DIOPT: Extremely Fast Classification Using Lookups and Optimal Feature Discretization

Johan Garcia and Topi Korhonen
Department of Mathematics and Computer Science
Karlstad University, Sweden
Email: `firstname.lastname@kau.se`

***Abstract*—For low dimensional classification problems we propose the novel DIOPT approach which considers the construction of a discretized feature space. Predictions for all cells in this space are obtained by means of a reference classifier and the class labels are stored in a lookup table generated by enumerating the complete space. This then leads to extremely high classification throughput as inference consists only of discretizing the relevant features and reading the class label from the lookup table index corresponding to the concatenation of the discretized feature bin indices. Since the size of the lookup table is limited due to memory constraints, the selection of optimal features and their respective discretization levels is paramount. We propose a particular supervised discretization approach striving to achieve maximal class separation of the discretized features, and further employ a purpose-built memetic algorithm to search towards the optimal selection of features and discretization levels. The inference run time and classification accuracy of DIOPT is compared to benchmark random forest and decision tree classifiers in several publicly available data sets. Orders of magnitude improvements are recorded in classification runtime with insignificant or modest degradation in classification accuracy for many of the evaluated binary classification tasks.**

I. INTRODUCTION

Machine learning is utilized to address a wide range of problems across a very wide range of scales. This work focuses on problem settings where the classification throughput in terms of classifications per second for the trained model is of major importance. Traffic classification in computer networks is one such setting where, as stated by [32], the maximum possible packet rate can be up to 14.2 million packets per second over common 10 Gbps links and the rate is correspondingly higher for the faster operator backbone links. Current machine learning techniques vary considerably in their computational demands for classification. While a Decision Tree (DT) such as C4.5 [30] has very low computational demands, it can be prone to overfitting and may not generalize well. As an extension of DT, Random Forests (RF) [3] and other similar ensemble-based classifiers have been proposed. In [7] RF variations were considered the best out of 179 evaluated classifiers from 17 families. Compared to DT, RF provides better classification performance, but also requires more computational resources to perform the classification. Here we study the prospect of using additional computational resources during the classifier construction phase, in order to instantiate a very high throughput classifier which also has appropriate classification performance. Our proposed solution entails performing a tailored supervised discretization and a careful optimization to obtain the most salient features, each represented with the ideal amount of discretization. The resulting discretized feature space is then completely enumerated by means of discretization bin index concatenation, thus creating a single index, which is used to realize a lookup table. This lookup table is filled by means of a reference classifier. The resulting index-enumerated lookup-table based classifier can perform classifications very rapidly as the only online operations necessary are discretization of the salient features, bin index concatenation, and lookup of the class label.

Based on the reliance of both discretization (DI) and optimization (OPT), we introduce the acronym DIOPT for the proposed approach. Key contributions of this work are:

1) the DIOPT structure of combining discretization, GA optimization and lookup-table pre-filling,
2) the KSD-MI discretization algorithm which employs KS distance to retain maximum discriminatory power,
3) an experimental evaluation comparing DIOPT with existing classifiers over a range of data sets to understand accuracy, computation time and memory trade-offs.

The results demonstrate that DIOPT can be competitive to random forests in accuracy but orders of magnitude faster at evaluation time. In the next section related work is discussed, followed by Section III which describes the overall DIOPT approach. Section IV covers the KSD-MI discretization algorithm, while Section V describes the hGAm optimization algorithm used for the crucial selection of features and their discretization level. Section VI provides an evaluation over a range of data sets, followed by discussion and conclusions in Section VII.

II. RELATED WORK

The need for high-speed inference in a number of application areas is illustrated by the research into using hardware such as GPUs [36], field programmable gate arrays (FPGA) [1], [26], [29], or clusters of FPGAs [25] to achieve very high speed classification for decision trees, or ensembles of them. In contrast, the our present work aims to achieve increased classification throughput on the CPU, without employing any specialized hardware.

At a conceptual level, the present work could be seen as related to the model distillation approach originally proposed by Hinton et al. [18]. Examples of model distillation include going from a large neural net to a smaller neural net [18], and from a neural net to a decision tree [9]. These approaches utilize a more complex model to aid in the construction of a smaller and faster model. The main benefit of the model distillation approach is that the small model obtained by model distillation has better classification performance than a model of the same size trained directly on the training data. Similarly to model distillation, the proposed DIOPT approach uses a more time-consuming model (random forest) to obtain a faster model (DIOPT) which has better classification performance than an almost similarly fast model (decision tree). However, as the DIOPT simplification is mainly a result of data discretization rather than model simplification the approaches are not equivalent. A precursor to DIOPT which utilizes a similar lookup-based approach but with other discretization and simple random search instead of a meta-heuristic is presented in [12].

Considering other fast prediction approaches, there have been strong interest in approaches that minimize the error while minimizing prediction time, and many of them focus on building "top-down" decision functions where features are included based on their utility [35], [40]. Many of those approaches are focusing on utilizing the feature capture cost, which however is not a consideration in our work. Pruning of the base classifiers in an ensemble such as RF have also been an area of active research [24], [39], [41]. In relation to DIOPT this can be viewed as implicitly reducing the number of bin boundaries created by the set of all split decisions in the ensemble. Approaching the issue of feature discretization from the perspective of model simplification, [17] proposes a method to 'defrag' the small subspaces created by the tree ensemble through means of Bayesian model selection. This allows the construction of simpler, and thus faster, models which retain much of the classification performance. Results are only provided for a few data sets, but a their reported results for the spambase data set indicate a classification performance likely worse, but more interpretable than DIOPT. High throughput classification has also been considered in [19], which uses various approaches including unsupervised equal-frequency binning in their FastBDT variation of Gradient Boosted Decision Trees. (GBDT). They report order of magnitude improvements in execution time relative to other GBDT implementations. However, their reported classification times are still orders of magnitude worse than DIOPT.

In addition to methods that essentially strive to simplify a complex ensemble model, an alternative approach is to create a relatively simple DT which has a close to optimal structure. This is the approach proposed by [2], which strives to locate the optimal splits. However, such approach have a worst-case exponential cost, and is limited in the number of training instances (10K instances give prohibitive cost, [2, sec. 6]. Some improvements of [2] in terms of scalability have been provided in [38]. Other approaches related to tree optimization have been reported in [4] which strive to optimize the misclassification error, jointly and iteratively, over all nodes. An alternate DT improvement approach is proposed by [31], which discusses an efficient approach to enumerate all unique DT as a way of building the optimal feature set. This approach does not have a limitation in the number of instances, but appears to give less of an improvement over the base DT. In the evaluation section the reported performance of several of the above works are compared to DIOPT for a subset of data sets. However, we note that few of the works ([19] is an exception) report results for large scale data sets which are of primary interest here.

III. THE DIOPT APPROACH

The proposed DIOPT approach employs discretization of feature values in a novel way, and utilizes optimization to strive towards using the most descriptive features at their most appropriate discretization level. This enables the construction of a classifier ($\mathbb{C}_{\mathrm{LU}}$) comprised of a lookup table populated with pre-predicted class labels, and the subsequent achievement of very high classification throughput.

A DIOPT classifier is created with a target maximum memory size, S_{max}, which is typically related to the target computing environment. Further, as $\mathbb{C}_{\mathrm{LU}}$ is populated using predictions from a reference classifier ($\mathbb{C}_{\mathrm{R}}$), a suitable $\mathbb{C}_{\mathrm{R}}$ needs to be chosen. While any arbitrary classifier such as Random Forest (RF), GBM [8], SVM [5], or one of their many derivatives could be employed as $\mathbb{C}_{\mathrm{R}}$, for larger S_{max} the computational requirements of $\mathbb{C}_{\mathrm{R}}$ become important. For the work presented here we are using RF for $\mathbb{C}_{\mathrm{R}}$ as it has been shown to provide excellent all-round classification performance [7], and can achieve low classification runtimes [13].

Now consider a classification task $\mathcal{T}$ with data $\mathbf{X}_{[N\times M]}$, where N is the number of instances and M is the number of features. Further, there are observed data instances $\mathbf{x}_i = \mathrm{row}[\mathbf{X}]_i$, feature values of the j:th feature $\mathbf{f}_j = \mathrm{col}[\mathbf{X}]_j = [f_{1,j}, \ldots, f_{N,j}]^T$, and class labels $\mathbf{y} = [y_1, \ldots, y_N]^T$.

To begin constructing $\mathbb{C}_{\mathrm{LU}}$ all numerical features need to be discretized. A per feature 'bit budget' A_j is introduced, signifying that a discretized feature $\hat{\mathbf{f}}_j$ is allocated A_j bits for representation. For example, if feature $\mathbf{f}_j$ should be discretized into 128 bins, that requires $A_j = \log_2(128) = 7$ bits so that the bin index value $\hat{\mathbf{f}}_j$ of the discretized feature can be represented. Further, there is a bit allocation vector $\mathbf{A} = [A_1, \ldots, A_M]$ denoting the number of bits allocated to each of the M features. The total total number of bits, L, allocated for some allocation $\mathbf{A}$ is simply $L_{\mathbf{A}} = \sum \mathbf{A}$. The amount of memory, $S_{\mathbf{A}}$, required to hold the $\mathbb{C}_{\mathrm{LU}}$ for some $\mathbf{A}$ is computed as $S_{\mathbf{A}} = 2^{L_{\mathbf{A}}-3}$ bytes for a 2-class classifier (where one byte can be viewed as holding $8 = 2^3$ binary label slots). In a multi class scenario, $S_{\mathbf{A}}$ is adjusted by a factor of $\lceil \log_2 n_c \rceil$ where n_c is the number of classes for $\mathcal{T}$. In the remainder, we focus the presentation on the 2-class case.

The process of generating $\mathbf{A}$ is denoted as 'bit allocation' and is of essential importance, simultaneously selecting features and their discretization levels within a limited bit budget. The memory constraint $S_{\mathbf{A}} \leq S_{\mathrm{max}}$ results in a total bit budget $L \leq \log_2(S_{\mathrm{max}}/\lceil \log_2 n_c \rceil) + 3$ when S_{max} is provided in bytes. For a 2-class example, when $S_{max} = 4$ GiB then $L \leq \log_2(2^2 \cdot 2^{30}/1) + 3 \implies L \leq 35$. Thus, DIOPT classifiers are restricted to relatively few features with somewhat modest discretization levels as in practice L is expected to be less than 40 in order to avoid very large lookup tables. However, it turns out that in many cases even small lookup tables are able to

Algorithm 1 DIOPT overview

```
Learning phase:
1: $\mathcal{B} \leftarrow$ learnBinBoundaries($\mathbf{f}_j^{\text{tr}}, \mathbf{y}^{\text{tr}}, t$)  $\forall j$
2: $\hat{\mathcal{F}} \leftarrow$ buildDiscretizedFeatureset($\mathcal{B}, \mathbf{f}_j$)  $\forall j$
3: $\mathbf{A} \leftarrow$ doBitAllocation($\hat{\mathcal{F}}, \mathbf{y}, L$)
4: $\hat{\mathbf{f}}_{\mathbf{A}} \leftarrow$ extractDiscretizedFeatures($\hat{\mathcal{F}}, \mathbf{A}$)
5: $\mathbb{C}_{\mathbb{R}} \leftarrow$ trainReferenceClassifier($\hat{\mathbf{f}}_{\mathbf{A}}, \mathbf{y}$)
6: $\mathbb{C}_{\mathbb{LU}} \leftarrow$ fillLookUpTable($\mathbb{C}_{\mathbb{R}}$)
Inference:
1: $\hat{\mathbf{x}} \leftarrow$ discretizeInstance($\mathbf{x}, \mathcal{B}, \mathbf{A}$)
2: $i \leftarrow$ generateLUindex($\hat{\mathbf{x}}$)
3: $c \leftarrow$ readClassLabel(LU, $i$)
```

capture the necessary information of the underlying data set and provide strong classification performance.

Now turning to the major steps in building a DIOPT classifier, Algorithm 1 provides an overview description. The first step is to learn bin boundaries which can be employed to discretize the features. This step can be seen as a supervised discretization training process, denoted as $B()$, which is done using a discretization training subset $\mathbf{f}_j^{\text{tr}}$ of the overall training set values for some feature j, by means of some specific discretization method. Here, discretization is performed by the KSD-MI method as described in the next section. From $B()$ the bin boundaries used to discretize a feature j into a number of bins corresponding to its bit allocation A_j is obtained, i.e. $B(A_j, \mathbf{f}_j^{\text{tr}}) = [b_{1,j}, \ldots, b_{2^{A_j}-1,j}] = \mathbf{b}_j$. In line 1 a set of sets, $\mathcal{B}$, is constructed in which every feature has a set of bin boundaries corresponding to all possible bit allocations $A_j \in \{1, \ldots, t\}$, i.e. starting from 1 bit and going up to some maximum number of bits per feature t. With the learned bin boundaries, a discretization function $D()$ can now be applied to all feature values, i.e. $D(\mathbf{b}_j, \mathbf{f}_j) = \hat{\mathbf{f}}_j$ thus mapping the numerical values of feature $\mathbf{f}_j$ to bin indexes $\hat{\mathbf{f}}_j$ using bin boundaries $\mathbf{b}_j$ so that, $b_{kj} \leq f_{i,j} < b_{k+1,j} \Rightarrow \hat{f}_{i,j} = k$. For each feature $\mathbf{f}_j$ the operation in line 2 creates a set of corresponding discretized features, one for each $A_j \in \{1, \ldots, t\}$, and all these features are held in $\hat{\mathcal{F}}$.

With the set of all discretized features $\hat{\mathcal{F}}$ enabling all possible allocation values $\{1, \ldots, t\}$ for all features, the optimal allocation of bits to each feature is searched for in line 3. For an allocation $\mathbf{A}$, the corresponding discretized feature set can be obtained as

$$\forall j \quad \text{col}[\hat{\mathbf{f}}_{\mathbf{A}}]_j := \begin{cases} D(B(A_j, \mathbf{f}_j^{\text{tr}}), \mathbf{f}_j) & \text{for } A_j \neq 0 \\ \text{discard otherwise.} \end{cases} \tag{1}$$

In our case the search is done with the hGAm optimizer as detailed in a later section. When the search has terminated and the best candidate bit allocation $\mathbf{A}$ for some task $\mathcal{T}$ and memory constraint $S_{\max}$ has been obtained, the step in line 4 extracts the corresponding discretized feature values $\hat{\mathbf{f}}_{\mathbf{A}}$. Then, in line 5 the reference classifier $\mathbb{C}_{\text{R}}$ is trained on the extracted discretized training data.

To create the $\mathbb{C}_{\text{LU}}$ classifier its lookup table needs to be populated. Thus, using $\mathbb{C}_{\text{R}}$, the instantiation of the enumerated classification model $\mathbb{C}_{\text{LU}}$ is performed. The complete discretized feature space is transformed to a single index by means of a function q. For ease of notation, let us denote $\mathscr{A}_j := A_1 + \cdots + A_j$. Then, a single overall index for the enumerated classifier considering the M' non-discarded features becomes $q(\hat{\mathbf{x}}) := \hat{\mathbf{f}}_1 2^0 + \hat{\mathbf{f}}_2 2^{\mathscr{A}_1} + \cdots + \hat{\mathbf{f}}_{M'} 2^{\mathscr{A}_{M'-1}}$ as the value domain of $\hat{\mathbf{f}}_j$ is $[0 \ldots 2^{A_j} - 1]$. In practice, the single index is constructed by bit shifting of the discretized values of the considered features. For the inference, the necessary steps are to discretize the features, do bit shifts to construct the single index, and lookup the class label. To obtain high-speed discretization during inference several variants of feature value to bin-index mapping have been implemented. Different variants are employed for different features as appropriate for their data type and value domain. For example, integer data types with limited value domains are discretized using a feature value to bin index lookup table which is a very time efficient operation. In summary, the simplicity of the necessary classification operations result in very low classification times.

IV. KSD-MI Supervised Discretization

As the memory available to hold $\mathbb{C}_{\text{LU}}$ is limited, discretizing the feature values to a limited number of bins becomes a central aspect. The feature discretization should preserve as much as possible of the features discriminative power between classes, thus striving to maximize classifier performance when the discretized features are employed. Many discretization approaches have been proposed in the literature, and surveys are available in [14], [22]. In this work we propose KSD-MI as an improvement over the KSD [11] approach for this usage domain. KSD-MI uses mutual information [33] as bin boundary selection mechanism, rather than the tightly integrated wrapper approach proposed in KSD. KSD-MI is a top-down discretizer utilizing the Kolmogorov statistic [20], which is also referred to as the Kolmogorov-Smirnov statistic after the widely used Kolmogorov-Smirnov statistical test [23]. Henceforth, it will be referred to as the Kolmogorov-Smirnov (KS) distance as the use here is not related to statistical testing, but rather to work as an aid for deciding bin placement. We now provide a succint description of KSD-MI and refer the reader to [11] for a more elaborate illustration of the KSD approach, and a comparison to other approaches.

Since KSD-MI considers only a single feature at a time, the feature index j has been excluded in the following description. The KS distance is used to iteratively place new candidate bin boundary value between two already existing bin boundary values. Two sets of values $\mathbf{f}^c := \{f_i \in \mathbf{f} | y_i = c\}$ are considered, that is, values f_i so that class $y_i = c$, where $c \in \{0, 1\}$. The KS distance is computed on the empirical cumulative probability distribution function (eCDF) which for some value v in the value domain of $\mathbf{f}$ is defined as $E_c(v) = n_c^{-1} \sum_{i=1}^{n_c} \mathcal{I}(f_i^c < v)$. Here, $\mathcal{I}(\omega)$ is the indicator function attaining value 1 if ω is true and 0 otherwise, f_i^c is the i:th observation of set $\mathbf{f}^c$, and $n_c = |\mathbf{f}^c|$. The two eCDF's are compared and the KS distance computed as $\text{KS} = |E_0(v) - E_1(v)|$. A new candidate bin boundary is then placed at the location v where the KS distance is maximized

$$v = \arg\max_v |E_0(v) - E_1(v)|. \tag{2}$$

Algorithm 2 KSD discretization algorithm

```
1: $\mathbf{b} \leftarrow [f_{\min}, f_{\max}]$
2: while $|\mathbf{b}| < 2^A + 1$ do
3:     $\mathbf{v} \leftarrow []$
4:     for $k$ in $1 \ldots |\mathbf{b}| - 1$ do
5:         $\mathbf{f}^{k,c} \leftarrow \{f \in \mathbf{f}^c | b_k \leq f^c < b_{k+1}\},\ c \in \{0,1\}$
6:         $n_{k,c} \leftarrow |\mathbf{f}^{k,c}|,\ c \in \{0,1\}$
7:         $E_{k,c}(v) \leftarrow \frac{1}{n_{k,c}} \sum_{i=1}^{n_{k,c}} I(f_i^{k,c} < v),\ c \in \{0,1\}$
8:         $\mathbf{v}[k] \leftarrow \arg\max_v |E_{k,0}(v) - E_{k,1}(v)|$
9:     end for
10:    $\mathbf{b} \leftarrow \mathbf{b} \cup \left( \arg\max_{v \in \mathbf{v}} I(D(\mathbf{b} \cup v, \mathbf{f}); \mathbf{y}) \right)$
11: end while
12: return $\mathbf{b} \setminus \{f_{\min}, f_{\max}\}$
```

Pseudocode for KSD-MI is provided in Algorithm 2 where $f^{k,c} := \{f \in \mathbf{f}^c | b_k \leq f^c < b_{k+1}\}$ are the observed values from f^c that are within the interval $[b_k, b_{k+1})$, and $n_{k,c} := |f^{k,c}|$ is the number of observations in the current interval. At the start the ultimate boundaries $f_{\min}$ and $f_{\max}$ are readily available and thus initially $\mathbf{b}$ becomes $[f_{\min}, f_{\max}]$. The inner loop in lines 4 to 9 constructs the eCDF's $E_{k,c}$ within the interval of the k:th of the current bins for both classes, and assigns v the value giving the maximal KS distance, as stated in Eq. (2). At each inner iteration a candidate boundary location v is recorded into the vector $\mathbf{v}$. After iterating over all current bins, the v maximizing the mutual information given the current boundaries is added to the existing boundaries, line 10. Which candidate bin boundary value in $\mathbf{v}$ to keep is thus determined by mutual information which for some discrete random variables X and Y is defined as

$$I(X;Y) = \sum_{x,y} p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)} \tag{3}$$

Here, we consider $I(D(\mathbf{b} \cup v, \mathbf{f}); \mathbf{y})$ i.e the mutual information between class labels $\mathbf{y}$ and the discretized variable $D(\mathbf{b} \cup v, \mathbf{f})$, where $\mathbf{b} \cup v$ are the current established bin boundaries $\mathbf{b}$ combined with an additional candidate bin boundary v. The boundaries are always ordered. The outer iteration is continued until the desired number of bins have been found or it is not feasible to introduce new bins, i.e., the number of bins has become equal to the number of unique values in feature $\mathbf{f}$. When finally returning the set of bin boundaries the initial extreme value boundaries are discarded, and when performing discretization the first and last bins extend to negative and positive infinity thus also capturing previously unseen extreme feature values. We note that DIOPT requires a discretization approach where the desired number of bins can be specified, a property not necessarily present for other supervised discretization approaches.

V. SELECTING OPTIMAL FEATURES AND DISCRETIZATION LEVELS

The initial discretization step in DIOPT produces multiple discretized representations for all features, which are stored in $\hat{\mathcal{F}}$. DIOPT uses $\hat{\mathcal{F}}$ when it, in addition to traditional feature selection [15], [21], also performs the selection of discretization

Algorithm 3 hGAm algorithm

```
1: $\mathsf{P} \leftarrow \mathsf{initalizePop}(\mathsf{popsize}, L, t)$
2: while nr non-improving generatns $\leq$ maxNonImprov do
3:     for i in $1 \ldots |\mathsf{P}|$ do
4:         $\mathbf{A} \leftarrow \mathsf{P_i}$
5:         $\mathsf{g[i]} \leftarrow G(\mathbf{\Theta}_d, \hat{\mathbf{f}}_{\mathbf{A}}, \mathbf{y})$
6:     end for
7:     $\mathsf{M} \leftarrow \mathsf{parentSelect}(\mathsf{P}, \mathsf{breedFrac}, \mathbf{g})$
8:     $\mathsf{C} \leftarrow \mathsf{recombine}(\mathsf{M}, \mathsf{moveFrac})$
9:     $\mathsf{C} \leftarrow \mathsf{mutate}(\mathsf{C}, \mathsf{mut})$
10:    $\mathsf{P} \leftarrow \mathsf{C} + \mathsf{eliteSelect}(\mathsf{P}, \mathsf{survFrac}, \mathbf{g})$
11:    if current generation number $\bmod$ locIntvl $= 0$ then
12:        $\mathsf{P_{worst}} \leftarrow \mathsf{localsearch}(\mathsf{P_{best}})$
13:    end if
14: end while
15: return $\mathsf{P_{best}}$
```

levels in order to efficiently manage the limited total number of bits, L, that result from the S_{max} memory constraint. To address this optimization problem we have designed a novel custom hybrid genetic algorithm (GA) [34, pg. 50] named hGAm which enforces the bit count preservation requirement of DIOPT. It is imperative to seek for the most descriptive combined discretized feature set $\hat{\mathbf{f}}_{\mathbf{A}}$ while accounting for the memory constraint. This process is called bit allocation and is responsible for selection of the used features and their discretization levels. Each candidate allocation is constructed utilizing the set of discretized features, $\hat{\mathcal{F}}$, in which each feature j exists with all possible discretization levels $A_j \in \{1, \ldots, t\}$.

The proposed bit allocation method is based on estimating the utility of some allocation vector $\mathbf{A}$ by directly observing the $\mathbb{C}_{\mathrm{R}}$ classification performance in an $\hat{\mathbf{f}}_{\mathbf{A}}$ validation set. As optimization target a metric G (here accuracy), is used so that we directly seek for $\mathbf{A} = \arg\max_{\mathbf{A}} G(\mathbf{\Theta}_d, \hat{\mathbf{f}}_{\mathbf{A}}, \mathbf{y})$, where $\mathbf{\Theta}_d$ define the classifier parameters. An overall description of hGAm is presented in Algorithm 3 and the various configuration parameters are given in Table I. The selection of configuration parameters was aided by meta-optimization on a multiple choice quadratic knapsack (MCQKP) surrogate problem, as further elaborated in [10] which also provide an extended presentation of hGAm.

The algorithm starts with creating a population P of randomly

TABLE I: hGAm algorithm parameters

Parameter	Short	Value
Initial population size	popsize	2000
Max nr of bits per feature	t	10
Max generations w/o improvement	maxNonImprov	5
Breeding fraction of the best candidates in population	breedFrac	0.5
During recombination move fraction of differing bits	moveFrac	0.15
Child mutation probability	mut	0.25
Nr of mutation loops per allocation	nMut	1
Elite survival fraction	survFrac	0.1
Local search interval	locIntvl	5
Number of the most important RFE features provided to GA	nFeat	15

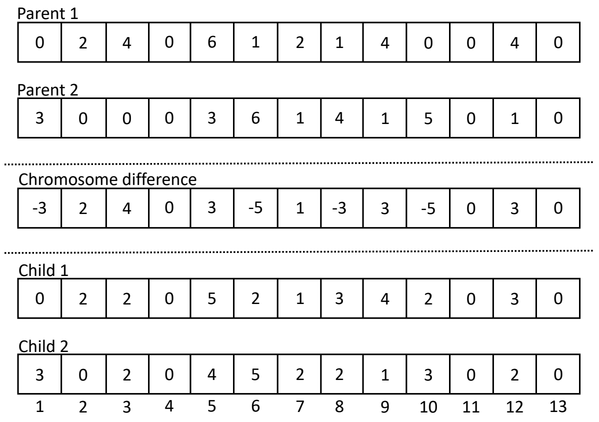


Fig. 1: hGAm recombination

generated candidate bit allocations where each individual $\mathsf{P_i} = rand(\mathbf{A}) | 0 \leq A_j \leq t\ \forall j,\ \sum_j A_j = L$ (line 1). The algorithm then evolves by obtaining the classification performance metric scores $\mathbf{g}$ for all individuals in P (lines 3 to 6), selecting parents M (line 7), creating children C (lines 8 to 10), and performing local search every $\mathsf{locIntvl}$ generations.

Selecting which parents to perform mating on is done by randomly choosing from $\mathsf{breedFrac}$ of the most fit individuals in P (line 7), that is, the ones having the largest scores g. Traditionally, a binary representation is used by GAs as this allows for a number of recombination operations and straightforward mutation procedures. However, for the DIOPT bit allocation task a bit-based GA representation is problematic due to the need to convey both the selection of features and the number of bits allocated to these features. To address this, hGAm uses a tailored chromosome representation. This specialized representation format also requires an adapted recombination operation, both to handle the specifics of the format as well as to ensure that feasible solutions are produced by the recombination. The recombination procedure of hGAm is illustrated in Figure 1. In this example $M = 13$ features, $t = 10$ bits, and $L = 24$ bits. To perform the recombination a chromosome difference vector $\Delta = \mathbf{A}_{\mathsf{p1}} - \mathbf{A}_{\mathsf{p2}}$ represents the difference in bit allocation between parents $\mathbf{A}_{\mathsf{p1}}$ and $\mathbf{A}_{\mathsf{p2}}$. Children $\mathbf{A}_{\mathsf{c1}}$,$\mathbf{A}_{\mathsf{c2}}$ are initially copies of $\mathbf{A}_{\mathsf{p1}}$, $\mathbf{A}_{\mathsf{p2}}$ before a fraction ($\mathsf{moveFrac}$) of the $\sum_{0<\Delta_j} \Delta_j$ differing bits are moved. Bits are moved from randomly selected genes $j \in \{j | 0 < \Delta_j\}$ in $\mathbf{A}_{\mathsf{c1}}$ to the same genes in $\mathbf{A}_{\mathsf{c2}}$. The above is repeated for $\Delta_j < 0$ now moving bits from $\mathbf{A}_{\mathsf{c2}}$ to the same genes in $\mathbf{A}_{\mathsf{c1}}$. This approach conserves bits, i.e., children have equal number of bits to parents, and resembles merging of the bit allocation characteristics of the parents. In the example in the figure, a fraction of five out of the $\sum_{0<\Delta_j} \Delta_j = 16$ differing bits are moved in each direction.

After recombination a randomly selected fraction (mut) of the children is further subjected to mutation. Mutation repeatedly (nMut) removes a bit from randomly selected feature j and adds it to another randomly selected feature k, i.e., $A_j = A_j - 1$ and $A_k = A_k + 1$ (line 9). Following mutation a new generation is formed, however, a fraction $\mathsf{survFrac}$ of the most fit individuals in the current generation are carried over to the next generation unchanged (line 10).

At $\mathsf{locIntvl}$ intervals, hGAm incorporates local search on the fittest individual $\mathsf{P_{best}}$. With the hGAm representation, local search encompasses moving a single bit at a time from all $A_j \neq 0$, to all $\{A_k | k \neq j, A_k < t\}$ and observing the effect on the metric G. If any such bit shift leads to an increase in G this allocation is set in place of $\mathsf{P_{worst}}$ (line 12). When the population stops evolving, i.e., $\max(\mathbf{g})$ remains the same between $\mathsf{maxNonImprov}$ subsequent generations, the best achieved bit allocation $\mathbf{A} = \mathsf{P_{best}}$is returned. The convergence speed of hGAm is improved by providing to it only a set of nFeat most important features as given by Recursive Feature Elimination (RFE) [16]. Although not further elaborated here due to space constraints, we note that the metaheuristic hGAm approach provided better results than alternate approaches we also implemented and evaluated. These included convex optimization of feature and discretization level importances, as well as mutual information based bit allocation. In contrast to these and others, the hGAm approach both inherently captures complex feature and discretization level interactions, as well as balances the bias-variance trade-off.

VI. PERFORMANCE EVALUATION

After having presented how discretization as well as feature and discretization level optimization is performed, we now turn to the evaluation of the resulting classifier. To perform the evaluation DIOPT was implemented and scikit-learn [27] version 0.20.0 was used to train the discretized RF reference classifier, $\mathbb{C}_{\mathrm{R}}$. The trained scikit model $\mathbb{C}_{\mathrm{R}}^{\mathrm{scikit}}$ was then exported to our custom-built RF/tree evaluator $\mathbb{C}^{\mathrm{fast}}$ implemented in C with compiler and other optimizations employed as further described in [13]. The $\mathbb{C}_{\mathrm{R}}^{\mathrm{fast}}$classification model was used for the $\mathbb{C}_{\mathrm{R}} \rightarrow \mathbb{C}_{\mathrm{LU}}$ table filling. Timings for DIOPT are then collected on the lookup based model $\mathbb{C}_{\mathrm{LU}}$. For the RF and tree classifiers used for comparisons, training was performed with scikit using undiscretized data, and timings measured on the $\mathbb{C}^{\mathrm{fast}}$implementation using the exported scikit model.

Experiments were performed on a workstation with an i7-6850K and 128Gb RAM running Ubuntu 16.04. To collect the inference runtime values each dataset was timed using 100000 instances obtained by sampling with replacement from the test set, and this was replicated ten times. Missing values were imputed by feature means, and features with only a single value were dropped. The comparison considers baseline RF and RF with feature selection using RFE [16], as well as Decision Trees (DT) and DT-RFE. For the RF variants, using $\{5, 10, 20, 50, 100\}$ trees were evaluated, and for DIOPT memory configuration footprints from 2 MiB to 32 GiB were considered. Ten-fold stratified cross validation is employed for testing. At each cross validation split the training set is used for the bin-boundary generation as well as deducing the RFE feature order. The classifier hyperparameters, including the feature selection, are optimized using 3-fold cross validation in the training set. The hyperparameters for RF are grid search optimized using 'max_depth': [10,20,30] and max_features: [.1, .4, .7, 1]. For the evaluations including a decision tree classifier the settings are 'max_depth': [10,20,30,40,50] and

TABLE II: Error rate with standard deviation and classification time for the classifier configuration providing 'Best error rate', and a 'Manual configuration choice' which selects an RF configuration providing an error rate close to the best DIOPT. DT: Decision Tree, RF: Random Forest, -RFE: With Recursive Feature Elimination, DIOPT: The proposed approach.

				Best error rate							**Manual configuration choice**	
			Class	Error rate(%)			Classif. time (ns)			DIOPT	Error rate (%)	Cl. time (ns)
Data set	Dim	Inst.	fraction	DT-RFE	RF-RFE	DIOPT	DT-RFE	RF-RFE	DIOPT	Config	RF-RFE	RF-RFE
Fried	10	40K	0.50	11.70±0.55	7.47±0.48	7.87±0.42	44	6927	29	32G	7.75±0.50	3247
Magic	11	19K	0.65	15.13±0.97	11.92±1.14	12.83±1.09	39	8323	54	32G	12.47±0.91	1237
EyeState	15	15K	0.45	15.39±0.93	6.72±0.72	9.37±0.53	68	7916	68	32G	9.52±0.58	539
Codrna	8	0.5M	0.33	4.57±0.13	1.48±0.14	2.01±0.17	37	6167	42	32G	1.91±0.12	486
Elevators	18	17K	0.69	14.85±0.92	12.39±0.76	12.78±0.57	47	7840	38	8G	12.65±0.66	3651
Ailerons	40	14K	0.42	14.20±0.79	11.47±0.85	11.57±1.02	56	3773	36	0.5G	11.47±0.85	3773
CellNet	26	3.2M	0.07	1.00±0.08	0.71±0.03	0.85±0.04	74	5096	40	32G	0.82±0.02	363

TABLE III: 'Fried' data: Multiple metrics overview

Metric	DT-RFE	RF-RFE	DIOPT-GA
Error rate (%)	11.70±0.55	7.47±0.48	7.87±0.42
EER (%)	11.67±0.58	7.61±0.57	7.85±0.45
Precision (%)	88.50±0.59	92.83±0.52	92.23±0.55
Recall (%)	87.98±0.69	92.15±0.63	91.98±0.56
F1-score (%)	88.24±0.55	92.48±0.49	92.10±0.42
Logloss	1.16±0.89	0.19±0.01	0.19±0.01
Classification time (ns)	44.47±7.88	6927.87±276.19	29.56±5.34

max_features: [.1, .4, .7, 1]. All timing results are when running on a single core. Public binary data sets from openML [37] and the UCI repositories [6] were examined, as well as one proprietary data set related to classification of traffic inside high-speed computer networks. This proprietary data set comprised of 2.2 billion packets, which after filtering, packet aggregation, and data subsampling were reduced to 3.2 million flow instances for this evaluation. This data set had 26 features, all of which were integers with small value domains, which allowed all features to be processed with the fastest possible non-typeconverting, non-scaling discretization pipeline.

A. Single data set perspective

The first part of the evaluation focuses on the characteristics and results for a single data set, 'Fried', which is listed on the top row of Table II. Table III additionally shows the results of the top performing classifiers in terms of a range of metrics computed as the mean of the ten replications, and also provides a measure of dispersion in the form of the standard deviation of the results from the replications. Most metrics are very close between RF-RFE and DIOPT-GA but somewhat surprisingly RF-RE and DIOPT-GA were equal for logloss, something which was not observed for any of the other data sets.

In the following, the error rate metric will be discussed which simply is 1-accuracy, although the accuracy metric has several noted issues [28]. Yet, accuracy and error rate can be intuitively understood and is in widespread use, and suffices for the current purpose of evaluating the relative performance between RF and DIOPT. To illustrate the dynamics of the different classifiers, Figure 2a shows how the accuracy evolves over the configurations. The lines in the figure follow the progression of configuration values, and thus do not necessarily represent the Pareto boundary of that variant.

It is clear from the graph that the DIOPT approaches have orders of magnitude less classification time, and can come very close in terms of accuracy to the best RF classifier.

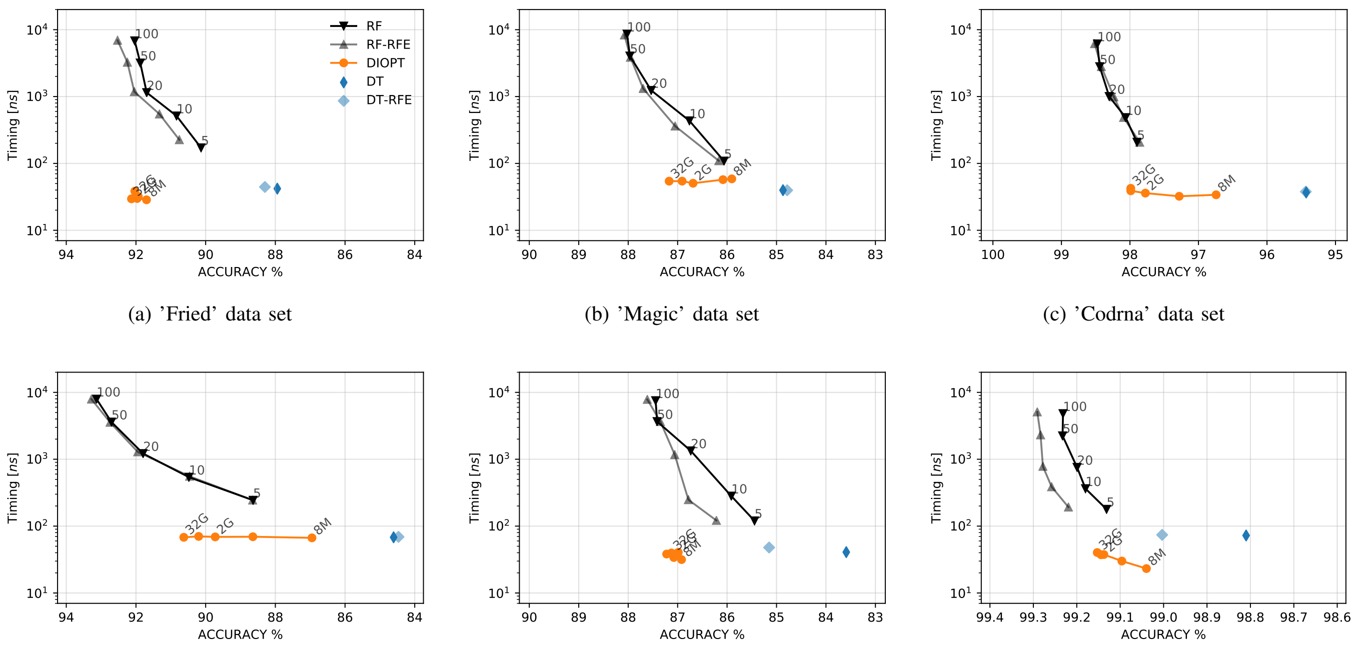
B. Multiple data set perspective

The characteristics of DIOPT were also evaluated over a wider range of data sets, as detailed in Table II. In Table II, the Dim, Inst., and Class Fraction columns denote the number of features, number of instances, and positive class fraction in a given dataset. The table has two major result columns, the leftmost 'best error rate' column reports the best results by any configuration of RF-RFE (over number of trees) and DIOPT (over memory sizes). We note that while the RF-RFE column always reports the lowest mean error rate, the DIOPT mean is within one standard deviation for the majority of data sets. DT-RFE is much behind in terms of error-rate, although it achieves timings that are similar to DIOPT.

From a runtime perspective there is strong correlation between the number of trees in an RF model and the resulting classification time. For DIOPT there is no such correlation between model complexity and runtime. Thus, the two rightmost 'Manual choice' columns report the error rate and runtime for the RF-RFE classifier with the tree number chosen such that the resulting RF-RFE error rate is as close as possible to the error rate of the best DIOPT configuration. As can be observed, RF-RFE classifiers with comparable classification performance to DIOPT still have orders of magnitude larger classification times.

Figure 2 further elaborate on the relationship between runtime and model complexity for a subset of datasets from Table II. It is clearly visible that while the runtime of the RF classifier varies considerably over almost two orders of magnitude when increasing the number of trees from 5 to 100, similar behavior is not observed for the DIOPT runtimes when the memory footprint is increased from 8MB to 32GB. This is expected as the inference runtime of DIOPT is mainly due to the discretization of the selected features, that is, from mapping the feature values of an instance to bin indices and concatenating them.

While the largest DIOPT memory configuration is often the best, in several cases the data set is such that all relevant information can be captured well also with a smaller memory size. Depending on the deployment target, the DIOPT memory configuration with the best error rate might not always be the most appropriate. The selection of the most appropriate classifier configuration for a particular high classification throughput scenario can thus be seen as a multi-objective optimization (MOO) problem involving 1) classification runtime, 2) classification performance in terms of a metric such



(a) 'Fried' data set (b) 'Magic' data set (c) 'Codrna' data set

(d) 'EyeState' data set (e) 'Elevators' data set (f) 'CellNet' data set

Fig. 2: Time per classification versus model accuracy over a range of number of trees (for RF) or memory configurations (for DIOPT). Note the logarithmic scale on the y-axis.

as error rate, and 3) memory requirements. How to weigh these parameters is scenario-dependent, but DIOPT is clearly superior when classification run-time is the major parameter. Table II and Figure 2 shows that there is a between 8 to 112 fold increase in classification time when going from DIOPT to a similarly performing RF classifier (note the logarithmic y-axis scale). For the particular use case of network traffic classification (CellNet), there is considerably more throughput using DIOPT as compared to RF-RFE with only minor degradation in classification performance. The best RF-RFE configuration of 100 trees has 7 misclassifications per 1000 classifications, whereas DIOPT with a 2GiB memory size has 8.6 misclassifications per 1000. To handle the same peak classification load, such an RF-RFE classifier would however need more than one hundred times the number of CPU cores as DIOPT would. Considering the Fried data set, the accuracy is also very close, while DIOPT here provides a classification rate increase at 239x for 'best' and 112x for 'manual' configurations.

Examining the results from the perspective of data set variability, on a high level it can be seen that the RF vs DIOPT relative classification performance varies considerably between the data sets. Notable differences can be observed for two of the data sets, 'Fried' and 'Elevators', where the classification performance of the different DIOPT memory configurations are much more 'compressed' in relation to the RF classification performance. Clearly, these data sets are less sensitive to heavy discretization, and the inherent class separability can be captured almost fully also for small lookup tables.

The DIOPT classification performance can also be contrasted to results reported in the literature reviewed in Section 2. In particular, we consider the DT-related works of [2], [31], [38], and the RF simplification approach of [17]. We note that the comparison is possibly handicapped by differences in hyper-parameter optimization approaches, evaluation protocol, etc. and that results for larger scale data sets typically are lacking. This comparison is thus performed based on eight smaller, public, openML/UCI data sets for which all the cited works provide results. The data sets consider binary classification tasks and have a dimensionality between 14 and 72, and and instance count between 351 and 48842. Overall, we for these data sets in general observe results in line with Table II, i.e. DIOPT achieves error rates close to RF while having inference runtimes similar to DT. Compared to the error rates reported in literature, DIOPT shows better performance than the alternate approaches for all data sets with one exception (credit-approval).

In summary, the results show the ability of the DIOPT approach to considerably increase classification throughput with minor impact on classification accuracy for many large scale data sets. While there are numerous classification problems that are not suited for DIOPT, a considerable fraction of practically relevant problems can benefit from DIOPT and achieve very high classification rates.

VII. CONCLUSIONS

In areas such as high-energy physics, management of communication networks, and other areas there is a need for very high-throughput classification. This work contributes the novel DIOPT approach where the new KSD-MI supervised discretization algorithm, as well as the hGAm feature selection and discretization optimization algorithm is employed. With these building blocks it is possible to create a DIOPT lookup-based classifier capable of achieving very fast classification times for problems of moderate dimensionality. Employing efficient run-time discretization implementations, we demonstrated that DIOPT can achieve orders of magnitude improvements in classification throughput while showing insignificant or modest classifications performance reductions as compared to Random Forest over a range of data sets. While DIOPT is not suitable for all classification problems, we have established that DIOPT can provide considerable improvements in classification throughput for appropriate problem domains, many of which also appear in practice.

ACKNOWLEDGMENTS

The authors wish to thank Sandvine for assisting with feature suggestions and data collection for the CellNet data set. Computations were performed on resources at Chalmers Centre for Computational Science and Engineering (C3SE) and High Performance Computing Center North (HPC2N) provided by the Swedish National Infrastructure for Computing. Funding for this study was provided by the HITS project grant from the Swedish Knowledge Foundation.

REFERENCES

[1] Mario Barbareschi, Salvatore Del Prete, Francesco Gargiulo, Antonino Mazzeo, and Carlo Sansone, 'Decision tree-based multiple classifier systems: An fpga perspective', in *International Workshop on Multiple Classifier Systems*, pp. 194–205. Springer, (2015).

[2] Dimitris Bertsimas and Jack Dunn, 'Optimal classification trees', *Machine Learning*, **106**(7), 1039–1082, (2017).

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

[4] Miguel A Carreira-Perpinán and Pooya Tavallali, 'Alternating optimization of decision trees, with application to learning sparse oblique trees', in *Adv. in Neural Information Processing Systems*, pp. 1211–1221, (2018).

[5] Corinna Cortes and Vladimir Vapnik, 'Support-vector networks', *Machine learning*, **20**(3), 273–297, (1995).

[6] Dheeru Dua and Efi Karra Taniskidou. UCI ML repository, 2017.

[7] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim, 'Do we need hundreds of classifiers to solve real world classification problems?', *The Journal of Machine Learning Research*, **15**(1), 3133–3181, (2014).

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

[9] Nicholas Frosst and Geoffrey Hinton, 'Distilling a neural network into a soft decision tree', *arXiv preprint arXiv:1711.09784*, (2017).

[10] Johan Garcia, 'A surrogate-assisted GA enabling high-throughput ML by optimal feature and discretization selection', in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO 20. ACM, (2020).

[11] Johan Garcia and Topi Korhonen, 'Efficient distribution-derived features for high-speed encrypted flow classification', in *Proceedings of the SIGCOMM 2018 Workshop on Network Meets AI & ML*, NetAI'18, pp. 21–27. ACM, (2018).

[12] Johan Garcia and Topi Korhonen, 'On runtime and classification performance of the discretize-optimize (DISCO) classification approach', *SIGMETRICS Perform. Eval. Rev.*, **46**(3), 167–170, (January 2019).

[13] Johan Garcia, Topi Korhonen, Ricky Andersson, and Filip Västlund, 'Towards video flow classification at a million encrypted flows per second', in *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pp. 358–365. IEEE, (2018).

[14] Salvador Garcia, Julian Luengo, José Antonio Sáez, Victoria Lopez, and Francisco Herrera, 'A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning', *IEEE Transactions on Knowledge and Data Engineering*, **25**(4), 734–750, (2013).

[15] Isabelle Guyon and André Elisseeff, 'An introduction to variable and feature selection', *J. Mach. Learn. Res.*, **3**, 1157–1182, (March 2003).

[16] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik, 'Gene selection for cancer classification using support vector machines', *Machine learning*, **46**(1-3), 389–422, (2002).

[17] Satoshi Hara and Kohei Hayashi, 'Making tree ensembles interpretable: A bayesian model selection approach', in *International Conference on Artificial Intelligence and Statistics*, pp. 77–85, (2018).

[18] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean, 'Distilling the knowledge in a neural network', *arXiv preprint arXiv:1503.02531*, (2015).

[19] Thomas Keck, 'Fastbdt: a speed-optimized multivariate classification algorithm for the belle ii experiment', *Computing and Software for Big Science*, **1**(1), 2, (2017).

[20] Andrey Kolmogorov, 'Sulla determinazione empirica di una legge di distribuzione', *Inst. Ital. Attuari, Giorn.*, **4**, 83–91, (1933).

[21] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu, 'Feature selection: A data perspective', *ACM Computing Surveys (CSUR)*, **50**(6), 94, (2018).

[22] H. Liu, F. Hussain, C.L. Tan, and M. Dash, 'Discretization: An enabling technique', *Data Mining and Knowl. Disc.*, **6**(4), 393–423, (2002).

[23] Frank J Massey Jr, 'The kolmogorov-smirnov test for goodness of fit', *Journal of the American statistical Association*, **46**(253), 68–78, (1951).

[24] Feng Nan, Joseph Wang, and Venkatesh Saligrama, 'Pruning random forests for prediction on a budget', in *Advances in neural information processing systems*, pp. 2334–2342, (2016).

[25] Muhsen Owaida, Amit Kulkarni, and Gustavo Alonso, 'Distributed inference over decision tree ensembles on clusters of fpgas', *ACM Trans. Reconfigurable Technol. Syst.*, **12**(4), 17:1–17:27, (September 2019).

[26] Muhsen Owaida, Hantian Zhang, Ce Zhang, and Gustavo Alonso, 'Scalable inference of decision tree ensembles: Flexible design for cpu-fpga platforms', in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–8. IEEE, (2017).

[27] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al., 'Scikit-learn: Machine learning in python', *Journal of Machine Learning Research*, **12**(Oct), 2825–2830, (2011).

[28] Foster J Provost, Tom Fawcett, Ron Kohavi, et al., 'The case against accuracy estimation for comparing induction algorithms.', in *ICML*, volume 98, pp. 445–453, (1998).

[29] Yun R Qu and Viktor K Prasanna, 'Scalable and dynamically updatable lookup engine for decision-trees on fpga', in *2014 IEEE High Performance Extreme Computing Conference*, pp. 1–6. IEEE, (2014).

[30] J. Ross Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[31] Salvatore Ruggieri, 'Enumerating distinct decision trees', in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2960–2968. JMLR. org, (2017).

[32] Pedro M Santiago del Rio, Dario Rossi, Francesco Gringoli, Lorenzo Nava, Luca Salgarelli, and Javier Aracil, 'Wire-speed statistical classification of network traffic on commodity hardware', in *Proceedings of the 2012 Internet Measurement Conference*, pp. 65–72. ACM, (2012).

[33] Claude E Shannon and Warren Weaver, 'The mathematical theory of communication, 117 pp', *Urbana: University of Illinois Press*, (1949).

[34] Dan Simon, *Evolutionary optimization algorithms*, John Wiley & Sons, 2013.

[35] Kirill Trapeznikov and Venkatesh Saligrama, 'Supervised sequential classification under budget constraints', in *Artificial Intelligence and Statistics*, pp. 581–589, (2013).

[36] Brian Van Essen, Chris Macaraeg, Maya Gokhale, and Ryan Prenger, 'Accelerating a random forest classifier: Multi-core, gp-gpu, or fpga?', in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pp. 232–239. IEEE, (2012).

[37] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo, 'Openml: Networked science in machine learning', *SIGKDD Explorations*, **15**(2), 49–60, (2013).

[38] Sicco Verwer and Yingqian Zhang, 'Learning optimal classification trees using a binary linear program formulation', in *33rd AAAI Conference on Artificial Intelligence*, (2019).

[39] Xin Xia, Tao Lin, and Zhi Chen, 'Maximum relevancy maximum complementary based ordered aggregation for ensemble pruning', *Applied Intelligence*, 1–12, (2018).

[40] Zhixiang Xu, Matt Kusner, Kilian Weinberger, and Minmin Chen, 'Cost-sensitive tree of classifiers', in *International Conference on Machine Learning*, pp. 133–141, (2013).

[41] Fan Yang, Wei-hang Lu, Lin-kai Luo, and Tao Li, 'Margin optimization based pruning for random forest', *Neurocomputing*, **94**, 54–63, (2012).