Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Confidence Ensembles: Tabular Data Classifiers on Steroids

Tommaso Zoppi[1]*, Peter Popov[2]

[1] Department of Mathematics & Computer Science, University of Florence, Viale Morgagni 65, 50134, Florence (IT)
[2] Computer Science Department, City St. George's, University of London, Northampton Square, EC1V 0HB, London, United Kingdom
Mail addresses: tommaso.zoppi@unifi.it, p.t.popov@city.ac.uk
* Corresponding author

A R T I C L E I N F O

Article history:
Submitted: 05/12/2024
Revision Submitted: 24/02/2025
Accepted: 15/03/2025

Keywords:
Confidence Ensembles
Classification Confidence
Ensemble Learning
Robust Classification
Tabular Data
Machine Learning

A B S T R A C T

The astounding amount of research conducted in the last decades provided plenty of Machine Learning (ML) algorithms and models for solving a wide variety of tasks for tabular data. However, classifiers are not always fast, accurate, and robust to unknown inputs, calling for further research in the domain. This paper proposes two classifiers based on *confidence ensembles*: Confidence Bagging (ConfBag) and Confidence Boosting (ConfBoost). Confidence ensembles build upon a base estimator and create base learners relying on the concept of "confidence" in predictions. They apply to any classification problem: binary and multi-class, supervised or unsupervised, without requiring additional data with respect to those already required by the base estimator. Our experimental evaluation using a range of tabular datasets shows that confidence ensembles, and especially ConfBoost, i) build more accurate classifiers than base estimators alone, even using a limited amount of base learners, ii) are relatively easy to tune as they rely on a limited number of hyper-parameters, and iii) are significantly more robust when dealing with unknown, unexpected input data compared to other tabular data classifiers. Amongst others, confidence ensembles showed potential in going beyond the performance of de-facto standard classifiers for tabular data such as Random Forest and eXtreme Gradient Boosting. ConfBag and ConfBoost are publicly available as PyPI package, compliant with widely used Python frameworks such as *scikit-learn* and *pyod*, and require little to no tuning to be exercised on tabular datasets for classification tasks.

1 Introduction

We are at the dawn of the fifth industrial revolution [1], where retail and business logic are meant to synergize with sustainability and inclusivity for building human-centered autonomous systems to relieve the human workforce from time-consuming, laborious, and often hazardous tasks for the social good. Examples include, but are not limited to: fully autonomous driving, computer-guided robotic surgery, mobile robots for inspections and surveillance, optimized power management and generation, and manufacturing robots. These cutting-edge innovations often require solving classification problems by designing and training Machine Learning (ML) algorithms that assign a discrete label to input data with little to no misclassifications. Classifiers should solve a wide variety of tasks processing either structured tabular data [2] or unstructured inputs (e.g., images) [3], [4] for binary or multi-class classification. Most applications deal with tabular data, comprising samples (rows) with the same set of features (columns). Tabular data is used in practical applications in many fields, especially when stakeholders aim at monitoring the behavior of ICT systems for the early detection of anomalies due to errors, intrusions, or upcoming failures. Monitoring activities generate very large amounts of (tabular) data that could be potentially used for classification; unfortunately, such data is often unlabeled. In this case, the classification process can only be unsupervised, leading to well-known detrimental effects on the overall classification performance [5], making it unfeasible apart from very specific "corner" cases [6], [7].

Regardless of the availability of labels, analyzing such a massive amount of tabular data is very challenging. That is why experts and researchers keep designing new algorithms and providing trained models that have the potential to outperform their baseline on specific scenarios, often under the assumption of Independent and Identically Distributed (IID) training, validation, and test data [8], [9]. Constraining experiments behind such assumptions may be acceptable and suitable for research purposes; however, real systems and infrastructures are prone to encountering unexpected operating conditions [10], [11] that violate the above mentioned IID assumption. Classifiers for these systems have to be designed and deployed for high accuracy, but also to be robust to unexpected inputs, and meant to operate reliably in the wild [8].

This paper proposes "Confidence ensembles", classifiers that are more accurate and robust than the respective baselines, and in our judgment close the gap between what practical applications require and what the current state-of-the-art in machine learning offers as of today. The proposed confidence ensembles, either Confidence Bagging (ConfBag) or Confidence Boosting (ConfBoost), can use any existing classifier as a "base estimator" to build an ensemble meta-classifier without requiring any additional information compared to what the base estimator needs. Thus, they can be applied to any existing classification task in a way that is almost transparent to the user and requires minimal tuning. ConfBag refines the traditional bagging [12], [13], [14], by allowing only the most confident base learners to contribute to the prediction, weighting their contribution to the prediction by their confidence score. ConfBoost creates base learners that are more and more specialized in classifying train data items for which existing base learners cannot make a confident prediction. This is a radically different approach from the typical boosting strategies [15], [16], which always require a labeled training set as (weak) base learners are subsequently trained using train data that got misclassified by existing base learners. ConfBag and ConfBoost:

- are compatible with any existing classifier as a base estimator, which is used to create base and meta learners;
- have a generic and quite straightforward architecture that applies to all classification tasks and scenarios: supervised or unsupervised, binary or multi-class;
- do not require additional input with respect to those already needed by the base estimator;
- require little to no parameter tuning, are easy to use even by non-experts, and are implemented in a public Python framework that exposes interfaces that comply with the de-facto standard libraries [17] such as *scikit-learn* (supervised learning) and *pyod* (unsupervised learning);
- provide a significant improvement in accuracy and robustness of classification compared to using traditional classifiers, measured through an extensive experimental analysis on many public tabular datasets and algorithms;
- can deal with many data types. This paper focuses on tabular data only, but the approach is general and can be used to instantiate classifiers of unstructured data, e.g. images [18].

The paper is organized as follows. Section 2 provides the background and summarizes the works related to classification and robustness to unknown input data or operating conditions. Section 3 describes the design of ConfBag and ConfBoost, their key features, and their differences against traditional bagging and boosting ensembles. Section 4 describes the experimental methodology to evaluate the performance of confidence ensembles against baselines, including standard bagging and boosting. Section 5 presents the results from the experimental campaigns, quantifying how confidence ensembles, and especially ConfBoost, are more accurate and far more robust to unexpected inputs than the base estimators. Section 6 conducts a statistical validation to strengthen experimental findings and provide statistically solid takeovers. Section 7 details threats to the validity of this study, letting Section 8 conclude the paper. Moreover, Appendix A provides a detailed description of the *conf-ensemble* Python framework and how to use ConfBag and ConfBoost, with Appendix B summarizing terms and acronyms used in the paper.

2 Background and Related Works

This section provides useful background about classifiers for tabular data and their robustness to unexpected input data. The terminology introduced therein is summarized in Appendix B.

2.1 Machine Learning Classifiers

Decades of research and practice on Machine Learning (ML) provided us with plenty of algorithms that can learn how to predict a discrete label for a data point i.e., classifiers. Supervised classifiers [2], [16], [19], and particularly Deep Learners [3], [4], [20] were demonstrated capable of achieving excellent classification performance in many application domains, whereas unsupervised classifiers are typically applied only when training data is not labelled [21], [22].

More formally, a classifier first devises a mathematical model from a *training dataset* [20], which contains a given amount of data points. Each data point contains a set of feature values, where each feature value describes a specific input of the classification problem. Once the model has learned (i.e. the ML algorithm has been trained), it can be used to predict the probabilities of the data point belonging to each class defined for the problem domain, of which the class with the highest probability is assigned as a label of the new data point.

The classification performance of the classifier is usually computed by applying the classifier to novel data points and computing metrics such as accuracy [23], i.e., the percentage of correct predictions of a classifier (typically on the *testing dataset*).

2.2 Tabular Datasets and ICT Systems

Data to be classified may result from monitoring activities of computer systems, where features are performance system indicators at hardware or low-level, system-level, input/sensor, environment, application-level, or even coding-level [24], [25]. Features can be textual or numeric: textual features (e.g., the name of a protocol) are always categorical, while numeric features may either be categorical (the ID of a system call), or continuous, describing a continuous ordinal range of values such as the percentage of memory used, the number of packets received from the network interface in a time-frame, etc. Classifiers may compute Euclidean distance (e.g., those based on the concept of neighbourhood as the k^{th} nearest neighbours - kNN), which may deliver misleading results when applied to categorical features: there is no meaning in computing the distance between the IDs of system calls, or between the names of network protocols. As such, categorical features usually require pre-processing before being fed to a classifier e.g., representing each value of a categorical feature with a vector of floating-point numbers, or dummy variables [26].

Datasets resulting from monitoring activities have specific properties compared to other tabular datasets. First, features can hardly be considered independent as they describe different viewpoints of the same system or different areas of the same system. This may become a problem whenever applying classifiers that are known to perform well under the assumption of (linear) independence amongst features. Whereas it may be possible to eliminate or reduce collinearity in multivariate datasets using mechanisms such as Principal Component Analysis, this is beyond the scope of this work and will not be elaborated further. Second,

monitoring activities usually happen over a quite stretched timespan: the amount of data points is far higher than those of features, which rarely exceed hundreds. Monitoring thousands of features may exceedingly slow down the execution of the regular tasks of the system, which should not be negatively impacted by monitoring and logging activities. A trade-off analysis is typically conducted to understand the balance between the completeness of information provided by monitored features and the resulting computational overhead.

2.3 Machine Learning Classifiers for Tabular Data

Supervised classifiers require training data for which the label (also called *class*) is known. Recent literature shows how ensembles of decision trees as Random Forests or Gradient Boosting classifiers can accurately and quickly extract a model from tabular data, and are considered the recommended option or de facto standard for classifying tabular data [2], [27], [28]. Other works propose Deep Neural Networks that are specifically crafted for the classification of tabular data [29], [30]. However, they do not reliably outperform tree-based ensembles, possibly due to *lack of locality, data sparsity (missing values), mixed feature types (numerical, ordinal, and categorical), and lack of prior knowledge about the dataset structure* [2]. This dichotomy led to a wide variety of benchmark studies in recent years. An extensive study [31] analyses the factors making DNN-based classifiers more likely to outperform the gradient-boosted decision trees (GBDTs) on tabular data. The study analyses 19 classification algorithms and 176 datasets and provides useful insight: i) the difference between the two classification methods is typically *insignificant*; ii) tuning hyperparameters usually is more important than the choice between the two classification methods; iii) DNNs tend to perform better on datasets with many features, whose values are normalized. The study also defines a new benchmark, called *TabZilla*; a similar benchmark, *TALENT*, is presented in [32]. The study, based on 300 tabular datasets, provides an intriguing insight that a core set of "meta-features" can be used to identify dataset heterogeneity, which in turn affect the classification performance.

When a labelled training dataset is not available, the only option is to conduct unsupervised classification. These ML algorithms model the expected (normal) behaviour of a system, and classify any deviation from the normal behaviour as an anomaly [33]; thus, they can only perform binary classification (i.e., is my system behaving unusually or not?) assuming no other information is provided. Options for unsupervised classification include, but are not limited to, clustering, statistical, angle, density, and neighbour-based algorithms [22]. Differently from supervised classification, unsupervised classification of tabular data has seen a decent usage of DNNs, namely autoencoders, that have been shown [34], [35] to perform similarly to other unsupervised classifiers.

2.4 Ensembles and Diversity

N-Version Programming (NVP) and design diversity have been successfully used for many decades and enforced using different design methods, including: programming languages, functional diversity, different forms of testing [36], [37]. In statistics and machine learning, *ensemble* methods use multiple learning algorithms *to obtain better predictive performance than could be obtained from any of the constituent learning model alone* [38]. An ensemble is a form of NVP as it consists of a finite set of alternative models, called base learners, orchestrated according to a meta-learning strategy [39] acting as data fusion strategy. Ensembles usually outperform individual ML algorithms and have been used successfully in a variety of domains [16], [17].

The enabling condition for an ensemble to outperform the individual classifiers is the diversity in the behaviour of the base learners: "*If perturbing the learning set can cause significant changes in the predictor constructed, then bagging can improve accuracy*" [14]. For ensembles and meta-learning strategies as stacking [40] that use different algorithms to create base learners, diversity is guaranteed to some extent, but may not suffice [41]. More generally, the study [42] summarizes three different *design diversity principles*: Data, Model, and Inference Diversification. *Data diversification* provides different training data to many instances of the same classifier, or single instances of many different classifiers. This concept is widely applied in Bagging (e.g., Random Forests) or Boosting (e.g., XGBoost) ensemble classifiers, and makes them more accurate than their Decision Tree, or Decision Stump, baseline. *Model diversification* can be implemented either by creating different instances of the same classifiers that are trained using different parameters (e.g., a different k value for kNN) or by using ensembles, which create multiple base learners from heterogeneous classifiers. Lastly, inference diversification is concerned with obtaining multiple outputs from each classifier, e.g. a classifier output consists of a set of possible decisions alongside a ranking or their confidence scores. This usually applies to object detection and recognition, where multiple labels may be assigned to a single input, each with a confidence score. Such a diversification, however, rarely applies to the classification of tabular data. Sometimes, design diversity is paired with metrics to experimentally quantify diversity [43].

2.5 Confidence and Uncertainty of Classification

ML researchers usually aim at maximizing classification performance. However, trusting each prediction of a classifier, to the extent that the prediction can be confidently propagated towards the encompassing system and used in a real system, is a different problem [44]. Confidence in classification requires estimating the prediction uncertainty and using it to suspect if the classifiers' predictions are likely to be misclassifications. Uncertainty is [45] a combination of *aleatoric* and *epistemic* uncertainty. The former refers to the notion of randomness, that is, the variability in the outcome of an experiment due to inherently random effects e.g., coin-flipping. The latter describes the uncertainty due to a lack of knowledge (i.e. in one's knowledge) of any underlying random phenomenon or due to methodological errors. In other words, epistemic uncertainty refers to the reducible part of the (total) uncertainty, whereas aleatoric uncertainty refers to the irreducible part and cannot be reliably estimated.

(Epistemic) uncertainty can be quantified using the Bayes theorem [46] or more complex approaches. Works as [47] estimate the uncertainty by using ensembles of neural networks: scores from the ensembles are combined in a unified measure that describes the agreement of predictions and quantifies uncertainty. In [48], authors processed *softmax* probabilities of neural networks to identify misclassified data points. A new proposal came from [49], where authors paired a k-Nearest Neighbour classifier with a neural network to compute the prediction uncertainty. The work [50] computed the cross-entropy on the probabilities of a neural network and used it to detect the out-of-distribution input data

likely to be misclassified. Many of these approaches are used in [51] to implement a safety wrapper.

2.6 Unexpected Inputs and Robustness

Even when estimating confidence, there is the risk of having classifiers that: "*[...] produce almost always high confidence predictions far away from the training data*" [52]. This is a relevant concern as a classifier deployed in its operational environment is likely to face input data that does not belong to the distribution used to train the classifier. Unexpected or non-IID inputs may be due to a wide variety of reasons, mostly environmental changes, distribution shifts, anomalous, Out-Of-Distribution (OOD) data, or adversarial attacks [53], [54]. OOD and adversarial data were primarily deemed relevant for computer vision and image classification. Recently, these sources of unknown inputs become of concern for tabular data [54], alongside with: i) novel or unplanned interactions between subsystems or interfaces, generating behavioural anomalies [33], ii) distribution and concept drifts [55], iii) the occurrence of previously unseen events (i.e., zero-day attacks in security [6]). Industrial standards such as the SOTIF (ISO/PAS 21448 – Safety Of The Intended Functionality [56]) recognize performance limitations of ML-based software due to these unknown inputs, concluding that the frequency of misclassifications due to these events should be reduced until it is considered acceptable.

Straightforwardly, the impact of unexpected input data must be considered when evaluating the performance of a classifier [6], [57], [58], [59], quantifying what is generally referred to as *robustness*. In [59], the authors review existing datasets and classifiers for attack detection. Amongst other findings, they conclude that classifiers may have the problem of adapting to new attacks, raising many false alarms, or having poor accuracy in the process. The study [58] applies the kNN classifier to a dataset of system calls and measures a clear degradation in the detection performance of kNN, whose Recall drops from 100% to 75% in the presence of unknown inputs. The study [6] measures the impact that zero-day attacks have on supervised and unsupervised classifiers and concludes that when the likelihood of zero-day attacks becomes very high, the performance of unsupervised intrusion detectors may become superior to supervised alternatives. Last, the work in [57] trains different classifiers with the CICIDS17 tabular dataset and evaluates them using an unseen dataset, measuring a clear drop in accuracy due to the trained model being neither general nor robust enough to unknown test data from a similar but different dataset.

The occurrence of unexpected inputs has a noticeable impact on classification performance, especially of supervised classifiers. A possible solution may be to pair supervised classifiers with unsupervised ones as suggested in [21], [60]. However, most of these solutions are very problem-specific and are difficult to generalize: combining the two approaches itself is not trivial, does not always improve performance, and can even have detrimental effects as argued in [41].

3 Confidence Ensembles

In this section we describe ConfBag and ConfBoost ensembles, which use the concept of prediction confidence for building the ensemble model.

3.1 Preliminaries and Notation

Confidence ensembles are created from a base estimator, which is used to derive k *base learners* bl_i, $1 \leq i \leq k$ within the training process of each confidence ensemble. The base estimator can be any existing classifier *clf* that implements the following functions.

fit: this function trains the classifier *clf* using a specific train set. The training can be supervised (with labelled training data) or unsupervised. The *fit* function generates a model and makes *clf* ready for prediction: unfitted classifiers cannot predict anything. The syntax of using the "fit" function is as follows:

clf.fit(train features) - for a unsupervised classifier, and
clf.fit(train features, train labels) – for a supervised classifier.

predict_proba: given an input data *data*, it predicts a probability distribution in the form of an array of probabilities $prob = \{p_i, 1 \leq i \leq |c|, 0 \leq pi \leq 1\}$ for a given set *c* of classes *c*. Probabilities in the *prob* array should sum up to 1, i.e. $\sum_{i=1}^{|c|} p_i = 1$ and represent the likelihood that the data item submitted for classification belongs with probability p_i to class i.

$$prob = clf.predict\_proba(data) \quad (1)$$

predict_confidence (optional): this function quantifies the confidence $0 \leq conf \leq 1$ in a specific prediction for a given input data *data*, with the following signature.

$$conf = clf.predict\_confidence(data) \quad (2)$$

Whenever this function is not provided (nor implemented), the confidence will be derived as the maximum probability from *predict_proba* as in the following Equation (3).

$$conf = max\{pi \in clf.predict\_proba(data),\ 1 \leq i \leq |c|\} \quad (3)$$

3.2 Confidence Bagging: ConfBag

Confidence Bagging (ConfBag) is described below and is depicted in Figure 1.

3.2.1 Basics of Bagging

Bagging was first proposed by Breiman [14]. It creates *k* sets where n_{bag} data is drawn with or without replacement from the training set of *n* data points. Then, it uses these sets to train *k* base learners which are instances of the same *base estimator* trained using different training sets. Once all base learners are trained the prediction for a new input data is exercised as an "*average over the versions when predicting a numerical outcome and a plurality vote when predicting a class*" [14].

3.2.2 Training ConfBag

The training of ConfBag follows the proposal of Breiman [14] with a small adjustment, as it selects a subset of both training data points and features to train the base learners. More precisely, it randomly draws (without replacement) a subset of $n_{bag} < n$ data points, each with $f_{bag} \leq f$ features that will be used by a base learner i.e., each base learner will see only a fraction of the training set composed of n_{bag} items and f_{bag} features.

3.2.3 Predictions with ConfBag

The prediction (inference) phase is very different from the original bagging [14]: ConfBag predicts a label by using the confidence scores in two different ways.

Confidence for selecting base learners. Each base learner knows

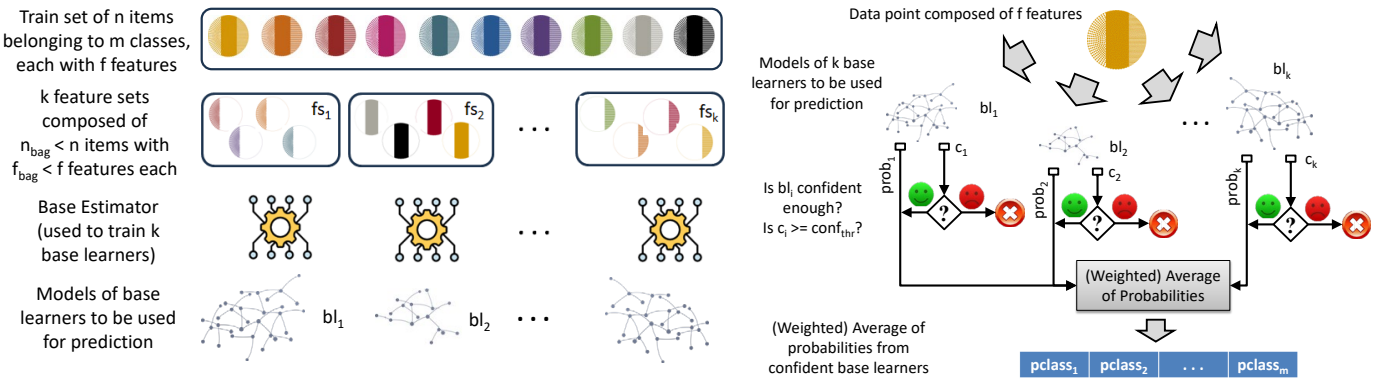


Figure 1: Train (left) and test (right) schemas for Confidence Bagging (ConfBag) at a glance.

the confidence in their prediction through the *predict_confidence* function. This may range from no uncertainty (high confidence) to high uncertainty (low confidence). Allowing the base learners with low confidence in their predictions to take part in the final decision may negatively affect the quality of the final decision. In ConfBag, the predictions of "uncertain" base learners are discarded as their opinion may not be trustworthy. Predictions of a base learner bl_i, $1 \leq i \leq k$, will contribute to the output of ConfBag only if the confidence by the base learner exceeds a given threshold:

$$conf_i = bl_i.predict\_confidence(data) \geq conf_{thr} \quad (4)$$

Clearly, the threshold $conf_{thr}$ in Equation (4) is essential and its value must be chosen with care as it may dramatically alter the prediction result. In cases when the optimal $conf_{thr}$ is difficult to derive, ConfBag may be configured to operate differently, e.g., the user may specify a number bl_n or the fraction bl_{frac} of base learners which should contribute to the final decision of the ConfBag. Then, ConfBag will compute an array of confidence scores, ordered by value, *scs* as in the following Equation (5).

$$scs=sort\_descending(\{bl_i.predict\_confidence(data),\ 1 \leq i \leq k\}) \quad (5)$$

For each prediction, a dynamic $conf_{thr}$ will be derived as the bl_{frac} (or bl_n / k) percentile of *scs*, or rather the value for which the exactly bl_n (or bl_{frac} * k) base learners predict with confidence greater than $conf_{thr}$. Overall, the threshold is computed as in Equation (6).

$$conf_{thr} = \begin{cases} conf_{thr}, if\ conf_{thr}\ is\ provided\ by\ the\ user \\ scs[bl_n], if\ bl_n\ is\ provided\ instead \\ scs[index], index = \lfloor bl_{frac} * k \rfloor, if\ bl_{frac}\ elsewhere \end{cases} \quad (6)$$

Confidence to weight predictions. Traditional bagging averages of probabilities. A ConfBag *conf_bag_clf* computes probabilities *cb_prob* for a data point *data* as shown in Equation (7).

$$cb\_prob = conf\_bag\_clf.predict\_proba(data) = \frac{1}{bl_n}\sum_{i=0}^{k} bl_i.predict\_proba(data)\ if\ bl_i.predict\_confidence(data) \geq conf_{thr} \quad (7)$$

ConfBag also offers the option to *weighting* the predicted probabilities with the confidence of each base learner, see Equation (8).

$$cb\_prob = conf\_bag\_clf.predict\_proba(data) = \sum_{i=0}^{k} bl_i.predict\_proba(data) * bl_i.predict\_confidence(data)\ if\ bl_i.predict\_confidence(data) \geq conf_{thr}$$
$$cb\_prob = cb\_prob\ /\ sum(c\_prob) \quad (8)$$

3.2.4 Parameters

ConfBag is configured via a set of parameters.

- *base_estimator* (classifier object): this is the classifier to be used to create base learners.
- k (integer ≥ 2): the number of base learners; for compliance with the existing libraries, k may be referred to as *n_estimators*.
- *f_bag* (0 < float ≤ 1): the proportion (as a percentage of features of the full training set) of training features to be provided to each base learner when learning its model.
- *sampling_ratio* (0 < float ≤ 1): the percentage of the items of the training set to be provided to each base learner in training (used to compute n_{bag})
- *weighted_pred* (a Boolean value): true if the final prediction has to be computed as the weighted average, false if only a simple average is used.
- *conf_thr* (0 < float ≤ 1): the value to be used as a confidence threshold for judging whether the prediction of each base learner is made with sufficiently high confidence or not.
- *bl_n* (0 < integer ≤ k): the number of base learners that will contribute to the final prediction by the ConfBag classifier.
- *bl_perc* (0 < float ≤ 1): the proportion of base learners that will contribute to the final prediction by ConfBag classifier.

Only one of the last three parameters is used: if *conf_thr* is provided, *bl_n* and *bl_perc* values will be ignored; otherwise, *bl_perc* will be used only if *bl_n* is missing.

3.3 Confidence Boosting: ConfBoost

Confidence Boosting (ConfBoost) is described below and is depicted in Figure 2.

3.3.1 Basics of Boosting

ConfBoost builds upon the proposal of Shapire [61], which introduced boosting as an ensemble of weak learners, where each base learner "*is only required to perform slightly better than random guessing*", building "*as strong as a model in which the learner's error can be made arbitrarily small*". Base learners are created sequentially and are specialized to classify areas of the input space for which existing base learners output misclassifications. Consequently, boosting is a supervised process that assumes knowledge of the labels to understand if a data point is misclassified by a base learner. This information is then used to prepare the training dataset for the next base learner. Boosting works well when using decision stumps as base estimators: decision stumps are shallow decision tree regressors that have constrained learning capabilities and work well as weak-learners.

3.3.2 Training ConfBoost

The biggest innovation of ConfBoost is to use confidence scores of base learners instead of their predictions to influence how data is sampled for training the next base learner. Instead of making misclassified training data more likely to be chosen for training the next base-learner, we target training data for which the existing base learners cannot output a confident prediction, no matter if it is correct or not. The training process of a base learner bl_i, $1 \leq i \leq k$, works as follows.

- Draw $n_{boost} \leq n$ data points from the training dataset *ts* according to their weights at step *i-1*. For bl_1 (first base-learner), weights are uniformly distributed: $w = \{w_j = 1/n,\ 1 \leq j \leq n\}$. This creates a training sub-dataset $ts_i \subset ts$.
- Train a copy of the *base_estimator* using ts_i. The resulting model will be referred to as the bl_i model.
- Score the confidence $conf_i$ of the bl_i model on each data point in the *full* training dataset *ts*. Then, update the weights *w* of the training dataset according to the rule in Equation (9) below.

$$w_j = \begin{cases} w_j * lr, if\ c_j < conf_{thr} \\ \frac{w_j}{lr} otherwise \end{cases}, 1 \leq j \leq n$$
$$w = \{\frac{w_j}{sum(w)}, 1 \leq j \leq n\} \quad (9)$$

- This increases the weights of the data points in the training dataset ts_i for which bl_i is not confident enough. The update speed is regulated by a parameter *lr* (learning rate): the higher the value of this parameter, the faster the weights will increase or decrease. After updating the weights, the array *w* gets normalized to sum up to 1.

These steps train the base learner bl_i and prepare for sampling the next training dataset ts_{i+1}, and train the next bl_{i+1}, thus iterating over all *k* base learners until the training process ends.

3.3.3 Predictions with ConfBoost

As it was originally proposed [15] - and applied to almost all known boosting algorithms to date [16], [17] - ConfBoost computes the final probabilities (and thus the predicted class) by averaging the probabilities computed by each base learner. Depending on the specific implementation, boosting algorithms may compute the final prediction as a weighted average of the base learners' predictions using different measures (e.g., the accuracy of a base learner on the validation/training dataset) as weights. Similarly to ConfBag, ConfBoost may perform a weighted average of probabilities using the confidence in the predictions of each base learner as weight.

3.3.4 Parameters

ConfBoost is configured via the following parameters.

- *base_estimator*(object): this is the classifier to be used to create base learners.
- *k* (integer ≥ 2): the number of base learners; for compliance with the existing libraries, *k* may be referred to as *n_estimators*.
- *sampling_ratio* (0 < float ≤ 1): the percentage of items of the training dataset to be sampled and used for training each base learner (used to compute n_{boost}).
- *lr* (float > 1): the learning rate, or rather the speed for which weights are updated. The bigger the *lr*, the faster the value of weights increases/decreases.
- *weighted_pred* (a Boolean): true if the final prediction by ConfBoost is computed as a weighted average, false - if only a simple average is used.
- *conf_thr* (0 < float ≤ 1): the value to be used i) as threshold to consider updating the sampling weights of the training set, and ii) for judging whether the prediction of each base learner is made with sufficiently high confidence or not.
- *bl_n* (0 < integer ≤ k): the number of base learners that will contribute to the final prediction by the ConfBoost classifier.
- *bl_perc* (0 < float ≤ 1): the proportion of base learners that will contribute to the final prediction by ConfBoost classifier.

Differently from ConfBag, *conf_thr* is mandatory as it is needed both in the training and inference phase. Should the user provide *bl_perc* or *bl_n* inputs, these will be used to recompute the confidence threshold to be used for inference (not for training) according to the formula in Equation (6).

3.4 Discussion and Issues of Confidence Ensembles

ConfBag and ConfBoost have a generic formulation that makes them applicable to any type of classification problem: they apply to both supervised and unsupervised classification problems, whereas traditional boosting is supervised by design. Also, they rely on a very basic formulation of confidence (i.e., the highest probability predicted for a given class) that can be extended at will depending on the needs, knowledge, and expertise of the user (e.g.,

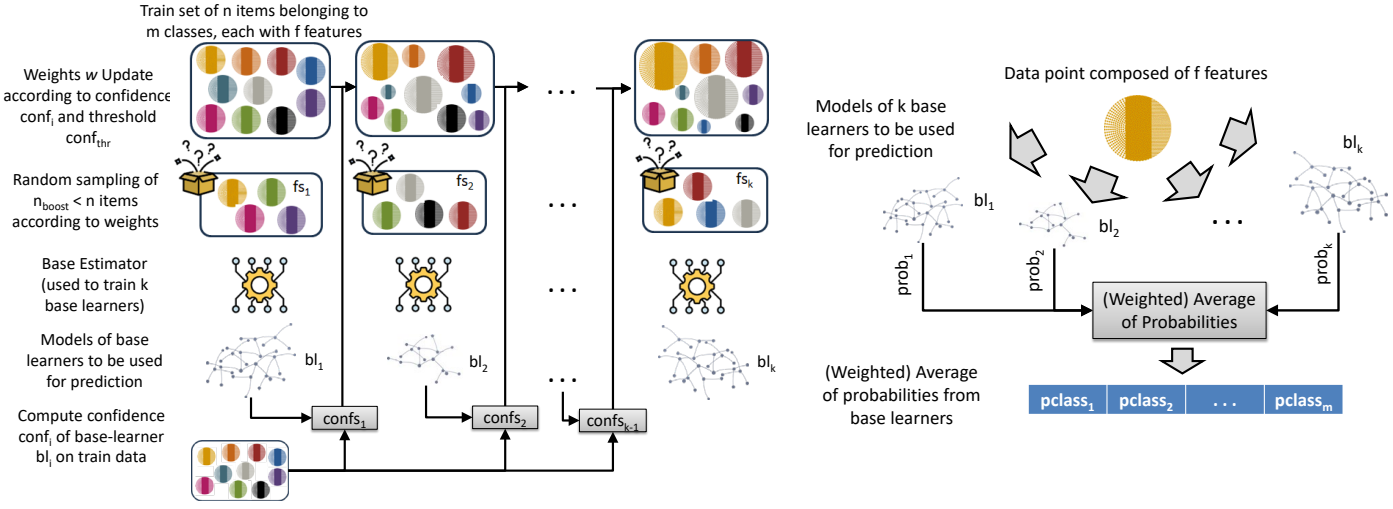


Figure 2: Train (left) and test (right) schemas for Confidence Boosting (ConfBoost) at a glance.

entropy of probabilities [71] or more complex confidence computations [44], [51]). The cognitive complexity for learning how to use confidence ensembles is very low and becomes negligible if the confidence ensembles are configured to use with default parameters. Given that confidence ensembles can virtually use any classifier as a base estimator, it is rather easy to quantify the improvements in classification performance when running a base estimator as a standalone classifier or when using it for building a confidence ensemble.

The downside of any ensemble method is bringing more complexity to the table: regardless of the implementation, the training and prediction times of confidence ensembles will be higher than those of the base estimator. While the former concern (setting up the ensemble) is a one-off overhead, the overhead due to generating multiple predictions cannot be avoided but can also be minimized by adequate multithreading programming and hardware support. Another potential concern may be related to the design of ConfBoost, which follows a boosting process but does not strictly require the base estimator to be a *weak* learner: the base estimator could even be a very heavy neural network which is far from being a weak learner according to its definition [61]. As per Shapire's definition, this may threaten the performance of ConfBoost, as base learners using strong base estimators may converge to the same model, learning the same things and thus providing very similar results, making ensembles ineffective. To tackle this, we remind how the study [42] highlights Data and Model Diversification as pillars of any classification ensemble. Confidence ensembles have their way of implementing data diversity, as they train the base learners using different subsets of the training set (and even subsets of features for ConfBag). The size of these subsets can be tuned: the smaller the subsets, the less overlap between different subsets and the less data a base learner would rely upon to learn its model. Considering very similar subsets may lead to crafting very similar base learners and be detrimental to the overall classification performance. Regarding model diversification, each base learner is an instance of the same base estimator (i.e., implements the same ML algorithm), but since it is trained using different portions of the training dataset, it should end up with a diverse (or unstable, using the terminology in [14]) behaviour.

3.5 Confidence as a Building Block

Last, but not least, we would like to emphasise the way the confidence is used for creating classifiers in this study, and why, to the best of our knowledge, this differs from the existing studies in literature. The confidence in classification is typically provided in the studies by others as an additional output to complement the class prediction, i) quantifying a margin [62] and understanding if a prediction should be rejected or should be trusted instead [63], ii) driving the selection of those classifiers that actually contribute to the inference process from a large classification ensemble (ensemble pruning, [64], [65]), or iii) crafting weights for weighted fusion of base-learners opinions to improve ensemble performance [64], [66], [67], [68].

Confidence ensembles build on these concepts as follows. ConfBag and ConfBoost have a *weighted_pred* parameter which, if true, makes different base-learners contribute to the final result according to the prediction confidence they have: those that are more confident will contribute more towards the ensemble prediction. Ensemble pruning techniques are integrated by design

Table I. Comparison of classifiers and fusion strategies with respect to the role of confidence in predictions. Ticks are starred when a capability depends on the parameter setup. The "static" tag is assigned to mechanisms that do not dynamically adapt decision-making to each individual prediction.

Classifier / Fusion Strategy	**Fits Unsupervised Learning**	**Confidence Used during Training**	**Weighted Contr. of Base-Learners**	**Ensemble Pruning**	**Built-in Prediction Rejection**
Bagging [13], [14]	✓		✓*		
Boosting [16], [73]			✓*		
EP-CC [75]			✓(static)	✓(static)	
EP-WL-CC [75]			✓(static)		
EP-WV-CC [75]				✓(static)	
Bayesian Voting [77]			✓	✓*	
Recovery Blocks [79], Delegating [80]	✓			✓*	✓
Cascade Generalization [79]	✓		✓*		
Stacking [40]	✓		✓*	✓*	
CEnsemble [18]		✓	✓*		
Pruning Individual Contribution [76]			✓(static)	✓(static)	
Learn++ [78]			✓(static)	✓(static)	
ConfBag (weighted=True)	✓		✓	✓	
ConfBag (weighted=False)	✓			✓	
ConfBoost (weighted=True)	✓	✓	✓	✓	
ConfBoost (weighted=False)	✓	✓		✓	

in both ConfBag and ConfBoost, and can be easily tuned via the parameters *bl_n*, *bl_perc* and *conf_thr* in both approaches. Importantly, the study [64] performs ensemble pruning exercising base-learners on a labelled pruning set (different from training and test sets), collecting their predictions, computing accuracy on the pruning set, and allowing only base-learners with high accuracy to take part in the ensemble prediction. Whereas this technique may be very useful, it has two important drawbacks: i) it needs a labelled pruning set, i.e. be applied to supervised learning only, whereas confidence ensembles may even work in unsupervised learning scenarios, and ii) tends to prune the same base-learners, if the pruning set does not vary, resulting in a static subset of base-learners that contribute to the decision, whereas confidence ensembles perform pruning at prediction level (i.e., different base-learners may contribute to each prediction). Regarding prediction rejection, confidence ensembles do not embed any mechanism to do so, as it happens in decision schemes such as recovery blocks [69] or delegating [70], but provide enough information for crafting such strategies.

Noticeably, all the mechanisms above assume that base-learners are already trained, and that the confidence is used only afterwards to fine-tune the fusion of the ensemble predictions. The most important innovation of ConfBoost is that the confidence is used to learn the model itself as an essential part of the training phase, altering the likelihood of data items to be selected for building the training set of the base-learners that are created iteratively. The only technical report (still unpublished) that uses confidence in learning the model is [18], which presents an approach that iteratively creates a chain of DNNs trained sequentially using only a subset of the training dataset for which

there is not enough confidence in predictions. ConfBoost, however, uses a different strategy to update the likelihood of the data points to be sampled for training base learners, allowing all data points (with non-uniform distribution) to take part in the learning process. The prediction mechanism of ConfBoost, too, is completely different from the proposal in [18].

These considerations are summarized in Table I above which shows the differences between confidence ensembles and the other known studies relying on confidence. Overall, confidence ensembles are designed to maximise the usage of confidence at inference time for optimal data fusion, even in unsupervised scenarios. In the case of ConfBoost, confidence is used also for learning the models of each base-learner in the ensemble, thus acts at both training and test time.

4 Experimental Campaign

This section describes the experimental analysis to answer the research questions stated in Section 4.1. The experimental methodology is outlined Section 4.2. Details on the experiments, assumptions, and inputs are detailed in Section 4.3 to Section 4.6.

4.1 Research Questions

RQ1. Do confidence ensembles outperform base estimators used for supervised or unsupervised classification of tabular data?

RQ2. Do confidence ensembles outperform traditional bagging and boosting ensemble classifiers, ideally using a restricted number of base learners?

RQ3. What is the impact of hyper-parameter values on the classification performance of confidence ensembles?

RQ4. What is the time overhead of confidence ensembles in comparison with base estimators?

RQ5. How do confidence ensembles compare with base estimators when dealing with unexpected inputs? Are they robust?

4.2 Methodology

Our experimental methodology had been deployed according to the following 4 steps.

S1. Select a set of supervised and unsupervised classifiers to compare against confidence ensembles. Unsupervised classifiers can only perform binary classification: when dealing with a dataset having data points of multiple classes, a conversion from a multi-class to a binary (normal vs others) label is needed to enable unsupervised classification.

S2. Prepare different instances of ConfBag and ConfBoost ensembles, varying their parameters to assess their capabilities on a range of configurations. These use supervised and unsupervised classifiers from step S1 as base estimators. For further comparisons, we crafted traditional bagging and boosting ensembles using the same base estimators and parameters as ConfBag and ConfBoost.

S3. Gather tabular datasets suitable for classification in real ICT systems. These could be datasets with either a binary or multi-class label.

S4. Create variants of datasets to evaluate the robustness of the classifiers to unexpected inputs (e.g. non-IID data points distributed differently from those in the training dataset).

Experiments have been conducted on a Dell Precision 5820 Tower with an Intel Xeon Gold 6250, GPU NVIDIA Quadro RTX6000 with 24GB VRAM, 192GB RAM and Ubuntu 18.04, NVIDIA driver 450.119.03 with CUDA 11.0.

Confidence ensembles are implemented in the library *conf-ensembles*, which is publicly available on GitHub (https://github.com/tommyippoz/confidence-ensembles [72]) and on PyPI (https://pypi.org/project/confidence-ensembles/). The usage of the library is detailed in Appendix A, and has interfaces similar to those of the well-known *scikit-learn* and *pyod* libraries.

4.3 Supervised and Unsupervised Classifiers

We selected 8 supervised classifiers that are widely used in the literature: 3 statistical classifiers as Naïve Bayes, Linear Discriminant Analysis, Logistic Regression and 5 tree-based classifiers that were benchmarked in [17]: Decision Trees, Random Forests, Extremely Randomized Trees, Logit Boost, and XGBoost. These are very diverse among themselves: some are ensembles, and others rely on very fast (but often unreliable) statistical mechanisms to predict classes. All classifiers are implemented in scikit-learn (https://scikit-learn.org) or in specific libraries whose interfaces comply with scikit-learn's.

Additionally, we selected 5 unsupervised classifiers that are implemented in *pyod* (https://pyod.readthedocs.io/, [73]): Histogram-based outlier score (HBOS), Isolation Forests, Cluster-Based Local Outlier Factor (CBLOF), Principal Component Analysis (PCA) and Isolation-based Anomaly Detection Using Nearest-Neighbor ensembles (INNE). Amongst the many alternatives from the library, we chose these since they rely on different heuristics, with some of them (Isolation Forest and INNE) being ensembles themselves.

This experimental evaluation aimed at benchmarking confidence ensembles against existing classifiers rather than reaching the best classification performance on a specific dataset. A preliminary and informal exploration did not reveal any major impact on the difference in performance between classifiers and confidence ensembles while varying classifiers' parameters. Thus, all classifiers/base estimators were run using their default *parameters*. As a special case, Logistic Regression provided multiple alerts of *failed convergence* during training, which we minimized using the following combination of parameters: *solver = 'sag'*, *max_iter = 1000* and *tol = 0.001*.

4.4 Confidence Ensembles

The 13 classifiers above will be used i) in isolation and as base estimators to build ii) confidence ensembles and iii) traditional bagging and boosting ensembles.

When crafting confidence ensembles, we varied the number of *estimators* (5, 10, 20), and the *weighted_pred* (true or false). For ConfBag, we set *f_bag* to (0.3, 0.5, 0.7), and *bl_perc* to (0.3, 0.5). For ConfBoost, we set the *lr* to 2, *sampling_ratio* (0.5, 0.3, 0.2), and make *conf_thr* range as (0.9, 0.8, 0.5). This choice was not arbitrary: it came after preliminary sensitivity analyses in which we found that assigning some parameter values always worsened classification performance. This happens when using *conf_thr* values below 0.5, when using exceedingly small subsets of the training set (*f_bag* below 0.3 or *sampling_ratio* below 0.2), or when the data for training base learners is not diverse enough i.e., *f_bag* above 0.7, *sampling_ratio* above 0.5). Regarding the number of estimators, we did not go beyond 20 as even a small number of base learners can build confidence ensembles that outperform other classifiers. All these setups will be applied for

each of the 13 base estimators from the previous section, resulting in a total of 13*162 = 2106 configurations of *ConfBag* and 13*54 = 702 of *ConfBoost*. We used the same parameters for training traditional bagging and boosting ensembles (*BaggingClassifier* and *AdaBoostClassifier* from scikit-learn) using the 13 classifiers as base estimators as this is required to answer RQ2. Noteworthy, confidence-related parameters as *conf_thr, weighted_pred* are not relevant for traditional ensembles and thus were not used there. Also, the *AdaBoostClassifier* could not be applied to unsupervised classifiers nor the supervised LDA due to bugs in the library (i.e., *pyod* classifiers are missing some required attributes to be used in some of the *scikit-learn*'s function).

4.5 Error, Attack and Failure Datasets

We selected 23 datasets for this study: 12 datasets of network traffic to be used for intrusion detection, 5 hardware monitoring datasets for failure prediction, and 6 datasets about error and anomaly detection in IoT and Industrial Control systems.

Table II summarizes the 23 datasets, reporting the domain, name, year, number of data points, number of features, and categories of anomalies, errors, or attacks. All datasets were labelled, structured in CSV format, and cropped to contain 200 000 data points at most, to speed up the evaluation. Datasets are publicly available and can be downloaded using the provided references.

4.5.1 Network Intrusion Detection (NIDS)

We selected labelled datasets on network intrusions looking at surveys, Kaggle, UCI, Zenodo, IEEEDataport and other online portals. Our selection process resulted in the following datasets: ADFANet [74], AndMal17 [75], BAIoT Doorbell [76], CICIDS17 [77], CICIDS18 [77], CIDDS [78], IoT Network [79], ISCX12 [80], NSLKDD [81], SDN20 [82], UGR16 [83], UNSW-NB15 [84]. All those datasets included records of normal traffic and records collected while the system was under attack. Features are mostly numeric, extracted by monitoring the network traffic in packets (e.g., bytes received per second, number of packets).

4.5.2 Hardware Failure Prediction

We gathered datasets related to monitoring hard disks, where the performance indicators and the status (failed or not) of a hard drive are logged as data points. The BackBlaze [85] manufacturer made many years of hard drive data available to the public. Another source of hard drive data came from the BAIDU [86] competition whose input datasets are still available.

4.5.3 Error/Anomaly Detection

The last group of datasets we considered came from monitoring IoT or industrial control systems (ICS): a distributed control systems of a power plant controlling a turbine [87], [88], malfunctions of metros in Portugal [89], railroad trucks equipped with sensors to monitor brake pressure [90], an edge device monitored for errors [91], the mechanical failure of electrical machinery in power plants [92].

4.5.4 Creating Datasets Variants

The RQ5 is devoted to quantifying the robustness of classifiers to *unexpected inputs* (see Section 2.6). Robustness is usually difficult to quantify but at the same time one of the most desirable properties of classifiers, especially those that may be deployed in real environments.

Table II. Name, release year, number of attack types, number of portions, and the amount of ordinal features *f* of used datasets.

Domain	Dataset Name	Year	Categories of Anomalies	# Features	Number of Data Points
Network Intrusion Detection	ADFANet	2015	5	3	132 002
	AndMal17	2017	4	75	100 522
	BAIoT Doorbell	2018	5	115	75 165
	CICIDS17	2017	4	75	200 000
	CICIDS18	2018	5	75	200 000
	CIDDS	2015	4	7	200 000
	IoT Network	2019	9	8	210 425
	ISCX12	2013	4	6	200 000
	NSLKDD	2009	4	37	148 517
	SDN20	2020	5	78	205 128
	UGR16	2016	5	7	207 256
	UNSW-NB15	2015	8	38	165 461
HW Monitor	BackBlaze 2017	2017	1	50	32 678
	BackBlaze 2019	2019	1	44	47 525
	BackBlaze 2021	2021	1	37	44 950
	BackBlaze 2023	2023	1	35	70 512
	BAIDU	2017	1	12	186 049
Error / Anom. Detection	Arancino Device	2023	9	119	154 000
	HAI Pressure	2019	1	54	200 000
	HAI ICS	2023	1	224	54 000
	Mechanical Failure	2018	1	18	7 906
	Metro PT	2022	2	20	173 824
	Scania Trucks	2016	1	170	76 000

In our experiments, unexpected inputs are anomaly, error, or attack categories that only appear in the test set. This makes the non-IID with respect to training data. We simulated unexpected inputs by removing specific categories from the training set of each dataset that has more than a single "anomalous" class. This allowed for the creation of as many training variants as the categories of anomalies (see Table II) contained in each dataset. The label used for training variants is always binary (normal vs anomaly): it will not be possible to have correct multi-class misclassification of a data point belonging to a class that does not appear in the training set. Importantly, the test set remains unchanged: classifiers trained on different training sets or training variants of a specific dataset will be validated using the same test set, in which unexpected inputs may be present, labelled as anomalies. This is similar to the process followed in [6].

4.6 Performance Metrics

The classification performance of classifiers is usually measured through the confusion matrix and the compound metrics derived from it such as accuracy i.e., the fraction of correct predictions over all predictions.

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

However, accuracy may deliver misleading results when datasets are unbalanced [93], which happens frequently in anomaly, error and intrusion detection: therefore, we mostly relied upon the Matthews Correlation Coefficient (MCC),

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

which was primarily developed for binary classification but adapts well to multi-class classification. The MCC quantification

allows for discussing RQ1, RQ2 and RQ3. For completeness, we report results using other well-known metrics for classification (i.e., Accuracy and F-Measure, or F1) in Appendix C.

To answer RQ4, we kept track of the training and the test time (in milliseconds) needed by each classifier, and of the size of the resulting model. Regarding RQ5, we computed the MCC of all the models learned using variants of the training set used to simulate the occurrence of unexpected inputs. Then, we quantified the recall, or coverage of unexpected inputs being correctly detected as a metric for robustness. We call this metric *rec-unk*.

$$rec - unk = \frac{\#\ of\ correctly\ classified\ unexpected\ inputs}{\#\ of\ unexpected\ inputs}$$

5 Results, Analysis, and Discussion

This section presents and discusses the results of our experimental campaign in light of the research questions RQ1 to RQ5 from Section 4.1.

5.1 RQ1 - Classification Performance of Confidence Ensembles

RQ1 aims to assess if the *confidence ensembles outperform the base estimators when performing supervised or unsupervised classification*.

5.1.1 Comparison against other Classifiers in the Study

Table III summarises the findings for each of the 13 classifiers in our experiments, and of ConfBag and ConfBoost ensembles using those classifiers as base estimators. Quantities in the table are an average of the maximum MCC score obtained on each dataset by base estimators and confidence ensembles, respectively, after trying all the possible parameters' combinations. For a fair comparison of unsupervised and supervised classifiers, the scores in the table refer to a binary classification, i.e. normal vs anomalies in all datasets. On the right-hand side of the table, we report the gain in MCC when using a confidence ensemble in comparison with a base estimator: the red background of cells points to a negative gain (i.e., a reduction of classification performance), while the green cells highlight the combinations for which the confidence ensembles provide an actual improvement in classification performance.

Looking at unsupervised classifiers in the top half of the table, it is quite clear that the confidence ensembles are a direct upgrade to base estimators. This is especially evident for ConfBoost, whereas ConfBag does not always show visible improvements (i.e., ConfBag has worse MCC than the base estimator PCA). ConfBoost, on the other hand, provides an outright improvement over the unsupervised classifier. This is a *significant result* as unsupervised classifiers are usually known for their poor classification performance: finding a generic solution that flat-out improves them is of utmost importance.

For supervised classifiers in the second (lower) half of Table III, there is a clear distinction between simple classifiers as Naïve Bayes, LDA, Logistic Regression, Decision Trees and ensemble methods LogitBoost, ExtraTrees, RandomForest and XGBoost. ConfBag often struggles to match the performance of ensemble methods: see negative gains in the bottom 4 rows of the table. Conversely, ConfBoost always outperforms the base estimator by a fair – and sometimes huge – amount. We note that tree ensembles are considered top choices for the classification of tabular data: thus, improving classification scores of Extra Trees, Random Forests, or XGBoost even by a small amount is still an important achievement.

More in detail, a ConfBoost of Decision Trees scored the highest average MCC of 0.805 over all classifiers, outperforming Random Forests (0.796) and XGBoost (0.791), and even their ConfBoost variants (MCCs of 0.798 In both cases). Tree ensembles are based on multiple decision trees (Extra Trees, Random Forests) or stumps (LogitBoost, XGBoost), which are created according to well-known procedures that for boosting heavily rely on labels. As Table III suggests, using confidence in predictions allows ConfBoost to have more discriminative power

Table III. Comparison of classification performance of supervised and unsupervised classifiers against ConfBag and ConfBoost ensembles using classifiers as base estimators. Bolded numbers highlight the highest MCC for a specific base classifier and corresponding ensembles.

	Classifier	*Average MCC of Base Estimator and Confidence Ensembles*					*MCC Gain of Confidence Ensembles w.r.t. Base Estimator*				
		Base Estimator	**ConfBag (weighted=False)**	**ConfBag (weighted=True)**	**ConfBoost (weighted=False)**	**ConfBoost (weighted=True)**	**ConfBag (weighted=False)**	**ConfBag (weighted=True)**	**ConfBoost (weighted=False)**	**ConfBoost (weighted=True)**
Unsupervised	CBLOF	0.232	0.225	0.233	0.279	**0.286**	-0.007	0.001	0.047	0.054
	Isolation Forest	0.250	0.251	0.265	**0.320**	0.317	0.001	0.015	0.070	0.067
	INNE	0.165	0.217	0.201	**0.269**	0.224	0.052	0.036	0.104	0.059
	HBOS	0.215	0.221	0.222	0.281	**0.283**	0.006	0.006	0.065	0.067
	PCA	0.288	0.265	0.267	**0.311**	0.303	-0.023	-0.021	0.023	0.015
Supervised	GNB	0.415	0.459	0.463	0.507	**0.517**	0.045	0.048	0.093	0.102
	Logistic Regression *	0.281	0.259	0.256	0.424	**0.427**	-0.022	-0.025	0.142	0.145
	LDA	0.447	0.462	0.448	**0.604**	**0.604**	0.015	0.001	0.157	0.158
	Decision Tree	0.747	**0.805**	0.795	**0.805**	0.802	0.058	0.046	0.058	0.054
	Logit Boost	0.753	0.746	0.75	0.765	**0.768**	-0.007	-0.003	0.012	0.015
	Extra Trees	0.773	0.762	0.772	0.778	**0.782**	-0.011	-0.001	0.005	0.009
	Random Forest	0.796	0.762	0.779	**0.798**	0.795	-0.034	-0.017	0.002	-0.001
	XGBoost	0.791	0.756	0.767	**0.798**	0.796	-0.035	-0.026	0.007	0.005
	Averages	*0.473*	*0.476*	*0.478*	***0.534***	*0.531*	*0.003*	*0.005*	***0.060***	*0.058*

* Logistic Regression delivered multiple alerts of failed convergence in many datasets regardless of the combination of parameters we tried.

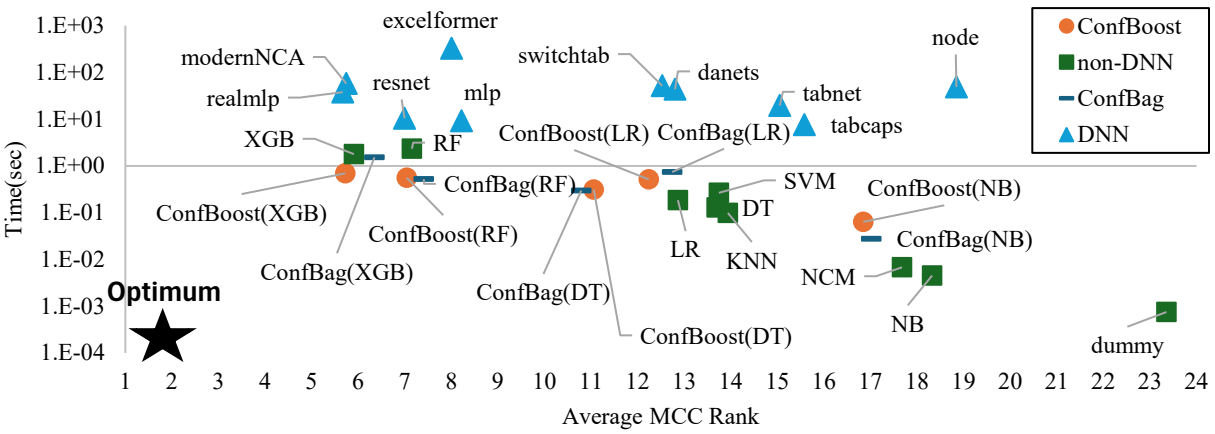


Figure 3: Results using the TALENT benchmark: ConfBag (blue dashes) and ConfBoost (orange circles) are exercised alongside non-DNN (green squares) and DNN (light-blue triangles) classifiers. The plot shows the MCC rank against training time, averaged across 100+ datasets.

than its competitors, making it applicable also to scenarios where labels are not available.

5.1.2 Benchmarking using TALENT

Our experimental analysis already includes many datasets and many classifiers to be used individually and as base estimators for confidence ensembles. However, in recent years there has been a quick development of benchmarking frameworks for classification and regression tasks for tabular datasets that cannot be ignored. Thus, we benchmarked confidence ensembles, both ConfBag and ConfBoost, within one of these frameworks, LAMDA-TALENT [32], which is available at https://github.com/qile2000/LAMDA-TALENT. The exact steps we followed to perform the benchmark are detailed in the conf-ensembles repository [72], which contains all the additional configuration files and scripts we crafted to exercise confidence ensembles within LAMDA-TALENT. In short, we extended the base classes to craft new classifiers, adding default configuration parameters and those that are selected in the optimization phase through grid searches, which are automatically executed within the framework. Also, the current version of the framework misses a script for plotting the figures that are in the paper [32], thus we logged metric scores for classifiers and datasets as CSV files, and edited them as Excel worksheets for data cleaning and plotting. Out of all the datasets available in the benchmark, we filtered out those related to regression tasks, and those that were triggering issues while training some classifiers (likely due to imperfect management of categorical features and missing values), which are only a few.

The results are shown in Figure 3, which plots scores of different classifiers, averaged across all datasets in the benchmark. Mimicking the documentation of LAMDA-TALENT, on the horizontal axis we report the MCC rank (i.e., rank *i* means that the classifier has the *i-th* best MCC on a dataset), where a lower rank indicates better performance, while the vertical axis shows the average training time in seconds, on a logarithmic scale. The results obtained for the benchmark are consistent with the results from Section 5.1.1: for some base estimators as Decision Tree (DT), Logistic Regression (LR), Naïve Bayes (GNB), both ConfBag and ConfBoost provide a far better classification performance (i.e., depicted closer to the left) than base estimators. For stronger learners as XGB and RF, ConfBoost allows for faster training times for either the same or a slightly improved classification capability. DNNs are behind in the benchmark, requiring more time to train and also having worse classification performance.

5.2 RQ2 – Comparison with other Bagging and Boosting Ensembles

The results in the previous section suggest that confidence ensembles have the potential to improve classification performance against existing classifiers but does not show how they compare against alternative existing bagging and boosting ensembles. When analysing any ensemble learner, it is important to focus on the number *k* of base learners that are trained and then used at inference time, slowing down inference. Table IV quantifies the MCC of confidence ensembles and regular bagging / boosting ensembles for supervised base estimators, varying the amount of base learners *k*. We left unsupervised base estimators out of the comparison as it is not possible to apply traditional boosting techniques to unsupervised classification by design, thus we would have missed the *AdaBoostClassifier* scores.

First, we can observe how confidence ensembles outperform traditional bagging and boosting ensembles in the vast majority of cases. Starting from the top of the table, the GNB classifier has an average MCC of 0.415 (the column with k=1, or single learner, has the same value as the "base estimator" in Table III), with traditional bagging it goes up to 0.469, with boosting it stops at 0.462, with ConfBag goes to 0.470, and with ConfBoost reaches 0.516. Furthermore, in general for the GNB, Logistic regression, LDA, Decision Tree classifiers we observe that:

- all ensembles outperform their base estimator;
- ConfBag has scores similar to the traditional *BaggingClassifier;*
- ConfBoost clearly outperforms traditional boosting, and it is almost always the top performing ensemble.

Second, we read the table as follows: the more a "best" score leans towards the right side, the more base learners are needed and thus the more time and resource intensive the classification process will be. The MCC scores on the bottom right of the table are those of supervised ensembles Logit Boost, Extra Trees, Random Forests, XGBoost that we ran with default parameter values, which is k=100 base learners. These supervised ensembles are considered the top performing classifiers and have average MCC as high as 0.796 (Random Forest with 100 base learners, see second-last row on the extreme right of Table IV). The reader should notice how ConfBag and ConfBoost ensembles of Decision Trees reach and even exceed this MCC score using only k=10 base learners (4th column, 20-21th rows of the table), and further

Table IV. Average MCC of confidence ensembles against supervised ensemble classifiers varying the number k of base learners.

k (# base learners)	1	5	10	20	100
GNB					
BaggingClassifier	0.415	0.469	0.468	0.469	
AdaBoostClassifier	0.415	0.422	0.425	0.462	
ConfBag	0.415	0.464	0.470	0.467	
ConfBoost	0.415	0.464	0.496	**0.516**	
Logistic Regression					
BaggingClassifier	0.281	0.310	0.299	0.308	
AdaBoostClassifier	0.281	0.271	0.277	0.291	
ConfBag	0.281	0.325	0.315	0.291	
ConfBoost	0.281	0.362	0.397	**0.425**	
LDA					
BaggingClassifier	0.447	0.445	0.441	0.443	
AdaBoostClassifier	n.a.	n.a.	n.a.	n.a.	
ConfBag	0.447	0.456	0.444	0.449	
ConfBoost	0.447	0.542	0.589	**0.609**	
Decision Tree					
BaggingClassifier	0.747	0.768	0.784	0.791	
AdaBoostClassifier	0.747	0.754	0.768	0.769	
ConfBag	0.747	0.775	0.796	**0.804**	
ConfBoost	0.747	0.782	0.799	0.803	
Ensembles					
Logit Boost		0.715	0.733	0.743	**0.753**
Extra Trees		0.756	0.763	0.769	**0.773**
Random Forest		0.776	0.782	0.789	**0.796**
XGBoost		0.717	0.752	0.777	**0.791**

← MCC using default estimators=100 (see Table III)

improve with k=20 (5th column, same rows), reaching an average MCC of 0.804. This *confirms the view that confidence ensembles are more accurate than the existing classifiers, and that they also require crafting significantly fewer base learners*, which is a huge achievement as the resulting model will require far less resources than those needed by the existing supervised ensembles.

5.3 RQ3 – Impact of Hyper-Parameters

This section illustrates how different values assigned to hyper-parameters affect the performance of confidence ensembles. Parameters such as *k, conf_thr, f_bag* and *weighted_pred*, may have an impact to ConfBag, while *k, conf_thr, sampling_ratio* and *weighted_pred* have a significant impact on ConfBoost.

We quantify and depict the impact of the various parameters in Figure 4, which plots the average MCC achieved by ConfBag (left) and ConfBoost (right), respectively, while varying the values of hyper-parameters listed above. Down-facing black arrows in each plot point to the highest column, corresponding to the best overall classification performance. For ConfBag (left of Figure 4), we observe bars that are progressively growing from front to bottom: the first three lines of bars correspond to a *conf_thr = 0.5*, while the farther ones correspond to *conf_thr = 0.7*. Also, bars in the middle of the plot (*k=10*) are higher than those on the left (*k = 20*) and on the right (*k=5*); having a *weighted_pred = True* (see where black arrows point to) usually allows for an additional slight improvement of classification performance. Overall, we conclude that ConfBag ensembles of *k=10* base learners with *weighted_pred=True* and *conf_thr* ≥ *0.7* should be preferred over other combinations.

For ConfBoost (right of Figure 4), we observe slightly different trends. First, the biggest impact on classification performance is due to the number of base learners. From front (*k* = 5) to bottom (*k* = 20) of the plot, we can notice bars becoming higher and higher, with major jumps between the second (orange-patterned cylinders) and third (striped parallelepipeds) series, and from the fourth (yellow cylinders) to the fifth (blue parallelepipeds) series; these mark the increases from 5 to 10 and from 10 to 20 base learners. Using a *weighted_pred* = True does not seem to have a major impact: the 1st, 3rd, 5th series (non-weighted prediction) are almost on par compared to the others using weighted predictions. The *sampling_ratio* has also a noticeable effect: if low (0.1, see right of the plot), it has a very detrimental impact. However, there are no major differences when using a sampling ratio of 0.3 or 0.5 (middle and left of the plot). Overall, we conclude that ConfBoost ensembles of *k* ≥ *20* base learners with *sampling_ratio* ≥ *0.3* should be preferred over others.

As a final remark in this section, we acknowledge that these findings are based on the "average case": the optimal performance in specific case studies may be reached using different combinations of parameters. Nevertheless, the results in Figure 4 are useful to derive general guidelines and for setting up default parameters for confidence ensembles.

5.4 RQ4 – Time Overhead

The main downside of ensemble learning is that resulting models are more complex and thus always require more time than individual non-ensemble classifiers. Confidence ensembles are no different: the higher the *k*, the more base learners are used in the ensemble, the more time is needed for their training. We note that

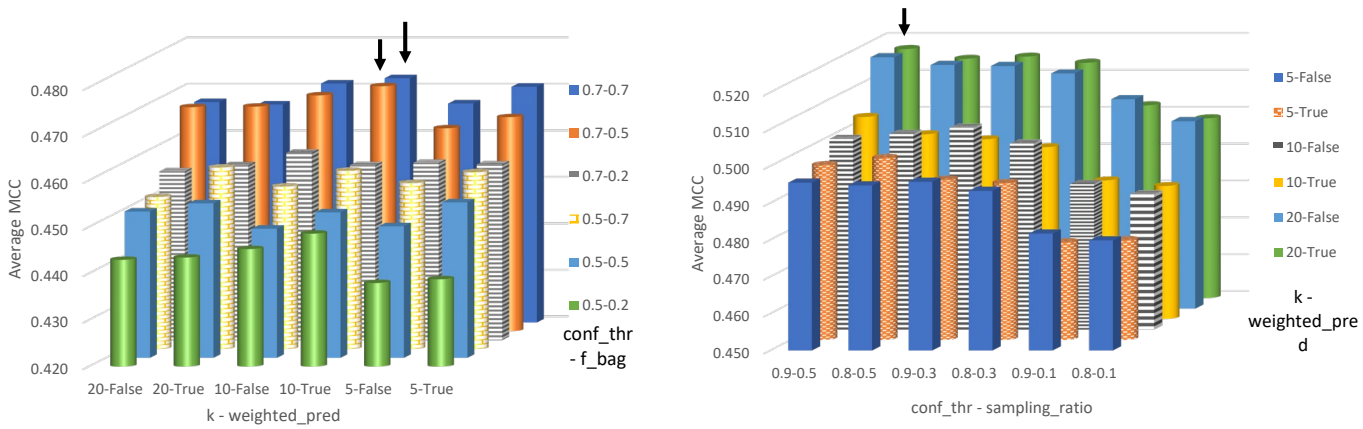


Figure 4: Variation of MCC scores for different combination of hyperparameters for ConfBag (left) and ConfBoost (right) ensembles.

training a classifier is typically considered a one-off overhead which, even if significant, may be acceptable. The need to retrain a classifier may occur, but is rare, e.g. to respond to new classes, or to match distribution shifts, which may emerge after the classifier has been deployed.

Figure 5 shows the train times required for a Decision Tree (DT) and for ConfBag and ConfBoost ensembles built using DT as a base estimator. We choose DT as it resulted in the highest MCC in Table III and Table IV: the expectation is that this trend will hold in terms of the relative increase in training time when changing the base estimator. The average time needed to train a DT in the 23 datasets in this study is 563 ms (left of Figure 5): a rough expectation could then be that confidence ensembles of size *k* will require k*563 ms for training i.e., if k=20, the train time may be 20*563 = 11260 ms. However, we can observe that ConfBoost ensembles with k=20 base learners (see labels over the x-axis) have a training time that ranges between 5000 and slightly over 8000 ms. This is because parameters *f_bag* and *sampling_ratio* affect the amount of data used to train each base learner, and thus the overall training time. Another important observation is that the times shown in Figure 5 were computed running experiments as a single-thread process, blocking parallelization. Training of ConfBag base learners can be parallelized which can lead to significant reduction of the overall time on training the base learners depending on the capabilities of the specific machine used for training.

The most significant overhead of using ensembles, and confidence ensembles in particular, is the one that occurs during inference. Similarly to training, this is proportional the number of base learners in the ensemble. However, the discussion of RQ2 (Section 5.2) suggests that the usage of a limited number of base learners may still be sufficient to provide improved classification capabilities of confidence ensembles, generating less overhead and requiring fewer resources than other ensemble classifiers.

5.5 RQ5 – Robustness to Unexpected Inputs

The last research question concerns the robustness to non-IID data or unexpected inputs of confidence ensembles. We quantified that by training and testing all classifiers in our study, including the confidence ensembles, using the dataset variants as defined in Section 4.5.4. We aggregated the scores obtained on all dataset variants and all classifiers in Table V, which reports *rec-unk* scores

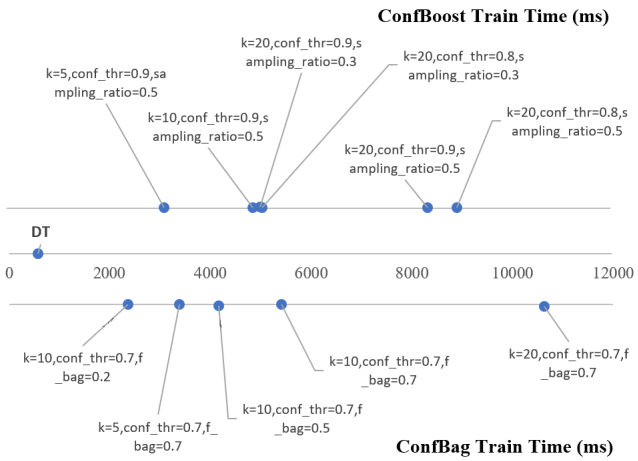


Figure 5: Variation of Train Time for different instances of ConfBag (down in the figure) and ConfBoost (up) built over Decision Trees as base estimator (DT, middle of the plot)

Table V. Comparison of rec-unk of supervised and unsupervised classifiers against confidence ensembles in the presence of unknowns. The numbers in bold highlight the highest rec-unk for a specific base estimator and corresponding confidence ensembles.

	Classifier	Base Estimator	ComfBag (weighted=False)	ComfBag (weighted=True)	ConfBoost (weighted=False)	ConfBoost (weighted=True)
Unsupervised	CBLOF	0.461	0.466	0.464	**0.486**	0.427
	Isolation Forest	**0.464**	0.416	0.430	0.442	0.451
	INNE	0.457	0.464	0.469	0.490	**0.508**
	HBOS	**0.425**	0.372	0.393	0.413	0.405
	PCA	0.430	0.427	0.414	0.422	**0.432**
Supervised	GNB	0.441	0.536	0.555	**0.583**	0.579
	Logistic Regression *	0.253	0.288	0.265	0.510	**0.514**
	LDA	0.326	0.433	0.480	0.598	**0.599**
	Decision Tree	0.426	0.512	0.490	**0.667**	0.661
	Logit Boost	0.434	0.480	0.483	0.562	**0.585**
	Extra Trees	0.412	0.480	0.481	0.622	**0.631**
	Random Forest	0.424	0.485	0.487	0.640	**0.658**
	XGBoost	0.434	0.491	0.495	0.599	**0.644**

* Logistic Regression delivered multiple alerts of failed convergence in many datasets regardless of the combination of parameters we tried.

of base estimators (one per row) and confidence ensembles.

Unsupervised classifiers (base estimators in the first half of the table, 2nd column) have a better *rec-unk* than the supervised classifiers even with lower classification performance overall (see Table III). This confirms the results of field studies [4], [5], pointing to a better capability of dealing with unknowns of unsupervised classifiers than supervised counterparts.

Deploying ConfBoost ensembles over supervised base estimators makes for a significant increase of *rec-unk*. This improvement is significant for Decision Trees (0.426 to 0.667), ensembles as Random Forests (0.424 to 0.658) and XGBoost (0.434 to 0.644). A ConfBoost with Decision Trees as base estimator has an average rec-unk of 0.667, meaning that exactly two-thirds of unexpected inputs are being correctly classified. This is a very significant improvement from the 4 out of 10 (rec-unk of 0.426) obtained by using decision trees alone.

Throughout years, researchers struggled to combine supervised and unsupervised classification mechanisms to obtain a unique complex classifier with optimal classification performance (that unsupervised classifiers do not achieve, when labels are available) and high robustness (which supervised classifiers often lack) [60], [41]. However, there is still no solution that is easy to deploy without excessive complexity and that consistently outperforms its competitors. Our results clearly suggest that ConfBoost ensembles create both *more accurate* and *more robust* classifiers thanks to their training process that relies on the concept of confidence in the classifier's predictions.

6 A Statistical Validation using Confidence Intervals

This section provides a statistical validation of the performance of confidence ensembles and their base estimators.

6.1 Computing Confidence Intervals

Rather than using the measured MCC value, as is done in the previous section, we account for the *uncertainty* attached to these point estimates, which is due to randomness in the training and testing processes. The analysis is based on *confidence intervals* computed for each point estimate of MCC. Confidence intervals were computed using the code used in [94] (available at *https://github.com/yukiitaya/MCC*) which computes *Simple, Fisher* and *Zou* confidence intervals [95] for a given confusion matrix (from which MCC point estimates have been computed) at a given confidence level. As suggested in [96], "*when the means of two independent samples are to be presented (graphically), it is a common practice to accompany the two points by error bars giving the 95% confidence intervals for each mean*". Thus, we set a 95% confidence level and computed the confidence interval for all classifiers and all datasets as a range

$$CI_{MCC} = [MCC_{pe} - \frac{c_{int}}{2};\ MCC_{pe} + \frac{c_{int}}{2}]$$

where MCC_{pe} is the point estimate of an MCC score calculated in classifier's testing, and c_{int} is the tightest of the three confidence intervals above. The resulting CI_{MCC} range will contain the "true MCC value" with a probability of 95%.

6.2 Possible Outcomes of the Analysis

For each combination of 23 datasets and 13 base estimators, we computed the confidence interval i) using the MCC_{pe} of the base estimator $CI_{MCC}(base)$, and ii) using the highest MCC_{pe} of ConfBag and ConfBoost in that combination i.e., $CI_{MCC}(ens)$. Comparing $CI_{MCC}(base)$ against $CI_{MCC}(ens)$ may result in one of the following outcomes.

There is no statistically significant difference between MCC scores of the base estimator and confidence ensembles when $CI_{MCC}(base)$ and $CI_{MCC}(ens)$ intervals overlap. With the chosen confidence level of 95%, the null hypothesis "the two-point estimates are identical" is statistically significant in this case. Conversely, $CI_{MCC}(base)$ and $CI_{MCC}(ens)$ may not overlap, leading to reject the null hypothesis. Confidence ensembles are better than the base estimator whether the interval $CI_{MCC}(ens)$ occupies a range of values that are strictly greater than the values occupied by $CI_{MCC}(base)$. The opposite happens when $CI_{MCC}(ens)$ occupies values strictly smaller than those of $CI_{MCC}(base)$; here, confidence ensembles are worse than the base estimator.

These 3 cases are illustrated in Figure 6 with examples from the study. From the left of the figure, applying CBLOF as the base estimator and for creating confidence ensembles in the ADFANet dataset resulted in overlapping $CI_{MCC}(ens)$ and $CI_{MCC}(base)$ intervals. The difference between the respective MCC point estimates is not statistically significant: there is *no evidence* that the confidence ensembles outperform the base estimator. The example in the middle of Figure 6 shows that confidence ensembles, built using PCA as a base estimator perform better than the base estimator, while the opposite is depicted on the right of the figure, where the base estimator Extra Trees has better classification performance than confidence ensembles on the ARANCINO dataset.

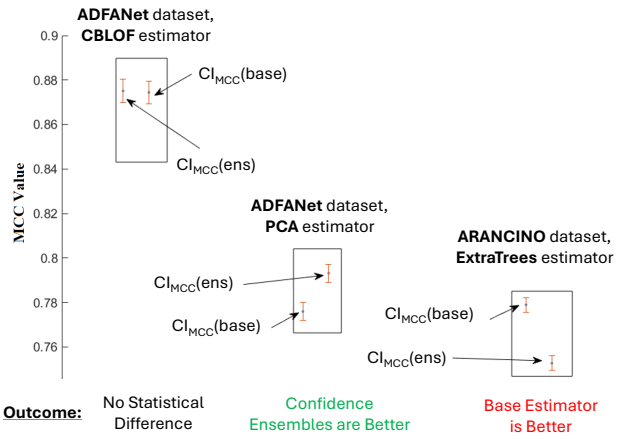


Figure 6: Examples of possible outcomes of the statistical validation: i) no statistical difference (left), ii) confidence ensembles are better than the base estimator (center), and iii) the base estimator is better than confidence intervals (right).

Considering all the combinations of datasets and base estimators in our experimental study, we obtain the following:

- there is no statistically significant difference between the confidence ensembles and the base estimator in 53 out of 285 (18.6%) combinations.
- Confidence ensembles are better than the respective base estimators in 222 out of 285 (77.9%) combinations.
- Confidence ensembles are worse than the base estimators in only 10 out of 285 (3.5%) combinations.

These results show further evidence of the superiority of confidence ensembles over the respective base estimators. The benefits are evident not only via the experimental MCC point estimates from Section 5, but also after accounting for the uncertainty in the MCC scores. In the vast majority of the cases (96.5%) the confidence ensembles are *not worse* than the respective base estimators and in a significant proportion of combinations (~78%) offer an outright benefit.

6.3 Impact of Confidence Ensembles on Base Estimators

As a last contribution, we analyse the statistical improvement of confidence ensembles over individual base estimators, depicted in Figure 7. From top to bottom of the bar chart, we depict red slices when the base estimators are statistically better than confidence ensembles, light-grey slices when there is no

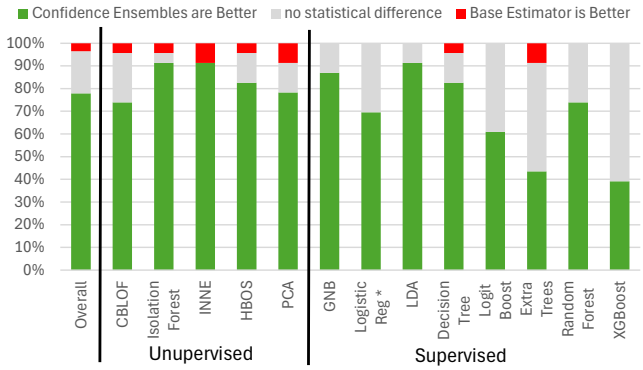


Figure 7: The bar chart shows the percentage of cases in which confidence ensembles are statistically better (green), worse (red), no statistical difference (light-grey) against base estimators.

significant statistical difference, and green slices when confidence ensembles are better than the base estimators.

The *overall* bar on the left of the chart summarizes the overall behaviour reported at the end of Section 6.2. When unsupervised classifiers are used as base estimators, the green slice grows more, simultaneously reducing the light-grey area. However, the most outstanding results can be observed for GNB, Logistic Regression, LDA, Logit Boost, Random Forests, and XGBoost base estimators, for which *confidence ensembles are never statistically worse than base estimators and still being beneficial in most of the cases* (i.e., no red slice). This is a critical improvement as decision tree ensembles as Random Forests and XGBoost are the de-facto standard for classification in tabular data [2], [16], [17], [27], [28] as they typically outperform their competitors. The two bars on the extreme right of Figure 7 show that confidence ensembles of Random Forests or XGBoost can further improve the performance of these two top-notch base estimators, with no drawbacks.

7 Threats to Validity and Reproducibility

We report here possible limitations to the validity and the applicability of our study.

7.1 Internal Validity

Internal validity is concerned with factors that may have influenced the results, but they have not been thoroughly considered in the study.

First, classifiers have hyperparameters whose tuning critically affects the classification performance: as explained in Section 4.3, this is not relevant for the base estimators, but it is for ConfBag and ConfBoost. Therefore, we exercise confidence ensembles under different parameters' combinations and discuss the results of these sensitivity analyses in Section 5.3 (RQ3). We discuss our plans for future work on confidence threshold in section 8 below.

Second, each classifier may encounter a wide variety of problems during training (e.g., under/overfitting, poor quality of features, feature selection to leave out noisy features, etc.). These problems may have a noticeable impact on the classification performance of a classifier. The use of multiple tabular datasets with different features helped us mitigate this problem, as the envisaged problems are unlikely to affect all datasets. In cases where we found that these problems were recurring consistently, we made sure to find solutions through a careful tuning of hyper-parameters, as with Logistic Regression.

7.2 External Validity

We cannot claim validity of this study for classifiers and domains other than those that we used in this study**.** One of the authors of this paper already conducted a preliminary analysis [18] using a technique similar to ConfBoost, and applying it using DNN base estimators for image classification. One of the major limitations was that the number of (DNN) estimators needed to be kept very low to avoid incurring in lack of resources or unfeasibly lengthy training and inference times. Since tree-based classifiers typically outperform or at least have comparable classification performance with DNNs for tabular data classification [2], [27], [28], [31], [32], there was no need to deploy confidence ensembles over DNNs. Indeed, confidence ensembles are agnostic to the base estimator by design, provided that there is a way to estimate the confidence in a specific prediction. Building confidence ensembles with DNNs as base estimators will result in resource-hungry classifiers that would limit their applicability in resource-constrained systems, which is a frequent condition in embedded or IoT systems, to name a few. Therefore, confidence ensembles using DNN base estimators should be used only in domains, mostly computer vision as image classification, where DNNs are clearly outperforming other competitors.

Theoretically, confidence ensembles may also be generalized to regression problems, but this process will require domain-specific knowledge of regression problems and is outside the scope of this paper.

7.3 Reproducibility

The usage of public data and public frameworks to build classifiers was a prerequisite for our analysis to enable the reproducibility of the analysis and scrutiny of the findings. All software used to obtain results presented therein is publicly available on GitHub [72] and PyPI, including the scripts to reproduce the experiments (see Appendix A for further descriptions). The framework does not include the datasets we used due to IP constraints; however, the manuscript provides the reader with references to all datasets, which are available to download at the owner/publisher's websites.

8 Conclusions and Future Work

This paper introduced *confidence ensembles*, ensembles of Machine Learning classifiers that use confidence in predictions within the training process to learn their model. Confidence ensembles craft their base learners as multiple instances of an existing classifier, (i.e., base estimator), without requiring any additional input by the user. Confidence estimation is considered model-agnostic, enabling the application of confidence ensembles to any classification task. The confidence ensembles ConfBag and ConfBoost stem from traditional bagging and boosting, respectively, but their behaviour is noticeably different compared to baselines due to employing confidence in learners' predictions.

After designing ConfBag and ConfBoost, we set up an experimental campaign that uses more than 20 public datasets collected by monitoring real or simulated ICT systems and infrastructures. Our studies include many supervised and unsupervised classifiers used for processing *tabular data*, a common data type in real-world applications, despite the formulation of confidence ensembles makes them applicable also to other classification (e.g., image) tasks. After discussing research questions RQ1 to RQ5, we found that:

- (RQ1) confidence ensembles are typically more accurate than the base estimators they are derived from. Especially, ConfBoost is a flat-out upgrade for any unsupervised classification task and allows also to improve the performance of tree-based supervised classifiers, which are the preferred choice for tabular data classification.
- (RQ2) compared to traditional bagging and boosting ensembles, ConfBag and ConfBoost require fewer base learners to provide improved classification performance. This is critical from an implementation standpoint, as it allows for more efficient usage of resources as fewer inference processes have to be carried out simultaneously.

- (RQ3) despite having many hyper-parameters that could affect the classification performance, only a few of them may be worth subjecting to tuning, which may be optional in some cases.
- (RQ4) the time - and complexity - overhead of using confidence ensembles over base estimators is linearly dependent on the number of base learners.
- (RQ5) confidence ensembles and especially ConfBoost are much more robust to *unexpected inputs* (i.e., non IID, belonging to distributions other than those used in training and validation) than the base estimators.
- An additional validation of experimental results using confidence intervals demonstrates how confidence ensembles are statistically better in more than half of the cases, and that confidence ensembles using LDA Logistic Regression, LDA, Logit Boost, Random Forests, and XGBoost *are never statistically worse* than base estimators, still b*eing beneficial in most of the cases*.

Wrapping up, we have both experimental and statistical evidence that ConfBag and, even more, ConfBoost have a clear potential to improve any classifier in terms of both accuracy and robustness of classification, with little to no drawbacks aside from the increased complexity. Confidence ensembles can be applied to a wide range of problems and can be deployed over many classifiers. The Python framework publicly available on GitHub and PyPI [72] makes them easy to use, mimicking interfaces from well-known frameworks as *scikit-learn*.

Among the directions for future work, our activities will mostly be devoted to the following aspects.

Confidence Thresholds and Optimization. We acknowledge the possibility to consider alternative ways of using confidence thresholds in our approach. Currently, this threshold is defined as a hyperparameter, and its value is expected to come from the user of our methods (a default value if also defined in the respective classes). An alternative approach would be to consider whether an *optimal value* of the confidence threshold can be automatically sought during the training process. We have conducted preliminary studies and established that the value of the threshold impacts the classification accuracy, thus this machinery has high potential. However, we realise that seeking an optimal threshold value in training, which defines tighter decision boundaries, may negatively affect the capability of confidence ensembles of dealing with unexpected inputs (as discussed in section 5.5). The optimization will depend on a train/validation set, which may not be distributed the same as test data, likely dropping classification performance in case of unknown, non IID inputs.

Usage of a Set of Base Estimators. Widespread and proven-in-use fusion and ensemble techniques as Bagging and Boosting assume that base-learners are created as variants of the same ML algorithms: this is also the case of confidence ensembles. The usage of different ML algorithms for crafting base learners found application in many techniques as stacking, cascading, cascade generalization, delegation, recovery blocks as it promotes diversity, which is the enabling condition [14], [38] for ensemble learning to outperform individual classifiers. When drafting confidence ensembles at a first stage, we were initially thinking of using different base estimators. The problem we found is that each classifier has their own probability distribution of predictions: the usage of a unique confidence threshold did not allow to fully take advantage of the diversity of base estimators. A possible solution could be to allow only perfectly, or semi-perfectly calibrated [97] classifiers to be used as base estimators, alleviating this issue. Calibration should make predicted probabilities comply to a common distribution, making them behave properly against the confidence threshold used by the confidence ensemble.

Framework refinement. The framework for confidence ensembles is publicly available at [72] and fully operational. However, there are ongoing coding optimizations and others to be made for increasing the quality of the code and its speed, relying on GPU supports whenever available.

Acknowledgments

This work was supported in part by the 202297YF75 PRIN 2022 project S2 under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. The CogniSafe3D EUROSTARS 3.6 project funded by the EU, and by the AUAS Project, funded by the EU ECSEL Programme (Grant Agreement 737475).

The authors are grateful to the editor and the anonymous reviewers for the thorough reviews and the numerous useful suggestions for improvements of the paper.

References

[1] Coelho, P., Bessa, C., Landeck, J., & Silva, C. (2023). Industry 5.0: the arising of a concept. Procedia Computer Science, 217, 1137-1144.

[2] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," Information Fusion, vol. 81, pp. 84–90, May 2022, doi: 10.1016/j.inffus.2021.11.011.

[3] S. Tan, Q. Li, L. Li, B. Li, and J. Huang, "STD-NET: Search of Image Steganalytic Deep-Learning Architecture Via Hierarchical Tensor Decomposition," IEEE Trans Dependable Secure Computing, pp. 1–18, 2023, doi: 10.1109/TDSC.2023.3267065.

[4] H. Zhang, H. Xu, X. Tian, J. Jiang, and J. Ma, "Image fusion meets deep learning: A survey and perspective," Information Fusion, vol. 76, pp. 323–336, Dec. 2021, doi: 10.1016/j.inffus.2021.06.008.

[5] R. Sathya and A. Abraham, "Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification," International Journal of Advanced Research in Artificial Intelligence, vol. 2, no. 2, 2013, doi: 10.14569/IJARAI.2013.020206.

[6] T. Zoppi, A. Ceccarelli, T. Puccetti, and A. Bondavalli, "Which algorithm can detect unknown attacks? Comparison of supervised, unsupervised, and meta-learning algorithms for intrusion detection," Computers and Security, vol. 127, 2023, doi: 10.1016/j.cose.2023.103107.

[7] M. Catillo, A. Pecchia, and U. Villano, "CPS-GUARD: Intrusion detection for cyber-physical systems and IoT devices using outlier-aware deep autoencoders," Computers and Security, vol. 129, p. 103210, Jun. 2023, doi: 10.1016/j.cose.2023.103210.

[8] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," in 2010 IEEE Symposium on Security and Privacy, IEEE, 2010, pp. 305–316. doi: 10.1109/SP.2010.25.

[9] X. Ma, J. Zhu, Z. Lin, S. Chen, and Y. Qin, "A state-of-the-art survey on solving non-IID data in Federated Learning," Future Generation Computer Systems, vol. 135, pp. 244–258, Oct. 2022, doi: 10.1016/j.future.2022.05.003.

[10] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," IEEE Trans Dependable Secure Computing, vol. 1, no. 1, pp. 11–33, Jan. 2004, doi: 10.1109/TDSC.2004.2.

[11] Z. Xu and J. H. Saleh, "Machine learning for reliability engineering and safety applications: Review of current status and future opportunities,"

Reliability Engineering and System Safety (RESS), vol. 211, p. 107530, Jul. 2021, doi: 10.1016/j.ress.2021.107530.

[12] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in 2008 Eighth IEEE International Conference on Data Mining, IEEE, Dec. 2008, pp. 413–422. doi: 10.1109/ICDM.2008.17.

[13] L. Breiman, "Random Forests," Mach Learn, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[14] L. Breiman, "Bagging predictors," Mach Learn, vol. 24, no. 2, pp. 123–140, Aug. 1996, doi: 10.1007/BF00058655.

[15] Y. Freund, "Boosting a Weak Learning Algorithm by Majority," Information and Computation, vol. 121, no. 2, pp. 256–285, Sep. 1995, doi: 10.1006/inco.1995.1136.

[16] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA: ACM, Aug. 2016, pp. 785–794. doi: 10.1145/2939672.2939785.

[17] S. González, S. García, J. Del Ser, L. Rokach, and F. Herrera, "A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities," Information Fusion, vol. 64, pp. 205–237, Dec. 2020, doi: 10.1016/j.inffus.2020.07.007.

[18] R. Rosales, P. Popov, and M. Paulitsch, "Evaluation of Confidence-based Ensembling in Deep Learning Image Classification," Mar. 2023, https://arxiv.org/abs/2303.03185.

[19] S. Dey, Q. Ye, and S. Sampalli, "A machine learning based intrusion detection scheme for data fusion in mobile clouds involving heterogeneous client networks," Information Fusion, vol. 49, pp. 205–215, 2019, doi: https://doi.org/10.1016/j.inffus.2019.01.002.

[20] M. A. Souza, R. Sabourin, G. D. C. Cavalcanti, and R. M. O. Cruz, "A dynamic multiple classifier system using graph neural network for high dimensional overlapped data," Information Fusion, vol. 103, p. 102145, 2024, doi: https://doi.org/10.1016/j.inffus.2023.102145.

[21] X. Mao, H. Yang, S. Huang, Y. Liu, and R. Li, "Extractive summarization using supervised and unsupervised learning," Expert Systems and Applications, vol. 133, pp. 173–181, Nov. 2019, doi: 10.1016/j.eswa.2019.05.011.

[22] M. Goldstein and S. Uchida, "A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data," PLoS One, vol. 11, no. 4, p. e0152173, Apr. 2016, doi: 10.1371/journal.pone.0152173.

[23] J. Lever, "Classification evaluation: it is important to understand both what a classification metric expresses and what it hides," Nat Methods, vol. 13, p. 603+, 2016, [Online]. Available: https://link.gale.com/apps/doc/A459507798/HRCA?u=anon~f33228a3&sid=googleScholar&xid=ceaf5104.

[24] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards Automated Log Parsing for Large-Scale Log Data Analysis," IEEE Trans Dependable Secure Computing, vol. 15, no. 6, pp. 931–944, Nov. 2018, doi: 10.1109/TDSC.2017.2762673.

[25] P. P. do Nascimento, P. Pereira, J. M. Mialaret, I. Ferreira, and P. Maciel, "A methodology for selecting hardware performance counters for supporting non-intrusive diagnostic of flood DDoS attacks on web servers," Computers and Security, vol. 110, p. 102434, Nov. 2021, doi: 10.1016/j.cose.2021.102434.

[26] P. Rodríguez, M. A. Bautista, J. Gonzàlez, and S. Escalera, "Beyond one-hot encoding: Lower dimensional target embedding," Image and Vision Computing, vol. 75, pp. 21–31, Jul. 2018, doi: 10.1016/j.imavis.2018.04.004.

[27] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?," Adv Neural Inf Process Syst, vol. 35, pp. 507–520, 2022.

[28] T. Zoppi, S. Gazzini, and A. Ceccarelli, "Anomaly-based error and intrusion detection in tabular data: No DNN outperforms tree-based classifiers," Future Generation Computer Systems, vol. 160, pp. 951–965, Nov. 2024, doi: 10.1016/j.future.2024.06.051

[29] S. Ö. Arik and T. Pfister, "TabNet: Attentive Interpretable Tabular Learning," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 8, pp. 6679–6687, May 2021, doi: 10.1609/aaai.v35i8.16826.

[30] S. Popov, S. Morozov, and A. Babenko, "Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data," in International Conference on Learning Representations, 2020. [Online]. Available: https://openreview.net/forum?id=r1eiu2VtwH

[31] McElfresh D, Khandagale S, Valverde J, et al. When do neural nets outperform boosted trees on tabular data?. Advances in Neural Information Processing Systems, 2024, 36.

[32] Liu, S. Y., Cai, H. R., Zhou, Q. L., & Ye, H. J. (2024). TALENT: A Tabular Analytics and Learning Toolbox. arXiv preprint arXiv:2407.04057.

[33] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: a Survey," ACM Computing Surveys, vol. 41, no. 3, pp. 1–58, Jul. 2009, doi: 10.1145/1541880.1541882.

[34] G. Li and J. J. Jung, "Deep learning for anomaly detection in multivariate time series: Approaches, applications, and challenges," Information Fusion, vol. 91, pp. 93–102, 2023, doi: https://doi.org/10.1016/j.inffus.2022.10.008.

[35] L. Erhan et al., "Smart anomaly detection in sensor systems: A multi-perspective review," Information Fusion, vol. 67, pp. 64–79, 2021, doi: https://doi.org/10.1016/j.inffus.2020.10.001.

[36] P. Popov and B. Littlewood, "The effect of testing on reliability of fault-tolerant software," in International Conference on Dependable Systems and Networks, 2004, IEEE, 2004, pp. 265–274. doi: 10.1109/DSN.2004.1311896.

[37] M. R. Lyu and A. Avižienis, "Assuring Design Diversity in N-Version Software: A Design Paradigm for N-Version Programming," 1992, pp. 197–218. doi: 10.1007/978-3-7091-9198-9_10.

[38] D. Opitz and R. Maclin, "Popular Ensemble Methods: An Empirical Study," Journal of Artificial Intelligence Research, vol. 11, pp. 169–198, Aug. 1999, doi: 10.1613/jair.614.

[39] B. Pfahringer, H. Bensusan, and C. G. Giraud-Carrier, "Meta-Learning by Landmarking Various Learning Algorithms.," in ICML, 2000, pp. 743–750.

[40] D. H. Wolpert, "Stacked generalization," Neural Networks, vol. 5, no. 2, pp. 241–259, Jan. 1992, doi: 10.1016/S0893-6080(05)80023-1.

[41] S. Džeroski and B. Ženko, "Is Combining Classifiers with Stacking Better than Selecting the Best One?," Mach Learn, vol. 54, no. 3, pp. 255–273, Mar. 2004, doi: 10.1023/B:MACH.0000015881.36452.6e.

[42] Z. Gong, P. Zhong, and W. Hu, "Diversity in Machine Learning," IEEE Access, vol. 7, pp. 64323–64350, 2019, doi: 10.1109/ACCESS.2019.2917620.

[43] L. I. Kuncheva and C. J. Whitaker, "Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy," Mach Learn, vol. 51, no. 2, pp. 181–207, 2003, doi: 10.1023/A:1022859003006.

[44] H. Jiang, B. Kim Google Brain, M. Y. Guan, and M. Gupta Google Research, "To Trust Or Not To Trust A Classifier." [Online]. Available: https://github.com/google/TrustScore

[45] E. Hüllermeier and W. Waegeman, "Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods," Mach Learn, vol. 110, no. 3, pp. 457–506, Mar. 2021, doi: 10.1007/s10994-021-05946-3.

[46] W. J. Krzanowski, T. C. Bailey, D. Partridge, J. E. Fieldsend, R. M. Everson, and V. Schetinin, "Confidence in Classification: A Bayesian Approach," J Classif, vol. 23, no. 2, pp. 199–220, Sep. 2006, doi: 10.1007/s00357-006-0013-3.

[47] D. Hendrycks and K. Gimpel, "A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks," Oct. 2016.

[48] J. R. Fonseca, M. I. Friswell, J. E. Mottershead, and A. W. Lees, "Uncertainty identification by the maximum likelihood method," J Sound Vib, vol. 288, no. 3, pp. 587–599, Dec. 2005, doi: 10.1016/j.jsv.2005.07.006.

[49] Z. Bilgin and M. Gunestas, "Explaining Inaccurate Predictions of Models through k-Nearest Neighbors," in Proceedings of the 13th International Conference on Agents and Artificial Intelligence, SCITEPRESS - Science and Technology Publications, 2021, pp. 228–236. doi: 10.5220/0010257902280236.

[50] Z. Xiao, Q. Yan, and Y. Amit, "Likelihood Regret: An Out-of-Distribution Detection Score For Variational Auto-encoder."

[51] T. Zoppi, A. Ceccarelli, and A. Bondavalli, "Ensembling Uncertainty Measures to Improve Safety of Black-Box Classifiers," 2023. doi: 10.3233/FAIA230635.

[52] M. Hein, M. Andriushchenko, and J. Bitterwolf, “Why ReLU Networks Yield High-Confidence Predictions Far Away From the Training Data and How to Mitigate the Problem,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2019.

[53] J. Guerin, K. Delmas, R. Ferreira, and J. Guiochet, “Out-of-Distribution Detection Is Not All You Need,” Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, no. 12, pp. 14829–14837, Jun. 2023, doi: 10.1609/aaai.v37i12.26732.

[54] K. He, D. D. Kim, and M. R. Asghar, “Adversarial Machine Learning for Network Intrusion Detection Systems: A Comprehensive Survey,” IEEE Communications Surveys & Tutorials, vol. 25, no. 1, pp. 538–566, Oct. 2023, doi: 10.1109/COMST.2022.3233793.

[55] L. Xu, X. Ding, H. Peng, D. Zhao, and X. Li, “ADTCD: An Adaptive Anomaly Detection Approach Toward Concept Drift in IoT,” IEEE Internet Things J, vol. 10, no. 18, pp. 15931–15942, Sep. 2023, doi: 10.1109/JIOT.2023.3265964.

[56] Sina Mohseni, Mandar Pitale, Vasu Singh, and Zhangyang Wang, “Practical Solutions for Machine Learning Safety in Autonomous Vehicles,” SafeAI workshop @ AAAI, 2019.

[57] M. Catillo, A. Del Vecchio, A. Pecchia, and U. Villano, “Transferability of machine learning models learned from public intrusion detection datasets: the CICIDS2017 case study,” Software Quality Journal, vol. 30, no. 4, pp. 955–981, Dec. 2022, doi: 10.1007/s11219-022-09587-0.

[58] Y. Liao and V. R. Vemuri, “Use of K-Nearest Neighbor classifier for intrusion detection,” Computers & Security, vol. 21, no. 5, pp. 439–448, Oct. 2002, doi: 10.1016/S0167-4048(02)00514-X.

[59] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, “Survey of intrusion detection systems: techniques, datasets and challenges,” Cybersecurity, vol. 2, no. 1, p. 20, Dec. 2019, doi: 10.1186/s42400-019-0038-7.

[60] T. Zoppi and A. Ceccarelli, “Prepare for Trouble and Make it Double. Supervised and Unsupervised Stacking for Anomaly Based Intrusion Detection,” arXiv. 2022. doi: 10.48550/arxiv.2202.13611.

[61] R. E. Schapire, “The strength of weak learnability,” Mach Learn, vol. 5, no. 2, pp. 197–227, Jun. 1990, doi: 10.1007/BF00116037.

[62] Bartlett, P., Freund, Y., Lee, W. S., & Schapire, R. E. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. The annals of statistics, 26(5), 1651-1686

[63] Zhang, X. Y., Xie, G. S., Li, X., Mei, T., & Liu, C. L. (2023). A survey on learning to reject. Proceedings of the IEEE, 111(2), 185-215.

[64] Li, L., Hu, Q., Wu, X., & Yu, D. (2014). Exploration of classification confidence in ensemble learning. Pattern recognition, 47(9), 3120-3131.

[65] Lu, Z., Wu, X., Zhu, X., & Bongard, J. (2010, July). Ensemble pruning via individual contribution ordering. In Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 871-880).

[66] Dietterich, T. G. (2000, June). Ensemble methods in machine learning. In International workshop on multiple classifier systems (pp. 1-15). Berlin, Heidelberg: Springer Berlin Heidelberg.

[67] Parikh, D., & Polikar, R. (2007). An ensemble-based incremental learning approach to data fusion. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 37(2), 437-450.

[68] Gama, J., & Brazdil, P. (2000). Cascade generalization. Machine learning, 41, 315-343.

[69] Randell, B., & Xu, J. (1995). The evolution of the recovery block concept. Software fault tolerance, 3, 1-22.

[70] Ferri, C., Flach, P., & Hernández-Orallo, J. (2004, July). Delegating classifiers. In Proceedings of the twenty-first international conference on Machine learning (p. 37).

[71] P. E. Hart, “Entropy and Other Measures of Concentration,” J R Stat Soc Ser A, vol. 134, no. 1, p. 73, 1971, doi: 10.2307/2343975.

[72] T. Zoppi, “confidence-ensembles GitHub ,” https://github.com/tommyippoz/confidence-ensembles.

[73] 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

[74] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho, “Flow-based benchmark data sets for intrusion detection,” 2017.

[75] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, “Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification,” in 2018 International Carnahan Conference on Security Technology (ICCST), IEEE, Oct. 2018, pp. 1–7. doi: 10.1109/CCST.2018.8585560.

[76] B. Meidan Yair and A. Shabtai, “detection of IoT botnet attacks N BaIoT.” 2018.

[77] W. Haider, J. Hu, J. Slay, B. P. Turnbull, and Y. Xie, “Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling,” Journal of Network and Computer Applications, vol. 87, pp. 185–192, Jun. 2017, doi: 10.1016/j.jnca.2017.03.018.

[78] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization,” in Proceedings of the 4th International Conference on Information Systems Security and Privacy, SCITEPRESS - Science and Technology Publications, 2018, pp. 108–116. doi: 10.5220/0006639801080116.

[79] H. Kang, D. H. Ahn, G. M. Lee, J. Do Yoo, K. H. Park, and H. K. Kim, “IoT network intrusion dataset.” IEEE Dataport, 2019. doi: 10.21227/q70p-q449.

[80] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, “Toward developing a systematic approach to generate benchmark datasets for intrusion detection,” Computers & Security, vol. 31, no. 3, pp. 357–374, May 2012, doi: 10.1016/j.cose.2011.12.012.

[81] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, IEEE, Jul. 2009, pp. 1–6. doi: 10.1109/CISDA.2009.5356528.

[82] M. S. Elsayed, N.-A. Le-Khac, and A. D. Jurcut, “InSDN: A Novel SDN Intrusion Dataset,” IEEE Access, vol. 8, pp. 165263–165284, 2020, doi: 10.1109/ACCESS.2020.3022633.

[83] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, and R. Therón, “UGR‘16: A new dataset for the evaluation of cyclostationarity-based network IDSs,” Computers & Security, vol. 73, pp. 411–424, Mar. 2018, doi: 10.1016/j.cose.2017.11.004.

[84] N. Moustafa and J. Slay, “UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set),” in 2015 Military Communications and Information Systems Conference (MilCIS), IEEE, Nov. 2015, pp. 1–6. doi: 10.1109/MilCIS.2015.7348942.

[85] BackBlaze, “BackBlaze HDD Data,” https://www.backblaze.com/cloud-storage/resources/hard-drive-test-data.

[86] Baidu Inc, “Baidu HDD - Baidu SMART Dataset for Seagate ST31000524NS drive model,” https://www.kaggle.com/datasets/drtycoon/hdds-dataset-baidu-inc.

[87] W. C. S. Y. J.-H. Shin Hyeok-Ki; Lee and B.-G. Min, “HAI security datasets.” 2023. [Online]. Available: https://github.com/icsdataset/hai

[88] H.-K. Shin, W. Lee, J.-H. Yun, and H. Kim, “HAI 1.0: HIL-based Augmented ICS Security Dataset,” in 13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20), USENIX Association, Aug. 2020. [Online]. Available: https://www.usenix.org/conference/cset20/presentation/shin

[89] N. Davari, B. Veloso, R. P. Ribeiro, P. M. Pereira, and J. Gama, “Predictive maintenance based on anomaly detection using deep learning for air production unit in the railway industry,” in 2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA), IEEE, Oct. 2021, pp. 1–10. doi: 10.1109/DSAA53316.2021.9564181.

[90] “APS Failure at Scania Trucks.” 2017 [online] https://www.kaggle.com/datasets/uciml/aps-failure-at-scania-trucks-data-set.

[91] T., Zoppi, G. , Merlino, A. , Ceccarelli, A. , Puliafito, and A. Bondavalli, “Anomaly Detectors for Self-Aware Edge and IoT Devices,” in 2023 IEEE International Conference on Software Quality, Reliability and Security (QRS), IEEE, 23AD.

[92] A. Agarwal, “Machine Failure Prediction.” Kaggle, 2018. [Online]. Available: https://kaggle.com/competitions/machine-failure-prediction

[93] D. Chicco and G. Jurman, “The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation,” BMC Genomics, vol. 21, no. 1, p. 6, Dec. 2020, doi: 10.1186/s12864-019-6413-7.

[94] Itaya, Y., et al. (2024) Asymptotic Properties of Matthews Correlation Coefficient. arXiv.stat.ME 24 DOI: 10.48550/arXiv.2405.12622
[95] Zou, G. Y. (2007). "Toward using confidence intervals to compare correlations." Psychological Methods Vol 12 (4): 399-413.
[96] Goldstein, H., & Healy, M. J. (1995). The graphical presentation of a collection of means. Journal of the Royal Statistical Society: Series A (Statistics in Society), 158(1), 175-177.
[97] Vaicenavicius, J., Widmann, D., Andersson, C., Lindsten, F., Roll, J., & Schön, T. (2019, April). Evaluating model calibration in classification. In The 22nd international conference on artificial intelligence and statistics (pp. 3459-3467). PMLR.

Appendix A

Structure of the Framework

The framework that implements ConfBag and ConfBoost is called *confidence-ensembles* and is publicly available on GitHub (https://github.com/tommyippoz/confidence-ensembles [72]) and on PyPI (https://pypi.org/project/confidence-ensembles/).

Appling the framework includes the following steps:

- Define a base class Classifier which provides the definition of the main interfaces and provides compatibility with scikit-learn and PYOD frameworks. Thus, confidence ensembles can be used in scikit-learn Pipelines, *GridSearchCV* for parameters, or make them part of complex estimators such as *StackingClassifier* or *VotingClassifier* without any issue.
- Define a class *ConfidenceBaggingClassifier* for ConfBag and a *ConfidenceBoostingClassifier* for ConfBoost. These inherit their type from Classifier and thus benefit from the features of the superclass.

Shared Methods

Confidence ensembles classes, either *ConfidenceBaggingClassifier* or *ConfidenceBoostingClassifier*, have their own constructors, which we detail in the next subsection. The classes also have the following additional methods:

`fit(self, X, y=None)`
This function is used for training a classifier using a training feature set (X) and labels(y), that are optional and set to None by default. The training feature set can be a 2D matrix (either *numpy.ndarray* or *pandas.DataFrame*) for tabular datasets, or an array of images for image classification. Calling this function is mandatory for calling any other function on the objects.

`predict_proba(self, X)`
This function returns a 2D *numpy.ndarray* of probabilities, composed of as many rows as the *items* in X, and as many columns as the classes of the problem (minimum of 2). Each row of the *ndarray* describes probabilities that an item in the test set belongs to a particular class, thus each row of probabilities sums up to 1. The highest value in each row of probabilities correspond to the class predicted for a specific item.

`predict(self, X)`
This function returns a *numpy.ndarray* of predicted classes composed of as many predictions as the *items* in X. Each item describes the class assigned by the classifier to each item in X.

`predict_confidence(self, X)`
This function returns a *numpy.ndarray* of floating point values where each value describes the confidence that the classifier has in each prediction for the items in X. The higher the floating point value, the more confident the classifier. Ideally, these floating point values should range from 0 (complete lack of any confidence) to 1 (certainty in the classification).

`classifier_name(self)`
Returns the name of the classifier as a string.

`get_diversity(self, X, y, metrics=None)`
This function outputs the value of all diversity metrics specified in *metrics*. To compute them, both a feature set X and associated labels y are needed, regardless of the type of learning (supervised, unsupervised) applied to the problem. Diversity metrics are implemented in the metrics.DiversityMetric class of confidence ensembles and use the metrics defined in [43]. The function outputs a dictionary of pairs *(metric name, value)*.

Constructor and Parameters of ConfBag

Creating a Pyton object implementing the ConfBag classifier requires a set of different parameters to be specified, which map those defined in Section 3.2.4.

- clf: specifies the algorithm to be used for creating base learners (*base_estimator* in Section 3.2.4).
- n_base (int): number of base learners (*k* in Section 3.2.4).
- max_features (float): specifies the percentage of features to be used at each iteration (*f_bag* in Section 3.2.4),
- sampling_ratio (float): specifies the percentage of the dataset to be used at each iteration (*sampling_ratio* in Section 3.2.4).
- conf_thr (float): defines the a floating point value to be used as a threshold used to distinguish between "confident enough" and "not confident enough" predictions (*conf_thr* in Section 3.2.4).
- perc_decisors (float): specifies the percentage of base learners to be used for prediction (*bl_perc* in Section 3.2.4).
- n_decisors (int): specifies the number of base learners to be used for prediction (*bl_n* in Section 3.2.4).
- weighted (bool): True if prediction has to be computed as a weighted sum of probabilities of base learners or False otherwise (*weighted_pred* in Section 3.2.4).

Listing A.1. Usage of ConfBoost for supervised learning.

```
import numpy
import sklearn.metrics as metrics
from sklearn.discriminant_analysis
        import LinearDiscriminantAnalysis
from src.classifiers.ConfidenceBoosting
        import ConfidenceBoosting

# Load dataset in the typical way,
# splitting in train (x_train y_train)
# and test (x_test, y_test) sets

# Creating classifiers
classifier = LinearDiscriminantAnalysis()
cb_clf = ConfidenceBoosting(clf=classifier,
                            n_base=10,
                            learning_rate=2,
                            sampling_ratio=0.5,
                            conf_thr=0.8)

# Exercising Classifier
classifier.fit(x_train, y_train)
clf_pred = classifier.predict(x_test)

# Exercising ConfBoost
cb_clf.fit(x_train, y_train)
cb_pred = cb_clf.predict(x_test)

print('LDA has accuracy of %.3f, whereas the
        ConfBoost(LDA) has accuracy of %.3f' %
        (metrics.accuracy_score(y_test, clf_pred),
         metrics.accuracy_score(y_test, cb_pred)))
```

Constructor and Parameters of ConfBoost

Creating a ConfBoost object requires a set of different parameters to be specified, which map those defined in Section 3.3.4.

- clf: specifies the algorithm to be used for creating base learners (*base_estimator* in Section 3.3.4).
- n_base (int): defines the number of base learners (*k* in Section 3.3.4).
- learning_rate (float): defines how fast weights of items of the training set are being updated (*lr* in Section 3.3.4).
- sampling_ratio (float): defines the percentage of the dataset to be used at each iteration (*sampling_ratio* in Section 3.3.4).
- conf_thr (float): defines the a floating point value to be used as a threshold used to distinguish between "confident enough" and "not confident enough" predictions, and update the weights of the training set accordingly (*conf_thr* in Section 3.3.4).
- perc_decisors (float): specifies the percentage of base learners to be used for prediction (*bl_perc* in Section 3.3.4).
- n_decisors (int): specifies the number of base learners to be used for prediction (*bl_n* in Section 3.3.4).
- weighted (bool): True if the prediction has to be computed as a weighted sum of probabilities of base learners (*weighted_pred* in Section 3.3.4) and False otherwise.

Usage Examples

We show the usage of the library for two simple analyses in Listing 1 and Listing 2, respectively. For both libraries, we assume that the dataset gets loaded according to user's preferences and show how to instantiate an object of respective classifier type, train it, and predict using the respective confidence ensemble. Listing 1 refers to supervised learning, while Listing 2 shows the usage of confidence ensembles for unsupervised learning.

Listing A.2. Usage of ConfBoost for unsupervised learning.

```
import numpy
import sklearn.metrics as metrics
from pyod.models.pca import PCA
from src.classifiers.ConfidenceBoosting
        import ConfidenceBoosting

# Load dataset in the typical way,
# splitting in train (x_train y_train)
# and test (x_test, y_test) sets

# Creating classifiers
classifier = PCA(contamination=an_perc)
cb_clf = ConfidenceBoosting(clf=classifier,
                            n_base=10,
                           learning_rate=2,
                           sampling_ratio=0.5,
                           conf_thr=0.8)

# Exercising Classifier
classifier.fit(x_train)
clf_pred = classifier.predict(x_test)

# Exercising ConfBoost
cb_clf.fit(x_train)
cb_pred = cb_clf.predict(x_test)

print('PCA has accuracy of %.3f, whereas the
        ConfBoost(PCA) has accuracy of %.3f' %
        (metrics.accuracy_score(y_test, clf_pred),
         metrics.accuracy_score(y_test, cb_pred)))
```

Repeatability of Experiments

All experiments reported in this paper can be reproduced by calling specific scripts available in the library. Each script sets a random seed at the beginning to avoid the nondeterminism due to pseudo-randomness; this makes the execution fully deterministic and repeatable. Results in the paper can be obtained using the following scripts:

- *tests/rq1-2.py* contains the script used to generate the data needed to answer RQ1 and RQ2. The script generates a large CSV file that can be then used to compute averages, produce plots, and to compute additional statistics.
- *tests/rq3.py* contains the script used to generate the data needed to answer RQ3, including the generation of the dataset's variants and the computation of rec-unk. The script generates a large CSV file that can then be used to compute averages, produce plots, and to compute additional statistics.

The XLSX file contains the results of these scripts, including the plots and tables presented in this paper, can be found in the root folder of the GitHub repository.

Appendix B

8.1 Definitions

This section reports the table of terms that are used throughout the paper, associated with the meaning they have within the manuscript.

Table B.I: terms used throughout the manuscripts.

Term	Meaning within the manuscript
System Indicator	A performance indicator (e.g., memory usage, bytes received from the network, number of opened files) of a target system.
Feature	A System Indicator to be monitored for analysis
Feature Value	The value of a Feature gathered at a specific time instant or when specific events occur
Data Point	A collection of Feature Values of different Features for the same Target System
Tabular Data	Structured data comprising of multiple Data Points (rows) containing values of multiple Features (columns)
Label	A class (categorical value) associated to a Data Point.
ML Algorithm	An algorithm that can learn classification or regression tasks from data and generalize their findings to unseen data.
Classifier	A ML algorithm that performs classification
Training Data	Collection of Data Points used for training a classifier
Inference	The process of assigning a label to a data point.
Model	An object containing what a classifier learns after training. It can be used for inference.
Prediction	The Label result of the inference of a classifier, i.e., the output of a Model when a Data Point is fed into it.
Unexpected Input	Data point to be classified that does not belong to the distribution of data points in training data
Supervised Classifier	A classifier whose training data is labelled
Unsupervised Classifier	A classifier whose training data is unlabelled
Base Learner	Classifier exercised within an ensemble classifier
Base Estimator	The classifier used to craft base learners. It is instantiated with different parameters or training set to create diverse base learners.
Uncertainty	The epistemic uncertainty to be estimated using different techniques.
Confidence	Floating point value that quantifies the Uncertainty in a Prediction
Confidence Ensembles	Either Confidence Bagging or Confidence Boosting
confidence-ensembles	The GitHub framework that implements Confidence Ensembles, available at https://github.com/tommyippoz/confidence-ensembles and on PyPI at https://pypi.org/project/confidence-ensembles/

8.2 Table of Acronyms

This section summarizes the acronyms used in this paper.

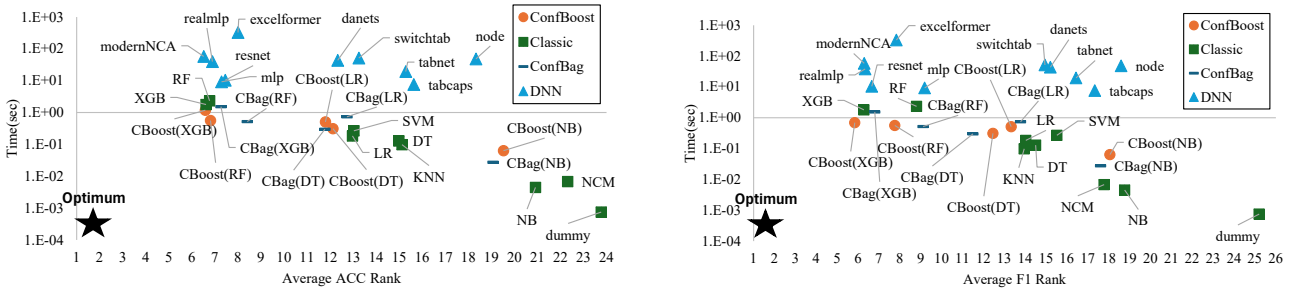
Table B.II: Table of acronyms.

Term	Meaning within the manuscript
ICT	Information and Communication Technology
ML	Machine Learning
DNN	Deep Neural Network
IID	Independent and Identically Distributed
OOD	Out-Of-Distribution
PCA	Principal Component Analysis (mostly referred to as a classifier, but also as and a dimensionality reduction technique in Section 2.2)
HBOS	Histogram-based outlier score
CBLOF	Cluster-Based Local Outlier Factor
INNE	Isolation-based Anomaly Detection using Nearest-Neighbor ensembles
kNN	k-th Nearest Neighbour
SOTIF	Safety Of The Intended Functionality
GNB	Gaussian Naïve Bayes
LDA	Linear Discriminant Analysis
XGBoost	eXtreme Gradient Boosting
ConfBag	Confidence Bagging
ConfBoost	Confidence Boosting
PyPI	Python Package Index
MCC	Matthews Correlation Coefficient

Appendix C

The paper mostly relies on the Matthews Correlation Coefficient (MCC) to quantify classification performance. For completeness, this section reports tables and plots discussed in Section 5.1 in which we present Accuracy and F1 (F-Measure) scores instead of MCC values. The reader may observe that changing the metrics does not change the takeovers of the paper. Note that results of the benchmark with LAMDA-TALENT are also available at the public GitHub [72], folder '*benchmark*'.

Results using the TALENT benchmark using Accuracy - Figure C.ACC (left) – and F-Measure – Figure C.F1 (right).



Comparison of classification performance of classifiers against ConfBag and ConfBoost ensembles using classifiers as base estimators using Accuracy (Table C.ACC, up), and F-Measure (Table C.F1, down).

	Classifier	*Average ACC of Base Estimator and Confidence Ensembles*					*ACC Gain of Confidence Ensembles w.r.t. Base Estimator*				
		Base Estimator	**ComfBag (weighted=False)**	**ComfBag (weighted=True)**	**ConfBoost (weighted=False)**	**ConfBoost (weighted=True)**	**ComfBag (weighted=False)**	**ComfBag (weighted=True)**	**ConfBoost (weighted=False)**	**ConfBoost (weighted=True)**
Unsupervised	CBLOF	0.790	0.814	0.812	0.829	0.839	0.024	0.022	0.039	0.049
	Isolation Forest	0.746	0.793	0.795	0.798	0.800	0.046	0.048	0.051	0.054
	INNE	0.737	0.771	0.770	0.783	0.764	0.034	0.033	0.046	0.027
	HBOS	0.747	0.791	0.791	0.802	0.805	0.044	0.044	0.054	0.057
	PCA	0.762	0.818	0.816	0.796	0.798	0.056	0.053	0.034	0.036
Supervised	GNB	0.733	0.842	0.839	0.899	0.877	0.109	0.106	0.165	0.144
	Logistic Regression *	0.858	0.875	0.869	0.926	0.926	0.018	0.011	0.069	0.068
	LDA	0.909	0.911	0.911	0.942	0.941	0.002	0.002	0.033	0.032
	Decision Tree	0.968	0.975	0.976	0.975	0.975	0.007	0.008	0.008	0.007
	Logit Boost	0.961	0.962	0.963	0.964	0.964	0.000	0.001	0.002	0.002
	Extra Trees	0.974	0.974	0.974	0.975	0.975	0.001	0.000	0.001	0.001
	Random Forest	0.974	0.974	0.974	0.975	0.975	0.000	0.000	0.002	0.002
	XGBoost	0.971	0.970	0.971	0.973	0.973	0.000	0.000	0.002	0.002
	Average	**0.856**	**0.882**	**0.882**	**0.895**	**0.893**	**0.026**	**0.025**	**0.039**	**0.037**

* Logistic Regression delivered multiple alerts of failed convergence in many datasets regardless of the parameters we tried.

	Classifier	*Average F1 of Base Estimator and Confidence Ensembles*					*F1 Gain of Confidence Ensembles w.r.t. Base Estimator*			
		Base Estimator	**ComfBag (weighted=False)**	**ComfBag (weighted=True)**	**ConfBoost (weighted=False)**	**ConfBoost (weighted=True)**	**ComfBag (weighted=False)**	**ComfBag (weighted=True)**	**ConfBoost (weighted=False)**	**ConfBoost (weighted=True)**
Unsupervised	CBLOF	0.511	0.521	0.522	0.550	0.556	0.010	0.011	0.039	0.045
	Isolation Forest	0.498	0.528	0.538	0.549	0.551	0.030	0.040	0.051	0.053
	INNE	0.451	0.493	0.491	0.515	0.492	0.042	0.040	0.064	0.040
	HBOS	0.481	0.518	0.519	0.525	0.532	0.037	0.038	0.044	0.051
	PCA	0.525	0.547	0.547	0.547	0.542	0.022	0.022	0.022	0.017
Supervised	GNB	0.574	0.668	0.663	0.730	0.727	0.094	0.089	0.156	0.153
	Logistic Regression *	0.570	0.601	0.596	0.694	0.694	0.031	0.027	0.125	0.124
	LDA	0.678	0.691	0.694	0.780	0.778	0.014	0.016	0.102	0.100
	Decision Tree	0.857	0.890	0.893	0.896	0.896	0.033	0.036	0.039	0.039
	Logit Boost	0.827	0.825	0.829	0.844	0.838	-0.003	0.001	0.016	0.011
	Extra Trees	0.868	0.876	0.868	0.881	0.881	0.008	0.000	0.013	0.013
	Random Forest	0.876	0.879	0.877	0.889	0.888	0.003	0.000	0.013	0.012
	XGBoost	0.862	0.855	0.857	0.878	0.877	-0.007	-0.005	0.017	0.016
	Average	**0.660**	**0.684**	**0.684**	**0.714**	**0.712**	**0.024**	**0.024**	**0.054**	**0.052**