STOCHASTIC INDUCTION OF DECISION TREES WITH APPLICATION TO LEARNING HAAR TREE

Anonymous authors
Paper under double-blind review

ABSTRACT

Decision trees are a convenient and established approach for any supervised learning task. Decision trees are used in a broad range of applications from medical imaging to computer vision. Decision trees are trained by greedily splitting the leaf nodes into a split and two leaf nodes until a certain stopping criterion is reached. The procedure of splitting a node consists of finding the best feature and threshold that minimizes a criterion. The criterion minimization problem is solved through an exhaustive search algorithm. However, this exhaustive search algorithm is very expensive, especially, if the number of samples and features are high. In this paper, we propose a novel stochastic approach for the criterion minimization. Asymptotically, the proposed algorithm is faster than conventional exhaustive search by several orders of magnitude. It is further shown that the proposed approach minimizes an upper bound for the criterion. Experimentally, the algorithm is compared with several other related state-of-the-art decision tree learning methods, including the baseline non-stochastic approach. The proposed algorithm outperforms every other decision tree learning (including online and fast) approaches and performs as well as the baseline algorithm in terms of accuracy and computational cost, despite being non-deterministic. For empirical evaluation, we apply the proposed algorithm to learn a Haar tree over MNIST dataset that consists of over $200,000$ features and $60,000$ samples. This tree achieved a test accuracy of 94% over MNIST which is 4% higher than any other known axis-aligned tree. This result is comparable to the performance of oblique trees, while providing a significant speed-up at both inference and training times.

1 INTRODUCTION

Decision trees are among the most established machine learning models in the scientific community and have been widely used in various applications, such as computer vision and medical imaging Criminisi & Shotton (2013); Freund et al. (1999); Breiman (2001); Tavallali et al. (2019). Decision trees are extensively used as a sub-procedure in ensemble models, such as boosting and forest Criminisi & Shotton (2013). Decision tree training consists of recursively splitting a leaf node into one split and two leaf nodes Breiman et al. (1984); Gehrke et al. (1999). The split node consists of a decision stump that routs the samples to either of its left or right children. The splitting of a leaf node is cast as selecting the best feature and threshold that minimizes a desired criterion. The criterion is an impurity function, such as Gini index Breiman et al. (1984), cross-entropy Quinlan (1986), uniform piece-wise constant densities Ram & Gray (2011), Gaussian differential (continuous) entropy Criminisi & Shotton (2013) and others Yang & Wong (2014); Liu & Wong (2014); Sheikhi & Babamir (2018; 2016). The best feature and threshold are selected through an incremental algorithm that evaluates combination of every possible feature and threshold. However, this algorithm becomes computationally very expensive if the dataset is large or high dimensional. The computational complexity of solving criterion minimization problem is $\mathcal{O}(DNlogN)$, where N and D are number of samples and dimensionality, respectively. The computational complexity becomes very high especially when N and D are high since N and D are multiplied in the computational complexity term ($\mathcal{O}(DNlogN)$). In this paper, we tackle the abovementioned issue by applying the splitting algorithm in a novel iterative and stochastic fashion, where less important features are discarded at each iteration.

The literature includes many proposals for speeding up the training procedures of various learning algorithms. One common approach is preprocessing the dataset by reducing the sample size or applying some feature selection technique Dash & Liu (1997); Jović et al. (2015); John et al. (1994). Pre-training feature selection will potentially lead to a less computationally expensive process for learning Jović et al. (2015); Kohavi & Sommerfield (1995); Koller & Sahami (1996); Korn et al. (2001). Feature selection algorithms are generally categorized in two groups of filter and wrapper approaches.

The filter approaches consist of selecting features with regard to a desired criterion Jović et al. (2015). Generally, filter approaches use statistical measures, such as minimum Redundancy maximum Relevance-(mRmR) Tang et al. (2014), correlation Yu & Liu (2003), chi-square Moh'd A Mesleh (2007), information gain Bhattacharyya & Kalita (2013), gain ratio Witten & Frank (2002), etc.

The wrapper approaches consist of selecting features and minimizing the loss function jointly Bhattacharyya & Kalita (2013); Bradley & Mangasarian (1998); Maldonado et al. (2014); Hastie et al. (2009). However, feature selection as a preprocessing method might not lead to an accurate final model since features are not guaranteed to contribute to lower loss Jović et al. (2015). Nevertheless, wrappers are practical only for fast modeling algorithms or greedy search strategies, such as linear SVM Liu et al. (2014), Nave Bayes Cortizo & Giraldez (2006), and Extreme Learning Machines Benoît et al. (2013).

In Embedded approaches, first a filter approach is applied to select a subset of features and then followed by a wrapper approach to select the best candidate feature Guyon & Elisseeff (2003); Das (2001). Different decision tree algorithms, such as CART Breiman et al. (1984), C4.5 Quinlan (2014), and random forest are commonly used as embedded methods Sandri & Zuccolotto (2006).

The feature selection as a preprocessing approach decreases the computational complexity of training algorithm. However, it is suboptimal w.r.t. the objective function of training, and is inefficient for large datasets since it does not decrease computation over number of samples Rodriguez-Lujan et al. (2010).

A well-known approach that tackles the issue of large datasets is online learning Utgoff (1989). Various studies consider applying online learning to decision trees Utgoff (1989); Schlimmer & Fisher (1986); Wang et al. (2003). Such algorithms are fast, simple, and often make few statistical assumptions. The objective of online learning is to update and train the model from a stream of data Shalev-Shwartz et al. (2011). At each time step, the model gets updated optimally with respect to the recently acquired samples. Online learning is more computationally efficient compared to offline learning where the entire trainset is used for training Shalev-Shwartz et al. (2011). However, these algorithms increase the size of the tree when visiting a new batch of the data.

Despite the extensive usages and applications of decision trees Criminisi & Shotton (2013), few algorithms have focused on decreasing the high computational cost of inducing a tree **??**. However, the literature leaves a gap in treating the high computational complexity of inducing the decision tree and exploring potential improvements. None of the previous research propose the computational complexity of their algorithm or provide any analysis of the goodness of features selected at the split node. To address this gap, we investigated both computational complexity of SDT and its efficiency in finding near optimum feature at the split node. In **?**, authors propose to randomly select a subset of features at the split node and then find the best feature among the set using all samples present at the node. However, due to randomness, the optimality of this process is not guaranteed.

The high computational cost is mainly rooted in the exhaustive search algorithm applied for solving the split criterion. This algorithm specially becomes computationally expensive if the dataset is large and high dimensional. It is because the computational complexity of solving the splitting criterion is $\mathcal{O}(DNlogN)$ Hoare (1961); Breiman et al. (1984).

To counter the aforementioned issues, we propose an accurate and fast Stochastic Decision Tree (SDT) induction algorithm that minimizes the splitting criterion efficiently. At a node, the proposed algorithm starts with an empty set S_t and all D features. At node j, the algorithm iteratively selects a tiny random set of samples from S_j (in order of $2^{-C} \times |S_j|$) and discards half of the less important features w.r.t S_t. S_j and $|S_j|$ are the set and number of samples present at the j^{th} node,

respectively. For the best feature and threshold found at each iteration, SDT minimizes an upper bound of the splitting criterion, monotonically. Overall, the algorithm steeply descends towards the minimum of the criterion function. It is also demonstrated by the results at the experiments section. Mathematically speaking, we show that the algorithm provides higher priority to more distinctive features. Essentially, the more distinctive a feature, the greater the chance of it being selected at the final iteration. The computational complexity of the algorithm for constructing a tree of depth Δ is $\mathcal{O}(\frac{\Delta DNlog\frac{N}{2^C}}{2^C})$. C is a hyperparameter and in our experiments we set it to 10. The computational complexity achieved by SDT is smaller than that of the original approach (with $\mathcal{O}(\Delta DNlogN)$) and its optimal implementation ($\mathcal{O}(\Delta DN + DNlogN)$) by several orders of magnitude. Our approach is faster because it focuses on breaking down the computational complexity created due to the large number of samples and features. Numerically speaking, SDT shows its advantage over other implementations when the sizes of samples and features are high (ND is very high). Experimentally, the proposed algorithm could achieve same or improved accuracy as the original greedy algorithm Breiman et al. (1984) while surpassing every other related state-of-the-art algorithms both in terms of accuracy and complexity by several orders of magnitude. For empirical evaluation, we applied our algorithm to learn a Haar tree over MNIST where Haar Viola & Jones (2001) filters were used to extract $200,000$ features from the dataset. The Haar tree could achieve competitive accuracy to more complicated trees, such as oblique trees and optimal oblique trees while its inference time was far smaller. Theoretically, an oblique tree needs $\mathcal{O}(D\Delta)$ operations to produce prediction, while Haar tree only needs $\mathcal{O}(D + \Delta)$. Δ is the maximum depth of the tree. In terms of size, Haar tree takes $\mathcal{O}(2^\Delta)$ space while oblique tree takes $\mathcal{O}(D2^\Delta)$ space.

2 PRELIMINARIES

2.1 INDUCTION OF A DECISION TREE

Every decision tree construction algorithm tends to minimize the following objective function

$$L_\alpha(T) = L(T) + \alpha \times leaves(T), \qquad \alpha > 0, \tag{1}$$

where, $L(T)$ is a loss function of the tree (T) over a trainset, $leaves(.)$ represents the number of leaves in a tree, and $\alpha > 0$ is cost complexity coefficient. After constructing the tree, pruning is performed to minimize the cost function $L_\alpha(T)$ over the validationset. Finding the optimum of this problem is NP-complete **?**. Therefore, state-of-the-art algorithms for constructing a tree consist of greedily splitting a leaf node into a split and two leaf nodes.

Assume a dataset S consisting N pairs of observation $\{(x_i, y_i)\}_{i=1}^N$ where $x_i \in \mathbb{R}^D$ and $y_i \in \{1, 2, 3, ..., M\}$. The function $f_j(x)$ at the j^{th} split node consists of a decision stump $f_j(x) = sign(x^p - th)$ where superscript p and th represent p^{th} feature and the threshold, respectively. At the growing phase of a decision tree, p and th are found optimally with respect to some criteria Breiman et al. (1984). Usually the tree is grown until some stopping criterion or maximum depth is reached.

The problem of constructing a split node consist of minimizing the desired criterion over the samples of each child of the node.

$$\begin{aligned} \min_{f_j} \quad & \sum_{f_j=\{-1,1\}} \frac{|S_j^{f_j}|}{|S_j|} H(S_j^{f_j}) \\ s.t \quad & f_j(x) = sign(x^p - th) \end{aligned} \tag{2}$$

where, $S_j^{f_j}$ is set of samples at the left child ($f_j = -1$) or right child ($f_j = 1$) of a node j, S_j is the set of samples present at the j^{th} node, and $|.|$ returns the number of samples at its input set. H is an entropy function. The optimum of problem equation 2 can be found in $\mathcal{O}(DN_jlogN_j)$ where N_j and D are number of samples at node j ($|S_j|$) and features, respectively. This is done by sorting all samples along each feature and incrementally finding the best threshold for each feature Breiman et al. (1984).

3 PROPOSED METHOD

The computational complexity of solving problem equation 2 is high, especially, if the number of features and samples are large. In this section, we propose an algorithm that minimizes problem equation 2 faster than original approach Breiman et al. (1984) by several orders of magnitude.

Basic idea: Problem equation 2 consists of selecting the best feature to partition the space into two parts optimally. State-of-the-art algorithms find the best feature through the exhaustive search mentioned in Section 2.1. In this section, we propose a stochastic exhaustive search method for solving problem equation 2 that has a low computational cost. The idea is based on 3 pieces of information. **Info. #1** Each feature can be approximately ranked based on the objective function of equation 2 over a small random subset of the samples. **Info. #2** A precise ranking of the features requires larger subset of the samples. **Info. #3** The good news is that only ranking of the most competitive features require high precision. As a result, by combining Info #1-3, the algorithm consists of discarding less important features and randomly selecting more samples from the dataset at each iteration. This procedure gradually increases the precision of ranking for more important features. We call this algorithm the stochastic induction of decision tree (SDT).

The algorithm Now we describe the proposed algorithm for solving criterion problem equation 2. The algorithm starts with $S_{j^c} = S_j$, an empty set S_{j^t}, and set of features $D^t = \{p\}_{p=1}^{D}$ features. p represents the p^{th} feature. The algorithm proceeds as follows:

Step 1 Randomly select and remove $\frac{N_j}{2^C}$ samples from S_{j^c} and add them to set S_{j^t}. C is a user defined hyper-parameter.

Step 2 $\forall p \in D^t$, at k^{th} iteration, find feature importance of p^{th} feature (FI_p^k) by solving

$$
\begin{aligned}
(FI_p^k)^{-1} = \min_{th} \quad & \sum_{f_j=\{-1,1\}} \frac{|S_{j^t}^{f_j}|}{|S_{j^t}|} H(S_{j^t}^{f_j}) \\
s.t \quad & f_j(x) = sign(x^p - th)
\end{aligned}
\tag{3}
$$

where, $S_{j^t}^{f_j}$ represents the set of samples from S_{j^t} routed to the left child ($f_j = -1$) or the right child $f_j = 1$. FI_p^k represents the feature importance of p^{th} feature at k^{th} iteration. Note that the optimization is only happening over th in equation 3 and $(FI_p^k)^{-1}$ is equal to the optimum of objective function in equation 3. Problem in equation 3 can be efficiently solved in $\mathcal{O}(|S_{j^t}|log(|S_{j^t}|))$ Breiman et al. (1984).

Step 3 Sort all the features in D^t based on FI_p^k and discard half of the features in D^t with lowest FI.

Step 4 Iterate over steps 1-3 for α times. α is a user-defined hyper-parameter.

Step 5 Solve problem equation 2 for only the remaining features in D^t over the S_j.

In practice, we select C to be around 10 and α to be around $7 - 10$, depending on the dataset. Intuitively, C and α are the hyper-parameters that control the computational complexity of the algorithm. C controls the number of samples to be used at each iteration and α controls the number of features to be used at the last iteration. With higher values for C and α, the computational complexity decreases further. However, there is a possibility that the model loses accuracy with extremely high C and α .

To keep the computational complexity low, one can keep the samples in S_{j^t} sorted for the remaining features (D^t) from the previous iterations at Step 2. Only the newly selected samples from Step 1 have to be sorted and merged with S_{j^t}. The merging will take at most $|S_{j^t}|$. In total, this will keep the computational complexity of step 2 at $\mathcal{O}(2D\frac{N_j}{2^C}log(\frac{N_j}{2^C}))$ for all the α steps (assuming $\alpha \to \infty$).

4 ANALYSIS OF THE PROPOSED ALGORITHM

In this section, several theoretical properties of the algorithm are explored and analyzed.

4.1 COMPUTATIONAL COMPLEXITY

The computational complexity of splitting a node and induction of a tree are analyzed in this Subsection.

In the following theorem, we assume that the samples of S_{j^t} from previous iterations are **NOT** kept sorted. After the theorem, we will show how keeping samples sorted can affect the node training complexity and storage overhead.

Theorem 1 (node training complexity)**.** *The computational complexity of splitting the node using SDT is* $\mathcal{O}(4DN_0logN_0)$*, where* $N_0 = \frac{N_j}{2^C}$.

Proof. At each iteration, N_0 samples are added to the set S_{j^t} and half of the features are discarded. To solve equation 3, samples along each remaining feature have to be sorted that takes $\mathcal{O}(|S_{j^t}|logS_{j^t})$. At the k^{th} iteration, $|S_{j^t}|$ is $(k+1)N_0$. Therefore, the total complexity of steps 1-4 is as follows:

$$\sum_{k=0}^{\alpha} \frac{D}{2^k}(k+1)N_0 log(k+1)N_0 \tag{4}$$

The summation in equation 4 can further be simplified to:

$$\sum_{k=0}^{\alpha} \frac{D}{2^k}(k+1)N_0(logN_0 + log(k+1)) \leq DN_0(logN_0 + log(\alpha+1)) \sum_{k=0}^{\alpha} \frac{k+1}{2^k} \tag{5}$$

Using the formula for the geometrical series of $\sum_{k=0}^{\infty} \frac{k+1}{2^k} = \sum_{k=0}^{\infty} \frac{1}{2^k} + \sum_{k=1}^{\infty} \frac{1}{2^k} + \sum_{k=2}^{\infty} \frac{1}{2^k} + ... = 2 + 1 + \frac{1}{2} + \frac{1}{4} + ... = 4$ in the right hand side of inequality of equation 5, we have

$$DN_0(logN_0 + log(\alpha+1)) \sum_{k=0}^{\alpha} \frac{k+1}{2^k} \leq 4DN_0(logN_0 + log(\alpha+1)) \tag{6}$$

Asymptotically speaking, the computational complexity is being driven by the term $4DN_0logN_0$. Note that $log(\alpha+1)$ is relatively very small compared to $logN_0$ (Numerically speaking, N_0 is in order of $100 - 200$ while α is in order of $7 - 10$). Therefore, the computational complexity asymptotically becomes $\mathcal{O}(4DN_0logN_0)$. □

The computational complexity of the baseline algorithm for solving equation 2 is $\mathcal{O}(DN_jlogN_j)$, which is 2^C times larger than the proposed stochastic approach.

Effect of keeping the S_{j^t} sorted Effect of keeping the S_{j^t} sorted is that $k+1$ is removed from the formula equation 4. Therefore, the computational complexity changes to $\mathcal{O}(2DN_0logN_0)$. However, it will increase the space usage slightly ($(k+1)N_0$ new variables need to be saved to keep the label of samples along each feature).

Assuming a maximum depth of Δ for the tree, the computational complexity of inducing the whole tree will be $\mathcal{O}(\frac{\Delta 4DNlog\frac{N}{2^C}}{2^C})$. It is trivial to prove by simply using the fact that the children of each node receive a disjoint set of samples from its parent and the fact that total samples at the same depth are at most N. This complexity is smaller than complexity of the baseline approach Breiman et al. (1984) by an order of 2^C.

4.2 RELATION TO OBJECTIVE FUNCTION

In this section, the value of the objective function in equation 2 is evaluated using the best combination of feature and threshold found by the SDT at each iteration over S_{j^t}. For simplicity, throughout this section, the objective function of equation 2 is represented by $L(S_j)$. Consequently, $L(S_{j^t})$ represents error over the set S_{j^t}. In this section, we assume the used criterion is misclassification error $H(S_j^{f_j}) = 1 - p_{m^*}^{f_j}$, where $p_m^{f_j}$ and m^* represent ratio of class m in the f_j child and majority class in the partition, respectively. All the theorems and analysis can be readily extended to Gini-index and Cross-entropy Breiman et al. (1984) by applying their corresponding differences. The differences and how to apply them are explained after next theorem.

Table 1: The used datasets Information

Index	Name	Size		Description
		Train	Test	
1	MNIST	60000 × 784	10000 × 784	A large database of handwritten digits 0 to 9 that is serving as a basis for classification algorithms
2	FASHIOMNIST	60000 × 784	10000 × 784	A dataset of Zalando's article images contaianes 10 different classes of gray images of T-shirt/top ,Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot
3	SIGNMNIST	27455 × 784	7172 × 784	The American Sign Language letter database of hand gestures represent a multi-class problem with 24 classes of letters (excluding J and Z which require motion)
4	ISOLET	5238 × 617	1559 × 617	ISOLET (Isolated Letter Speech Recognition) is the name of each letter of the alphabet twice that is spoken by 52 speakers and a good domain for a noisy, perceptual task
5	SATIMAGE	4825 × 36	1610 × 36	The database consists of the multi-spectral values of pixels in 3×3 neighbourhoods in a satellite image, and the classification associated with the central pixel in each neighbourhood
6	COVERTYPE	435759 × 54	145253 × 54	This dataset contains tree observations from four areas of the Roosevelt National Forest in Colorado. All observations are cartographic variables

Theorem 2 (Monotonic decrease of an upper bound)**.** *The algorithm will monotonically decrease an upper bound over the objective function of equation 2.*

Proof. At each iteration k, the following equality holds:

$$|S_j|L^k(S_j) = |S^k_{j^t}|L^k(S^k_{j^t}) + |S^k_{j^c}|L^k(S^k_{j^c}) \tag{7}$$

where, the superscript k for sets and $L^k(.)$ represents the set of samples at the k^{th} iteration and the value of error with optimum parameters for $S^k_{j^t}$ over the input set, respectively. The loss $L(.)$ is always bounded from above (e.g., for misclassification, it is 1) by a constant value L_{max}. By replacing $L^k(S^k_{j^c})$ with L_{max} in equation 7, for iteration k we have

$$|S_j|L^k(S_j) \leq |S^k_{j^t}|L^k(S^k_{j^t}) + |S^k_{j^c}|L_{max} = |S_j|L^k_{upper}(S_j) \tag{8}$$

At each iteration, the first term in equation 8 is minimized. Therefore, the new optimum parameters at each iteration provide lower error to the first term of equation 8 than the optimum parameters of previous iteration which is $|S^{k+1}_{j^t}|L^k(S^{k+1}_{j^t}) \geq |S^{k+1}_{j^t}|L^{k+1}(S^{k+1}_{j^t})$. Note that the superscript of the loss L is different for both sides of the inequality. Also, $|S^{k+1}_{j^c}| < |S^k_{j^c}|$. Therefore,

$$\frac{|S^{k+1}_{j^t}|L^{k+1}(S^{k+1}_{j^t}) + |S^{k+1}_{j^c}|L_{max}}{|S_j|} = L^{k+1}_{upper}(S_j) \leq L^k_{upper}(S_j) \tag{9}$$

□

For the case of Gini-index or Cross-entropy, formula equation 7 should be replaced with $L^k(S_j) \leq L^k(S^k_{j^t}) + L^k(S^k_{j^c})$. The reason for this change is that the algorithm finds optimum over $S^k_{j^t}$ and $min(L^k(S^k_{j^t}), L^k(S^k_{j^c})) \leq L^k(S_j) \leq max(L^k(S^k_{j^t}), L^k(S^k_{j^c}))$.

Average Decrease The Theorem equation 2 is based on worst case scenario. The proof of Theorem equation 2 assumes that the samples selected at each iteration do not have any correlation with the rest of samples. In reality, this is not necessarily correct and set of S_{j^t} can partially represent the whole trainset S_j and as size of S_{j^t} increases through iterations, the FI_p is measured more correctly. In our experiments, we observed a very steep decrease in $L^k(S_{j^t})$ after each iteration and after several iterations, it almost converged to the loss of the best possible feature and threshold. More discussion and experiments are presented in supplemental materials.

Theorem 3 (Probability of discarding a feature)**.** *Assume a binary classification dataset where samples are generated from some distribution along each feature and features are independent. We assume the samples of both classes have varying degrees of overlap along each feature. Probability of a sample being generated from the overlap area is P^p_o for p^{th} feature. The probability of discarding a feature depends on P^p_o.*

Proof. Picking N_0 samples randomly from an infinitely large N_j is equivalent to directly sampling from the distribution itself. Selecting samples is the same as flipping a coin with probability of P^p_o to be 1 and $1 - P^p_o$ to be 0. Expectation of observing 1 is P^p_o and essentially, on average, we expect to observe $P^p_o \times N_0$ samples from the overlap area. Probability of deviating from average case and observing FI^k_p being worse than the true FI_p (please note that FI corresponds to goodness of a

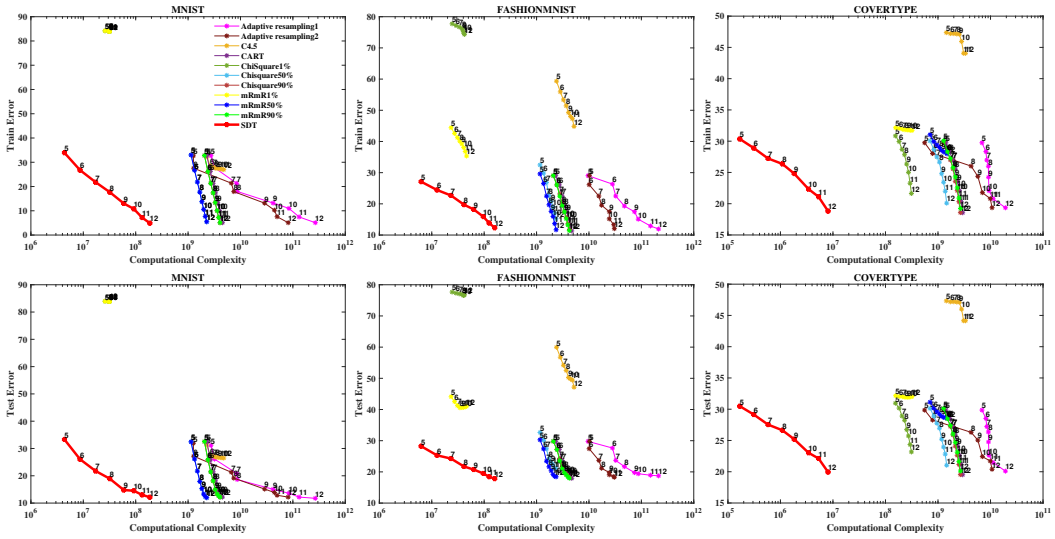

Figure 1: Error ratio versus computational complexity for various models.

feature) over the whole trainset is equivalent to probability of observing **at least** 1 extra (observing $P_0^p \times N_0 + 1$ from overlap area) sample from the overlap area. Therefore, $P(FI_p^k < FI_p) = 1 - (1 - P_o^k)^{N_0(1-P_o^k)}$ (or $1-$ probability of observing no extra sample from overlap area) which is an upper bound to probability of discarding a feature. □

The probability of overlap area P_o^p is inversely related to FI_p^k since samples with lower overlap area (lower P_o^p) are more distinctive; thus, such features have higher FI_p^k. The Theorem 3 implies that the features with higher true FI_p are more likely to stay in the iterations on average. For example, assume a feature that completely separates samples of both classes ($P_o^p = 0$) – any random set of samples will represent this distinction because of having no overlap. Such a feature will achieve $FI_p^k = \infty$, hence, the feature will remain in the D^t.

Theorem 4 (Consistency of SDT)**.** *The SDT is consistent and can learn the function that generated the classes.*

Theorem 5 (Decrease of the cost)**.** *The objective function of equation 2 decreases as new nodes are added.*

The proofs are provided in the supplemental materials. It is worth mentioning that both the theorems of 4 and 5 are true for original decision trees (such as CART) under some mild assumptions and here we show that SDT follows them too.

5 Experimental Results

In this section, we present results of experiments to show the merits of our proposed method. The SDT is compared with several other state-of-the-art algorithms and the baseline tree induction. Figures 1 and 2 present error ratio of train and test versus the computational complexity of each studied algorithm. The existing algorithms selected in the study consist of adaptive re-sampling Iyengar et al. (2000), using feature selection as pre-training procedureTang et al. (2014); Moh'd A Mesleh (2007), C4.5 Quinlan (2014), and CART Breiman et al. (1984). The adaptive re-sampling consists of iteratively training a tree from the scratch with respect to a tiny set of samples and expanding the set with wrongly classified samples by the tree at each iteration. We applied the adaptive re-sampling in two different setups. In one setup, we applied the adaptive re-sampling of Iyengar et al. (2000) until the algorithm achieves better accuracy or similar to that of SDT (Adaptive re-sampling 2). In the other setup, we followed the same steps as provided by Iyengar et al. (2000) (Adaptive re-sampling 2). For feature selection as a pre-training setup, we used ChiSquare and mRmR Tang et al. (2014); Moh'd A Mesleh (2007). The pre-training feature selections were applied to select $1\%, 50\%$, and

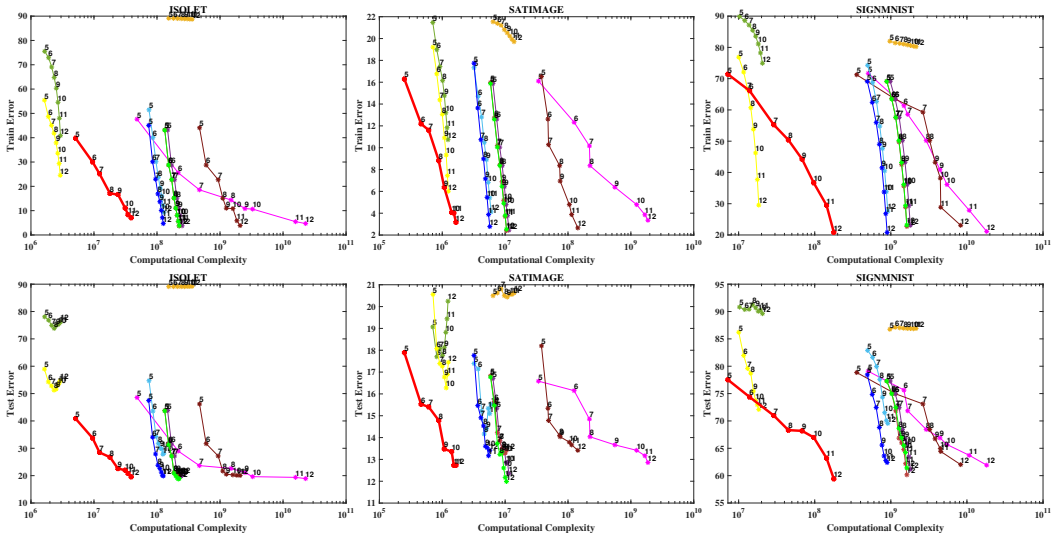

Figure 2: Error ratio versus computational complexity for various models.

90% of features. The C4.5 was trained using the same setup as Quinlan (2014). The decision tree curve represents the CART algorithm Breiman et al. (1984). All the trees were trained for depths of 5-12. In figures 1 and 2, the number at each point on the curve represents the depth of the tree. The vertical axis shows the error ratio and horizontal axis shows the computational complexity of the trained model. For SDT, we set $N_0 = 20$ for all experiments. The SDT iterated until the number of features remained in D^t are around 0.5% of the total features. Description of datasets are presented in table 1.

Figures 1 and 2 depict that SDT outperformed other fast tree training algorithms in both accuracy and computational complexity. The training complexity of SDT is smaller by several orders of magnitude (10^{2-3}). It is worth mentioning that although SDT uses less computation, its accuracy is the same as CART algorithm. Therefore, SDT has served its purpose which was decreasing the complexity of training while maintaining the accuracy.

5.1 HAAR TREE

Haar-like features are well-known in the literature of image processing and have been extensively used for face detection Viola & Jones (2001); Jones & Viola (2003); Demirkir & Sankur (2004); Mita et al. (2005); Pham & Cham (2007), object detection Lienhart & Maydt (2002); Park & Hwang (2014); Chen et al. (2007); Choi (2012) and ensembles Moghimi et al. (2018); Kwak et al. (2013); Larios et al. (2010). The Haar-like features are extracted by convolving various sizes of the Haar-filters with the image. Haar features are explained further in the supplemental materials. However, the number of features extracted by Haar-like features is very high; thus, training a model using these features can be very challenging and expensive. SDT is designed to overcome such a challenging task and we trained trees over Haar-like features using SDT. We call such tree as Haar tree. We applied

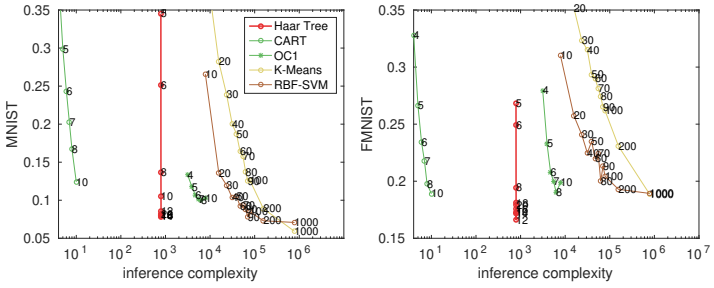

Figure 3: Inference complexity versus error ratio over test set for each model.

Haar-like features to MNIST and FMNIST to extract new features. The size of each region of used Haar filters are $(3n, 3m)$ for $m, n = 1, 2, ..., 8$. 121000 features were extracted from images of 28×28.

Figure 3 presents inference complexity versus error ratio for each model. We compared Haar trees with oblique trees Heath et al. (1993), axis-aligned trees Breiman et al. (1984), RBF-SVM and nearest neighbor k-means. The RBF-SVM consists of training SVM over various Radial Basis Functions (RBF). The center of RBFs are found using k-means algorithm. The width of the RBFs were found using a validationset (trainset was split to 80% train and 20% validation). The k-means model consisted of training a k-means and then using centroids as a nearest neighbor model (used similar to Tavallali et al. (2020)). The label of centroids were selected as the majority of label of samples assigned to each centroid. The horizontal axis in figure 3 represents the computational complexity of evaluating the model for an input (inference complexity). The vertical axis shows the error ratio over the testset. The number at each point on a curve represents the depth of the tree or number of centroids. We observed that the Haar trees were able to achieve better accuracy than the other trees and also better or similar accuracy to other nearest neighbor-based models. The Haar trees are the first axis-aligned trees to achieve accuracy higher than any other axis-aligned trees as per our study. Other axis-aligned trees have never passed the accuracy of 90% over MNIST while Haar Tree could achieve accuracy of 93%. Asymptotically, the inference of a Haar Tree is $\mathcal{O}(D+\Delta)$ because first an integral image is calculated and then at each node, the desired feature is calculated only using the integral image Viola & Jones (2001). The inference of oblique tree (OC1), k-means, and RBF-SVM are $\mathcal{O}(D\Delta)$, $\mathcal{O}(DK)$ and $\mathcal{O}(D(K+1))$, respectively. As can be observed, Haar Trees are faster than other models asymptotically while Haar Trees could achieve smaller test error.

6 LIMITATIONS:

This paper provides the first stochastic approach for induction of a decision tree. Consequently, the model and the proposed algorithm suffers from limitations that a decision tree might suffer. However, in the context of decision tree and ensemble models it potentially can enhance the state of the field, since the approach enables faster and more computationally efficient creation of decision tree and forest models. The approach essentially opens the path for introduction of tree-based convolution filters.

7 CONCLUSION

The tree training is expensive because of the exhaustive search for solving the criterion minimization problem. Despite the vast usage of decision trees in a broad range of applications such image processing, computer vision, and medical diagnosis, the mentioned problem has not been objectively approached. We tackled the problem by proposing a novel stochastic approach for tree induction. To the best of our knowledge, the SDT is the first stochastic algorithm designed for induction of trees through greedy approach. Asymptotically, the SDT improved the decision tree induction complexity by orders of magnitude. It was shown that the algorithm is likely to find the best feature and threshold at the split node. It was also shown that the algorithm minimizes an upper bound for the criterion problem. Experimentally, the SDT could achieve same accuracy as the baseline tree CART and outperformed other similar methods (e.g., online and fast tree induction). In the experiments, SDT could train a tree faster than other algorithms by several orders of magnitude. Finally, using SDT, we tackled the challenging problem of training Haar Trees. The Haar Trees could achieve better accuracy than other more complicated machine learning models, such as oblique trees and nearest neighbor-based models.

8 REPRODUCIBILITY

The experiment codes (currently attached as supplementary materials) will be published on a Github repository to facilitate reproduction and extension of our results. Furthermore, the datasets used in our experiments are publicly available via the UCI repository.

REFERENCES

Frénay Benoît, Mark Van Heeswijk, Yoan Miche, Michel Verleysen, and Amaury Lendasse. Feature selection for nonlinear models with extreme learning machines. *Neurocomputing*, 102:111–124, 2013.

Dhruba Kumar Bhattacharyya and Jugal Kumar Kalita. *Network anomaly detection: A machine learning perspective*. Crc Press, 2013.

Paul S Bradley and Olvi L Mangasarian. Feature selection via concave minimization and support vector machines. In *ICML*, volume 98, pp. 82–90, 1998.

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

Leo J. Breiman, Jerome H. Friedman, R. A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, Calif., 1984.

Qing Chen, Nicolas D Georganas, and Emil M Petriu. Real-time vision-based hand gesture recognition using haar-like features. In *2007 IEEE instrumentation & measurement technology conference IMTC 2007*, pp. 1–6. IEEE, 2007.

Jaesik Choi. Realtime on-road vehicle detection with optical flows and haar-like feature detectors. Technical report, 2012.

José Carlos Cortizo and Ignacio Giraldez. Multi criteria wrapper improvements to naive bayes learning. In *International Conference on Intelligent Data Engineering and Automated Learning*, pp. 419–427. Springer, 2006.

Antonio Criminisi and Jamie Shotton. *Decision Forests for Computer Vision and Medical Image Analysis*. Advances in Computer Vision and Pattern Recognition. Springer-Verlag, 2013.

Sanmay Das. Filters, wrappers and a boosting-based hybrid for feature selection. In *Icml*, volume 1, pp. 74–81, 2001.

Manoranjan Dash and Huan Liu. Feature selection for classification. *Intelligent data analysis*, 1(3): 131–156, 1997.

Cem Demirkir and B Sankur. Face detection using boosted tree classifier stages. In *Proceedings of the IEEE 12th Signal Processing and Communications Applications Conference, 2004.*, pp. 575–578. IEEE, 2004.

Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition*, volume 31. Springer Science & Business Media, 2013.

Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

Johannes Gehrke, Venkatesh Ganti, Raghu Ramakrishnan, and Wei-Yin Loh. Boatoptimistic decision tree construction. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of Data*, pp. 169–180, 1999.

Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. New York, NY: Springer, second edition, 2009.

David Heath, Simon Kasif, and Steven Salzberg. Induction of oblique decision trees. In *IJCAI*, volume 1993, pp. 1002–1007, 1993.

Charles Antony Richard Hoare. Algorithm 64: quicksort. *Communications of the ACM*, 4(7):321, 1961.

Vijay S Iyengar, Chidanand Apte, and Tong Zhang. Active learning using adaptive resampling. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 91–98, 2000.

George H John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Machine Learning Proceedings 1994*, pp. 121–129. Elsevier, 1994.

Michael Jones and Paul Viola. Fast multi-view face detection. *Mitsubishi Electric Research Lab TR-20003-96*, 3(14):2, 2003.

Alan Jović, Karla Brkić, and Nikola Bogunović. A review of feature selection methods with applications. In *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO)*, pp. 1200–1205. Ieee, 2015.

Ron Kohavi and Dan Sommerfield. Feature subset selection using the wrapper method: Overfitting and dynamic search space topology. In *KDD*, pp. 192–197, 1995.

Daphne Koller and Mehran Sahami. Toward optimal feature selection. Technical report, Stanford InfoLab, 1996.

Flip Korn, B-U Pagel, and Christos Faloutsos. On the" dimensionality curse" and the" self-similarity blessing". *IEEE Transactions on Knowledge and Data Engineering*, 13(1):96–111, 2001.

Ju-Hyun Kwak, Il-Young Woen, and Chang-Hoon Lee. Learning algorithm for multiple distribution data using haar-like feature and decision tree. *KIPS Transactions on Software and Data Engineering*, 2(1):43–48, 2013.

Natalia Larios, Bilge Soran, Linda G Shapiro, Gonzalo Martínez-Muñoz, Junyuan Lin, and Thomas G Dietterich. Haar random forest features and svm spatial matching kernel for stonefly species identification. In *2010 20th International Conference on Pattern Recognition*, pp. 2624–2627. IEEE, 2010.

Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In *Proceedings. international conference on image processing*, volume 1, pp. I–I. IEEE, 2002.

Chao Liu, Dongxiang Jiang, and Wenguang Yang. Global geometric similarity scheme for feature selection in fault diagnosis. *Expert Systems with Applications*, 41(8):3585–3595, 2014.

Linxi Liu and Wing Hung Wong. *Multivariate density estimation based on adaptive partitioning: Convergence rate, variable selection and spatial adaptation*. Department of Statistics, Stanford University, 2014.

Sebastián Maldonado, Richard Weber, and Fazel Famili. Feature selection for high-dimensional class-imbalanced data sets using support vector machines. *Information sciences*, 286:228–246, 2014.

Takeshi Mita, Toshimitsu Kaneko, and Osamu Hori. Joint haar-like features for face detection. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pp. 1619–1626. IEEE, 2005.

Mohammad Mahdi Moghimi, Maryam Nayeri, Majid Pourahmadi, and Mohammad Kazem Moghimi. Moving vehicle detection using adaboost and haar-like feature in surveillance videos. *arXiv preprint arXiv:1801.01698*, 2018.

Abdelwadood Moh'd A Mesleh. Chi square feature extraction based svms arabic language text categorization system. *Journal of Computer Science*, 3(6):430–435, 2007.

Ki-Yeong Park and Sun-Young Hwang. An improved haar-like feature for efficient object detection. *Pattern Recognition Letters*, 42:148–153, 2014.

Minh-Tri Pham and Tat-Jen Cham. Fast training and selection of haar features using statistics in boosting-based face detection. In *2007 IEEE 11th International Conference on Computer Vision*, pp. 1–7. IEEE, 2007.

J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

Parikshit Ram and Alexander G Gray. Density estimation trees. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 627–635. ACM, 2011.

Irene Rodriguez-Lujan, Charles Elkan, Carlos Santa Cruz, and Ramón Huerta. Quadratic programming feature selection. *Journal of Machine Learning Research*, 2010.

Marco Sandri and Paola Zuccolotto. Variable selection using random forests. In *Data analysis, classification and the forward search*, pp. 263–270. Springer, 2006.

Jeffrey C Schlimmer and Douglas Fisher. A case study of incremental concept induction. In *AAAI*, volume 86, pp. 496–501, 1986.

Shai Shalev-Shwartz et al. Online learning and online convex optimization. *Foundations and trends in Machine Learning*, 4(2):107–194, 2011.

Sanaz Sheikhi and Seyed Morteza Babamir. A predictive framework for load balancing clustered web servers. *The Journal of Supercomputing*, 72(2):588–611, 2016.

Sanaz Sheikhi and Seyed Morteza Babamir. Using a recurrent artificial neural network for dynamic self-adaptation of cluster-based web-server systems. *Applied Intelligence*, 48(8):2097–2111, 2018.

Jiliang Tang, Salem Alelyani, and Huan Liu. Feature selection for classification: A review. *Data classification: Algorithms and applications*, pp. 37, 2014.

Pooya Tavallali, Mehran Yazdi, and Mohammad Reza Khosravi. Robust cascaded skin detector based on adaboost. *Multimedia Tools and Applications*, 78(2):2599–2620, 2019.

Pooya Tavallali, Peyman Tavallali, Mohammad Reza Khosravi, and Mukesh Singhal. Interpretable synthetic reduced nearest neighbor: An expectation maximization approach. In *2020 IEEE International Conference on Image Processing (ICIP)*, pp. 1921–1925. IEEE, 2020.

Paul E Utgoff. Incremental induction of decision trees. *Machine learning*, 4(2):161–186, 1989.

Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, volume 1, pp. I–I. IEEE, 2001.

Haixun Wang, Wei Fan, Philip S Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 226–235. AcM, 2003.

Ian H Witten and Eibe Frank. Data mining: practical machine learning tools and techniques with java implementations. *Acm Sigmod Record*, 31(1):76–77, 2002.

Kun Yang and Wing Hung Wong. Density estimation via adaptive partition and discrepancy control. *arXiv preprint arXiv:1404.1425*, 2014.

Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pp. 856–863, 2003.

A APPENDIX

This is the Appendix (supplemental materials) for Stochastic Induction of Decision Trees with Application to Learning Haar Tree.

A.1 PROPERTIES OF THE SDT

Theorem 4 is a result of theorem 20.1 and Lemma 20.1 in Devroye et al. (2013).

Proof sketch of theorem 5:

Proof Sketch: Assuming the data IID. The criterion is bounded by 0 from below. At each node, a partition that is minimum of the objective function over a set of features is found. As a result, after partitioning each node, the error will decrease.

A.2 HAAR FEATURES

The exact haar features used for generating new features are presented in figure 4. The sizes of row pixels and column pixels are written right to each filter. For each filter, various filters with every combination of possible row and column pixels are created and convolved with the image to extract create new features.

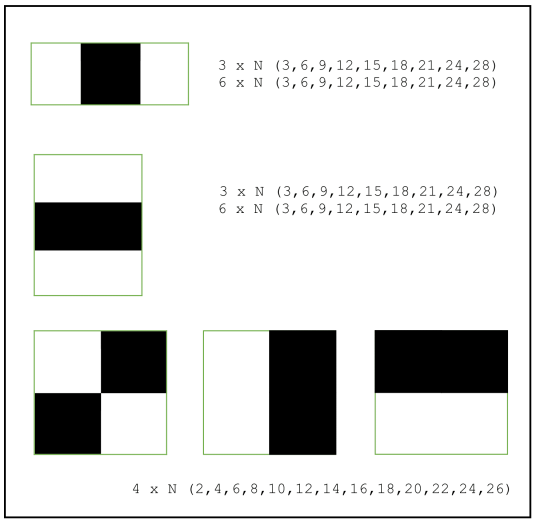

Figure 4: Haar Filters

A.3 EXPERIMENTAL RESULTS

In this section, several experiments that demonstrate properties of SDT are presented.

Figure 5 shows the reduction in impurity (criterion in formula 2) at each iteration of the algorithm for MNIST dataset. Each point on the curve represents the impurity of the best feature and threshold found by the algorithm ($L^k(S_j)$). As figure 5 demonstrate, at each iteration, the loss function has decreased drastically and eventually has converged to the true best feature and threshold.

Figures 6 and 7 present the importance of features at each iteration for problem of inducing one single split node for 2 classes. The feature importances were normalized to between 0 and 1.

From figures 6 and 7, it is observable that the SDT gradually discards the less discriminative features and gives higher value to more distinctive features.

Figures A.3 and A.3 represents computational complexity versus error ratio of train and test. In figures A.3 and A.3 various values are used for N_0 to explore the effect of N_0 over the final tree.

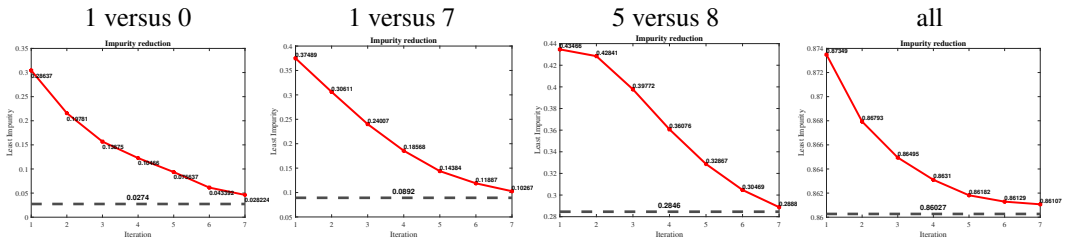

Figure 5: The horizontal axis shows the iteration number of SDT for a single node. Vertical axis shows the impurity. The dashed line represents the impurity of the best feature and threshold. The points on the curves represent exact impurity of the best split found by the algorithm at that iteration. The text above each plot shows the classes used for creating the split node.

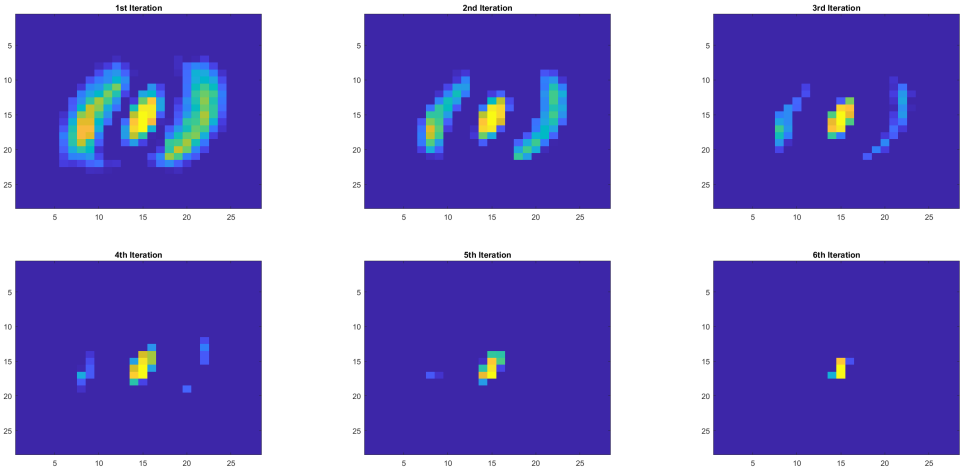

Figure 6: Feature importance of features by SDT at each iteration for problem of 0 versus 1. Brighter pixels show higher value of FI and darker pixels show lower value of FI.

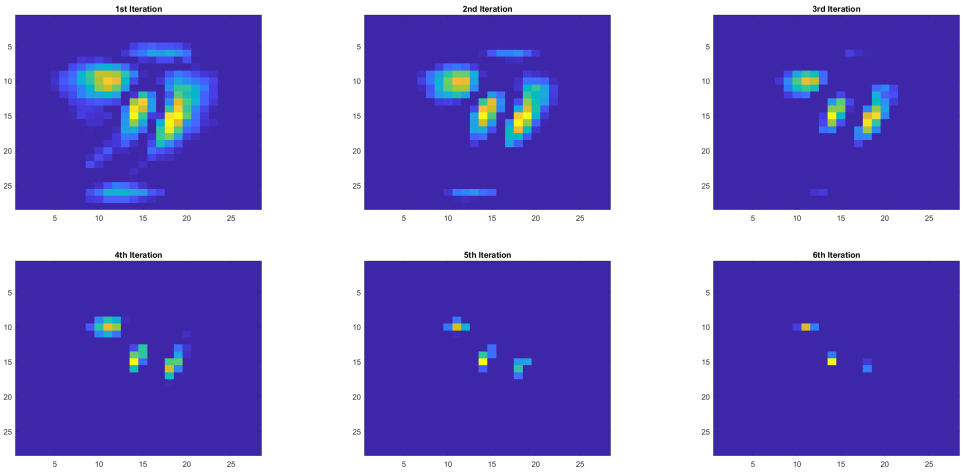

Figure 7: Feature importance of features by SDT at each iteration for problem of 1 versus 7. Brighter pixels show higher value of FI and darker pixels show lower value of FI.

Each curve represents the N_0 by its color. The N_0 for each color is defined at the legend of the plot. The depth of the tree is shown at the points on each curve.

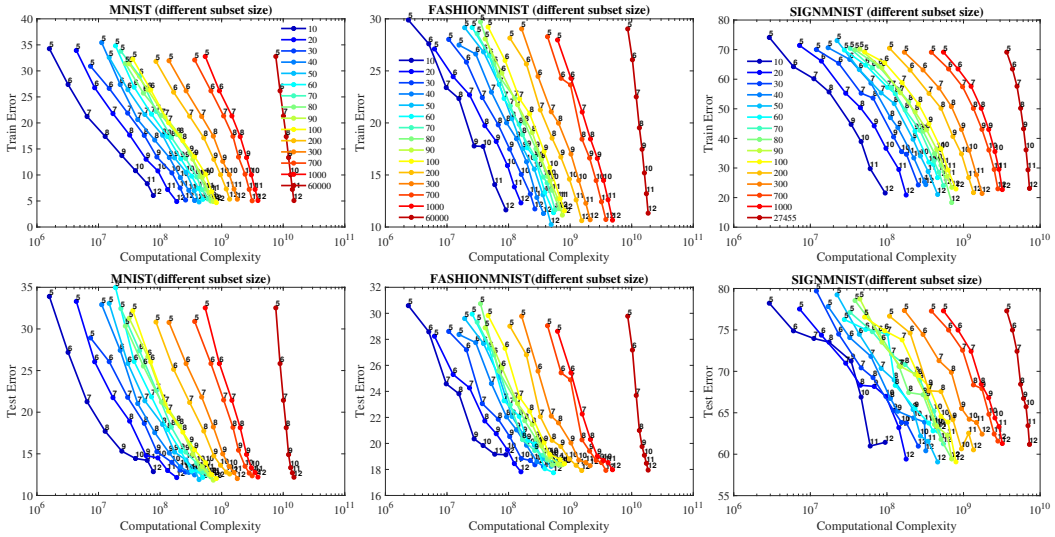

Figure 8: Horizontal axis represents the computational complexity of training a model. The vertical axis shows the error ratio. The colors represent the used value for N_0.

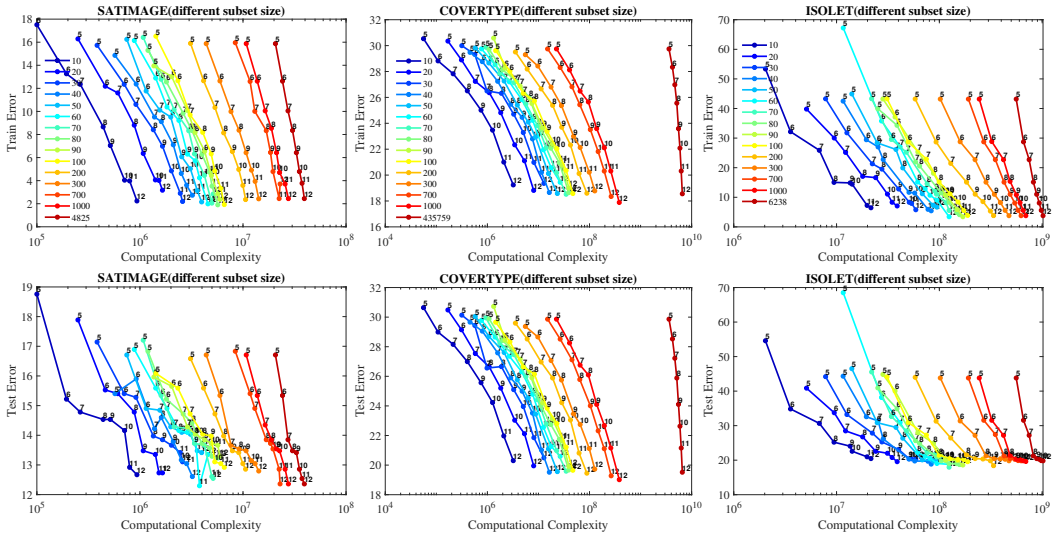

Figure 9: Horizontal axis represents the computational complexity of training a model. The vertical axis shows the error ratio. The colors represent the used value for N_0.