Efficient Multiclass Classification with Duet

Shay Vargaftik
VMware Research

Yaniv Ben-Itzhak
VMware Research

Abstract

In the upcoming era of edge computing, the capability to perform fast training and classification at the edge is an increasing need due to limited connectivity, hardware resources, privacy concerns, profitability, and more.

Accordingly, we propose a new classifier termed Duet. Duet incorporates the advantages of bagging and boosting decision-tree-based ensemble methods (DTEMs) by using two classifiers instead of a monolithic one. A simple bagging model is trained using the entire training dataset and is responsible for capturing the easier concepts. Then, a boosting model is trained using only a fraction of the dataset representing the concepts the bagging model finds hard. To make the whole process resource efficient, we develop a new heuristic approach to rank data with respect to concepts that the bagging model finds hard. We use this approach, termed *data instance predictability* to determine the dataset fraction for the boosting model training.

We implement Duet as a scikit-learn classifier. Evaluation using datasets from different domains and with different characteristics indicates that Duet offers a better tradeoff between classification accuracy and system performance than monolithic DTEMs. Moreover, in an evaluation over a resource-constrained Raspberry Pi 3 device Duet successfully completes all training tasks, where some monolithic models fail due to insufficient resources, indicating broader applicability of Duet to resource-constrained edge devices.

Duet is a part of an effort for advancements in resource-efficient classification, and its scikit-learn implementation can be found in https://research.vmware.com/projects/efficient-machine-learning-classification.

ACM Reference Format:
Shay Vargaftik and Yaniv Ben-Itzhak. 2022. Efficient Multiclass Classification with Duet. In *2nd European Workshop on Machine Learning and Systems (EuroMLSys '22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3517207.3526970

EuroMLSys '22, April 5–8, 2022, RENNES, France

ACM ISBN 978-1-4503-9254-9/22/04…$15.00
https://doi.org/10.1145/3517207.3526970

1 Introduction

In edge computing, the capability of a resource-constrained edge device to perform local and fast training and classification is of increasing need due to many emerging use cases such as limited connectivity and hardware resources or when local computation is required due to security privacy and even profitability concerns.

Decision-tree-based ensemble methods (DTEMs) are the de-facto standard used for tabular data multiclass classification. An arising challenge in such classification is the continued growth of datasets [25] in terms of the number of features, classes and, data instances, alongside the demand for greater accuracy. This growth results in larger model memory footprints, longer training times, and slower classification latencies at lower throughput.

We aim at alleviating this challenge by relying on well-established DTEMs and merely augmenting them to obtain a better resource/performance tradeoff. The leading idea behind our approach is to leverage the inherent differences in the two main principle strategies employed in existing DTEM classifiers, *i.e.,* bagging and boosting. Specifically, we seek to combine bagging and boosting DTEMs in a way that incorporates their advantages and mitigates their weaknesses.

A bagging DTEM (*e.g.,* tree bagging [3], random forest [5], extra trees [16]) has the advantage of a fast (and parallel) training. It also offers a controlled variance allowing it to learn relatively simple concepts at low effort. A boosting DTEM (*e.g.,* AdaBoost [14], GBDT [15, 18], XGBoost [9], LightGBM [21], CatBoost [31]), on the other hand, is considerably slower to train and more amenable to overfitting. Nevertheless, it offers a controlled bias allowing it to learn hard concepts.

In that light, we propose a new classifier we term Duet. The two main building blocks of Duet are two DTEM classifiers. The first classifier is a simple bagging DTEM that is trained using the entire training dataset. During classification, all data instances are classified by this classifier, and only the hard data instances (*i.e.,* cases in which this classifier is not sufficiently confident) are forwarded to the second classifier. The second classifier is a boosting DTEM. It is trained using only a fraction of the training dataset, picked with an emphasis on the concepts for which the first DTEM under-performs.

To execute the above recipe, we need to *efficiently* capture and rank data instances with respect to the concepts that the bagging DTEM did not learn satisfactorily. To do so, we develop a heuristic approach that relies on a new metric we term *data instance predictability*, whose computation relies purely on the bagging DTEM and therefore has a low overhead. To compute the predictability of a labeled data instance, we classify it by the bagging DTEM and obtain its probability

distribution over the classes. Then, predictability is defined as inversely proportional to the L2 norm distance between the class distribution vector of the instance and the instance's perfect distribution (*i.e.,* a probability distribution vector with a probability of one in the correct class).

Intuitively, a labeled data instance predictability measures how "difficult" a specific instance to the bagging DTEM is. Namely, the predictability measure indicates whether a specific labeled instance is captured by the bagging DTEM as a typical representative of its class (*i.e.,* has high predictability and therefore may offer lower training value to the boosting DTEM) or otherwise (*i.e.,* has low predictability and therefore may offer higher training value for the boosting DTEM).

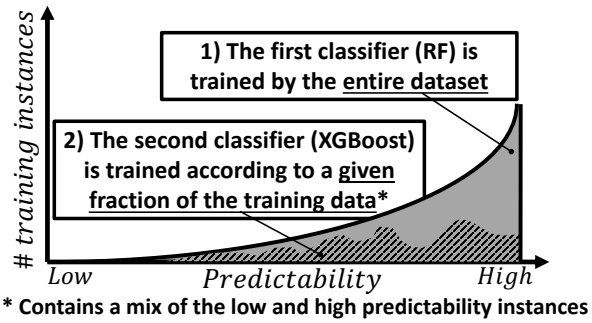
We implement Duet as a scikit-learn classifier [7] (Python 3) with only 240 lines of code [43]. Due to their broad applicability and usage, we chose a random forest (RF) as the bagging DTEM and XGBoost as the boosting DTEM. We then use various datasets from different domains and with different characteristics for evaluation. We find that Duet consistently offers a better tradeoff between classification accuracy and system performance than monolithic random forest and XGBoost. For example, evaluation over a commodity AWS server reveals that for similar classification accuracy, Duet's training and classification times are up to 63× and 42× lower than that of a monolithic XBGoost model, and its memory footprint is up to 47× smaller than that of a monolithic RF.

Repeating these experiments on a resource-constrained Raspberry Pi 3 device shows similar gains. Moreover, some RF and XGBoost monolithic models did not successfully complete their training due to insufficient CPU and memory resources, while Duet successfully completed training all models, indicating broader applicability to weaker and resource-constrained edge devices.

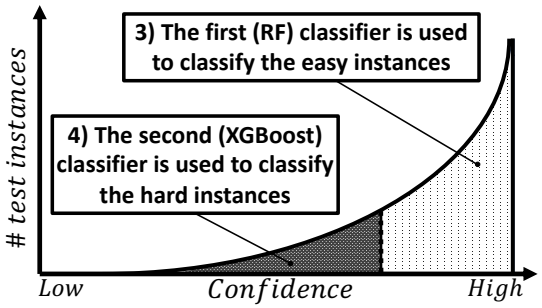
These gains can be attributed to the following design choices and hyperparameter tuning guidelines we detail in the paper:

- Low memory footprint: we limit the memory footprint of the random forest mainly by restricting the size of each tree and also the number of trees. A small model is sufficient since we only seek to learn the easier concepts.
- Low training time: the training time of the RF model is low due to its limited size and simplicity. Also, we train the XGBoost model using only a predictability-driven fraction of the training dataset. Thus, its training time is considerably lower. Other operations within Duet's training (*e.g.,* compute predictability) are negligible compared to the training of the RF and XGBoost models.
- Low classification latency: typically, most of the queries are easy and therefore are classified only by the RF.

To summarise, Duet's training and classification are illustrated in Figures 1(a) and 1(b) respectively.



(a) Training Duet.



(b) Classification by Duet.

Figure 1. Duet's architecture. 1) We train a small RF using the entire training dataset and then use it to compute the training data predictability. 2) Then, we train an XGBoost classifier using only a predictability-driven fraction of the training dataset. 3) During classification, instances are first classified by the RF, and 4) only the hard (low-confidence) instances are reclassified by the XGBoost model.

2 Predictability

We develop and propose *predictability* to rank how valuable a labeled data instance may be to the training of the boosting classifier of Duet. As mentioned, after we have built the bagging model, we use it to efficiently compute the predictability of the training instances. To do so, we first *classify* the training dataset by the bagging model and obtain the class probability distributions of each instance. Predictability of an instance is then given by a distance function that measures how far the resulting class probability distribution vector is from the perfect distribution vector with respect to the true label of that instance. That is, a perfect distribution vector should have a probability of one in the correct label (i.e., class).

2.1 Distance functions

A natural choice for a distance metric would be using a *norm* (*e.g.,* , L_1, L_2, L_∞). For example, using the L_1 norm, predictability of a labeled instance (x, y) can be defined as

$$Predictability(x, y) = 1 - \frac{1}{2} \cdot ||pp(x) - \hat{y}||_1, \qquad (1)$$

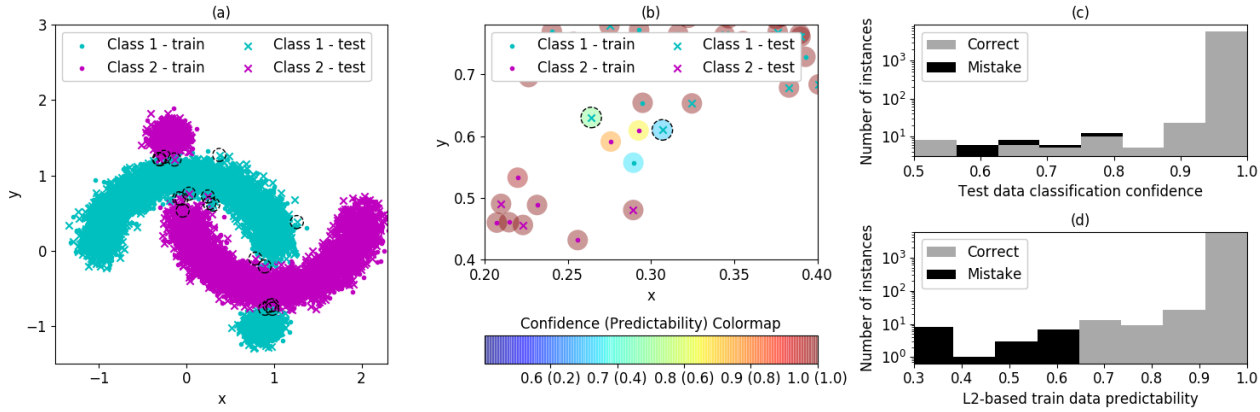


Figure 2. Demonstrating Predictability principles using a toy example.

where $pp(x)$ is the instance's class distribution vector[1] by the bagging classifier and $\hat{y}$ is the perfect distribution vector. Also, to obtain a normalized measure, we choose the function constants such that predictability is always in $[0, 1]$. This way, a perfect distribution receives a predictability of 1. On the other hand, being confident with probability 1 (*i.e.,* the highest possible confidence) in the wrong class has a predictability of 0. Likewise, predictability can be defined via L_∞ as

$$Predictability(x, y) = 1 - ||pp(x) - \hat{y}||_\infty. \quad (2)$$

The latter is essentially a metric that linearly depends on the classification confidence level (*i.e.,* the top-1 entry in $pp(x)$). Some flavors of the *instance hardness* [39] measure are based on it. Yet, both these measures are not the best fit for our goal, as we convey via an example.

Consider a classification task with six classes $0, 1, \ldots, 5$ and assume two distribution vectors $p(x_1) = (0.5, 0.5, 0, 0, 0, 0)$ and $p(x_2) = (0.5, 0.1, 0.1, 0.1, 0.1, 0.1)$ by the bagging model where the correct class of both is the first one (*i.e.,* $y = 0$ for both). Thus, the perfect distribution vector for both is $\hat{y} = (1, 0, 0, 0, 0, 0)$. Clearly, x_1 and x_2 are indistinguishable with respect to both predictability measures that are based on L_1 and L_∞. However, in this case, we would like x_1 to receive a lower predictability than x_2 since the former's confidence in a wrong class is considerably larger, and the bagging model is on the verge of making a classification mistake. Intuitively, such an instance is more confusing to the bagging DTEM and may hold an important concept that was not learned by it satisfactorily. Therefore, we use a predictability measure based on the L_2 norm that captures such properties. Formally,

$$Predictability(x, y) = 1 - \frac{1}{\sqrt{2}} \cdot ||pp(x) - \hat{y}||_2. \quad (3)$$

It is worth emphasizing that predictability is an inherently different measure than a classification confidence level. For example, if an instance's classification has high confidence but it is misclassified, it would have a low predictability. On the other hand, if it has high confidence, but it is correctly classified, it would have a high predictability.

[1]Denoted by `predict_proba(x)` in scikit-learn's [30] classifiers.

2.2 Example

We use a toy example to illustrate Duet's principles and provide intuition for the predictability measure.

Setup. Figure 2(a) illustrates a toy 2D dataset with two classes. The training data is marked with "·", where the test data is marked with "×". Using the training data, we build an RF with 10 decision trees, each limited to 20 leaf nodes. Then, we use this RF to classify both the training and test data. Namely, we obtain the classification distributions to calculate the predictability (using Equation (3)) for the training data and to obtain the top-1 confidence level for the test data.

Results. In Figure 2(a) and 2(b) (zoom-in area from 2(a)), the black dotted circles mark the test classification mistakes. In Figure 2(b), the background circle colors show the predictability of each training data instance and the top-1 classification confidence level of each test data instance. Blue color means low confidence or predictability, and red means high confidence or predictability, as shown by the color map. It is evident how typical misclassifications have low confidence and are closely located to low predictability training data instances. Finally, Figures 2(c) and 2(d) show histograms of the confidence level of the test data instances and the predictability of the training data instances. It is evident by the logarithmic scale how the majority of test instances are easy (*i.e.,* with high confidence level), whereas misclassifications are *typically* hard. Likewise, the majority of train instances have high predictability, whereas *all* training classification mistakes have low predictability.

Discussion. Intuitively, as depicted in Figure 2(b), low predictability training data contains mainly two types of instances: (1) borderline instances that are valuable for the training process; (2) outliers that are located deeper in the territory of the opposite class. Unlike in this simple example,

Algorithm 1: Duet training

```
Duet.fit((X,Y)):
1: Duet.RF.fit(X,Y)
2: pp = Duet.RF.predict_proba(X)
3: pred = Predictability(X,Y,pp) ▷ Eq. (3)
4: (X',Y') = Filter((X,Y),pred)
5: Duet.XGBoost.fit(X',Y')
6: return Duet
```

Algorithm 2: Duet classification

```
Duet.predict(x):
1: dist = Duet.RF.predict_proba(x)
2: if max(dist)>confidence:
3:     return argmax(dist)
4: else
5:     return Duet.XGBoost.predict(x)
```

Algorithm 3: Per-class and global instance selection

```
Filter((X,Y),pred):
1:classes = Unique(Y)
2:(X',Y') = []
3:For i in classes:
4:  (X',Y').append(Filter_h((X,Y==i),
pred[Y==i],fraction[i]))
5:(X",Y"),pred"=(X,Y),pred|(x,y) ∉ (X',Y')
6:r=fraction[tot]-len((X',Y'))/len((X,Y))
7:(X',Y').append(Filter_h((X",Y"),pred",r))
8:return (X',Y')
```

differentiating between these two types is a major research challenge to date with emerging tools and heuristic solutions [50]. Outlier detection is out of the scope of our work. Our goal is not to lose valuable information for the training of the boosting model, *i.e.,* concepts that the bagging model did not capture. Thus, we give preference to low predictability instances that contain both aforementioned types. We rely on correct parameter tuning of the boosting model to offer good generalization properties[2]. In contrast, high predictability instances may be sub-sampled without losing such valuable information and, in turn, without degrading the ML performance of the boosting model.

3 Model

We next overview in detail Duet's training and classification and discuss its hyperparameters (marked in bold in Algorithms 1-4). Finally, due to their broad applicability, we focus on a RF as the bagging DTEM and XGBoost as the boosting DTEM.

[2]Indeed, our evaluation indicates that this leads to favorable results for Duet versus the monolithic models.

Algorithm 4: Predictability-based instance selection

```
Filter_h((X,Y),pred,C):
 1: Bins, Len = Hist((X,Y),pred,nbins)
 2: Bin-search for maximum k such that:
 3:     sum([min(i,k),i∈Len]) ≤ C·len((X,Y))
 4: (X',Y') = []
 5: for B, L in Bins, Len:
 6:     if L ≤ k:
 7:       (X',Y').append(B)
 8:     else:
 9:       (X',Y').append(rand(B,k))
10: return (X',Y')
```

3.1 Training

Algorithm 1 describes Duet's training. It begins with the training of an RF model (line 1). Since we seek to learn the easier concepts and admit low resource usage, the RF is preferred to have trees of limited depth or a limited number of leaves.

We then use the RF model to classify the training data and obtain the class distribution vector of each labeled instance (line 2). With the distributions and the correct labels at hand, we continue to calculate the labeled data instances predictability by Equation 3 (line 3).

Next, we prepare the training dataset for the XGBoost model (line 4). Specifically, we choose a subset of the original training dataset by Algorithm 3 according to the **fraction** hyperparameter. Its resulting size is only a fraction of its original size, and the training instances are chosen with respect to their predictability as we describe in Section 3.3. As mentioned, we give more emphasis to low predictability instances. Notice that this subset of the training dataset is not an unbiased sample, as it contains specific instances chosen by their predictability value (see also §4.1).[3]

3.2 Classification

Algorithm 2 describes Duet's classification. First, we classify the unlabeled instances by the RF model (line 1). Then, we check if the classification confidence by the RF is greater than a predefined confidence threshold (*i.e.,* **confidence** – line 2). As mentioned, even though the RF model is rather simple, the classification is expected to be correct when requiring high confidence. Only if the confidence is insufficient (*i.e.,* there is a concern that the RF model may be mistaken), the query is passed to the XGBoost model (that is trained for such particular situations). In this case, the classification is determined solely by the XGBoost model, as the classification by the RF model is not informative (due to its low confidence).

[3]Another option is to use predictability for unbiased *importance sampling*. However, this is not aligned with our goal of having the two classifiers performing *different* tasks.

3.3 Predictability-based training data selection

There are many sensible ways to use predictability to pick data instances for the XGBoost model. The best recipe often depends on the dataset and the hyperparameters of Duet. Next, we describe the heuristic we develop and implement. We find it to be robust and work well in practice.

Roughly speaking, we aim to keep representative samples across all predictability levels of the different classes to avoid losing unique concepts. For example, we may sample with lower probability high predictability instances of the same large class. On the other hand, we may often take all low predictability instances of a small class.

We do so since we aim to reduce the dataset size with minimal information loss with respect to the concepts that the bagging model did not capture but without losing the general properties and structure of the data.

Algorithm 3 describes how we perform the selection of the data instances. To that end, we introduce a hyperparameter termed **`fraction`**. It is a vector that contains an entry for each class and an additional entry termed `total`. **`fraction`**`[total]` $\in$ [0,1] is a fraction of the original dataset that we want to use for the training of the XGBoost model. **`fraction`**`[i]`$_{i \in classes}$ are smaller fractions such that $\sum_{i \in classes}$**`fraction`**`[i]`$\leq$**`fraction`**`[total]` stating what is the minimal fraction offered to each class. For example, in a 3-class classification problem, if

- **`fraction`**`[total]`=0.06,
- **`fraction`**`[0]`=**`fraction`**`[1]`=**`fraction`**`[2]`=0.01,

then the total training data fraction that will be used to train the XGBoost model is 6% of the original dataset (`X,Y`) and at least `min{0.01·len((X,Y)),len((X,Y==i))}` is offered to each class *i*.

Algorithm 4 describes the sub-routine that is repeatedly invoked by Algorithm 3. It is invoked for each class independently with its fraction and the instances of that particular class. Then, it is invoked with the remaining fraction and *all* yet unpicked instances. Within each such call, instances are divided into bins according to their predictability. Then, for each bin, we either take the entire bin or sample from it uniformly at random, where the goal is to take roughly the same number of instances from each bin. Algorithm 4 also includes the hyperparameter **`nbins`** which is the number of bins to which we divide instances according to their predictability (line 1). In practice, 10 (our default) appears to work well.

The purpose of such "balanced" selection is to find a compromise between class and predictability imbalance via the **`fraction`** hyperparameter. Our default, which appears to work well in practice, is to set a value of 0 to all individual classes and subsample only according to predictability. We do find cases where setting non-zero values to different classes improves performance, especially in the presence of considerable class imbalance.

	Name	Instances	Dimention	Classes
0	abalone	4024	8	14
1	adult	48842	14	4
2	car	1728	6	4
3	cmc	1473	9	3
4	communities	1994	127	3
5	creditcardfraud	284807	30	2
6	krkopt	28029	6	17
7	letter	20000	16	26
8	mammography	11183	6	2
9	mushroom	8124	22	2
10	seismic	2584	18	2
11	yeast	1484	9	10
12	kddcup99	494021	41	23

Table 1. Evaluation datasets.

	Duet	**Duet - XGBoost only**
seismic	0.934984±0.0	0.93421±0.0
mammography	0.989091±0.0	0.987749±0.0
letter	0.95245±0.0	0.9438±0.0
mushroom	0.983131±0.02	0.961593±0.04
car	0.861111±0.01	0.851269±0.01
communities	0.795896±0.02	0.586791±0.05

Table 2. Duet versus Duet's XGBoost.

Dataset	**RF**	**XGBoost**	**Duet**
seismic	0.922218±0.02	0.888563±0.09	0.934984±0.0
mammography	0.987839±0.0	0.988196±0.0	0.989091±0.0
letter	0.9669±0.0	0.9646±0.0	0.95245±0.0
yeast	0.993935±0.0	0.999327±0.0	1.0±0.0
mushroom	0.896694±0.14	0.898296±0.14	0.983131±0.02
cmc	0.536991±0.03	0.564155±0.02	0.559412±0.02
krkopt	0.151309±0.05	0.301045±0.05	0.251776±0.05
car	0.849587±0.05	0.84031±0.04	0.861111±0.01
abalone	0.268146±0.02	0.277841±0.03	0.272623±0.03
adult	0.576041±0.0	0.576062±0.0	0.569469±0.0
communities	0.817947±0.02	0.817439±0.02	0.795896±0.02
creditcardfraud	0.999301±0.0	0.996812±0.01	0.999357±0.0
kddcup99	0.999773±0.0	0.999822±0.0	0.999848±0.0

Table 3. Macro F1 of Duet vs. monolithic RF and XGBoost.

4 Evaluation

We implement Duet as a fully compatible *scikit-learn* classifier [7] with 240 lines of code that is available in [43].

We use 13 datasets for our evaluation from the UCI [12] and Kaggle [26] repositories. To cover different use-cases, these include datasets from different domains as well as different sizes, number of features, and classes. These are summarized in Table 1

Dataset	Method	Fit time [sec]	Predict time [sec]	Model size [MB]
seismic	RF	1.723±0.02	0.106±0.0	2.96
	XGBoost	9.316±0.05	0.090±0.0	0.59
	Duet	2.071±0.02	0.102±0.0	0.22
mammography	RF	4.247±0.05	0.207±0.0	2.17
	XGBoost	37.605±0.49	0.876±0.04	1.1
	Duet	6.083±0.04	0.198±0.0	1.59
letter	RF	–	–	–
	XGBoost	2541.0±29.89	95.88±0.67	24.2
	Duet	523.59±0.44	31.27±0.92	70.88
yeast	RF	1.323±0.01	0.107±0.0	2.47
	XGBoost	19.29±0.05	1.475±0.0	3.05
	Duet	4.336±0.03	0.532±0.06	2.19
mushroom	RF	1.938±0.04	0.146±0.0	0.33
	XGBoost	24.09±1.47	0.135±0.01	0.33
	Duet	3.209±0.05	0.154±0.0	0.38
cmc	RF	6.838±0.04	0.496±0.0	41.71
	XGBoost	7.557±0.03	0.595±0.0	1.82
	Duet	1.365±0.01	0.081±0.0	0.29
krkopt	RF	–	–	–
	XGBoost	1214.59±9.06	124.55±3.15	49.83
	Duet	284.33±1.06	112.95±2.71	70.34
car	RF	11.54±0.02	0.886±0.0	30.67
	XGBoost	13.90±0.14	0.908±0.02	5.83
	Duet	5.663±0.04	0.422±0.05	8.63
abalone	RF	–	–	–
	XGBoost	115.35±0.12	8.762±0.02	7.91
	Duet	35.32±0.1	8.753±0.05	10.8
adult	RF	387.8±1.99	10.29±0.01	168.09
	XGBoost	1735.0±13.54	65.948±0.46	30.93
	Duet	150.20±0.34	26.576±0.07	2.62
communities	RF	19.33±0.14	0.622±0.01	16.2
	XGBoost	260.17±2.45	1.352±0.03	9.11
	Duet	68.148±0.74	0.439±0.04	6.21
creditcardfraud	RF	352.95±16.07	7.947±0.11	2.85
	XGBoost	5449.3±114.07	17.990±0.3	0.83
	Duet	807.41±84.75	10.422±0.89	2.78
kddcup99	RF	–	–	–
	XGBoost	–	–	–
	Duet	349.12±4.37	40.552±0.0	19.99

Table 4. Duet vs. monolithic RF and XGBoost. System performance comparison over Raspberry Pi 3 B+.

Dataset	Model	Fit time [sec]	Predict time [sec]	Model size [MB]
seismic	RF	0.2250±0.0	0.016±0.0	2.98
	XGBoost	1.4255±0.01	0.02±0.0	0.58
	Duet	0.2611±0.0	0.015±0.0	0.23
mammography	RF	0.6671±0.01	0.029±0.0	2.18
	XGBoost	5.9221±0.08	0.150±0.0	1.09
	Duet	0.9052±0.01	0.027±0.0	1.60
letter	RF	20.078±0.16	1.2368±0.01	1236.19
	XGBoost	327.78±0.8	11.116±0.5	24.18
	Duet	80.818±0.39	3.6092±0.18	70.88
yeast	RF	0.1599±0.0	0.014±0.0	2.48
	XGBoost	3.4518±0.01	0.096±0.0	3.05
	Duet	0.8773±0.01	0.036±0.0	2.19
mushroom	RF	0.2125±0.01	0.019±0.0	0.34
	XGBoost	3.3613±0.23	0.035±0.01	0.32
	Duet	0.3585±0.01	0.02±0.0	0.39
cmc	RF	0.8387±0.0	0.075±0.0	41.78
	XGBoost	1.3554±0.01	0.039±0.0	1.82
	Duet	0.1458±0.0	0.01±0.0	0.30
krkopt	RF	17.169±0.23	1.1869±0.09	3328.46
	XGBoost	213.27±1.59	10.622±0.29	49.83
	Duet	54.129±0.42	9.9678±0.15	70.28
car	RF	1.2056±0.01	0.131±0.01	30.79
	XGBoost	3.0606±0.06	0.056±0.0	5.83
	Duet	1.2725±0.02	0.036±0.0	8.85
abalone	RF	5.8190±0.01	0.310±0.0	320.79
	XGBoost	18.759±0.11	0.6339±0.04	7.91
	Duet	6.2120±0.06	0.734±0.01	10.81
adult	RF	32.742±0.07	1.4587±0.0	168.20
	XGBoost	235.86±0.76	4.46720±0.08	30.93
	Duet	19.257±0.08	1.7114±0.03	2.62
communities	RF	3.2046±0.03	0.083±0.01	16.26
	XGBoost	44.454±0.55	0.1127±0.0	9.11
	Duet	14.282±0.15	0.0446±0.0	6.38
creditcardfraud	RF	96.382±4.12	0.3871±0.02	2.86
	XGBoost	654.51±21.18	2.93963±0.11	0.83
	Duet	204.11±10.71	0.53967±0.06	2.79
kddcup99	RF	22.176±0.27	1.4043±0.0	17.04
	XGBoost	5320.6±28.86	74.7549±2.47	5.37
	Duet	83.702±3.22	1.5192±0.01	22.62

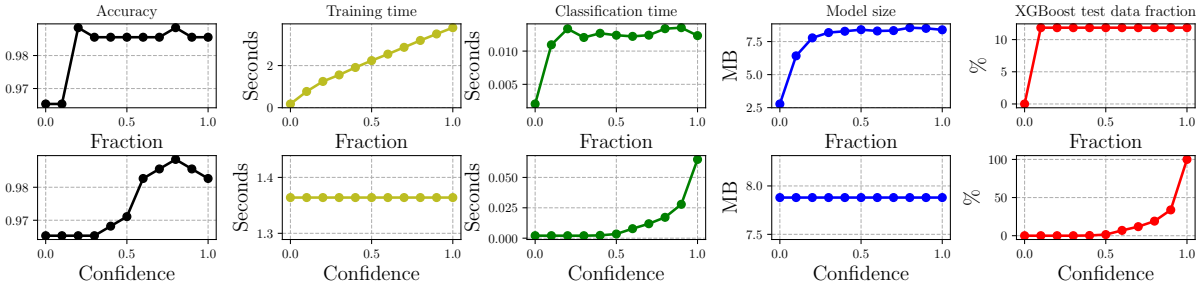
Table 5. Duet vs. monolithic RF and XGBoost. System performance comparison over `t2.2xlarge` AWS EC2 instance.

We are interested in the ML and system performance of Duet both on standard commodity servers and on weaker edge device with memory and compute constraints. Accordingly, we conducted our experiments over two different platforms: (1) a `t2.2xlarge` AWS EC2 instance with 8 vCPUs and 32 GB RAM running Ubuntu 16.04 LTS; (2) Raspberry Pi 3 B+ with a 1.4 GHz 64-bit quad-core ARM Cortex-A53 processor, 512 KB shared L2 cache, and 1GB SDRAM [33]. All classifiers are configured to run on a single core for fair training and classification time comparison.

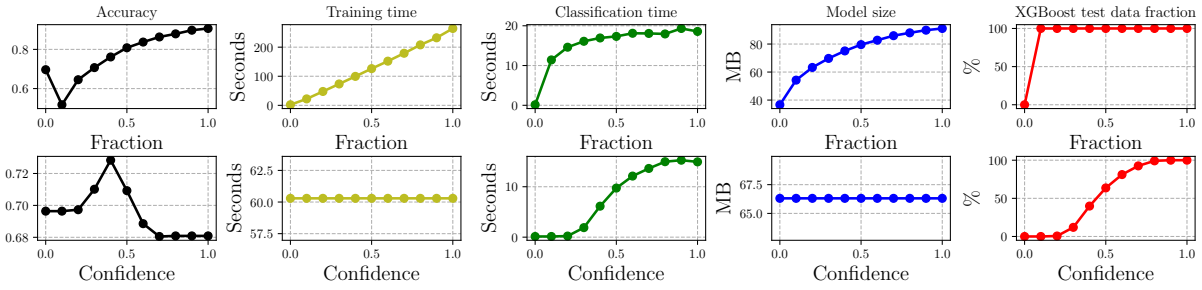
For all classifiers, we perform a grid-search with 5-fold cross-validation for parameter tuning. While both RF and XGBoost have well-established hyperparameters, Duet introduces new hyperparameters we tune: the data fraction vector (**`fraction`**) and the confidence threshold (**`confidence`**). Both parameters are fairly easily tunable. Common values we find to work well in practice are **`fraction`**`[total]` $\in [0.1, 0.25]$, **`fraction`**`[i]` $= \alpha \cdot \frac{\texttt{fraction[total]}}{\#classes}$, $\alpha \in [0, 1)$ for all `i`, and **`confidence`** $\in [0.7, 0.99]$. We also find that taking sensible monolithic XGBoost hyperparameters and using them in Duet works well.

4.1 Is it really a Duet?

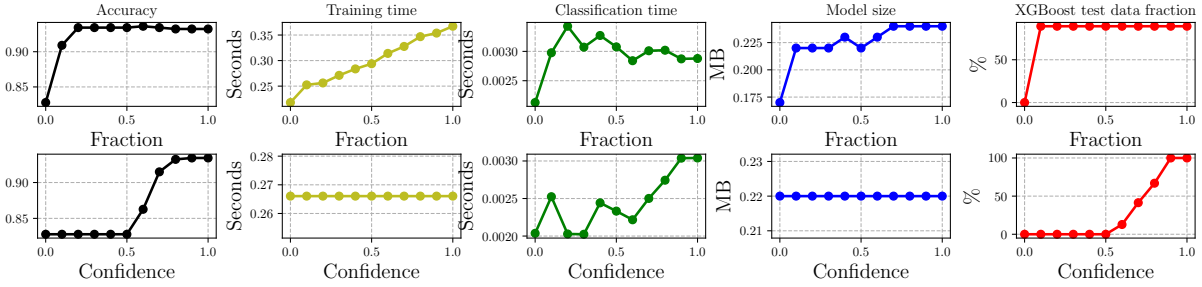
Before comparing Duet to monolithic RF and XGBoost models, we verify that often it is indeed the combination of both RF and XGBoost in Duet that yields the best benefits. Clearly, in some cases, Duet can rely purely on the XGBoost model and achieve similar ML capabilities with less memory consumption. Namely, the memory consumed by the RF model can be released after the XGBoost model is trained, and all classification queries can be forwarded directly to the XGBoost model. Often, we find this particular setting to be unfavorable. Using predictability purely as a sub-sampling approach is not always consistent and occasionally admits a higher variance in classification accuracy. It often does not allow a meaningful under-sampling factor without degrading ML performance. We illustrate this in Table 2. It shows how the complete Duet model is often more accurate than using



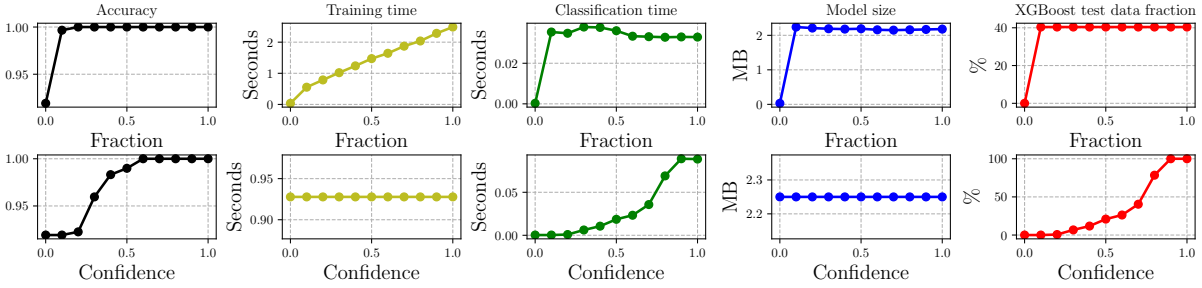
(a) car



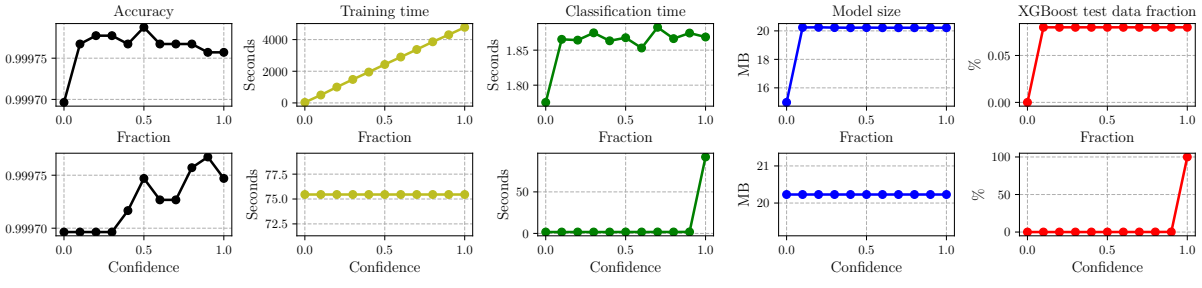
(b) krkopt



(c) seismic



(d) yeast



(e) kddcup99

Figure 3. The effect of Fraction and confidence parameters on Duet's performance.

only the resulting XGBoost model. It is worth noting that in our tests, we did not encounter an example in which using only the XGBoost model outperformed the complete Duet model in terms of accuracy in a non-negligible manner.

4.2 Duet vs. monolithic models

We evaluate Duet over Raspberry Pi 3 B+. It consists of a 1.4 GHz 64-bit quad-core ARM Cortex-A53 processor, 512 KB shared L2 cache, and 1GB SDRAM [33]. Table 4 summarises the results. It is evident how Duet often offers a better tradeoff between resource usage and classification accuracy.

On a more general note, RF often suffers from a large memory footprint (*e.g.,* cmc, communities) and XGBoost from long training and classification times (*e.g.,* letter, adult, creditcardfraud). Duet significantly reduces all – up to 11.5×, 7× and 143× for the training time, classification time, and the memory footprint, respectively. More rarely (*e.g.,* krkopt), the classification time of Duet is competitive to that of XGBoost. This happens when most classification queries are forwarded to the XGBoost model of Duet. Moreover, RF (4 datasets) and XGBoost (1 dataset) were not able to complete the training due to insufficient resources (marked with "–"). In particular, only Duet completed training over the kddcup99 dataset.

We also evaluated Duet over `t2.2xlarge` AWS EC2 instance with 8 vCPUs and 32 GB RAM running Ubuntu 16.04 LTS. Similarly, Duet significantly reduces the training time up to 63×, classification time up to 42× , and the memory footprint up to 47×. For example, for seismic, Duet achieves the best accuracy while its classification time is the fastest, its training time is 5.4× faster that of an XBGoost model and its memory footprint is 12.9× smaller than that of a RF.

4.3 How the fraction and confidence parameters affect Duet performance?

In Figure 3 we perform a parameter sweep over Duet parameters to illustrate how these affect ML and system performance.

We display the results over 5 datasets. While some results are more noisy than others, there are several evident trends that fall in line with the intuition. The accuracy increases with the dataset fraction that is forwarded to the training of the fine-grained model. For most datasets, this fraction can be quite small to achieve maximal accuracy (e.g, 10-25%). The confidence level for a valid classification by the coarse-grained model also admits sweet spots, usually in the range 0.6-0.95 yet again indicating that the confidence level is informative.

As for the system performance, as expected, it is the general trend that training times, classification times and memory footprint increase with the fraction and confidence. This is expected since larger fraction means longer training for the fine-grained model whereas higher confidence means that more test instances are classified by first the coarse-grained model and then by the fine-grained model due to insufficient classification confidence by the coarse-grained model.

5 Related work

In this section, we briefly overview most closely related work.

RADE. [44] is mostly related to our work and shares the idea of having a smaller model followed by expert models that are trained using only subsets of the data. However, RADE is designed and optimized for anomaly detection and does not support multiclass classification.

Instance hardness. Instance hardness (IH) was recently presented in [39] and later extended in [8, 19, 37, 38, 46, 49]. These works mainly target class imbalance. IH and our predictability measure are different concepts with different goals.

Cascading models. Cascading models are used for object detection (*e.g.,* image, video) [13, 36, 45], ranking [22], anomaly detection [27] and medicine [20, 47]. The main target of cascading models is fast classification at the price of resource-consuming training and increased memory footprint.

Stacking. Stacking [4, 17, 41, 48] differs from bagging and boosting by considering heterogeneous weak learners and combining them using a meta-model. Duet inherently differs from stacking where the $i+1$'th model is trained using the classification output of the i'th model.

Feature selection. [11, 23, 40] aim at selecting a subset of features that are sufficient for distinguishing instances that belong to different classes (orthogonal to Duet).

Sampling. To reduce training complexity one may use sampling [24, 29, 32]. As discussed in Section 4.1, these are inherently different than our use of predictability– *i.e.,* to build a boosting model that *complements* the bagging model and not replaces it. Also, these techniques are orthogonal to any classifier and hence applicable to Duet.

DTEM optimization. Previous work suggested approaches to tackle the drawbacks of bagging [2, 6, 42] and boosing [1, 9, 10, 21, 28, 31, 34, 35] DTEMs. Duet is mostly orthogonal to these, and we may leverage them for Duet's two classifiers.

6 Conclusions

We presented Duet, a DTEM-based classifier utilizing the power of data instance predictability. By combining the advantages of bagging and boosting DTEMs, Duet often achieves a better tradeoff between accuracy and system performance in terms of memory footprint, training time, and classification time compared to monolithic DTEMs.

Evaluation results indicate that Duet's training and classification times are up to 63× and 42× faster than that of a monolithic XBGoost model and its memory footprint is up to 47× smaller than that of a monolithic RF. Moreover, using a Raspberry Pi device, we have shown how datasets whose training failed with standard monolithic methods succeeded with Duet, widening the set of classification tasks that can be executed of resource limited edge devices.

Duet's scikit-learn compatible implementation and further details can be found in https://research.vmware.com/projects/efficient-machine-learning-classification.

References

[1] Ron Appel, Thomas Fuchs, Piotr Dollár, and Pietro Perona. 2013. Quickly boosting decision trees–pruning underachieving features early. In *International conference on machine learning*. 594–602.

[2] Nima Asadi, Jimmy Lin, and Arjen P De Vries. 2014. Runtime optimizations for tree-based machine learning models. *IEEE Transactions on Knowledge and Data Engineering* 26, 9 (2014), 2281–2292.

[3] L Breiman. 1996. ARCING classifiers (Technical report). *University of California, Department of Statistics* (1996).

[4] Leo Breiman. 1996. Stacked regressions. *Machine learning* 24, 1 (1996), 49–64.

[5] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[6] James Browne, Tyler Tomita, Disa Mhembere, Randal Burns, and Joshua Vogelstein. 2018. Forest Packing: Fast, Parallel Decision Forests. *arXiv preprint arXiv:1806.07300* (2018).

[7] Lars Buitinck, Gilles Louppe, Mathieu Blondel, et al. 2013. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 108–122.

[8] George DC Cavalcanti and Rodolfo JO Soares. 2020. Ranking-based instance selection for pattern classification. *Expert Systems with Applications* 150 (2020), 113269.

[9] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 785–794.

[10] Fang Chu and Carlo Zaniolo. 2004. Fast and light boosting for adaptive mining of data streams. In *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 282–292.

[11] Manoranjan Dash and Huan Liu. 1997. Feature selection for classification. *Intelligent data analysis* 1, 1-4 (1997), 131–156.

[12] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[13] Pedro F Felzenszwalb, Ross B Girshick, and David McAllester. 2010. Cascade object detection with deformable part models. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2241–2248.

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

[15] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[16] Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Machine learning* 63, 1 (2006), 3–42.

[17] Magdalena Graczyk, Tadeusz Lasota, Bogdan Trawiński, and Krzysztof Trawiński. 2010. Comparison of bagging, boosting and stacking ensembles applied to real estate appraisal. In *Asian conference on intelligent information and database systems*. Springer, 340–350.

[18] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. Unsupervised learning. In *The elements of statistical learning*. Springer, 485–585.

[19] Ahmedul Kabir, Carolina Ruiz, and Sergio A Alvarez. 2018. Mixed bagging: A novel ensemble learning framework for supervised classification based on instance hardness. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1073–1078.

[20] Asha Gowda Karegowda, MA Jayaram, and AS Manjunath. 2012. Cascading k-means clustering and k-nearest neighbor classifier for categorization of diabetic patients. *International Journal of Engineering and Advanced Technology* 1, 3 (2012), 147–151.

[21] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*. 3146–3154.

[22] Branislav Kveton, Csaba Szepesvari, Zheng Wen, and Azin Ashkan. 2015. Cascading bandits: Learning to rank in the cascade model. In *International Conference on Machine Learning*. 767–776.

[23] Nojun Kwak and Chong-Ho Choi. 2002. Input feature selection for classification problems. *IEEE transactions on neural networks* 13, 1 (2002), 143–159.

[24] Rui Leite and Pavel Brazdil. 2004. Improving progressive sampling via meta-learning on learning curves. In *European Conference on Machine Learning*. Springer, 250–261.

[25] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server.. In *OSDI*, Vol. 14. 583–598.

[26] Machine Learning Group - ULB. 2013. Credit Card Fraud Detection Dataset. https://www.kaggle.com/isaikumar/creditcardfraud. [Online; accessed 26-August-2019].

[27] Amuthan Prabakar Muniyandi, R Rajeswari, and R Rajaram. 2012. Network anomaly detection by cascading k-Means clustering and C4. 5 decision tree algorithm. *Procedia Engineering* 30 (2012), 174–182.

[28] Matthew Olson. 2017. JOUSBoost: An R Package for Improving Machine Learning Classifier Probability Estimates.

[29] Srinivasan Parthasarathy. 2002. Efficient progressive sampling for association rules. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* IEEE, 354–361.

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

[31] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*. 6638–6648.

[32] Foster Provost, David Jensen, and Tim Oates. 1999. Efficient progressive sampling. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 23–32.

[33] Raspberry Pi 3. 2018. *Model B+*. [Online; accessed 7-September-2019].

[34] Abolfazl Ravanshad. 2018. *Medium Corporation. Gradient Boosting vs Random Forest*. [Online; accessed 7-September-2019].

[35] Mojtaba Seyedhosseini, António RC Paiva, and Tolga Tasdizen. 2011. Fast adaboost training using weighted novelty selection. In *The International Joint Conference on Neural Networks*. IEEE, 1245–1250.

[36] Haichen Shen, Seungyeop Han, Matthai Philipose, and Arvind Krishnamurthy. 2017. Fast video classification via adaptive cascading of deep models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3646–3654.

[37] William C Sleeman IV and Bartosz Krawczyk. 2019. Bagging Using Instance-Level Difficulty for Multi-Class Imbalanced Big Data Classification on Spark. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2484–2493.

[38] Michael R Smith and Tony Martinez. 2016. A comparative evaluation of curriculum learning with filtering and boosting in supervised classification problems. *Computational Intelligence* 32, 2 (2016), 167–195.

[39] Michael R Smith, Tony Martinez, and Christophe Giraud-Carrier. 2014. An instance level analysis of data complexity. *Machine learning* 95, 2 (2014), 225–256.

[40] Jiliang Tang, Salem Alelyani, and Huan Liu. 2014. Feature selection for classification: A review. *Data classification: Algorithms and applications* (2014), 37.

[41] Kai Ming Ting and Ian H Witten. 1997. Stacking bagged and dagged models. (1997).

[42] Brian Van Essen, Chris Macaraeg, Maya Gokhale, and Ryan Prenger. 2012. Accelerating a random forest classifier: Multi-core, GP-GPU, or FPGA?. In *20th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 232–239.

[43] Shay Vargaftik and Yaniv Ben-Itzhak. 2022. Duet's classifier (v1.0) - scikit-learn compatible. https://research.vmware.com/projects/efficient-machine-learning-classification.

[44] Shay Vargaftik, Isaac Keslassy, Ariel Orda, and Yaniv Ben-Itzhak. 2021. Rade: Resource-efficient supervised anomaly detection using decision tree-based ensemble methods. *Machine Learning* 110, 10 (2021), 2835–2866.

[45] Paul Viola, Michael Jones, et al. 2001. Rapid object detection using a boosted cascade of simple features. *CVPR (1)* 1, 511-518 (2001), 3.

[46] Felipe N Walmsley, George DC Cavalcanti, Dayvid VR Oliveira, Rafael MO Cruz, and Robert Sabourin. 2018. An ensemble generation method based on instance hardness. In *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[47] Yefeng Wang and Jon Patrick. 2009. Cascading classifiers for named entity recognition in clinical notes. In *Proceedings of the workshop on biomedical information extraction*. Association for Computational Linguistics, 42–49.

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

[49] Jianjun Zhang, Ting Wang, Wing WY Ng, Shuai Zhang, and Chris D Nugent. 2019. Undersampling Near Decision Boundary for Imbalance Problems. In *2019 International Conference on Machine Learning and Cybernetics (ICMLC)*. IEEE, 1–8.

[50] Yue Zhao, Zain Nasrullah, and Zheng Li. 2019. Pyod: A python toolbox for scalable outlier detection. *arXiv preprint arXiv:1901.01588* (2019).