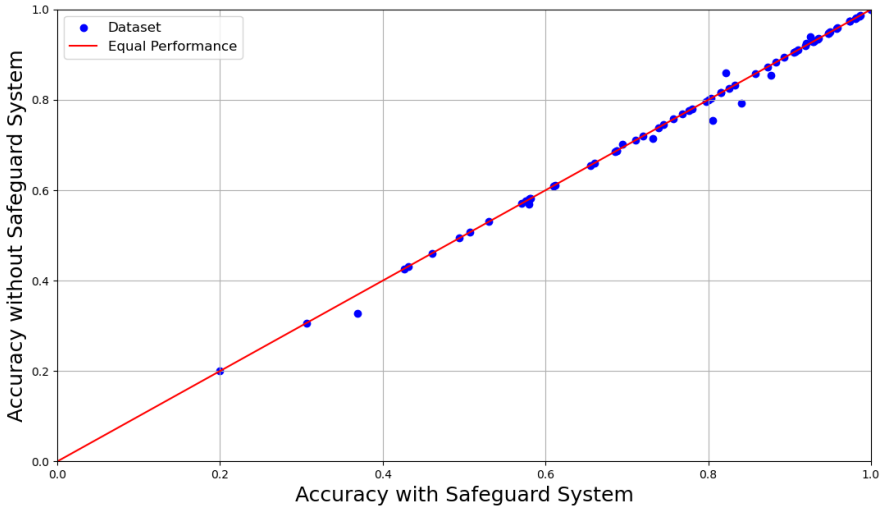

Fig. 6.7.: Performance Comparison of Safeguard vs Non-Safeguard Models: OpenML100 high quality

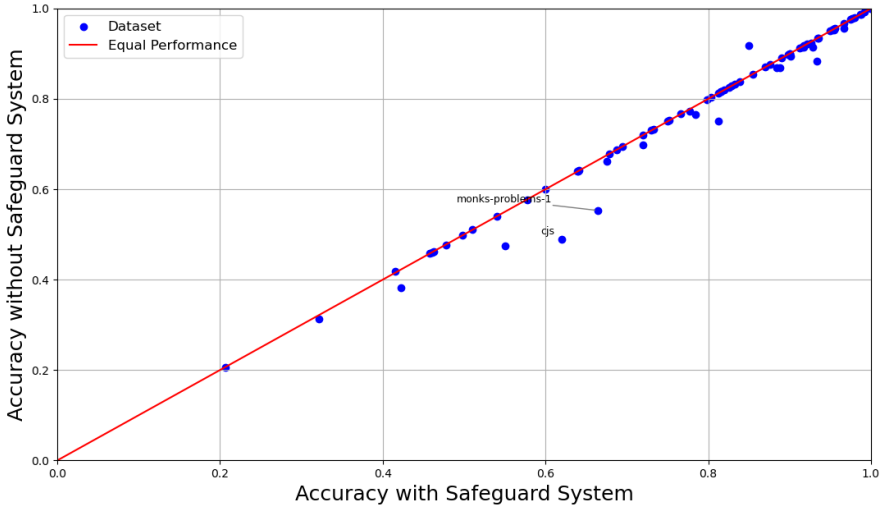

Fig. 6.8.: Performance Comparison of Safeguard vs Non-Safeguard Models: OpenML100 medium quality

6.2 Linear-Ensembling versus Autogluon's Built In Semi-Supervised Learning Support (RQ2)

In this section, we answer the question: Does our linear-ensembling approach perform favorably to AutoGluon's built in support for semi-supervised learning? As described in Section 3.2, AutoGluon has experimental support for unlabeled data to be used for training using a semi-supervised approach. We want to compare our approach (see Chapter 4) to two different variations. The first variation is comparing our approach (linear-ensembling, S1) to AutoGluon's support for semi-supervised learning (S2). The second variation is a hybrid approach (S3), where we use AutoGluon's support to train M1 and then using our approach to train M2. We also compare S1's M1 model to S2's model, as the safeguard system is not in effect, so sometimes the S1's M1 model might have better results than S1's M2 model. We compare the performance using the "accuracy" measurement. The number of datasets compared is fewer than the total number of datasets because of time limits and pymfe errors. The percentage after the winning approach ("Winner") are the following formulas (for S1 & S2 and S1 & S3):

$$winner\% = 100 * \begin{cases} \frac{|S1|}{|S1|+|S2|} & |S1| > |S2| \\ \frac{|S2|}{|S1|+|S2|} & otherwise \end{cases}$$

$$winner\% = 100 * \begin{cases} \frac{|S1|}{|S1|+|S3|} & |S1| > |S3| \\ \frac{|S3|}{|S1|+|S3|} & otherwise \end{cases}$$

As we can see in the following tables, the performance is almost always better after linear-ensembling (Table 6.7) and always better than just AutoGluon's semi-supervised support (Table 6.6). However, the performance of S1's M1 model compared to the S2 model is mixed (Table 6.5). For Table 6.8, S2 is sometimes better than S1's M2, but not always. This means that the purely semi-supervised support is sometimes better than our approach, excluding the safeguard system. What we have learned is that semi-supervised support has worse performance than our linear-ensembling approach, and it provides marginal benefits utilizing it for training M1.

We don't show the graphs comparing safeguard vs non-safeguard performance, as they change in performance is not really visible: there are almost no outliers.

Datasets	$\|S1\|$ better	$\|S2\|$ better	$\|EqualPerf\|$	Winner
OpenML100	15	21	43	S2 (58.3%)
OpenML-CC18	12	14	28	S2 (53.8%)
CD1	11	2	12	S1 (84.6%)
CD2	39	28	24	S1 (58.2%)
CD3	6	8	8	S2 (57.1%)

Tab. 6.5.: Comparing S1's M1 model to the S2 model

Datasets	$\|S1\|$ better	$\|S3\|$ better	$\|EqualPerf\|$	Winner
OpenML100	30	24	25	S1 (55.6%)
OpenML-CC18	20	18	16	S1 (52.6%)
CD1	17	6	2	S1 (73.9%)
CD2	36	28	27	S1 (56.2%)
CD3	7	3	12	S1 (70.0%)

Tab. 6.6.: Comparing S1's M1 model to S3's M2 model

Datasets	$\|S1\|$ better	$\|S3\|$ better	$\|EqualPerf\|$	Winner
OpenML100	19	16	44	S1 (54.3%)
OpenML-CC18	9	16	29	S3 (64.0%)
CD1	11	5	9	S1 (68.8%)
CD2	35	28	28	S1 (55.6%)
CD3	6	2	14	S1 (75.0%)

Tab. 6.7.: Comparing S1's M2 model to S3's M2 model

Datasets	$\|S1\|$ better	$\|S2\|$ better	$\|EqualPerf\|$	Winner
OpenML100	31	27	21	S1 (53.4%)
OpenML-CC18	25	18	11	S1 (58.1%)
CD1	9	14	2	S2 (60.8%)
CD2	33	37	21	S2 (52.8%)
CD3	7	7	8	Equal (50%)

Tab. 6.8.: Comparing S1's M2 model to S2

6.3 Meta-Features (RQ3 and 4)

In this section, we answer two research questions. The first: Is there a way to predict the performance of a certain dataset? This means predicting how much the accuracy of M2 will change in comparison to M1, when training on a certain dataset. If the accuracy increases (positive change), then the predictor should predict "plus", otherwise (negative change) it should predict "minus".

As we can see in Figure 6.9, we are able, to some degree of accuracy, predict which dataset will behave how when running our linear-ensembling pipeline.

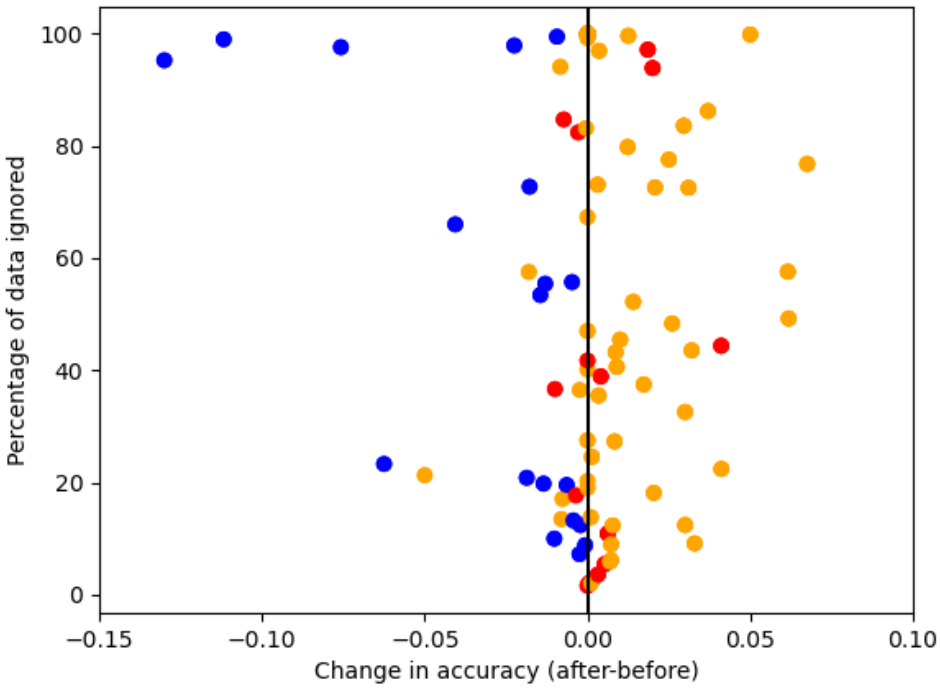

Fig. 6.9.: Percentage of data ignored vs change in accuracy ($accuracy_{M2} - accuracy_{M1}$). Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

We achieve this by training a classifier on 1) meta-features of the dataset and 2) the outcome of multiple dataset runs. This classifier system powers our safeguard system (see Section 4.2), which uses the predicted outcomes to decide whether to select M1 or M2 as the final model, and whether it is worth it to train M2.

We therefore ask our second research question: which meta-features perform the best for the safeguard system?

We have tried a few different meta-features to try and see which provide the best output from the classifier. The very first tests were done using the meta-features found on `openml.org` by parsing the html code using the `BeautifulSoup` library[1]. This was less successful, as the amount of overlap between meta-features on the different pages was very low, and it relied on the dataset being on `openml.org`. The second tests were done using the "general" [Alc+20] group of meta-features from pymfe. This group of meta-features did not provide a great deal of insight for the predictor, as most of the features are very simple and not very meaningful (e.g., number of items in the dataset). This leads us to trying the "model-based" and "landmarking" meta-features, whereby simple ML models are trained and the characteristics/performance metrics of these models is returned [Alc+20]. Although these models are very simple, they can give an insight into what characteristics the dataset will have, and is therefore perfect for our safeguard system, which tries to infer based on the performance of models. We also utilized "clustering" based meta-features, as these were also utilized in the AutoSSL approach as described in Section 3.1.

Our experiments show that a lot of "landmarking" features have negative impact on the model output (Figure 6.10). The reason it does not perform well is that it only returns the performance values of certain algorithms, which is not as meaningful to infer how the data will behave for linear-ensembling. Furthermore, no matter the settings for landmarking, we do not achieve very good performance on CD3, as there are a lot of false negatives.

"Model-based" has more impactful features, but each individually has less impact (Figure 6.11).

"Clustering" has impactful features, only one has no impact (Figure 6.12). This feature (sc, i.e., size of cluster) has no impact because it "[c]ompute[s] the number of clusters with size smaller than a given size"[@Ede], the default being 15, but for our results this never seems to happen.

Ultimately we have found that both "clustering" and "model-based" meta-features provide good and similar performance for the safeguard system. We therefore combined the two: Figure 6.13.

We have also learned that training with more datasets leads to better results in OpenML100, but not necessarily CD3.

[1] `https://www.crummy.com/software/BeautifulSoup/`

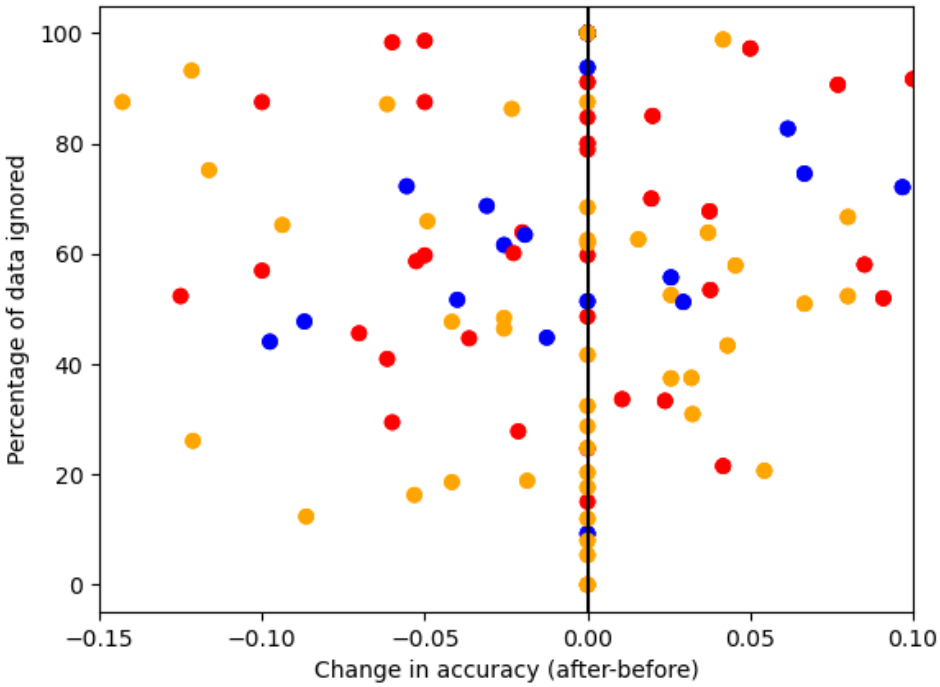

Fig. 6.10.: RandomForestClassifier Result (Landmarking): Trained on OpenML100, OpenML-CC18, CD1, CD2, run on CD3 → bad generalization. Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

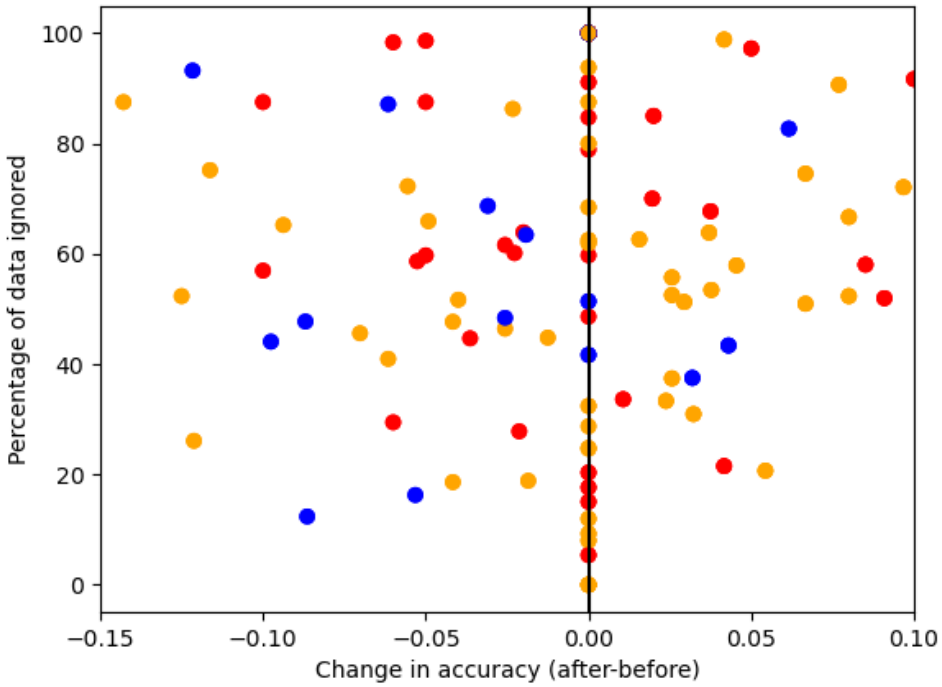

Fig. 6.11.: RandomForestClassifier Result (Model-Based): Trained on OpenML100, OpenML-CC18, CD1, CD2, run on CD3 → ok generalization. Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

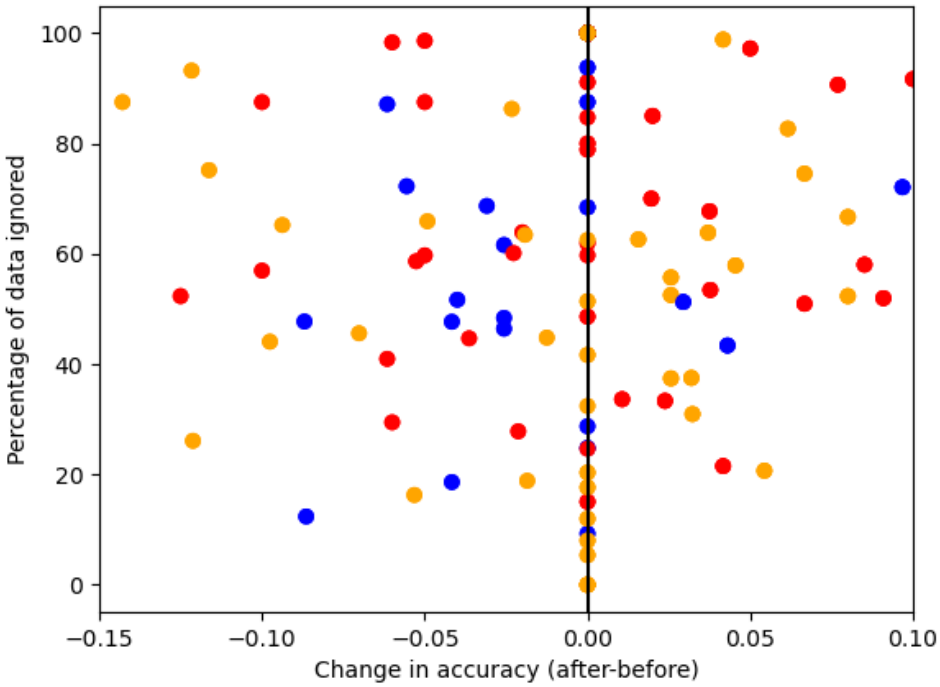

Fig. 6.12.: RandomForestClassifier Result (Clustering): Trained on OpenML100, OpenML-CC18, CD1, CD2, run on CD3 → ok generalization. Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

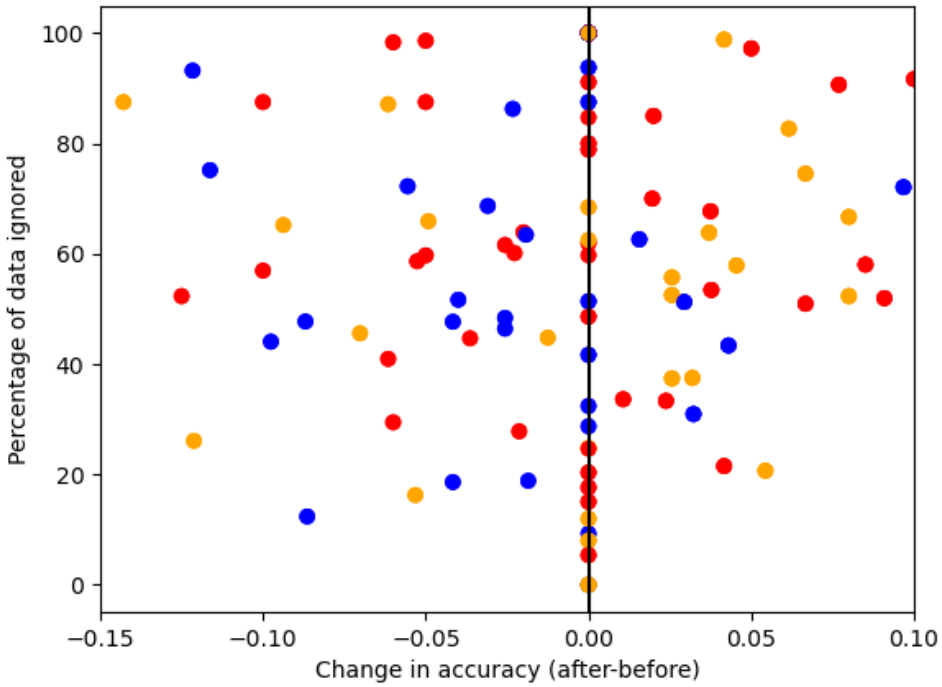

Fig. 6.13.: RandomForestClassifier Result (Clustering + Model-Based): Trained on OpenML100, OpenML-CC18, CD1, CD2, run on CD3 → better generalization. Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

Furthermore, good performance on OpenML100 or CC18 do not necessarily lead to good results on the custom datasets and vice-versa. We see this when comparing Figure 6.13 to Figure 6.14, where in Figure 6.13 there are a lot more "minus" outliers in the positive area.

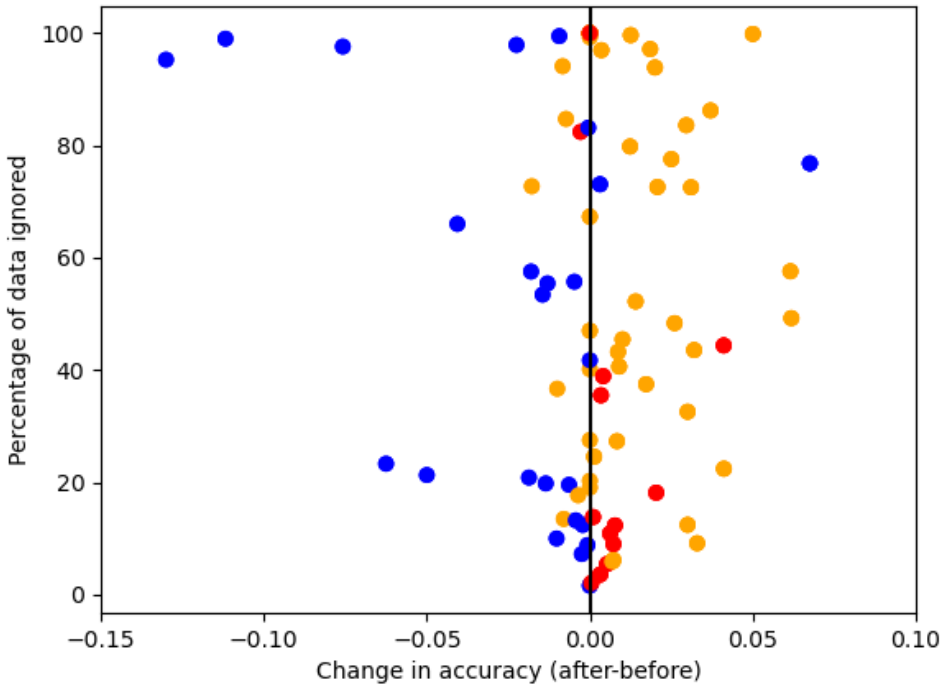

Fig. 6.14.: RandomForestClassifier Result (Clustering + Model-Based): Trained on OpenML100, OpenML-CC18, CD1, CD2, run on OpenML100 → good result

The preceding figures are `RandomForestClassifier` based. We will elaborate on why this classifier was chosen over the AutoGluon one in the next section.

6.4 Safeguard System (RQ5)

In this section, we answer the research question: which models and settings influence the performance of the safeguard system?

We compare the performance along 2 directions. The first is which classifier we use for the safeguard system, scikit-learn's `RandomForestClassifier` (Subsection 6.4.2), or AutoGluon's `TabularPredictor` (Subsection 6.4.1). For the AutoGluon classifier we also look at custom metrics as defined in Section 5.3 to modify the behavior of the classifier. The second direction is looking at meta-feature performance and

relevance, including using metrics from the SHAP library, as described in Subsection 5.4.1.

To measure the benefit of the safeguard system, we measure the mean accuracy with and without utilizing the safeguard system. If the mean accuracy is better or the same, this means the safeguard system has a positive benefit overall, if not, it has a negative benefit. Over a total of 32 runs, 8 runs had worse performance. This includes versions of the safeguard system with less than optimal settings.

When we compare AutoGluon to scikit-learn looking at Figure 6.16 vs Figure 6.15, we can see that AutoGluon barely makes any "minus" (blue circle) predictions, i.e., it almost never thinks that the performance will be decrease, which makes it pretty useless as a predictor.

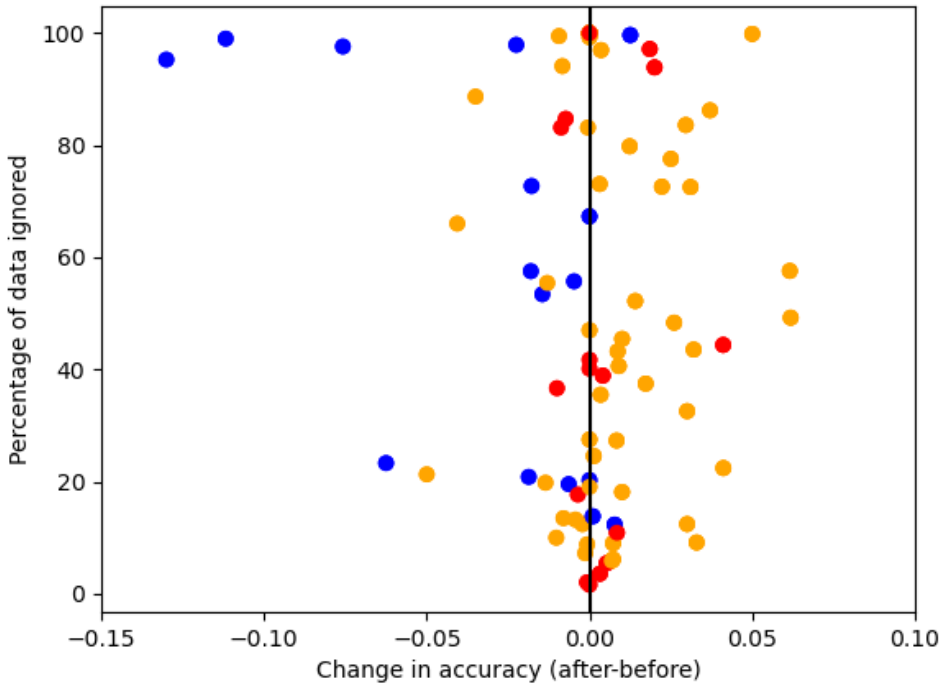

Fig. 6.15.: RandomForestClassifier: Percentage of data ignored vs change in accuracy ($accuracy_{M2} - accuracy_{M1}$). Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

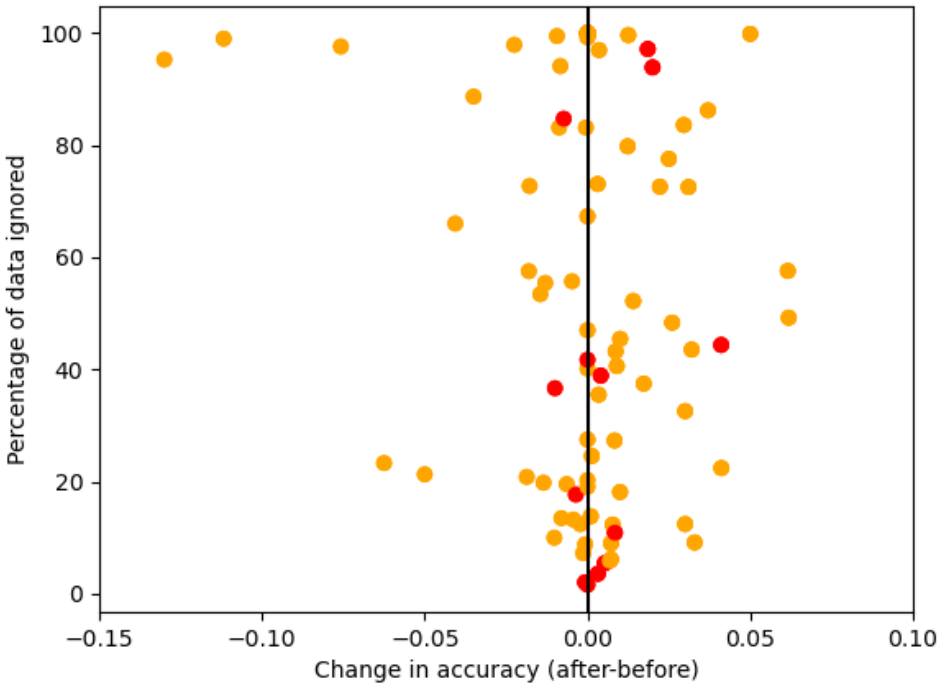

Fig. 6.16.: AutoGluon: Percentage of data ignored vs change in accuracy ($accuracy_{M2} - accuracy_{M1}$). Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

6.4.1 AutoGluon

The performance of the safeguard system implemented using AutoGluon's `TabularPredictor`. Of the 7 runs using the AutoGluon safeguard system, 3 had worse performance.

Even though AutoGluon has mostly positive performance, the predictor does not work very well. It mostly predicts "plus", i.e., that the performance will be better, even when it will not be. As described before, we look at custom metrics to try and influence this behavior, but it has little effect. The following models and the resulting figures are trained on OpenML100 and OpenML-CC18, and run on CD3. They utilize the model-based meta-features.

One way we try to influence the classifier is to highly negatively weigh false positives (FP, "plus"), to try and get more true negatives. The results can be seen in Figure 6.17. As we can see, the results are still not great.

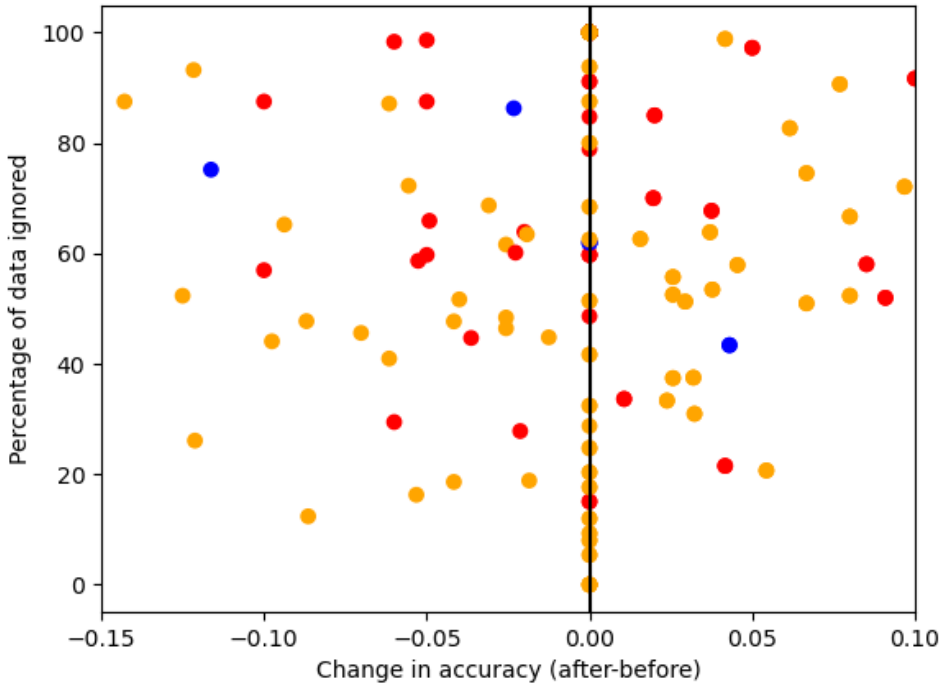

Fig. 6.17.: Custom Metric: Percentage of data ignored vs change in accuracy ($accuracy_{M2} - accuracy_{M1}$). Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

Another approach is using f_β scoring, whereby we set $\beta = 2$ and $\beta = 0.5$, so that we are able to increase the roll of recall and precision. As can be seen in Figure 6.18, setting $\beta = 2$ has little effect, as the classifier still predicts almost all "plus".

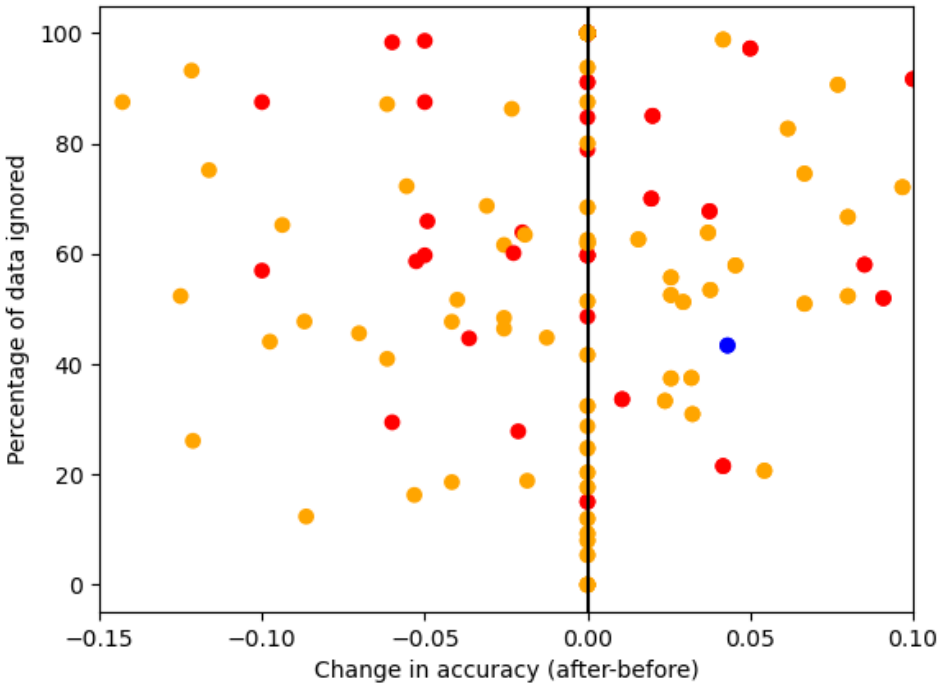

Fig. 6.18.: f_β ($\beta = 2$) metric: Percentage of data ignored vs change in accuracy ($accuracy_{M2} - accuracy_{M1}$). Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

Similarly, as can be seen in Figure 6.19, setting $\beta = 0.5$, also has no effect. There are fewer datapoints and a lot are clustered around $(0, 100\%)$, as this was run with `high quality` and was not able to complete in the time limit.

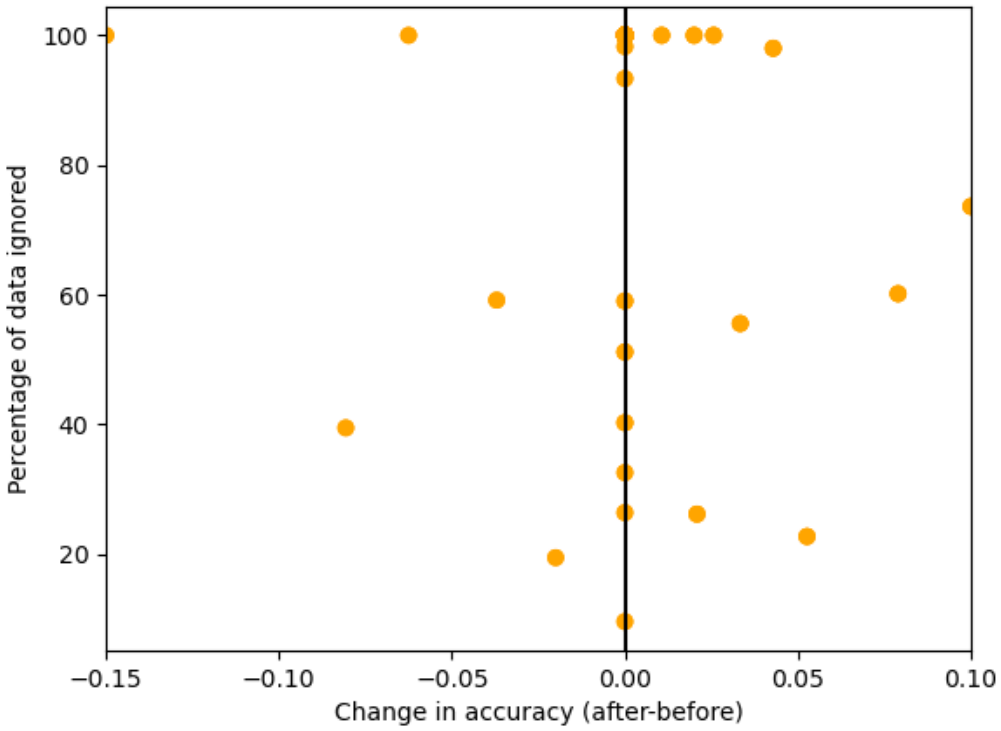

Fig. 6.19.: f_β ($\beta = 0.5$) metric: Percentage of data ignored vs change in accuracy ($accuracy_{M2} - accuracy_{M1}$). Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

The third approach, which is a little better, is using F1 scoring, as described in Section 5.3. The results can be seen in Figure 6.20

We can also see this phenomenon of the safeguard system not providing good performance with "landmarking" (Figure 6.21) and "clustering" (Figure 6.22) meta features by looking at the performance comparison of safeguard vs non-safeguard models.

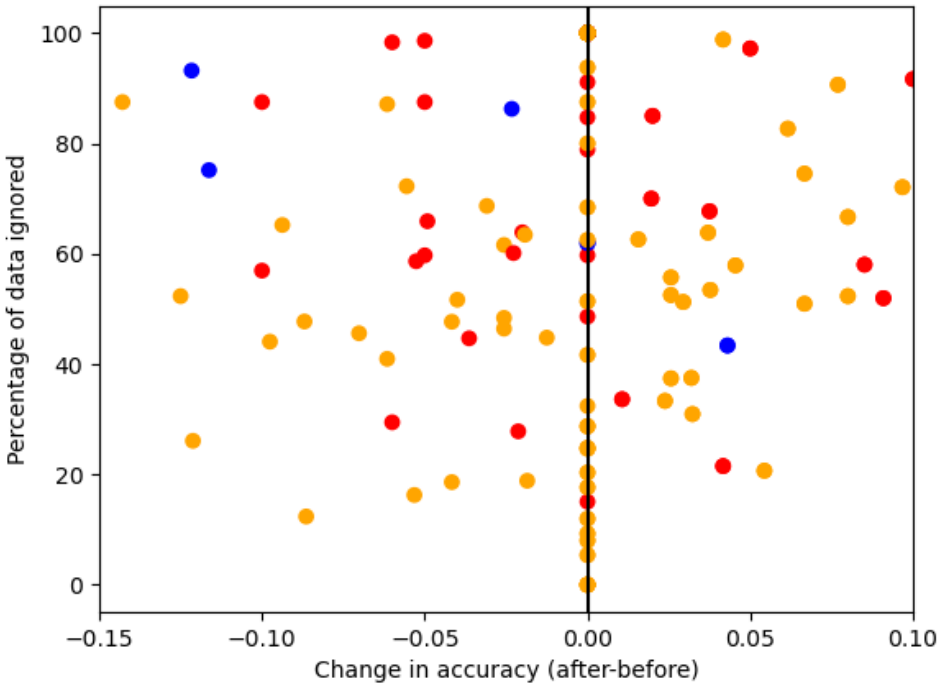

Fig. 6.20.: F1 metric: Percentage of data ignored vs change in accuracy ($accuracy_{M2} - accuracy_{M1}$). Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

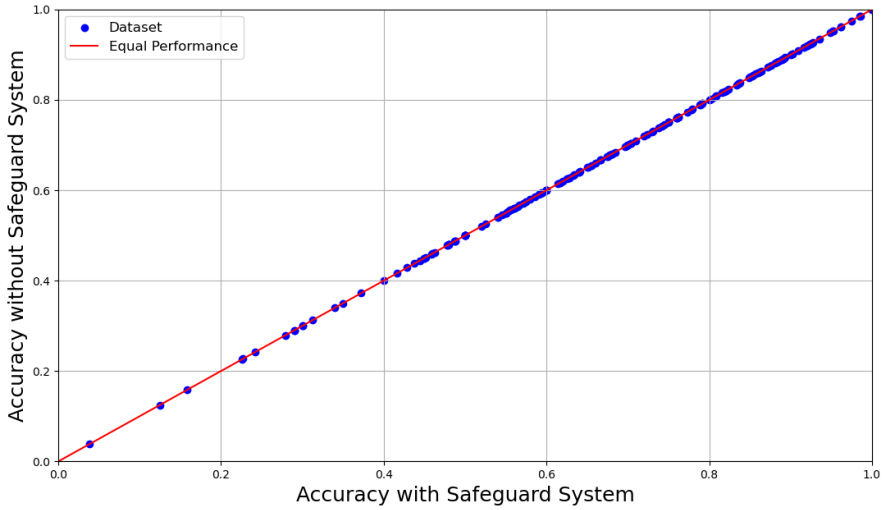

Fig. 6.21.: AutoGluon: Performance Comparison of Safeguard vs Non-safeguard Models (Landmarking)

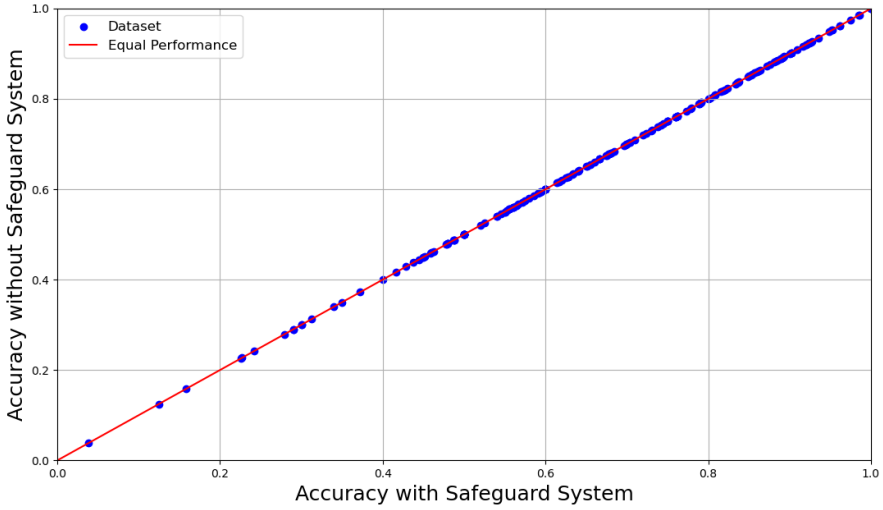

Fig. 6.22.: AutoGluon: Performance Comparison of Safeguard vs Non-safeguard Models (Clustering)

As the safeguard system always predicts plus, the mean is the same for both (Table 6.9) and each dataset has equal performance.

Meta-Features	Mean with Safeguard System	Mean	Is Same?
Landmarking	0.6675	0.6675	True
Clustering	0.6675	0.6675	True

Tab. 6.9.: AutoGluon: Comparison of Mean with Safeguard System and Mean for Different Meta Features

In conclusion, the AutoGluon predictor does not seem to be the optimal solution for our safeguard system. We therefore look at scikit-learn's `RandomForestClassifier` in the next section.

6.4.2 RandomForestClassifier

In this section, we take a look at the performance of the safeguard system implemented using scikit-learn's `RandomForestClassifier` [@devd]. For the 25 runs using the `RandomForestClassifier`, 5 had worse performance. This includes versions of the safeguard system with less than optimal settings, 4 of which had worse performance.

As we can see in the following figures and table (Table 6.10), the scikit-learn based safeguard system almost always performs better with it than without.

Corresponding Figure	Mean with Safeguard System	Mean	Is Better?
Figure 6.23	0.7184	0.7103	True
Figure 6.24	0.7831	0.7764	True
Figure 6.25	0.7121	0.7019	True
Figure 6.26	0.7160	0.7101	True
Figure 6.27	0.6953	0.6966	False
Figure 6.28	0.7918	0.7867	True

Tab. 6.10.: AutoGluon: Comparison of Mean with Safeguard System and Mean. "Is Better?" being "True" means that the mean with safeguard system is larger than the mean without it.

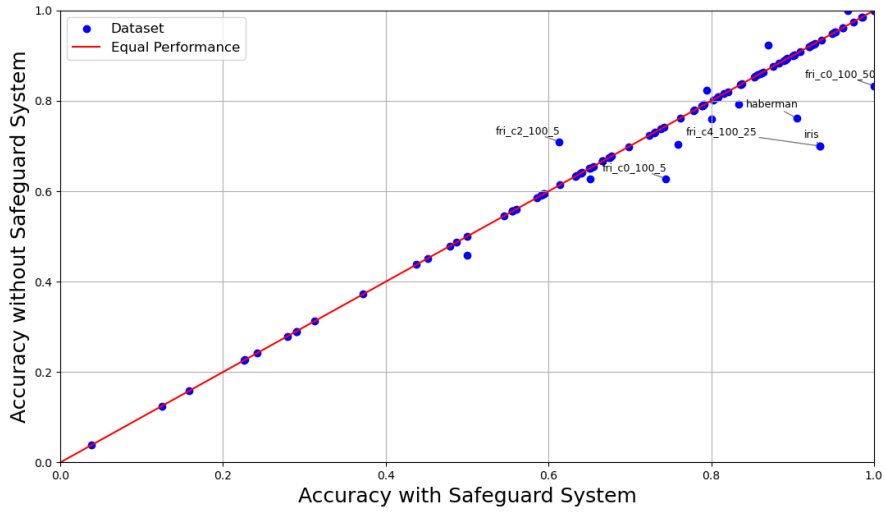

Fig. 6.23.: scikit-learn: Performance Comparison of Safeguard vs Non-safeguard Models (Clustering)

The only example where this is not the case (Figure 6.27) shows the limitation of purely model-based meta-features in certain cases. However, changing to using `bootstrap=False` as an option, the performance is better with the safeguard system (Figure 6.28).

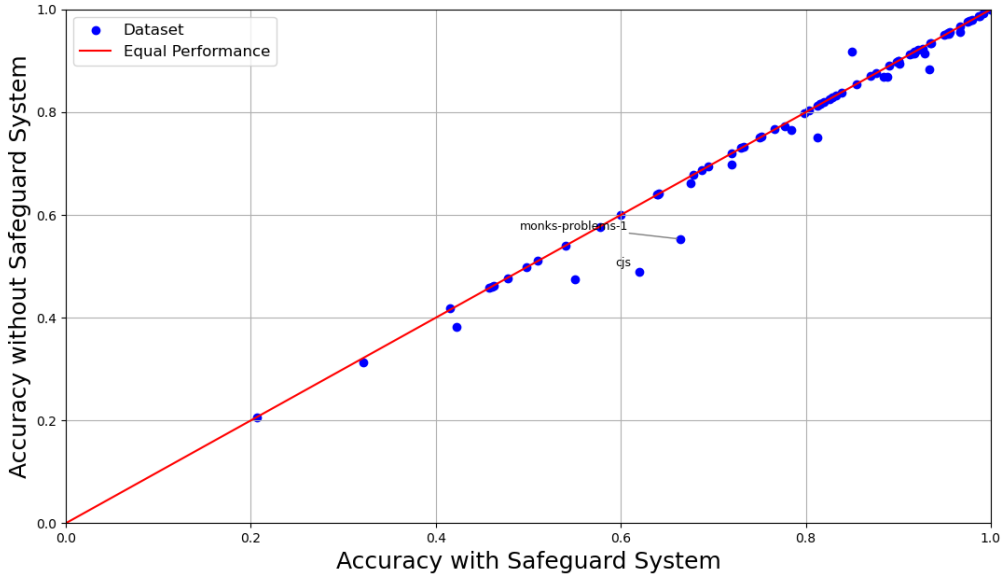

Fig. 6.24.: scikit-learn: Performance Comparison of Safeguard vs Non-safeguard Models (Model-Based + Clustering): Run on OpenML100

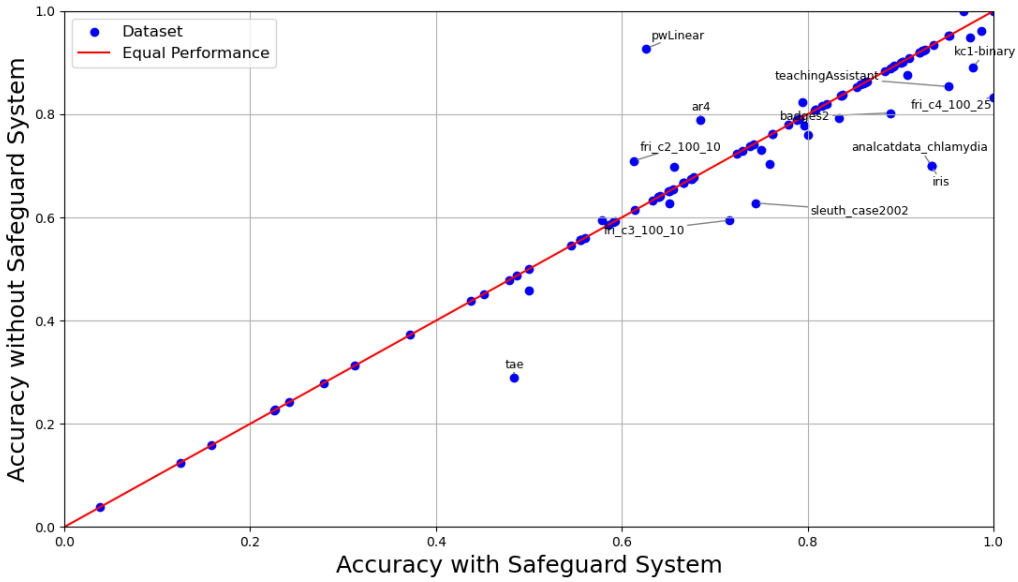

Fig. 6.25.: scikit-learn: Performance Comparison of Safeguard vs Non-safeguard Models (Model-Based + Clustering): Run on CD3

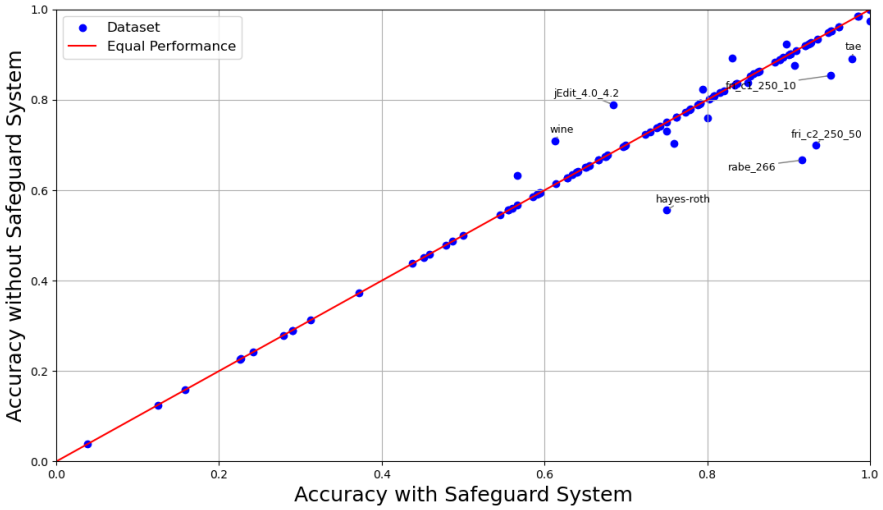

Fig. 6.26.: scikit-learn: Performance Comparison of Safeguard vs Non-safeguard Models (Landmarking)

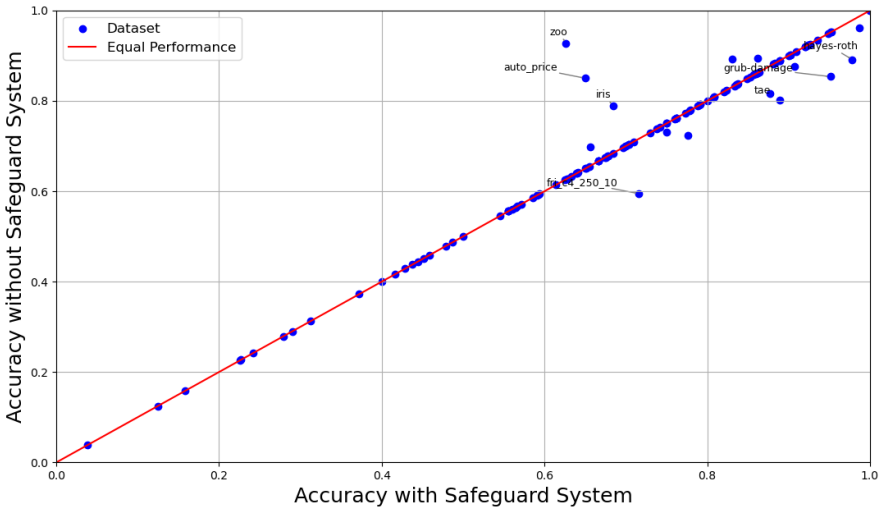

Fig. 6.27.: scikit-learn: Performance Comparison of Safeguard vs Non-safeguard Models (Model-Based) → Bad Performance

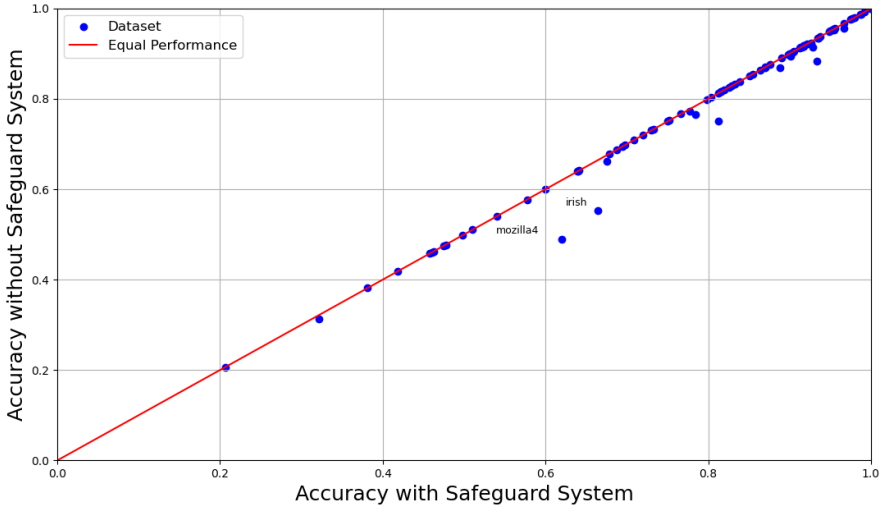

Fig. 6.28.: scikit-learn: Performance Comparison of Safeguard vs Non-safeguard Models (Model-Based) → Good Performance

We can also see the benefits of no bootstrapping in the following graphs (Figure 6.29 and Figure 6.30), where the classifier predicts more datasets correctly. Furthermore, we can see how important keeping model-based meta-features (Figure 6.30 vs Figure 6.31) is, as clustering and model-based lead to different dataset subsets being marked as negative correctly (blue circles). If we solely used clustering or model-based there would be quite a few more datasets which get overlooked for each.

We can also see how well clustering performs on both OpenML100 (Figure 6.32) and CD3 (Figure 6.33) compared to AutoGluon and in general.

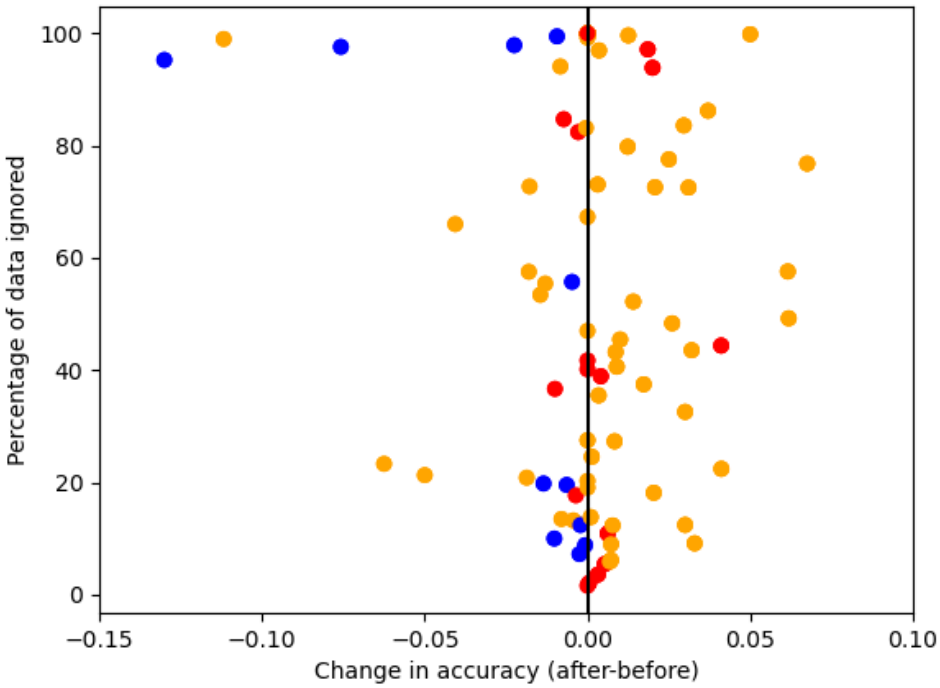

Fig. 6.29.: scikit-learn: Percentage of data ignored vs change in accuracy. Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

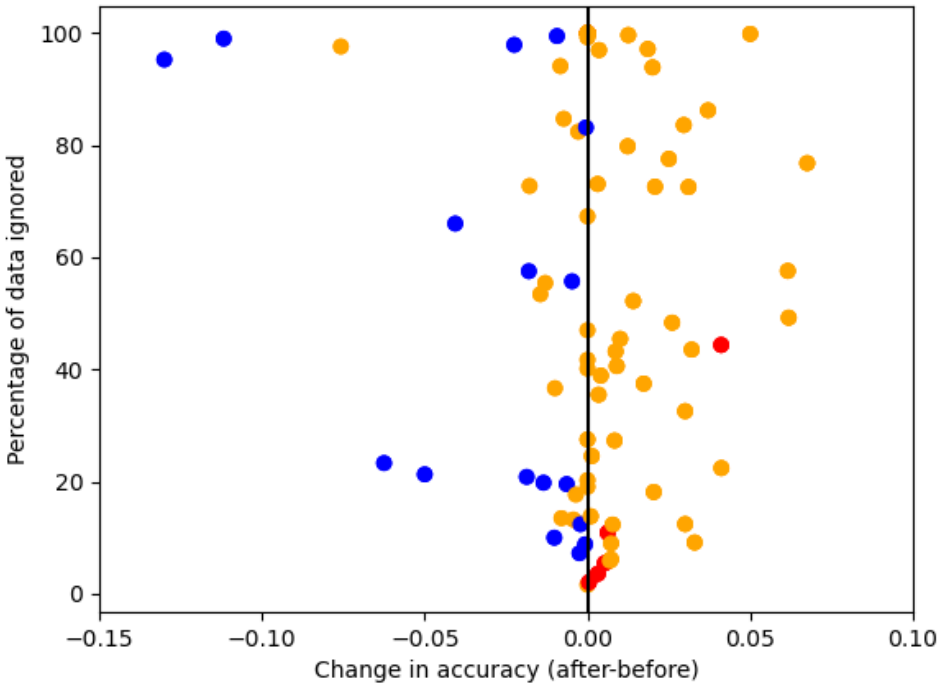

Fig. 6.30.: scikit-learn: Percentage of data ignored vs change in accuracy. Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

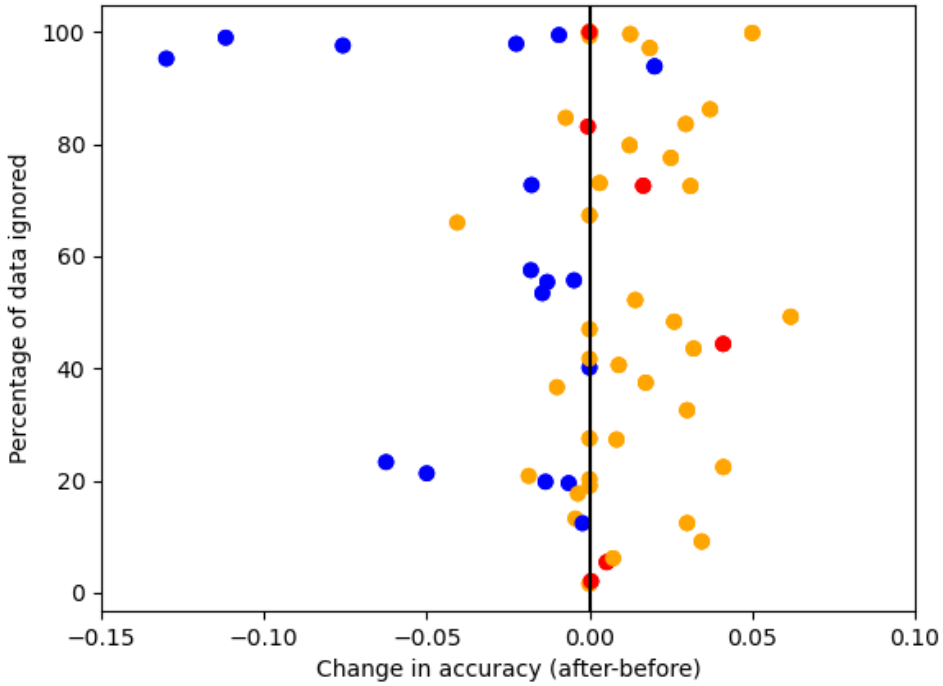

Fig. 6.31.: scikit-learn: Percentage of data ignored vs change in accuracy. Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

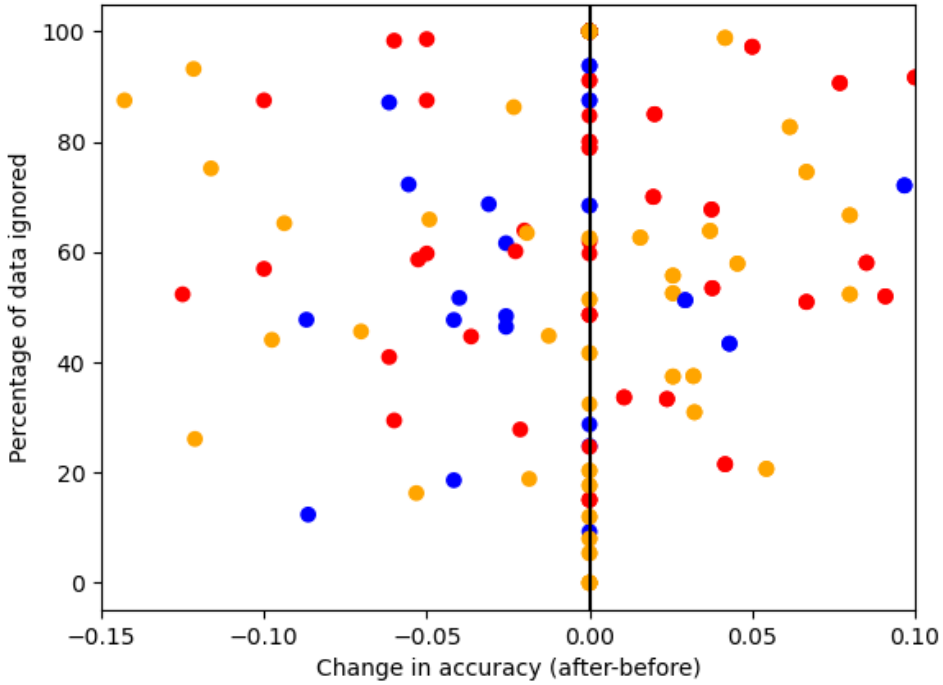

Fig. 6.32.: scikit-learn: Percentage of data ignored vs change in accuracy. Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

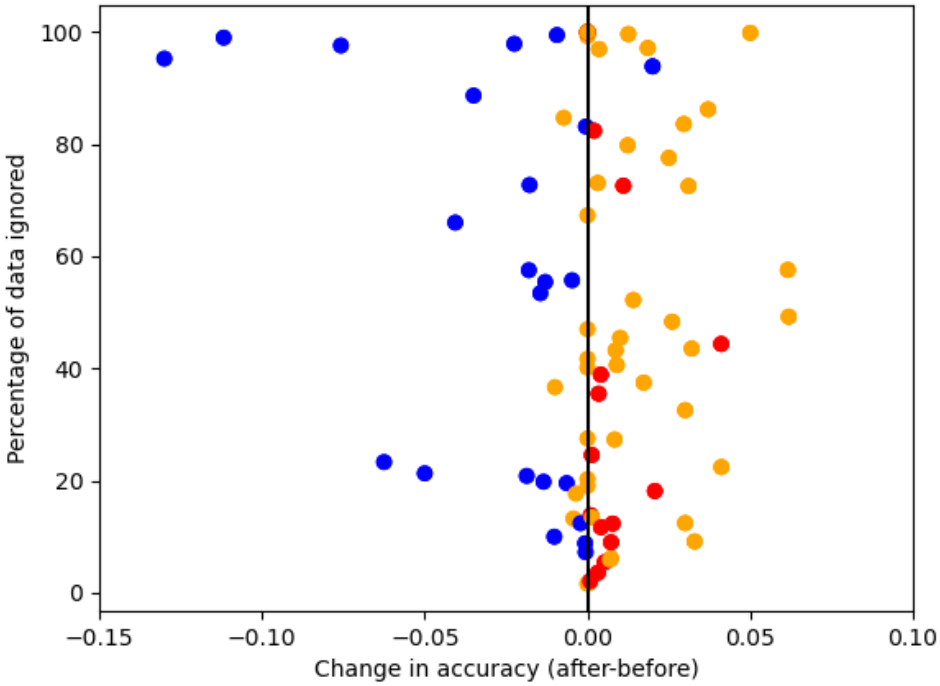

Fig. 6.33.: scikit-learn: Percentage of data ignored vs change in accuracy. Each dot represents a dataset. Orange is classifier predicting a positive change. Blue is negative change. Red is an error with the safeguard system.

In conclusion, we can see how well the scikit-learn based `RandomForestClassifier` performs. With only one minor tweak (setting `bootstrap=False`), the classifier performs well and is useful as a safeguard system. This is in contrast to the AutoGluon system, which provided worse predictive performance by choosing "plus" almost always (orange circles). Furthermore, the `RandomForestClassifier` was much easier to debug and tune, as it is very simple. We therefore selected the `RandomForestClassifier` over the AutoGluon based version.

6.4.2.1. SHAP and MDI Evaluation

In this section, we will look at the meta-features importance values as defined by their SHAP value and their MDI (mean decrease in impurity), as described in Section 5.4.

As can be seen in Figure 6.34 and Figure 6.35 there is a lot of variance around the mean for both model-based and landmarking based meta-features. This means that

their importance is inconsistent, which leads to less predictability. Furthermore, no feature (landmarking or model based) is very prominent.

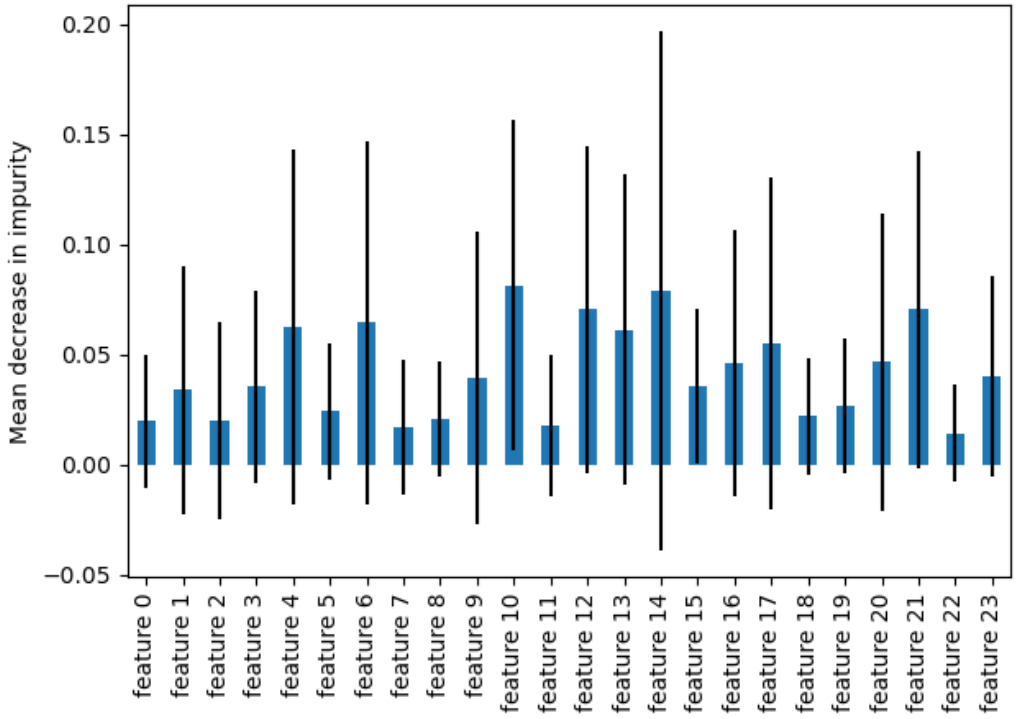

Fig. 6.34.: Feature Importances (Model-based) for OpenML100 using MDI

This is in comparison to clustering, as can be seen in Figure 6.36, where there is not a lot of variance. Feature 4 is the SC feature, as already defined in Section 6.3 and it is the only one which has no significance.

Looking now at the mean SHAP value graphs, we can again see that impact of each individual feature is very low for model-based (Figure 6.37).

This is somewhat different for landmarking based features (Figure 6.38), where we can see that `elite_nn.sd` and `one_nn.mean` are more significant features in terms of their mean SHAP values.

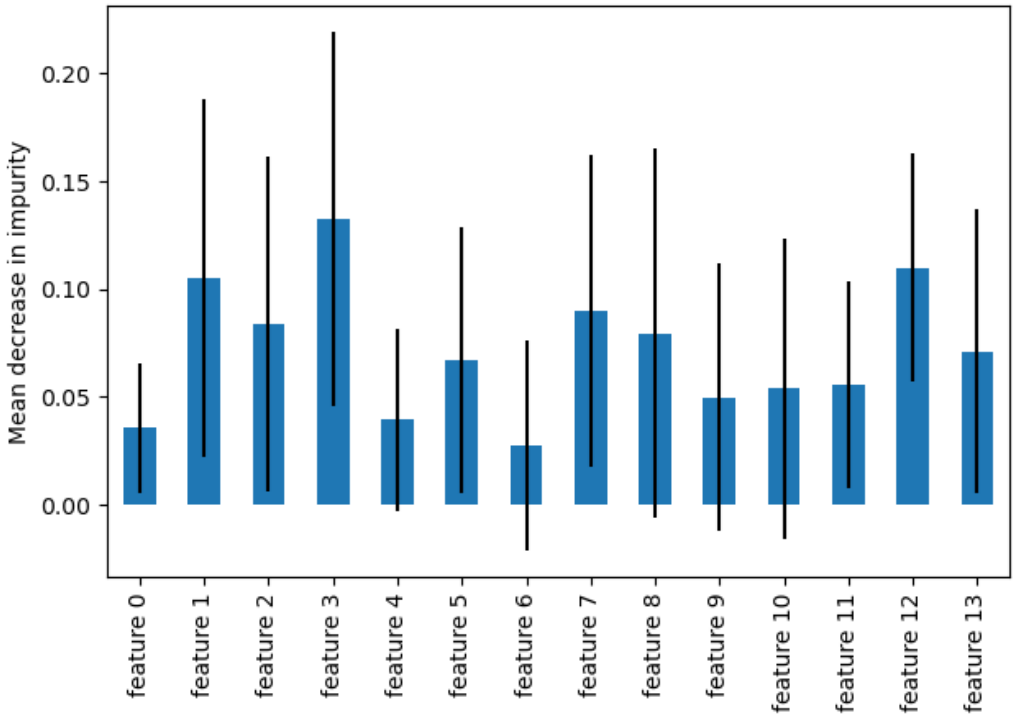

Fig. 6.35.: Feature Importances (Landmarking) for OpenML100 using MDI

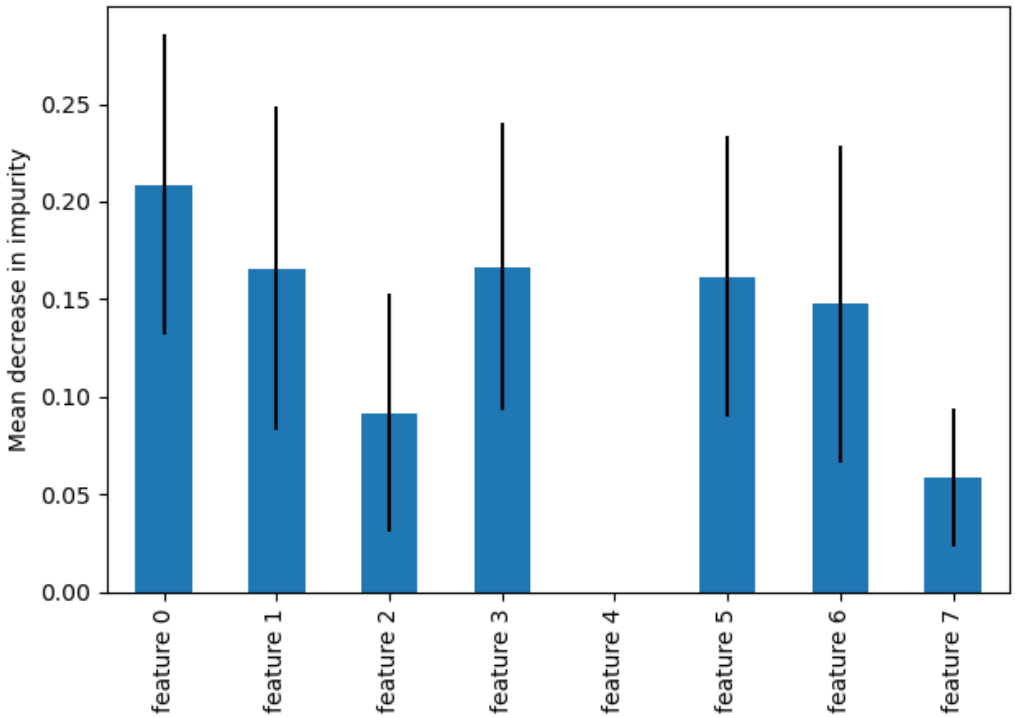

Fig. 6.36.: Feature Importances (Clustering) for OpenML100 using MDI

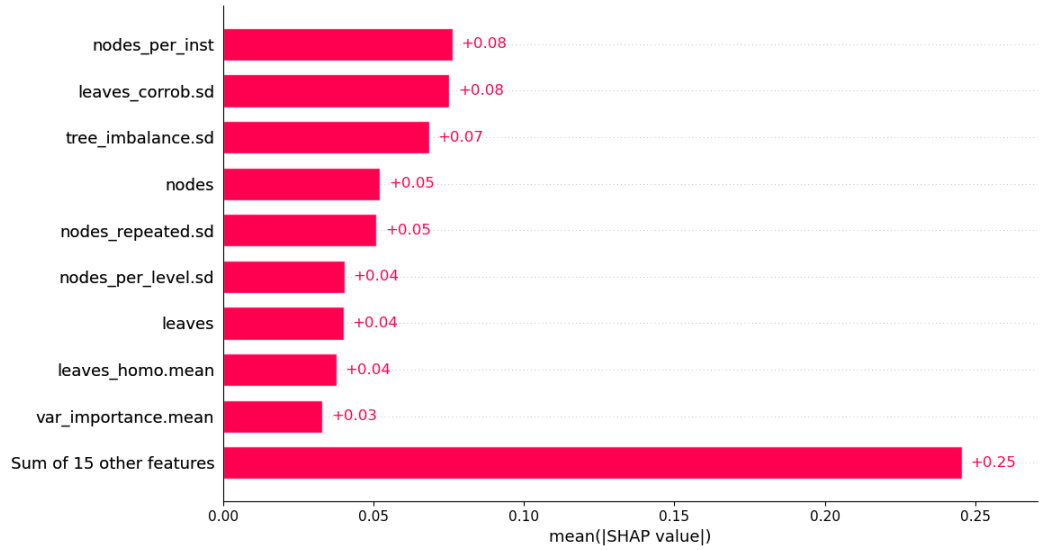

Fig. 6.37.: Mean SHAP Values (Model-Based)

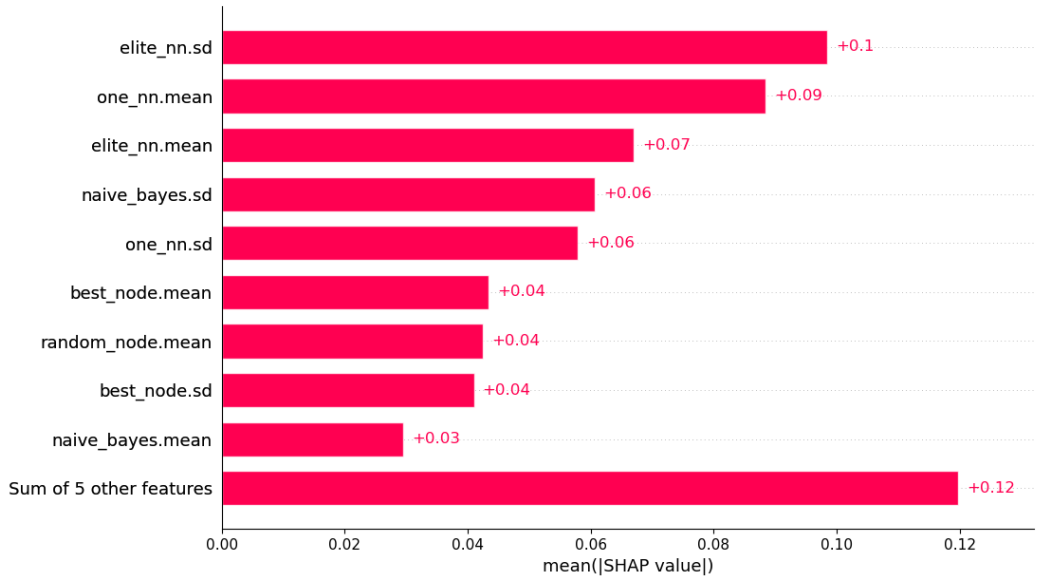

Fig. 6.38.: Mean SHAP Values (Landmarking)

Finally we see that the clustering based features (Figure 6.39) have around the same mean SHAP values, other than `sc` (see above).

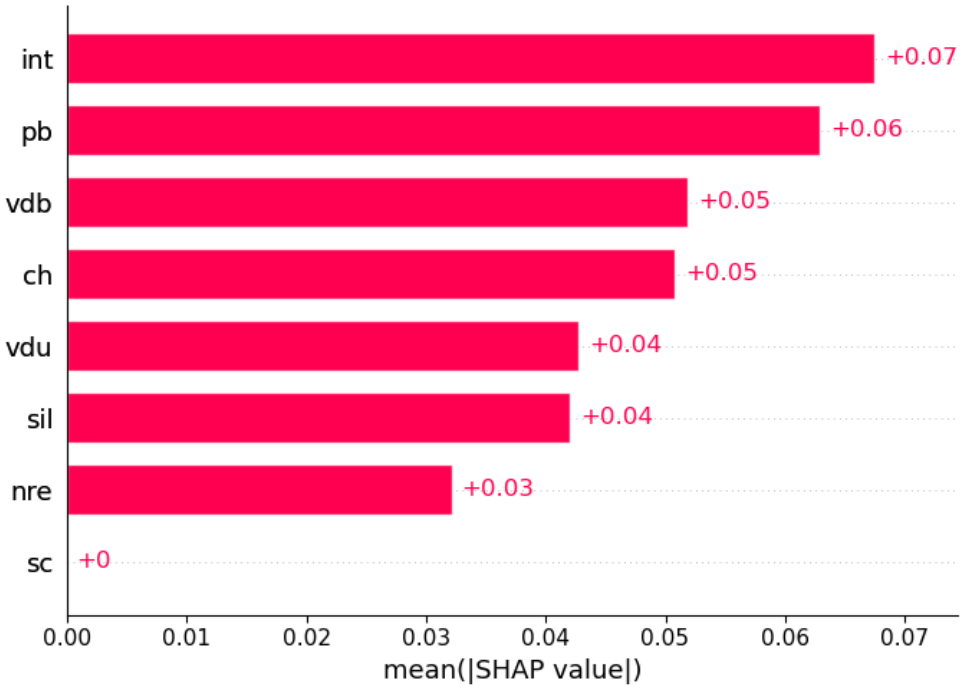

Fig. 6.39.: Mean SHAP Values (Clustering)

The final graphs (beeswarm) we examine are: Figure 6.40, Figure 6.41 and Figure 6.42, which show the actual SHAP values and their impact on model output.

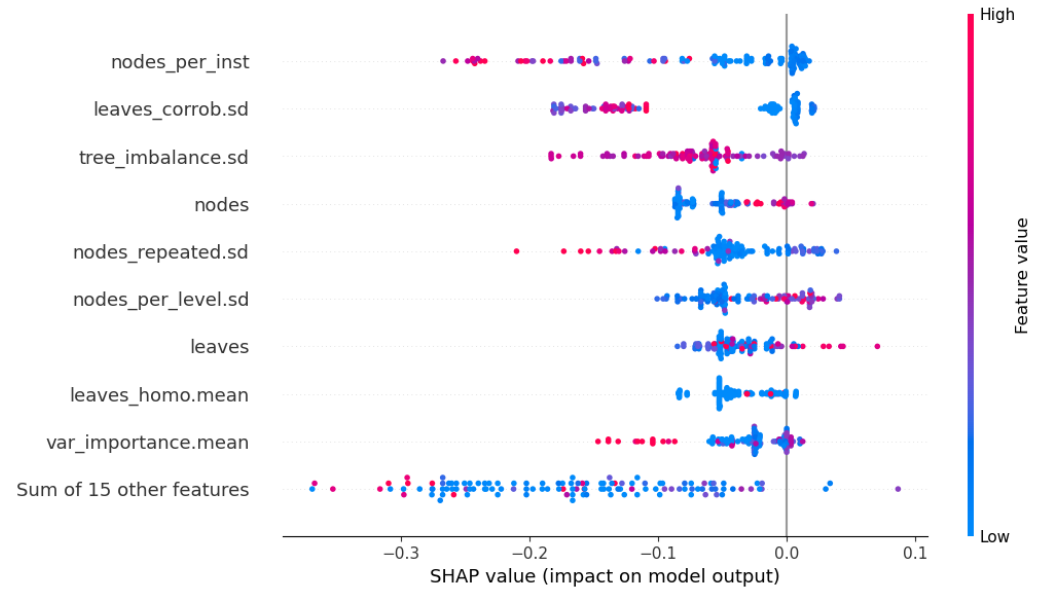

Fig. 6.40.: SHAP Values (Model-Based)

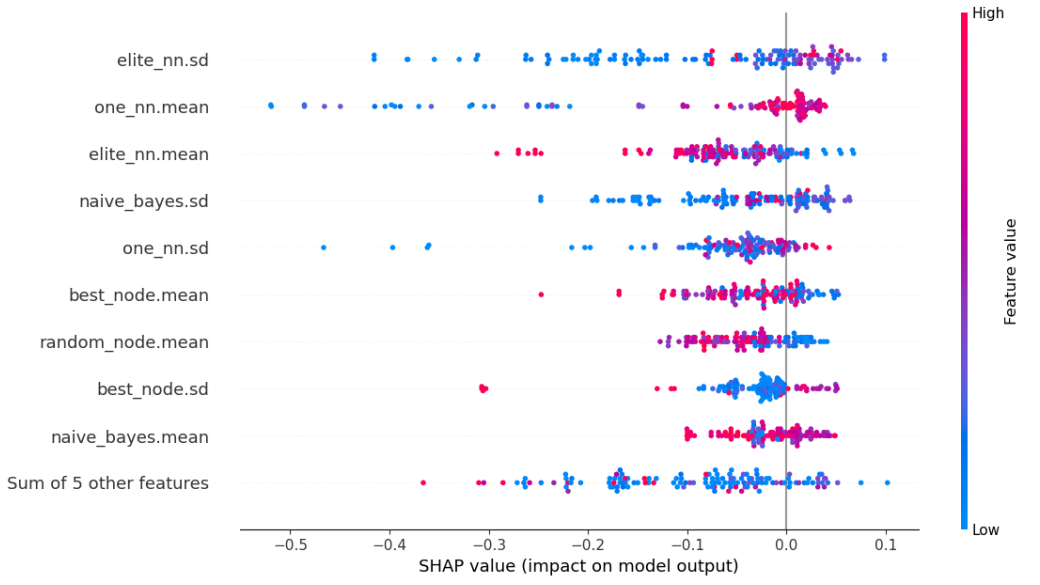

Fig. 6.41.: SHAP Values (Landmarking)

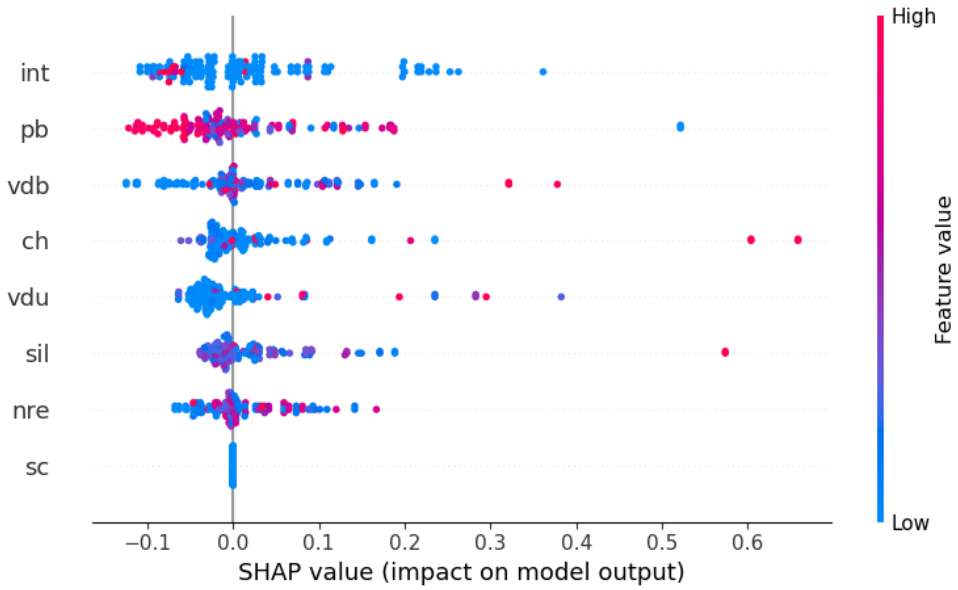

Fig. 6.42.: SHAP Values (Clustering)

This includes a spectrum of blue and red, whereby the color is the "original value of a feature" [@Lunb]. This means for each data point the original value of each feature, e.g., for sc, the value is always zero in the above graphs (so all blue). As we can see, landmarking (Figure 6.41) has a lot of features with large negative and spread out SHAP values. This is in contrast to model-based (Figure 6.40) and clustering (Figure 6.42), where there are not as many spread out values.

In conclusion, using the information provided by the SHAP library and the MDI, we can deduce the model-based and clustering meta-features provide very good and predictable information, and should therefore be used.

6.4.3 Conclusion

We can see that the safeguard system provides a net benefit for most applications. It enables savings in compute time by eliminating the need for computing M2, with the `RandomForestClassifier` introducing very little overhead. We have found which settings affect the performance of M2, mainly the threshold and quality parameters. Furthermore, we have found that our approach compares favorably to AutoGluon's built-in semi-supervised learning support.

Another aspect we have explored is how to predict dataset performance using meta-features and which ones perform the best for our safeguard system. Our results show that model-based and clustering meta-features are the best for our approach. Finally, we have explored the models and settings which impact our safeguard system, leading to the selection of the better performing `RandomForestClassifier`.

Overall, our results demonstrate that better performance can be achieved by utilizing unlabeled data, which has been labeled by a separately trained model.

7 Conclusion

We have shown how semi-supervised models can have better performance when trained on small amounts of labeled data. Through our novel approach, known as linear-ensembling, we are able to provide better performance overall, compared to both a supervised learning and AutoGluon's semi-supervised approach, despite not achieving 100% accuracy.

We have investigated which factors could lead to poor performance at the end of a learning process, such as the threshold or AutoGluon's quality configurations and the meta-features of the dataset to be trained on. We have identified clustering and model-based meta-features as particularly meaningful in predicting how a dataset will perform. To take advantage of this information, we have introduced our safeguard system. This system is able to leverage the improved accuracy from our linear-ensembling approach, while reducing computational resources and run times by eliminating unnecessary model training.

Furthermore, our findings have shown that the issues with performance loss can be eliminated without time-consuming steps needing to be taken.

Finally, we provide valuable guidance for future research, by identifying which settings and meta-features are the most useful and provide the best results. This includes which factors affect the performance of the safeguard system itself, such as the choice between scikit-learn and AutoGluon. These insights offer a clear direction forward for further research with diverse datasets and applications.

7.1 Future Work

There are several directions for future work. A further improvement and generalization of the safeguard system can include training it on more datasets, tuning the various settings of the `RandomForestClassifier`, or possibly selecting a different classifier model. Another area of research could be further refining the selection

of the meta-features and increasing the robustness of the pymfe system to make it more useful for the safeguard system. A third direction concerns a more hybrid approach, where a human would verify/change some pseudo-labeled results, e.g., the lowest confidence ones, to improve the performance of M2. The fourth direction is expanding our work to non-classifier based labeling systems with AutoML.

Appendix A

A.1 Custom Datasets

These are the list of all datasets used from [FSE18] as lists of `openml.org` dataset ids. They are divided into 3 dataset groups: CD1 (Custom Data-group 1) (Listing A.1), CD2 (Listing A.2) and CD3 (Listing A.3).

```
[3, 6, 12, 14, 16, 18, 20, 21, 22, 26, 28, 30, 32, 36, 44, 46, 60,
151, 155, 161, 162, 180, 182,
183, 184, 197, 209, 279, 287, 294, 300, 312, 375, 389, 391,
395, 398, 720, 958, 959, 962, 971, 976, 978, 979, 980,
991, 995, 1019, 1020, 1021, 1022, 1036,
1038, 1040, 1041, 1043, 1044, 1046, 1050, 1067, 1116, 1120,
1169, 1217, 1236, 1237, 1238, 1457,
1459, 1460, 1471, 1475, 1481, 1483, 1486, 1487, 1489, 1496,
1501, 1503, 1505, 1507, 1509, 1527,
1528, 1529, 1531, 1533, 1534, 1535, 1537, 1538, 1539, 1540,
1557, 1568, 4134, 4135,
4534, 4538, 40474, 40475, 40476, 40477, 40478]
```

Listing A.1: CD1

```
[11, 15, 23, 29, 31, 37, 50, 54, 181, 223, 292, 307,
313, 333, 334, 335, 377, 383, 385,
386, 392, 394, 400, 401, 458, 469, 478,
679, 715, 717, 718, 723, 740, 741, 742, 743,
749, 750, 751, 766, 770, 774, 779, 792,
795, 797, 799, 805, 806, 813, 824, 825, 826,
827, 837, 838, 841, 845, 849, 853, 855,
866, 869, 870, 872, 879, 884, 886, 888, 896,
903, 904, 910, 912, 913, 917, 920, 926,
931, 934, 936, 937, 943, 949, 954, 970, 983,
987, 994, 997, 1004, 1014, 1016, 1049,
1063, 1137, 1145, 1158, 1165, 1443, 1444, 1451,
1453, 1454, 1464, 1467, 1472, 1510, 1542,
1543, 1545, 1546, 3560, 3904,
3917]
```

Listing A.2: CD2

```
[8, 10, 39, 40, 41, 43, 48, 49, 53, 59, 61, 62, 164,
187, 285, 329, 336, 337, 338, 384,
387, 388, 397, 444, 446, 448, 461, 463,
464, 475, 685, 694, 714, 716, 719, 721, 724,
726, 730, 732, 733, 736, 744, 745, 746,
747, 748, 753, 754, 756, 762, 763, 768, 769,
771, 773, 775, 776, 778, 782, 783, 784,
788, 789, 793, 794, 796, 801, 808, 811, 812,
814, 818, 820, 828, 829, 830, 832, 834,
850, 851, 860, 863, 865, 867, 868, 873, 875,
876, 877, 878, 880, 885, 889, 890, 895,
900, 902, 906, 907, 908, 909, 911, 915, 916,
918, 921, 922, 924, 925, 932, 933, 935,
941, 952, 955, 956, 965, 969, 973, 974, 996,
1005, 1006, 1011, 1012, 1013, 1025, 1026,
1045, 1048, 1054, 1059, 1061, 1064, 1065,
1066, 1071, 1073, 1075, 1077, 1078, 1080,
1084, 1100, 1106, 1115, 1121, 1122, 1123,
1124, 1125, 1126, 1127, 1129, 1131, 1132,
1133, 1135, 1136, 1140, 1141, 1143, 1144,
1147, 1148, 1149, 1150, 1151, 1152, 1153,
1154, 1155, 1156, 1157, 1159, 1160, 1162,
1163, 1164, 1167, 1412, 1413, 1441, 1442,
1446, 1447, 1448, 1449, 1450, 1455, 1473,
1482, 1488, 1498, 1500, 1508, 1512, 1513,
1519, 1520, 1556, 1565, 1600, 3902, 3913,
4153, 4340]
```

Listing A.3: CD3

A.2 Safeguard System Training Procedure

Using the output from previous runs of a dataset group, CSV files are generated which contain the meta features of each dataset (see Section 2.5) and `plus` or `minus` indicating whether M2 performed better than M1 according to the "accuracy" performance measurement. These CSV files are then used to train the safeguard classifier, using a variety of performance metrics, as described in Section 5.3. This

classifier can then be used for new datasets that it has not seen previously and provide guidance whether M2 will perform better and is worth training.

Bibliography

[Alc+20] Edesio Alcobaça, Felipe Siqueira, Adriano Rivolli, et al. "MFE: Towards reproducible meta-feature extraction". In: *Journal of Machine Learning Research* 21.111 (2020), pp. 1–5 (cit. on pp. 6, 7, 35).

[And10] Răzvan Andonie. "Extreme data mining: Inference from small datasets". In: *International Journal of Computers, Communication & Control* (2010) (cit. on p. 3).

[Ara+20] Eric Arazo, Diego Ortego, Paul Albert, Noel E. O'Connor, and Kevin McGuinness. "Pseudo-Labeling and Confirmation Bias in Deep Semi-Supervised Learning". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. 2020, pp. 1–8 (cit. on p. 15).

[Ast+15] Marco Aste, Massimo Boninsegna, Antonino Freno, and Edmondo Trentin. "Techniques for dealing with incomplete data: a tutorial and survey". In: *Pattern Analysis and Applications* 18 (2015), pp. 1–29 (cit. on p. 2).

[Bar89] Horace B Barlow. "Unsupervised learning". In: *Neural computation* 1.3 (1989), pp. 295–311 (cit. on p. 1).

[BK96] Barry Becker and Ronny Kohavi. *Adult*. UCI Machine Learning Repository. 1996 (cit. on p. 2).

[Bis+21] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, et al. *OpenML Benchmarking Suites*. 2021. arXiv: `1708.03731 [stat.ML]` (cit. on p. 18).

[Car19] M Carlisle. "A Boston housing dataset controversy". In: `https://medium.com/@docintangible/racist-data-destruction-113e3eff54a8`. 2019 (cit. on p. 2).

[CR18] Alexandra Chouldechova and Aaron Roth. "The frontiers of fairness in machine learning". In: *arXiv preprint arXiv:1810.08810* (2018) (cit. on p. 1).

[Dat] Datagen. "Data-Centric AI: The New Frontier". In: `https://datagen.tech/guides/data-training/data-centric-ai/` (cit. on p. 3).

[Del96] Delve. "The Boston Housing Dataset". In: `https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html`. 1996 (cit. on p. 2).

[Din+21] Frances Ding, Moritz Hardt, John Miller, and Ludwig Schmidt. "Retiring adult: New datasets for fair machine learning". In: *Advances in neural information processing systems* 34 (2021), pp. 6478–6490 (cit. on p. 2).

[EH21] Jesper E. van Engelen and Holger H. Hoos. “Semi-supervised Co-ensembling for AutoML”. In: *Trustworthy AI - Integrating Learning, Optimization and Reasoning*. Ed. by Fredrik Heintz, Michela Milano, and Barry O’Sullivan. Cham: Springer International Publishing, 2021, pp. 229–250 (cit. on pp. 2, 9, 10, 13).

[Eri+20] Nick Erickson, Jonas Mueller, Alexander Shirkov, et al. “AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data”. In: *arXiv preprint arXiv:2003.06505* (2020) (cit. on pp. 5, 6).

[Feu+15] Matthias Feurer, Aaron Klein, Katharina Eggensperger, et al. “Efficient and Robust Automated Machine Learning”. In: *Advances in Neural Information Processing Systems 28 (2015)*. 2015, pp. 2962–2970 (cit. on pp. 2, 5, 9).

[FSE18] Nicolo Fusi, Rishit Sheth, and Melih Elibol. “Probabilistic Matrix Factorization for Automated Machine Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, et al. Vol. 31. Curran Associates, Inc., 2018 (cit. on pp. 18, 65).

[Gij+23] Pieter Gijsbers, Marcos L. P. Bueno, Stefan Coors, et al. *AMLB: an AutoML Benchmark*. 2023. arXiv: `2207.12560 [cs.LG]` (cit. on p. 6).

[HNP09] Alon Halevy, Peter Norvig, and Fernando Pereira. “The unreasonable effectiveness of data”. In: *IEEE intelligent systems* 24.2 (2009), pp. 8–12 (cit. on p. 1).

[HR78] David Harrison Jr and Daniel L Rubinfeld. “Hedonic housing prices and the demand for clean air”. In: *Journal of environmental economics and management* 5.1 (1978), pp. 81–102 (cit. on p. 2).

[He+19] Junxian He, Jiatao Gu, Jiajun Shen, and Marc’Aurelio Ranzato. “Revisiting self-training for neural sequence generation”. In: *arXiv preprint arXiv:1909.13788* (2019) (cit. on pp. 1, 3).

[Kot+21] Miloš Kotlar, Marija Punt, Z. Radivojević, M. Cvetanović, and V. Milutinovic. “Novel Meta-Features for Automated Machine Learning Model Selection in Anomaly Detection”. In: *IEEE Access* 9 (2021), pp. 89675–89687 (cit. on p. 7).

[Lat+17] Masitah Abdul Lateh, Azah Kamilah Muda, Zeratul Izzah Mohd Yusof, Noor Azilah Muda, and Mohd Sanusi Azmi. “Handling a small dataset problem in prediction model by employ artificial data generation approach: A review”. In: *Journal of Physics: Conference Series*. Vol. 892. 1. IOP Publishing. 2017, p. 012016 (cit. on p. 3).

[LeC98] Yann LeCun. “The MNIST database of handwritten digits”. In: *http://yann. lecun. com/exdb/mnist/* (1998) (cit. on p. 1).

[Lee+13] Dong-Hyun Lee et al. “Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks”. In: *Workshop on challenges in representation learning, ICML*. Vol. 3. 2. Atlanta. 2013, p. 896 (cit. on p. 6).

[Li+19] Yu-Feng Li, Hai Wang, Tong Wei, and Wei-Wei Tu. “Towards automated semi-supervised learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 4237–4244 (cit. on pp. 9, 10).

[LL17] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, et al. Curran Associates, Inc., 2017, pp. 4765–4774 (cit. on pp. 22, 23).

[Maa+11] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, et al. "Learning Word Vectors for Sentiment Analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150 (cit. on p. 1).

[MWH18] Felix Mohr, Marcel Wever, and Eyke Hüllermeier. "ML-Plan: Automated machine learning via hierarchical planning". In: *Machine Learning* 107.8-10 (2018), pp. 1495–1515 (cit. on p. 5).

[Mol20] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020 (cit. on p. 23).

[Mur12] Kevin P Murphy. *Machine learning: a probabilistic perspective*. 1st ed. MIT press, 2012 (cit. on pp. 1, 2).

[Ped+11] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 5, 6).

[PD19] Bruno Almeida Pimentel and Andre CPLF De Carvalho. "A new data characterization for selecting clustering algorithms using meta-learning". In: *Information Sciences* 477 (2019), pp. 203–219 (cit. on p. 7).

[Pla98] John Platt. "Using analytic QP and sparseness to speed training of support vector machines". In: *Advances in neural information processing systems* 11 (1998) (cit. on p. 1).

[Riv+18] Adriano Rivolli, Luis PF Garcia, Carlos Soares, Joaquin Vanschoren, and André CPLF de Carvalho. "Towards reproducible empirical research in meta-learning". In: *arXiv preprint arXiv:1808.10406* (2018), pp. 32–52 (cit. on p. 7).

[Sch+22] Reva Schwartz, Apostol Vassilev, Kristen Greene, et al. "Towards a standard for identifying and managing bias in artificial intelligence". In: *NIST special publication* 1270.10.6028 (2022) (cit. on pp. 1, 10).

[Scu65] Henry Scudder. "Probability of error of some adaptive pattern-recognition machines". In: *IEEE Transactions on Information Theory* 11.3 (1965), pp. 363–371 (cit. on p. 1).

[SLH22] Seonguk Seo, Joon-Young Lee, and Bohyung Han. "Unsupervised learning of debiased representations with pseudo-attributes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 16742–16751 (cit. on p. 1).

[TGS10] Nguyen Thai-Nghe, Zeno Gantner, and Lars Schmidt-Thieme. "Cost-sensitive learning methods for imbalanced data". In: *The 2010 International joint conference on neural networks (IJCNN)*. IEEE. 2010, pp. 1–8 (cit. on p. 20).

[Tho+13] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms". In: *Proc. of KDD-2013*. 2013, pp. 847–855 (cit. on p. 5).

[TE11] Antonio Torralba and Alexei A Efros. "Unbiased look at dataset bias". In: *CVPR 2011*. IEEE. 2011, pp. 1521–1528 (cit. on p. 1).

[VEJ21] Sahil Verma, Michael Ernst, and Rene Just. "Removing biased data to improve fairness and accuracy". In: *arXiv preprint arXiv:2102.03054* (2021) (cit. on pp. 2, 10).

[Wol96] David H Wolpert. "The lack of a priori distinctions between learning algorithms". In: *Neural computation* 8.7 (1996), pp. 1341–1390 (cit. on p. 2).

[WM97] David H Wolpert and William G Macready. "No free lunch theorems for optimization". In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82 (cit. on p. 2).

[Yao+18] Quanming Yao, Mengshuo Wang, Yuqiang Chen, et al. "Taking human out of learning applications: A survey on automated machine learning". In: *arXiv preprint arXiv:1810.13306* (2018) (cit. on pp. 1, 3).

[ZL18] Ying Zhang and Chen Ling. "A strategy to apply machine learning to small datasets in materials science". In: *Npj Computational Materials* 4.1 (2018), p. 25 (cit. on p. 3).

[Zho+17] Lina Zhou, Shimei Pan, Jianwu Wang, and Athanasios V Vasilakos. "Machine learning on big data: Opportunities and challenges". In: *Neurocomputing* 237 (2017), pp. 350–361 (cit. on pp. 1, 3).

[ZG09] Xiaojin Zhu and Andrew B Goldberg. "Introduction to Semi-Supervised Learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 3.1 (2009), pp. 1–130 (cit. on p. 3).

Webpages

[@Aut] Autogluon authors. *predictor.py*. accessed on 26-07-2024. URL: `https://github.com/autogluon/autogluon/blob/0e3bc0e54ab4b0edf865c8b99e1418472006d6b7/tabular/src/autogluon/tabular/predictor/predictor.py` (cit. on p. 10).

[@Coo] Aidan Cooper. *Explaining Machine Learning Models: A Non-Technical Guide to Interpreting SHAP Analyses*. URL: `https://www.aidancooper.co.uk/a-non-technical-guide-to-interpreting-shap-analyses/` (cit. on p. 23).

[@deva] scikit-learn developers. *f1_score*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html` (cit. on pp. 19, 20).

[@devb] scikit-learn developers. *fbeta_score*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.fbeta_score.html (cit. on pp. 19, 20).

[@devc] scikit-learn developers. *precision_score*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html (cit. on p. 19).

[@devd] scikit-learn developers. *RandomForestClassifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (cit. on p. 47).

[@deve] scikit-learn developers. *recall_score*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html (cit. on p. 19).

[@Ede] Felipe Siqueira Edesio Alcobaça. *Meta-feature Description Table*. accessed on 26-07-2024. URL: https://pymfe.readthedocs.io/en/latest/auto_pages/meta_features_description.html (cit. on pp. 7, 35).

[@Eri+24] Nick Erickson, Jonas Mueller, Alexander Shirkov, et al. *autogluon.tabular.TabularPredictor.fit*. 2024. URL: https://auto.gluon.ai/stable/api/autogluon.tabular.TabularPredictor.fit.html (cit. on p. 18).

[@Luna] Scott Lundberg. *bar plot*. URL: https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/plots/bar.html (cit. on p. 23).

[@Lunb] Scott Lundberg. *beeswarm plot*. accessed on 30-07-2024. URL: https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/plots/beeswarm.html (cit. on p. 60).

[@Lunc] Scott Lundberg. *Welcome to the SHAP documentation*. accessed on 02-08-2024. URL: https://shap.readthedocs.io/en/latest/index.html (cit. on p. 23).

[@Rec] Leibniz Rechenzentrum. *Available SLURM clusters and features*. accessed on 26-07-2024. URL: https://doku.lrz.de/available-slurm-clusters-and-features-11483939.html (cit. on p. 17).

[@Resa] Google Researchers. *Classification: Accuracy*. URL: https://developers.google.com/machine-learning/crash-course/classification/accuracy (cit. on p. 19).

[@Resb] Google Researchers. *Classification: Precision and Recall*. URL: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall (cit. on p. 19).

[@sci] scikit-learn developers. *Feature importances with a forest of trees*. URL: https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html (cit. on p. 22).

[@SI20] Seagate and IDC. *Rethink Data*. 2020. URL: https://www.seagate.com/files/www-content/our-story/rethink-data/files/Rethink_Data_Report_2020.pdf (cit. on p. v).

[@Sha] Adam Shafi. *Random Forest Classification with Scikit-Learn*. URL: https://www.datacamp.com/tutorial/random-forests-classifier-python (cit. on p. 6).

List of Figures

List of Tables

Glossary

auto-sklearn . pp. 5, 6, 9, 10

Auto-WEKA . p. 5

AutoGluon . pp. 5, 13, 16–18, 27, 32, 40–43, 46–48, 51, 54, 60, 63, 76, 77, 79

AutoML . pp. 2, 5, 6

CD1 . pp. 18, 33, 36–40, 65, 76

CD2 . pp. 18, 33, 36–40, 65, 76

CD3 . pp. 18, 28–30, 33, 35–39, 43, 49, 51, 65, 75–77, 79

ML-Plan . p. 5

OpenML-CC18 . pp. 18, 33, 36–40, 43, 76

OpenML100 . pp. 18, 27–29, 31, 33, 35–40, 43, 49, 51, 55, 56, 75–79

pymfe . pp. 7, 18, 20, 22, 28, 32, 35, 64

scikit-learn . pp. 5, 6, 16, 18, 40, 41, 47–54, 63, 77

Colophon

This thesis was typeset with LaTeX 2_ε. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.