Evaluating and Comparing Heterogeneous Ensemble Methods for Unsupervised Anomaly Detection

1st Simon Klüttermann
Chair of Data Science and Data Engineering
TU Dortmund University
Dortmund, Germany
Simon.Kluettermann@cs.tu-dortmund.de

2nd Emmanuel Müller
Chair of Data Science and Data Engineering
TU Dortmund University
Dortmund, Germany
emmanuel.mueller@cs.tu-dortmund.de

***Abstract*—Ensembles are one of the most promising research directions for unsupervised anomaly detection. But combining many different models into such an ensemble requires good combination procedures that are able to combine the strengths of many different submodels. To find, evaluate and understand these procedures, we create the biggest experiment to date, including multiple orders of magnitude more ensembles than each of our competitors.**

Using this high number of comparisons, we also study the effect different normalization methods have on the combination procedure and extract conditional performances of individual models. We use this, to develop a simple set of best practices to create good and reliable anomaly detection ensembles.

***Index Terms*—anomaly detection, ensembles, unsupervised machine learning**

I. Introduction

Anomaly detection, defined as finding strange or unusual samples, has long been an important data analysis task with many applications, ranging from fraud detection to healthcare [1]–[3].

More recently, unsupervised anomaly detection has become its most interesting subfield. There we want to find abnormal samples without knowledge about any examples of how they might look like. This allows applying anomaly detection to applications where labeling recorded samples is costly, where unknown types of anomalies might appear, or where anomalies have not been recorded. And since anomalies are by definition rare, studying and improving unsupervised anomaly detection is required for applying machine learning to many more datasets.

But this also complicates the task, as we can not assume much about the anomalies we want to find. We effectively have to learn how to detect almost every type of anomaly, compared to separating two sets of limited size in the supervised case.

Still many algorithms exist which try to find outliers by employing a plentitude of various approaches. Some algorithms try to model the distribution of known data samples [4], [5], other methods isolate anomalies [6], [7] and even other methods use statistical properties to indicate anomalousness [8], [9].

For each method, there are anomalies, that are easy to find. And thus there tends to be an optimal algorithm for a given anomaly detection task, dataset, and the associated type of anomalies.

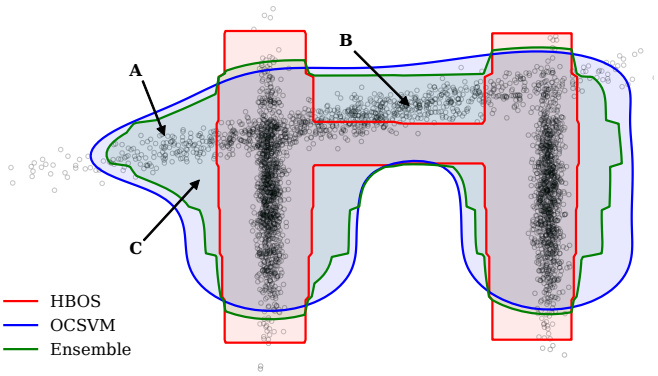

Fig. 1. Regions shown which are considered normal by two algorithms, *OCSVM* and *HBOS*, as well as of an ensemble of both. This ensemble considers the average score of both models and can create a more accurate representation of the underlying distribution: While *HBOS* fails to capture the distribution at point A and clearly makes a mistake at point B, it also captures the vertical parts of the distribution much better (see point C). The ensemble handles points A and B right and captures the distribution at C better than the OCSVM. Still, the combination is not perfect; finding the right function to build an ensemble is a complicated task.

This is a problem, as evaluating which method works best, is a supervised task and thus requires knowledge about the anomalies we want to find. As this is by definition not given for unsupervised anomaly detection, and so choosing the best anomaly detection algorithm is usually impossible.

Instead, applications rely on generalized results. But recent surveys struggle to find algorithms that are significantly different than others [10].

Ensembles can provide a solution to this. While for supervised anomaly detection, ensembling techniques allow combining many methods into a better one, for unsupervised tasks this is not guaranteed and their main use is to assure that a reasonable performance to each type of anomaly [11] is achieved. This is because of the unsupervised nature of our task. We are not able to test how well an individual method works, and so, when we are given many models, we are not able to weigh their influences based on their performance. Instead unsupervised ensembles have to rely on statistical effects: When one model mistakenly considers a sample as

too anomalous, it is less likely that all models make the same mistake, and so the error gets averaged out, increasing the reliability of our detection.

Unsupervised ensembles can be separated into homogeneous and heterogeneous ensembles [12]. Homogeneous ensembles [6], [13], [14] use very similar individual models, resulting in easily combinable submodels, but limiting how different individual models can be, and thus how much they can benefit from ensembling. Heterogeneous ensembles, like those we study in this paper, use different known models instead and profit from the high number of anomaly detection concepts suggested by the literature. But combining highly different methods also makes normalizing and aggregating their results quite challenging.

Regarding this limitation, many functions that combine the outputs of different anomaly detection algorithms have been proposed. Simple approaches like the average of their outputs can be effective, but recently also more advanced ideas, for example based on psychology [15], have been suggested.

Still, an strong unbiased comparison of these methods is missing. To help with this, we create the biggest known comparison to date, evaluating more than 6 billion ensembles, which to the best of our knowledge is multiple orders of magnitude more than every other paper before.

We are also the first to study the effect different normalization functions have on the resulting combination and we use the high number of ensembles we study, to also extract information about the best submodels to use.

By doing so, we provide a complete guide for creating effective unsupervised anomaly detection ensembles.

II. Related Work

A. Anomaly Detection

Anomaly detection, the task of finding abnormal or strange samples, has no singular perfect solution. Instead, whenever any method behaves unusually, it can be seen as a sign of an anomaly and thus be extracted into an anomaly detection algorithm. This means that there are many different methods following many different approaches to finding anomalies. And because each one finds slightly different types of anomalies, it is hard to say if one algorithm is more effective than another [10], [16].

The existing methods can be broadly separated into those that use deep learning and those that don't. Shallow algorithms include those based on the distance to close samples, like *kNN* [17] and *lof* [18] or those modeling densities like a *kde* [5]. Other solutions include an *isolation forest* [6], which tries to separate isolated points, or *Abod* exploiting angular differences caused by separations. Deep learning allows learning more complicated relations, for example, to model densities through a *GAN* [4], to learn a better data representation through *DeepSVDD* [19] or to enable reconstruction-based algorithms, like an *autoencoder* [20].

For our comparison we want to use as many algorithms as possible, so in addition to those already mentioned, we also employ an *ocsvm* [21], a *variational autoencoder* [22], a *pca* [23] algorithm and train *Inne* [7], *Sod* [24], *HBos* [25], *Cblof* [26], *Loda* [27], *COF* [28], *Copod* [9], *Ecod* [29], *Lmdd* [30] algorithms.

B. Ensembles

The high number of different approaches to anomaly detection invites their combination into ensembles. In the best case, combining many different models can allow to average out errors, while aggregating different knowledge from multiple sources.

For supervised machine learning, this has long proven useful. Methods like bagging, boosting and stacking [31] are effective at using supervised evaluations to create more efficient algorithms.

But for unsupervised tasks, we are not able to test if a model is performing better or worse than another during training. This makes creating ensembles much harder, and certain methods, like for example boosting, almost impossible.

Combining many models is still useful, as long as errors are rare and we can benefit from many different approaches [12], [13]. This means common errors are even more rare, and thus mistakes of individual models cancel out. But compared to the supervised task, we no longer have any guarantee that our ensemble performs better than individual models, as we can not evaluate a combination of models directly. But since we could also not simply choose the best individual model, we consider an ensemble effective if it outperforms the average performance of its submodels: The average performance with an ensemble has to be higher than the average performance without it.

Achieving this invites creating advanced combination functions, to be able to extract the most amount of information from individual models. But this is a complicated task, especially for heterogenous ensembles, as individual models produce drastically differently distributed anomaly scores. We will briefly explain common approaches in Section III, before comparing them in the rest of this paper.

C. Comparisons

Other papers compare different combination functions for unsupervised anomaly detection.

Each of the papers that introduce the combination functions of the next Section also compares them to other methods [12], [15], [32]. Still, their results tend to contradict each other, limiting the trust we can have in their conclusions. Also, their evaluation is often limited to a low number of ensembles, further limiting how well we can rely on these comparisons. One exception to this is the paper introducing the *IRT* combination function [15], which evaluates combination functions on 10000 different ensembles and inspired our evaluation. Still even compared to this high number of ensembles, we are creating 10000 times more ensembles, while using more datasets and individual submodels.

Another notable evaluation is given by [11], which compares different methods in an unbiased way, but their limited evaluation makes this paper also less useful in our opinion.

To the best of our knowledge, we are also the first to systematically study the effect of different normalization functions, and the benefit choosing different submodels provides.

III. Combination functions

In this Section, we will briefly review different ensemble combination methods.

A. Simple functions

Reference [12] suggests using the average of model prediction, or the maximum model prediction. Reference [13] uses a median of the model predictions, while Reference [14] uses a quadratic mean.

In addition to these, we also test the minimum model prediction ($ens(x) = min_i(model_i(x))$ and both a cubic and quartic mean ($ens(x)^a = mean_i(model_i(x)^a)$, $a = 3, 4$).

Additionally, [12] also suggests combining average and maximum model prediction into an average of maxima (AoM) or maximum of average (MoA). But using those requires choosing many additional options: Which combination to use and which submodel to assign to which of how many groups. Because of this, we leave these functions for a dedicated study.

B. Stacking

The three most common ensembling methods for supervised tasks are bagging, boosting and stacking [31]. In the unsupervised setting, we mostly look at bagging, as it is much easier to do compared to boosting [12]. But we do not see a reason prohibiting stacking: We can understand the individual models as learning a feature representation space in which outliers are more easily visible. And on this representation, we can train another anomaly detection algorithm.

Still stacking has not been studied well in the literature, except for supervised anomaly detection [33].

To understand why there is still not much research into stacking, we test a few common anomaly detection methods as combination functions. For this, we train our stacking model on 20% of our normal training data. We refer to this dataset as our normalization samples, as we will also use the same samples to calculate normalization constants in the next section. We train 5 different *kNN* [17] models with various values of k (1,3,5,10,100), and an isolation forest [6]. We also use the (the inverse) probability density function obtained by modeling the distribution as a single multi-variate gaussian distribution.

C. Combination through clustering

Similar to the stacking approach, here we try to model the resulting anomaly detection representation space. But instead of training an anomaly detection algorithm, we cluster these points. This means when we have multiple groups of anomalies (one algorithm only finds one kind of anomaly, while another one only finds a different kind of anomaly), we assume that points in the center of each of these clusters are normal, while points with a high distance to them are not. To evaluate this, we use *Affinity propagation* [34] to calculate the clusters. For computational reasons, we restrict ourselves to using *Zscore* normalization.

D. Threshold combinations

A threshold combination of submodels [12] is defined as $ens_{thresh}(x) = mean_i(relu(model_i(x) - t) + t)$ (with $relu(x) = \frac{1}{2} \cdot (x + \|x\|)$). So high anomaly scores are not changed, while low anomaly scores are cut off and brought to a constant level. Anomaly detection usually only cares about high ranges of scores, which represent anomalies. This means that the combination can be perturbed by small fluctuations in how normal a sample is considered, which can be fixed by threshold ensembles. We implement this here only with the original normalization of *Z-score*, as this allows us to understand the Threshold value t as Z scores by assuming the distribution of anomaly scores to be Gaussian. We test here 5 different values for $t = -2, -1, 0, 1, 2$.

E. Greedy ensembles

A greedy ensemble [32] is very similar to an approach like boosting for unsupervised anomaly detection. This would usually not be possible without true labels, so the approach here is to construct an approximative label vector from the submodels themself. This works, as we can be reasonably sure that the most abnormal samples (we assume here 5) found by each algorithm are true anomalies. Using this vector, we can try to average only a subset of models. We choose those that maximize a weighted correlation to our constructed vector in a greedy manner. We restrict ourselves to a *01* normalization for computational reasons.

F. Item response theory

Finally, Reference [15] suggests using *IRT*, a method commonly used in psychology to extract hidden variables, by treating an optimal anomaly score as hidden truth. To understand this further, please take a look at their excellent original paper.

IV. Normalization functions

We compare 5 different common normalization methods. For each of these methods, we use a subset of our training samples containing 20% of the points to calculate the individual normalization constants. When we demand all our samples to be in the range [0,1], all of our normalization samples are in this range, but for our training and test samples this is not guaranteed. Still most (normal) samples are in this range.

We think this approach matches the task of unsupervised anomaly detection best. When we require our model to work best on new data, we can not guarantee them to be in the same range, and we don't want to remove information.

The only exception is the *IRT* ensemble because their calculation requires every score to be between 0 and 1. Here we normalize our samples on the full dataset.

A. 01 normalization

By applying *01* normalization, we demand each model output to be between 0 and 1.

$$x_i^{01} = \frac{x_i - min_j(x_j)}{max_j(x_j) - min_j(x_j)} \tag{1}$$

B. Zscore normalization

When applying *Zscore* normalization, we want to fix the mean and the standard deviation of each model's output to be 0 and 1.

$$x_i^{Z} = \frac{x_i - mean_j(x_j)}{std_j(x_j)} \tag{2}$$

C. Simple normalization

Using no normalization would make little sense for most models, as their scale is often quite different. Instead, we simply divide each value by the mean of all predictions. We use this normalization mostly to test how important careful normalization is.

$$x_i^{simple} = \frac{x_i}{mean_j(x_j)} \tag{3}$$

D. Clipped normalization

Finally, we suggest an extension to *Zscore* normalization, in which values are limited to a given absolute value c.

$$x_i^{clipped} = min(max(x^Z, -c), c) \tag{4}$$

This means that the influence of a single model is limited, as it can no longer have an arbitrarily high contribution. We test this normalization once with $c = 1$ and with $c = 2$. Assuming a Gaussian distribution, this should alter $\approx 32\%$ and $\approx 4.6\%$ of samples respectively.

V. Experiments

A. Experimental setup

We design our experiments so that we can consider the highest number of ensembles available, by covering as many datasets and submodels as possible.

On every of the 182 datasets currently contained in the yano library [35], we apply each of the 21 models described in section II-A. All models are trained using the pyod library [36] with their originally suggested hyperparameters. Using these results, we take a look at every possible combination of models to build our ensembles. We only consider ensembles that contain at least 5 submodels. Some models are also incompatible with some datasets, producing faulty or erroneous results; we simply ignore those combinations and build ensembles from the remaining submodels.

In total, we look at 124.053.401 different ensemble-building tasks. We apply every of our 50 different pairs of combination (Section III) and normalization functions (Section IV) to each of them. This means, in total we look at more than 6 billion different ensembles to produce our conclusions.

B. Evaluating ensembles

A single anomaly detector is usually evaluated using a ROC-AUC score [37]. It is defined as the probability of random anomaly samples being assigned a higher anomaly score than a random normal one.

If we want to extend this to comparison ensembles, we can look at its rank-1 accuracy [15]: In how many cases does a given ensemble produce the highest possible AUC score?

But we think this is not the best choice, as this ignores the magnitude of change and introduces an unrealistic scenario. Most of the time, we only care that an ensemble reliably performs well and not that no other ensemble performs slightly better. And as the stated rank-1 accuracy depends on the number of combination functions we consider, which complicates parallelizing our calculations, we evaluate our combination functions differently.

For this, we propose 10 simple metrics, which not only allow us to find the best combination method, but also to learn more about how ensembles work for unsupervised tasks.

The first metric that we want to consider, is the reliability of the ensemble. How often is using an ensemble a good choice, compared to a random model($AUC(ens(x)) \geq mean_i(AUC(model_i(x)))$). And similarly, we also state the probability that the ensemble improves on the best and the worst individual model it is built from (*metric 2-4*).

But reliability alone is not enough. We also care about the size of the improvement. For this, we can directly look at the average ROC-AUC score (*metric 1*). But this value stays fairly constant around 0.75, as we have some datasets on which models commonly reach a good AUC score of ≈ 1.0 and some datasets where they only reach a low score ≈ 0.5. Because of this, it is hard to interpret any differences. So, we additionally also use the same comparisons as before and give the average difference in ROC-AUC between the ensemble and the worst/average/best submodel (*metric 5-7*).

Still, this weights badly-performing models differently from well-performing models. This is because improving an AUC score from 0.8 to 0.9 compared to 0.98 to 0.99, has a much bigger effect. Even though, in both cases, the probability that a random normal sample is considered more anomalous is halved. Because of this, we define a new metric to capture these probability changes:

$$\Delta Err = log(\frac{1 - AUC_1}{1 - AUC_2}) = log(1 - AUC_1) - log(1 - AUC_2) \tag{5}$$

Here a model halving the error rate always has a fixed $\Delta Err = log(2) \approx 0.3$. We also calculate this error rate compared to the performance of the worst/average/best submodel (*metric 8-10*).

C. Comparison of combination functions

We state these 10 metrics for each of our 50 ensemble procedures in Table I.

We achieve the best ensembles with a simple maximum combination function normalized with a *01* normalization.

TABLE I
ANOMALY DETECTION ENSEMBLE COMPARISON

Methods	Mean AUC	Bigger Than			ΔAUC to			ΔErr compared to		
		Minimum	***Mean***	***Maximum***	***Minimum***	***Mean***	***Maximum***	***Minimum***	***Mean***	***Maximum***
maximum (01)	**0.766**	**99.837**%	75.39%	4.445%	**0.28**	**0.036**	**-0.081**	**1.741**	**0.955**	**-1.386**
maximum (z)	0.758	99.248%	72.99%	3.258%	0.271	0.027	−0.09	1.601	0.814	−1.527
maximum (simple)	0.723	96.634%	64.153%	4.05%	0.237	−0.007	−0.124	1.37	0.584	−1.757
maximum (clipped 1)	0.66	91.431%	23.552%	0.282%	0.174	−0.07	−0.188	0.487	−0.299	−2.64
maximum (clipped 2)	0.712	95.002%	53.596%	0.739%	0.226	−0.018	−0.136	0.814	0.027	−2.314
mean (01)	0.754	96.794%	84.015%	2.64%	0.267	0.023	−0.094	1.599	0.813	−1.528
mean (z)	0.756	96.796%	84.857%	2.633%	0.27	0.026	−0.092	1.604	0.817	−1.524
mean (simple)	0.734	96.801%	69.156%	3.767%	0.248	0.004	−0.114	1.505	0.718	−1.623
mean (clipped 1)	0.747	96.8%	82.687%	1.308%	0.26	0.016	−0.101	1.199	0.412	−1.928
mean (clipped 2)	0.754	96.794%	**85.903**%	1.576%	0.267	0.023	−0.094	1.321	0.534	−1.807
median (01)	0.752	96.708%	82.115%	1.742%	0.265	0.021	−0.096	1.401	0.614	−1.727
median (z)	0.755	96.764%	84.038%	1.803%	0.268	0.024	−0.093	1.447	0.661	−1.68
median (simple)	0.744	96.659%	74.613%	2.805%	0.258	0.014	−0.104	1.381	0.595	−1.746
median (clipped 1)	0.743	96.761%	78.658%	0.475%	0.257	0.013	−0.104	1.056	0.269	−2.072
median (clipped 2)	0.753	96.759%	83.664%	1.065%	0.267	0.023	−0.095	1.295	0.509	−1.832
minimum (01)	0.673	95.942%	36.93%	0.902%	0.187	−0.057	−0.174	0.687	−0.1	−2.441
minimum (z)	0.671	96.439%	35.013%	0.682%	0.185	−0.059	−0.177	0.66	−0.126	−2.467
minimum (simple)	0.742	97.023%	71.487%	4.143%	0.255	0.011	−0.106	1.203	0.417	−1.924
minimum (clipped 1)	0.667	96.466%	29.391%	0.643%	0.181	−0.063	−0.181	0.629	−0.158	−2.499
minimum (clipped 2)	0.672	96.431%	34.76%	0.671%	0.186	−0.058	−0.175	0.66	−0.126	−2.467
l2mean (01)	0.753	97.075%	77.929%	3.079%	0.267	0.023	−0.095	1.649	0.862	−1.479
l2mean (z)	0.729	95.947%	52.75%	4.131%	0.243	−0.001	−0.119	1.311	0.524	−1.817
l2mean (simple)	0.735	97.753%	65.479%	2.892%	0.249	0.005	−0.113	1.458	0.671	−1.67
l2mean (clipped 1)	0.703	93.547%	37.37%	2.09%	0.216	−0.028	−0.145	0.803	0.016	−2.325
l2mean (clipped 2)	0.731	94.967%	53.955%	2.933%	0.244	0.0	−0.117	1.087	0.3	−2.041
l3mean (01)	0.751	97.333%	76.353%	3.242%	0.265	0.021	−0.097	1.637	0.851	−1.49
l3mean (z)	0.728	96.185%	52.526%	4.219%	0.242	−0.002	−0.12	1.314	0.528	−1.813
l3mean (simple)	0.734	97.618%	65.393%	3.095%	0.247	0.003	−0.114	1.444	0.657	−1.684
l3mean (clipped 1)	0.706	93.834%	39.472%	2.095%	0.22	−0.024	−0.142	0.827	0.04	−2.301
l3mean (clipped 2)	0.734	95.336%	56.223%	3.013%	0.247	0.003	−0.114	1.115	0.329	−2.012
l4mean (01)	0.75	97.537%	75.127%	3.361%	0.263	0.019	−0.098	1.604	0.817	−1.523
l4mean (z)	0.728	96.396%	52.426%	4.203%	0.241	−0.003	−0.12	1.316	0.53	−1.811
l4mean (simple)	0.733	97.599%	65.068%	3.15%	0.247	0.003	−0.114	1.436	0.649	−1.692
l4mean (clipped 1)	0.708	94.011%	40.809%	2.117%	0.222	−0.022	−0.14	0.844	0.058	−2.283
l4mean (clipped 2)	0.735	95.912%	57.487%	3.016%	0.249	0.005	−0.112	1.132	0.346	−1.995
Threshold ($t = -1$)	0.757	96.8%	85.036%	2.655%	0.271	0.027	−0.091	1.608	0.821	−1.52
Threshold ($t = -2$)	0.756	96.796%	84.934%	2.652%	0.27	0.026	−0.091	1.604	0.817	−1.524
Threshold ($t = 0$)	0.756	96.734%	81.793%	3.009%	0.27	0.026	−0.091	1.599	0.813	−1.528
Threshold ($t = 1$)	0.746	96.797%	70.396%	2.296%	0.26	0.016	−0.101	1.5	0.713	−1.628
Threshold ($t = 2$)	0.719	97.651%	47.855%	3.156%	0.233	−0.011	−0.129	1.295	0.508	−1.833
knn ($k = 1$)	0.757	97.326%	68.474%	4.434%	0.271	0.027	−0.091	1.543	0.757	−1.584
knn ($k = 3$)	0.757	97.532%	68.761%	4.434%	0.27	0.026	−0.091	1.537	0.751	−1.59
knn ($k = 5$)	0.756	97.664%	69.132%	4.074%	0.27	0.026	−0.092	1.523	0.736	−1.605
knn ($k = 10$)	0.752	97.068%	68.905%	3.22%	0.266	0.022	−0.096	1.48	0.693	−1.648
knn ($k = 100$)	0.664	78.755%	44.347%	3.768%	0.178	−0.066	−0.184	1.044	0.258	−2.083
IFor	0.754	97.283%	69.757%	2.502%	0.268	0.024	−0.093	1.352	0.566	−1.775
Gaussian	0.754	97.807%	66.289%	**6.148**%	0.268	0.024	−0.094	1.397	0.611	−1.73
Clustered	0.752	96.74%	76.265%	3.991%	0.265	0.021	−0.096	1.694	0.907	−1.434
Greedy	0.751	96.73%	83.008%	2.167%	0.264	0.02	−0.097	1.339	0.552	−1.789
IRT	0.731	96.511%	70.052%	2.624%	0.245	0.001	−0.117	1.125	0.338	−2.003

Overall simple functions perform quite well, many of them outperforming each of the more complicated methods.

It is the case, that most ensembles contain a submodel that performs better than the combined ensemble, showing that heterogenous ensembles for unsupervised anomaly detection work substantially different compared to supervised tasks.

At least ensembles outperforming an average model is fairly likely when using a good combination function. Still, even this is not guaranteed. There also is even a small chance, that the ensemble performs worse than each individual model. But using the right methods, it only happens for less than 1 out of every 500 different ensembles, and thus can likely be safely ignored.

From the simple methods we evaluate, the maximum seems to be clearly the best. But interestingly this depends on the normalization applied; when using a clipped normalization it performs worse than every other model (by most metrics). We think this happens because removing extreme values also removes the reason for making the maximum efficient: Only a single model has to consider a sample anomalous, but when $\approx 16\%$ of samples are equally as anomalous, this loses meaning.

Because of this high dependency on normalization effects,

we still only recommend a maximum combination function if at least some anomalies are known, to make sure this ensemble works well. Still, it can produce the best ensembles according to 8 out of 10 different metrics.

In the truly unsupervised case, a simple mean might be more reliable. As long as a common normalization is used, it does not matter much which one is chosen. And removing very extreme values, we can create the model with the highest probability of outperforming the average performance. Still, the average ROC-AUC performance is about 1% worse than for the best ensemble.

Higher powers of the mean don't seem to serve many purposes compared to a simple mean. Also using the minimum of all models might at most be useful for very specialized tasks.

On the other hand, the threshold ensemble can outperform a simple mean very slightly.But since the improvement is minor and this ensemble also interacts with a clipped normalization, it is probably not possible to increase it much further. The best possible threshold seems to be $t = -1$, where the $\approx 16\%$ most normal samples are modified (assuming a Gaussian distribution), contrary to the original paper [12], which suggests $t = 0$ (effecting 50% of samples).

Stacking could be quite promising. For the knn, a low value of k is the best, performing better than each other the isolation forest, a Gaussian fit, or even the clustering approach (in most metrics). It is able to outperform each simple mean, except in the likelihood of outperforming the mean case, nor in the Δerr metrics. This implies that it performs best in cases where the anomaly detection task is complicated. We see these as specialized high-risk high-reward methods. These models require the right situation (for example many normalization samples for the knn), but then can perform very well. And considering that we only looked at 4 simple approaches for stacking, we think there might be even better-dedicated algorithms for our task, and we look forward to revisiting this with future research.

The Gaussian fit deserves special attention, as it has by far the highest probability of outperforming every single submodel and thus might be able to benefit from the ensemble similar to how supervised algorithms work. Combining multiple definitions of anomalies and recognizing which parts of which definition is helpful. But before this is the case, we still have far to go, as the Gaussian fit is not very reliable, assumes a specific distribution of our samples, and does not work well with few data points.

Our last methods, the greedy ensemble, and the IRT approach perform well, but also not exceptionally so. And since in most cases we can achieve better results with less work, they don't seem worth the extra effort.

Finally, when we look at the last columns, we can reach an ΔErr of almost 1. This means that when a random model misclassifies a sample, on average there is only a $e^{-0.955} \approx 38\%$ chance that the *maximum (01)* ensemble also misclassifies it. So unless in very special situations, using an ensemble is generally a good idea.

D. Model contributions

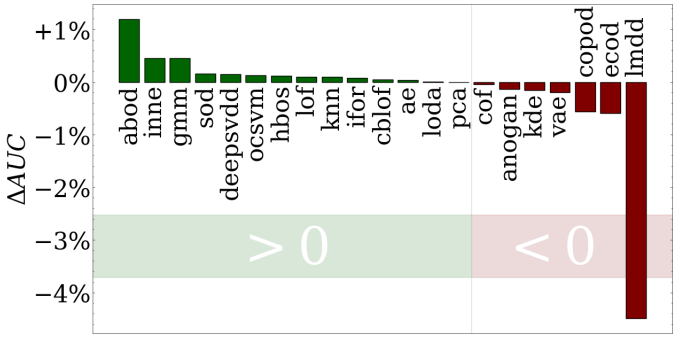

Fig. 2. Average ROC AUC score of ensembles using a specific submodel, compared to the average of all models considered. Here only shown for the *maximum (01)* ensembles, as this produces the best ensembles, and the rest look very similar.

Next to choosing the right combination procedure, we also study which models create the best ensembles. For this, we calculate the average AUC score for each ensemble considering a given model. We are no longer able to use all our ten metrics, as most metrics would compare the combined AUC score to the performance of individual models, which also changes when we remove/add a given model.

In Figure 2, we show the difference in the average performance when using all models for the *maximum* combination function and *01* normalization. We only show this dependency, as it has the best overall performance and the other ensemble methods show similar relations.

Abod seems to be by far the most helpful method to include in an ensemble. We think this is happening because its approach of considering angles is fairly unique, and so many models can profit from the new concept it provides. Overall, *Abod* should not be neglected in any ensemble, as its inclusion alone affects the resulting ensemble as much as choosing a slightly better ensemble procedure.

Other algorithms seem to actively hurt the ensemble performance. We do not suggest adding the *lmdd* algorithm to any ensemble, and would also suggest not employing *copod* or *ecod* except in special situations.

Interestingly, algorithms with similar concepts also perform relatively similarly. Both the *autoencoder* and the *Pca* are based on reconstruction errors, the *anogan*, *kde* and *variational autoencoder* all model densities and *DeepSVDD* is partially based on the concepts of the *Ocsvm*. All of these groups perform quite similarly, reinforcing our belief that the diversity of concepts is an important goal for each ensemble.

To study this further, we take a look at higher-order combinations. Namely, we look at the average AUC score for each ensemble that also considers two specific models simultaneously. This reduces our statics, but we still consider about 280 thousand ensembles for each combination of models.

In Figure 3 we show the average ROC AUC score of every ensemble that includes two different submodels, again only

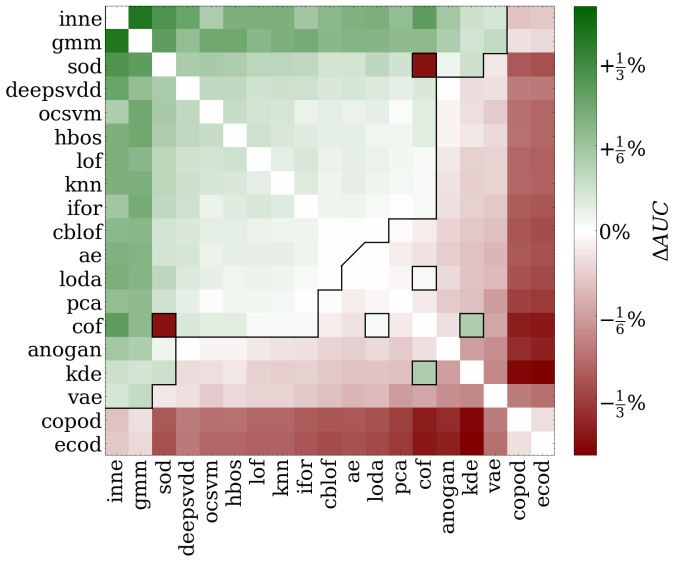

Fig. 3. Average AUC score of ensembles also considering two different models. We compare this to the expected score considering the average performance of both models individually. This means that each diagonal line always has a difference in AUC score of 0.0. We removed the two most extreme models *Abod* and *Lmdd*, to increase the readability of this plot.

for *maximum (01)* ensembles. We compare this AUC score to the expected performance considering each submodel.

$$AUC(m_1, m_2, \ldots) - \frac{(AUC(m_1, \ldots) + AUC(m_2, \ldots))}{2} \quad (6)$$

A positive score (green color) means ensembles using both models perform better than the expectation, while a negative score (red color) means they perform worse than expected. Overall, we see a group of models (from *Abod*, *Inne* till *autoencoder*) that only have positive interactions with each other. Because of this, we suggest using them to build an effective ensemble.

But overall, this relation looks quite similar to the contribution of a single model. The same six models are either much more or less useful than the rest. The only exception to this are a few interactions that behave strangely, all including the *COF* model. This also implies, that the strengths of the *SOD* model might be undervalued in Figure 2.

VI. Conclusion

In this paper, we created the most extensive comparison of combination procedures of heterogenous ensembles for unsupervised anomaly detection by far. To the best of our knowledge, we are also the first to study not only the effect of different combination functions but also of various normalizations and the contributions of individual models.

Using insights from each of these, we extract the following best practices for the creation of such ensembles:

1) From the set of algorithms (Abod, Inne, gmm, SOD, DeepSVDD, OCSVM, HBos, Lof, knn, ifor, cblof, autoencoder) train as many models on the training data as possible.
2) When computation time is a limiting factor, ignore some of the later models.
3) Choose a combination procedure
   - When it is possible to estimate the performance of the resulting ensemble, choose a *maximum* combination function with a *01* normalization.
   - If this is not the case, or the resulting ensemble performs poorly, rely instead on a simple *mean* of the model outputs, as well as a *Zscore* normalization.
4) Normalize each model output using the chosen normalization and combine the resulting values using the chosen combination function.
5) Use the resulting ensemble to score new samples.

From a research direction, we find that the benefit of complicated combination functions is limited. Very simple combination functions can outperform all of them. The *maximum* combination can be the best, but it is also very sensitive to the effects of normalization functions. The opposite is achieved by the *mean* combination function, which performs slightly worse, but does so more reliably and less dependent on normalization effects.

The most practical complicated combination functions are threshold ensembles, which extend the mean by removing the effects very normal samples have on the ensemble. Still, also their benefit is limited and we do not see many possible extensions to them.

We were also able to show that stacking for unsupervised anomaly detection is a rather promising research direction. Stacking methods can outperform each other combination function in at least one of the metrics we considered. We think they are only limited, because our approaches here are still quite simple, and we want to encourage the reader to think about more advanced stacking approaches.

To help with this, we publish our code at https://github.com/KDD-OpenSource/EvaluatingAnomalyEnsembles and combine our ensembling methods into the python library https://pypi.org/project/anoens. We also publish the results of our individual models at https://tu-dortmund.sciebo.de/s/tnCoUy9c6kknC18 to simplify including new methods into a similar comparison.

Acknowledgment

This work was supported by the Research Center Trustworthy Data Science and Security, an institution of the University Alliance Ruhr.

The authors gratefully acknowledge the computing time provided on the Linux HPC cluster at TU Dortmund University (LiDO3), partially funded in the course of the Large-Scale Equipment Initiative by the German Research Foundation (DFG) as project 271512359.

This research has been supported by the Federal Ministry of Education and Research of Germany and the state of North-Rhine Westphalia as part of the Lamarr-Institute for Machine Learning and Artificial Intelligence.

REFERENCES

[1] L. Dong, S. LIU, and H. ZHANG, "A method of anomaly detection and fault diagnosis with online adaptive learning under small training samples," *Pattern Recognition*, vol. 64, pp. 374–385, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320316303843

[2] W. Hilal, S. A. Gadsden, and J. Yawney, "Financial fraud: A review of anomaly detection techniques and recent advances," *Expert Systems with Applications*, vol. 193, p. 116429, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417421017164

[3] E. Šabić, D. Keeley, B. Henderson, and S. Nannemann, "Healthcare and anomaly detection: Using machine learning to predict anomalies in heart rate data," *AI andSOCIETY*, vol. 36, no. 1, p. 149–158, 2020.

[4] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," in *Information Processing in Medical Imaging*. Cham: Springer International Publishing, 2017, pp. 146–157.

[5] M. Rosenblatt, "Remarks on Some Nonparametric Estimates of a Density Function," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832 – 837, 1956. [Online]. Available: https://doi.org/10.1214/aoms/1177728190

[6] F. T. Liu, K. Ting, and Z.-H. Zhou, "Isolation forest," 01 2009, pp. 413 – 422.

[7] T. R. Bandaragoda, K. M. Ting, D. Albrecht, F. T. Liu, Y. Zhu, and J. R. Wells, "Isolation-based anomaly detection using nearest-neighbor ensembles," *Computational Intelligence*, vol. 34, no. 4, pp. 968–998, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/coin.12156

[8] H.-P. Kriegel, M. Schubert, and A. Zimek, "Angle-based outlier detection in high-dimensional data," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 444–452. [Online]. Available: https://doi.org/10.1145/1401890.1401946

[9] Z. Li, Y. Zhao, N. Botta, C. Ionescu, and X. Hu, "Copod: Copula-based outlier detection," in *2020 IEEE International Conference on Data Mining (ICDM)*, 2020, pp. 1118–1123.

[10] S. Han, X. Hu, H. Huang, M. Jiang, and Y. Zhao, "Adbench: Anomaly detection benchmark," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[11] A. Chiang, E. David, Y.-J. Lee, G. Leshem, and Y.-R. Yeh, "A study on anomaly detection ensembles," *Journal of Applied Logic*, vol. 21, 12 2016.

[12] C. Aggarwal, "Outlier ensembles: Position paper," *SIGKDD Explorations*, vol. 14, pp. 49–58, 01 2012.

[13] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga, "Outlier detection with autoencoder ensembles," in *SIAM International Conference on Data Mining*, 06 2017, pp. 90–98.

[14] B. Böing, S. Klüttermann, and E. Müller, "Post-robustifying deep anomaly detection ensembles by model selection."

[15] S. Kandanaarachchi, "Unsupervised anomaly detection ensembles using item response theory," *Information Sciences*, vol. 587, pp. 142–163, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025521012639

[16] L. Ruff, J. Kauffmann, R. Vandermeulen, G. Montavon, W. Samek, M. Kloft, T. Dietterich, and K.-R. Müller, "A unifying review of deep and shallow anomaly detection," *Proceedings of the IEEE*, vol. PP, pp. 1–40, 02 2021.

[17] X. Gu, L. Akoglu, and A. Rinaldo, *Statistical Analysis of Nearest Neighbor Methods for Anomaly Detection*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[18] M. Breunig, P. Kröger, R. Ng, and J. Sander, "Lof: Identifying density-based local outliers." vol. 29, 06 2000, pp. 93–104.

[19] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, D. E. Rumelhart and J. L. Mcclelland, Eds. Cambridge, MA: MIT Press, 1986, pp. 318–362.

[21] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support vector method for novelty detection," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 1999. [Online]. Available: https://proceedings.neurips.cc/paper/1999/file/8725fb777f25776ffa9076e44fcfd776-Paper.pdf

[22] D. Kingma and M. Welling, "Auto-encoding variational bayes," 12 2014.

[23] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang, "A novel anomaly detection scheme based on principal component classifier," 01 2003.

[24] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "Outlier detection in axis-parallel subspaces of high dimensional data," in *Advances in Knowledge Discovery and Data Mining*, T. Theeramunkong, B. Kijsirikul, N. Cercone, and T.-B. Ho, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 831–838.

[25] M. Goldstein and A. R. Dengel, "Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm," 2012.

[26] Z. He, X. Xu, and S. Deng, "Discovering cluster-based local outliers," *Pattern Recognition Letters*, vol. 24, no. 9, pp. 1641–1650, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865503000035

[27] T. Pevný, "Loda: Lightweight on-line detector of anomalies," *Mach. Learn.*, vol. 102, no. 2, p. 275–304, feb 2016. [Online]. Available: https://doi.org/10.1007/s10994-015-5521-0

[28] J. Tang, Z. Chen, A. W.-c. Fu, and D. W. Cheung, "Enhancing effectiveness of outlier detections for low density patterns," in *Advances in Knowledge Discovery and Data Mining*, M.-S. Chen, P. S. Yu, and B. Liu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 535–548.

[29] Z. Li, Y. Zhao, X. Hu, N. Botta, C. Ionescu, and G. Chen, "Ecod: Unsupervised outlier detection using empirical cumulative distribution functions," 01 2022.

[30] A. Arning, R. Agrawal, and P. Raghavan, "A linear method for deviation detection in large databases," in *Knowledge Discovery and Data Mining*, 1996.

[31] R. Odegua, "An empirical study of ensemble techniques (bagging, boosting and stacking)," 03 2019.

[32] E. Schubert, R. Wojdanowski, A. Zimek, and H. Kriegel, "On evaluation of outlier rankings and outlier scores," in *Proceedings of the Twelfth SIAM International Conference on Data Mining, Anaheim, California, USA, April 26-28, 2012*. SIAM / Omnipress, 2012, pp. 1047–1058. [Online]. Available: https://doi.org/10.1137/1.9781611972825.90

[33] M. O. Sandim, "Using stacked generalization for anomaly detection," Ph.D. dissertation, 2017.

[34] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1136800

[35] S. Klüttermann, "yano," Jan. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7520448

[36] Y. Zhao, Z. Nasrullah, and Z. Li, "Pyod: A python toolbox for scalable outlier detection," *Journal of Machine Learning Research*, vol. 20, no. 96, pp. 1–7, 2019. [Online]. Available: http://jmlr.org/papers/v20/19-011.html

[37] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (roc) curve." *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.