RAPID: Rapid and Precise Interpretable Decision Sets

Sunny Dhamnani*, Dhruv Singal†, Ritwik Sinha‡, M Tharun§ and Manish Dash¶
*Adobe Research, Bangalore, India
†Columbia University, New York, USA
‡Adobe Research, San Jose, USA
§Adobe Systems, Noida, India
¶IBM, Bangalore, India
Email: {dhamnani.sunny, manishdash12}@gmail.com, {risinha, tharun}@adobe.com, dhruv.singal@columbia.edu

***Abstract*—Interpretable Decision Sets (IDS) is an approach to building transparent and interpretable supervised machine learning models. Unfortunately, IDS does not scale to most commonly encountered big data sets. In this paper, we propose Rapid And Precise Interpretable Decision Sets (RAPID), a faster alternative to IDS. We use the existing formulation of decision set learning and propose a time-efficient learning framework. RAPID has two major improvements over IDS. First, it uses a linear-time randomized Unconstrained Submodular Maximization algorithm to optimize the objective function. Second, we design special data structures, based on Frequent-Pattern (FP) trees to achieve better computational efficiency. In this work, we first perform a time complexity analysis of IDS and RAPID, and show the significant advantages of the proposed method. Next we run our algorithm, along with baselines, on three public datasets. We show comparable accuracy for RAPID, with 10,000x improvement in running time over IDS. Additionally, due to the significant improvements in running time of RAPID, we can run more extensive hyperparameter search algorithms, leading to comparable accuracy with competitive baseline models.**

***Index Terms*—big data, interpretable, supervised machine learning, decision rules**

I. Introduction

Machine Learning (ML) models have achieved a significant position within our world, everything from loan approvals to objects detection within images are based on such models. ML and statistical models have served one of two purposes [1], to classify a data point, or explain an observation. Both explanation and classification serve important roles, but in more recent years, the role of classification or prediction has been a more dominant theme within computer science research [2]. Large black-box models have dominated these tasks. For instance, it has been noticed that the best models (with respect to classification accuracy), over hundreds of datasets are not easily interpreted [3]. While this trend towards large models has been motivated by the push to increase the prediction accuracy of these tasks, a casualty of this is the fact that humans who work with these models often fail to understand why a model took the decision that it did.

More recently, this challenge has led to an effort to try to provide interpretability to machine learning models, especially for those dealing with big data systems [4]. The authors in this paper argue that interpretable models help in these following ways. First, they help build systems that are safe, and are taking sound decisions. Second, it helps to debug these systems. Third, it helps science by helping human knowledge understand something it did not know before. Fourth, it helps overcome issues with mismatched objectives and trade-offs due to having multiple objectives. Finally, it helps address the potential to identify legal and ethical boundaries an automated decision system may be crossing. Large scale black-box ML algorithms that take automated decisions about human lives using big data have received increased scrutiny.

Interpretable Decision Sets (IDS) [5] extends decision trees and decision lists via itemset mining. All of which are supervised ML models that are interpretable by design. IDS provides a framework that constructs an objective function composed of terms that assign importance to both accuracy and interpretability. It generates transparent ML models and is shown to achieve interpretability when shown to human end-users of the model. It also achieves this with almost no loss of accuracy, compared to the best predictive models. Unfortunately, the proposal is extremely slow and fails to scale to big data (for example, in our experiments, IDS failed when the dataset had more than a few thousand rows). This limits the applicability of IDS to big data applications that are ubiquitous today.

The main contribution of this work is the proposal of RAPID, an approach that enables interpretable ML on big data. First, we analyze the time complexity of IDS. Next, introducing tailor-made data structures and adopting a state of the art unconstrained submodular maximization routine, we accelerate the process of obtaining interpretable decision sets. We further analyze the time complexity of RAPID, and show the improvements over IDS. Improved run times for RAPID enables us to use more extensive hyper-parameter search. Finally, we test the accuracy and run time of RAPID and comparing it with IDS and other approaches on three publicly available datasets.

Next, the literature related to this work is discussed.

II. Related Work

In large areas of ML application, the data is a collection of observations of human behavior. Important examples include

medicine and social sciences (like education, justice systems and marketing). In such areas, recent effort has involved building ML models which satisfy interpretability, instead of performing an *ex post* approximation of the best black-box model [4].

While providing interpretation as a post-processing task especially for deep neural network models has received much attention [6]–[9], these models typically fall short of providing a fully "transparent" explanation of model predictions [10]. This is primarily because these are generally model agnostic approaches in which the explanations are completely disjoint from the model. For example, LIME [11] and Anchors [12] provide a generic framework to generate local explanations to black-box models in a model-agnostic manner. However, a local interpretation is often limited in its applicability when the goal is to get a broad understanding of the model. In big data applications where predictions are being generated for millions of observations, a locally interpretable model does not provide a scalable model exploration framework.

Furthermore, there are other works that argue for interpretation through examples. In this approach, the functioning of a model is described by identifying specific datapoints. Some studies focus on generating counterfactual explanations [13], [14], where the goal is to identify minimal changes required to change the model's prediction. Another study [15] attempts to find a small number of representative data instances; scrutinizing outcomes of these instances can aid in understanding the working of a model. However, such approaches are often not suitable for big data applications with tabular data, where a data point can comprise of hundreds or even thousands of imperceptible features.

In particular, big data research is increasingly looking towards methods which incorporate interpretability intrinsically in predictive models. For example, [16] propose using interpretable features to augment the performance of automated algorithms in malicious activity detection in enterprise security (which typically entails sifting through millions of data records). To generate the interpretable features, they use itemset mining implicit in the RIPPER [17] algorithm, showing superior recall on comparison with other feature generation techniques (which do not promise interpretability). Similarly, an active area of study for building interpretable ML models uses decision sets [5]. This extends the areas of Decision Trees [18], [19] and Decision Lists [20], [21]. Decision Trees and Decision Lists [22] are naturally transparent because the model's decision process for a given data point can be understood by either tracing the path along the tree or identifying the right rule from a list.

However, algorithms based on such techniques only have practical applicability as long as they are efficient and are able to scale to big data systems. Naturally, scaling such approaches has been a focus of ongoing research efforts [23], [24]. The first of these provides substantial reduction in the search space for candidate pattern sequences in a high utility sequential pattern mining setting, while second introduces novel pruning techniques and augmented pattern-growth tree data structures to perform weighted frequent itemset mining efficiently. Our work adds to this literature by leveraging heuristics involving reducing the search space and augmented pattern growth data structures to improve the efficiency of a leading interpretable ML model which uses decision sets, namely, IDS [5].

IDS is a joint framework for building predictive models that are accurate, but also interpretable. IDS provides a flexible system for achieving this objective and satisfies the simultaneous objectives of accuracy and interpretation. IDS shows improvements over Bayesian Decision Lists [20], CN2 [25] and Classification Based on Associations (CBA) [26]. In ablation and human judgment studies, the models generated by IDS are also shown to be interpretable. While IDS is clearly shown to achieve the required benchmark for human interpretability, the solution to building interpretable decision sets fails to scale to a number of problems where this approach is very applicable and likely to bring value. The largest example datasets in [5] are of the size of $150,000$ data points. This is a grossly small dataset in many problems that are of interest to data science. Datasets in big data applications can easily scale to millions or billions of points, with tens or hundreds of attributes. One bottle neck in IDS stems from the use of the Smooth Local Search algorithm [27] for optimizing the algorithm. The run time of this algorithm is quadratic in the number of input rules. Additionally, IDS fails to achieve other optimizations that may be gleaned by leveraging the characteristics of the problem at hand. The next section describes IDS in some detail.

A. Interpretable Decision Sets

IDS generates an interpretable classification model for multi-class classification tasks, without sacrificing accuracy significantly. In this section, we review the notation and describe the framework of IDS.

```
If Age <= 51 and PR <= 3,    then Chemotherapy
If Age > 51 and PR <= 3,     then Chemotherapy and Tamoxifen
```

Fig. 1: An illustrative example of a decision set containing two rules. Predicates are presented in **bold** and the output labels are in red or blue.

Central to IDS is a decision set, defined as a collection of classification rules, statements of the form "if-then". Figure 1 provides an example. Note that a decision set does not have to be exhaustive; a default class is the decision for observations that fail to meet any rule. We have summarized the terms and notation in Table I. Many of these are common to [5] and we introduce a few additional terms for easier exposition.

Decision sets are easy to comprehend because when considering a rule, reasoning about all the other rules is not required. Decision sets are defined in terms of itemsets. Rules in a decision set are independent and applied individually to an itemset. An itemset s is a conjunction of predicates of the form (attribute, operator, value), for example, $(x_1 < 8)$. We say x satisfies s when all the predicates in s output *True* when evaluated on x. Further a rule r is a tuple of itemset s and class label c, that is, $r = (s, c)$.

TABLE I: Summary of notation

Notation	Description
$\mathcal{D}$	Dataset $\{(x_1, y_1), ..., (x_N, y_N)\}$
x	Observed attribute values of a data point
y	Class label of a data point
N	Number of data points
M	Number of features associated with each x
$\mathcal{C}$	Set of class labels in $\mathcal{D}$
c	A particular class label from $\mathcal{C}$
p	A predicate (attribute-operator-value tuple), e.g., Age $\geq$ 50
s	An itemset (conjunction of one or more predicates), e.g., Age $\geq$ 50 and Gender = Female
$\mathcal{S}$	Input set of itemsets
r	A rule – an itemset-class pair (s, c)
k	Size of input rule-set, $\mathcal{S} \times \mathcal{C}$
L_{max}	Maximum rule length, across all rules in $\mathcal{S} \times \mathcal{C}$
$\mathcal{R}$	Decision set – a set of rules $(s_1, c_1), ..., (s_k, c_k)$

TABLE II: Objective terms (from [5])

Objective	Formula
Set size	$f_1(\mathcal{R}) = k - \text{size}(\mathcal{R})$
Rule length	$f_2(\mathcal{R}) = L_{max} \cdot k - \sum_{r \in \mathcal{R}} \text{length}(r)$
Overlap	$f_3(\mathcal{R}) = N \cdot k^2 - \sum_{r_i, r_j \in \mathcal{R}; c_i = c_j} \text{overlap}_{\mathcal{D}}(r_i, r_j)$
Overlap	$f_4(\mathcal{R}) = N \cdot k^2 - \sum_{r_i, r_j \in \mathcal{R}; c_i \neq c_j} \text{overlap}_{\mathcal{D}}(r_i, r_j)$
Coverage	$f_5(\mathcal{R}) = \sum_{c' \in \mathcal{C}} \mathbb{1}(\exists r = (s, c) \in \mathcal{R} \text{ s.t. } c = c')$
Precision	$f_6(\mathcal{R}) = N \cdot k - \sum_{r \in \mathcal{R}} \lvert\text{incorrect-cover}_{\mathcal{D}}(r)\rvert$
Recall	$f_7(\mathcal{R}) = \sum_{(x,y) \in \mathcal{D}} \mathbb{1}\{\exists r \text{ s.t. } (x, y) \in \text{correct-cover}_{\mathcal{D}}(r)\}$

Following [5], a decision set $\mathcal{R}$ assigns a class label c to attribute values x as follows. If x satisfies exactly one itemset s_i, then its class label is c_i. If x satisfies zero itemsets then its class label is assigned using a default label, and if x satisfies more than one itemset, it is assigned a class using a tie-breaking function. The choice of default class label and class tie-breaking function is up to the user. For our experiments, we report results with the following choices. For data points that satisfy no itemsets, we predict the majority class label in the training data, and for data points that satisfy more than one itemset, we predict using the rule with the highest F1-score on the training data. However, other choices of default class labels and class tie-breaking functions can be easily incorporated. For example, we could use the smallest minority class as the default label.

B. Objective Function

IDS proposes a joint objective that scores decision sets based on how interpretable and how accurate they are. The objective is composed of non-negative reward functions, some of these reward explainability while others reward accuracy. The functions are summarized in Table II. We elaborate on these reward functions next.

1) Explainability: The following five terms reward decision sets which are explainable.

a) Set size: Decision sets with smaller size are favoured. Here, size($\mathcal{R}$) is the size of decision set $\mathcal{R}$.

b) Rule length: Shorter rules are more desirable for interpretability. Here length(r) for rule $r = (s, c)$ is the number of predicates in s.

c) Overlap (two terms): A pair of objective terms to reward decision sets with rules that do not overlap in the feature space. cover$_{\mathcal{D}}(r)$ denotes the set of data points in $\mathcal{D}$ which satisfy the itemset used in r. Further, overlap$_{\mathcal{D}}(r_i, r_j)$ is the number of points in $\mathcal{D}$ which are covered by both rules r_i and r_j.

d) Class coverage: This term in the objective function is used to encourage the decision set to have at least one rule that predicts each class.

2) Accuracy: The next two terms reward accuracy of the classifier.

1) Precision is encouraged by penalizing rules for misclassification. This objective is formalized by defining incorrect-cover$_{\mathcal{D}}(r)$, which is the set of points in $\mathcal{D}$ which are incorrectly classified by rule $r \equiv (s, c)$, that is, the set of points which satisfy the itemset s but which have a different class label than class label c. Smaller incorrect cover is favored to promote precision.
2) Recall is encouraged by defining correct-cover$_{\mathcal{D}}(r)$, which is the set of points in $\mathcal{D}$ that are correctly classified by rule $r \equiv (s, c)$, that is, the set of points which satisfy the itemset s and whose class label is identical to the class label c. Decision sets which correctly cover data points with at least one rule, are favored to increase recall.

The full learning objective is constructed by combing the seven reward functions in the following manner,

$$f((\lambda_1, \lambda_2, \cdots, \lambda_7), \mathcal{R}) = \sum_{i=1}^{7} \lambda_i f_i(\mathcal{R}), \tag{1}$$

where $\lambda_1, \lambda_2 \cdots, \lambda_7$ are non-negative weights that scale the relative importance of the terms. The final decision set is given by

$$\underset{\mathcal{R} \subseteq \mathcal{S} \times \mathcal{C}}{\text{argmax}}\, f((\lambda_1, \lambda_2, \cdots, \lambda_7), \mathcal{R})$$

The learning objective is maximized using the Smooth Local Search algorithm [27], which is an unconstrained submodular maximization method. The algorithm to finding an optimal decision set involves starting with the empty set and adding to it, each time computing the objective function for a decision set, and updated several times within in each iteration.

III. Time Complexity of IDS

A precursor to running IDS is to get a set of candidate sets to choose from. IDS proposes using Frequent Itemset Mining (FIM) algorithms [28] like Apriori [29] with a support threshold of $(\epsilon \cdot |\mathcal{D}|)$, to ensure that the candidate itemsets are present in at least ϵ proportion of data points. The output of an FIM algorithm is the set of the form $\mathcal{S}$.

TABLE III: Time complexity of objective terms in IDS

Objective Function	Complexity
Set size (f_1)	$\Theta(1)$
Rule length (f_2)	$\Theta(k)$
Overlap (f_3)	$\Theta(\lvert\mathcal{R}\rvert^2 \cdot N \cdot (M + \log N))$
Overlap (f_4)	$\Theta(\lvert\mathcal{R}\rvert^2 \cdot N \cdot (M + \log N))$
Coverage (f_5)	$\Theta(\lvert\mathcal{R}\rvert)$
Precision (f_6)	$\Theta(\lvert\mathcal{R}\rvert \cdot N \cdot M)$
Recall (f_7)	$\Theta(\lvert\mathcal{R}\rvert \cdot N \cdot (M + \log N))$

For the time complexity of the overall algorithm, we multiply the time complexity associated with objective function evaluation with number of times this function evaluation routine is invoked. Note that the analysis is entirely based on implementation code as released by the authors of IDS [30].

We first describe the time complexity of all the terms in the objective, summarized in Table III. Function f_1 computes the difference in size between the input rule set and the decision set, a constant time operation. Function f_2 iterates over the input rule set $\mathcal{S} \times \mathcal{C}$ to find L_{max}, and then iterates over $\mathcal{R}$ to sum the length of all rules. Since $|\mathcal{R}| \leq k$, the time complexity of f_2 is $\Theta(k)$. The functions f_3 and f_4, compute overlap between a pair of distinct rules from $\mathcal{R}$, based on the class of the rules. These functions iterate over pairs of rules ($|\mathcal{R}| \times |\mathcal{R}|$), and for each pair they sort the data points in $\Theta(N \log N)$ time and find which of the N rows are covered in $\Theta(NM)$ time. The function f_5 iterates over $\mathcal{R}$ to find the number of classes covered by this decision set.

The function f_6 calculates size of incorrect cover by iterating over each rule in $\mathcal{R}$. In each iteration, rows which are incorrectly covered by a particular rule are determined in $\Theta(N \cdot M)$ time. The last function, f_7, finds the number of points which are correctly covered by at least one rule in decision set $\mathcal{R}$. The function iterates over each rule in $\mathcal{R}$, and in each iteration, sorts the data points in $\Theta(N \log N)$ time, and finds which of the N rows satisfy the rule in $\Theta(NM)$ time.

Hence, the time complexity of computing the objective function T_f for a decision set $\mathcal{R}$ is given as,

$$T_f = \Theta(|\mathcal{R}|^2 \cdot N \cdot (M + \log N)). \tag{2}$$

Next, we consider the number of times the objective function is computed in the SLS algorithm. The objective function is first invoked to compute the estimate of the optimal value of the objective OPT and then iteratively over each successive candidate rule set. Consider T_{OPT}, the time complexity of computing OPT. Here, f is evaluated for a decision set of size $k/2$, hence we can replace $|\mathcal{R}|$ with k in Equation (2).

$$T_{OPT} = \Theta(k^2 \cdot N \cdot (M + \log N)), \tag{3}$$

Next consider T_{loop}, the total time spent iterating over the candidate rule sets. The i^{th} iteration evaluates the objective function for some decision set $\mathcal{R}_i$, therefore using equation (2),

$$T_{loop} = \sum_i \Theta(|\mathcal{R}_i|^2 \cdot N \cdot (M + \log N)). \tag{4}$$

Say that the maximum size of decision set attained across all the iterations is $\mathcal{R}_{max}$. Also, we know that maximum number of iterations is $O(k^2)$ [31], leading to the upper bound in time complexity of,

$$T_{loop} = \mathcal{O}(k^2 \cdot |\mathcal{R}_{max}|^2 \cdot N \cdot (M + \log N)). \tag{5}$$

Note that the decision set is initialized with the empty set ϕ. Since the size of decision set increases by a single unit in each iteration, reaching $\mathcal{R}_{max}$ will involve computations on decision sets of sizes $0, 1, 2, \ldots, \mathcal{R}_{max}$. Thus, a lower bound for Equation (4) can be calculated as follows, using the fact that $\sum_{i=1}^{n} i^2 = (n(n+1)(2n+1))/6$,

$$T_{loop} = \Omega(|\mathcal{R}_{max}|^3 \cdot N \cdot (M + \log N)). \tag{6}$$

The overall time complexity of IDS, $T_{overall}$, is the sum of T_{fim} (cost of finding itemsets in the dataset), T_{OPT} and T_{loop}, that is,

$$T_{overall} = T_{fim} + T_{OPT} + T_{loop}. \tag{7}$$

From Equations (3), (5) and (7) we see that the upper bound in the time complexity of IDS is

$$T_{overall} = \mathcal{O}(k^2 \cdot |\mathcal{R}_{max}|^2 \cdot N \cdot (M + \log N)) + T_{fim}. \tag{8}$$

From Equations (3), (6) and (7), we see that the lower bound for this time complexity is

$$T_{overall} = \Omega((k^2 + |\mathcal{R}_{max}|^3) \cdot N \cdot (M + \log N)) + T_{fim}. \tag{9}$$

IV. Rapid and Precise Interpretable Decision Sets

A key parameter in IDS is the threshold ϵ to run FIM. A lower ϵ will lead to a larger $\mathcal{S}$, implying a bigger value of k ($|\mathcal{S}|$). The lower bound of the time complexity of IDS involves k^2 operations and is multiplied by $N \log N$. For most classification problems on big data today, N ranges from hundreds of thousands to billions. Also, for most practical problems in real datasets, ϵ is conservatively chosen as 0.1 (10% of the dataset). With tens of attributes, this often leads to tens of thousands of itemsets in $\mathcal{S}$. This renders IDS not viable in realistic scenarios. For example, using the author-supplied code from IDS [30], the largest dataset we were able to analyze with IDS[1] had a mere $2,000$ data points, 10 attributes, and $\epsilon = 0.1$. This motivates the need to rethink IDS, to make it usable to build interpretable ML models for modern problems.

In Section II-A, we saw that IDS starts with an input of itemsets generated by the Apriori algorithm. It then calculates the objective function f on rule sets in a direct manner, to compute the set size, rule length, overlap (between pairs of rules), prediction for all classes, precision, and correct cover. Calls to this function are then incorporated within each iteration of the SLS algorithm. In this work, we propose the Rapid and Precise Interpretable Decision Sets (RAPID) algorithm, which improves over IDS in three ways. First, we leverage FIM algorithms that perform better than Apriori. Second, RAPID is based on a better unconstrained submodular

[1]On a Linux machine with 61 GB of memory and running an Intel Xeon E5-2686 v4 8 Core Broadwell Processor with base frequency of 2.3 GHz.

maximization algorithm, when compared to SLS which is used by IDS. Third, the computational efficiency of the reward functions of IDS can be significantly improved.

A. Frequent Itemset Mining

IDS proposes using the Apriori algorithm to generate the input itemset, $\mathcal{S}$. While Apriori was the first algorithm proposed to generate frequent itemsets [29], it has been superseded in efficiency by multiple newer proposals like Frequent Pattern Growth (FP-Growth), Linear time Closed item set Miner (LCM), and Eclat [28], [32], [33]. It has been empirically shown that FP-Growth has orders of magnitudes lower running times than the Apriori algorithm [34].

We further improve the computational efficiency of candidate set generation by selecting a narrower set of *candidate rules* for each class c. The input itemset $\mathcal{S}_c$ is obtained by running FIM only on transactions for class (c) (using a relative threshold ratio of ϵ). This is as opposed to mining rules on the whole transaction database and taking a cross product with the set of all classes ($\mathcal{S} \times \mathcal{C}$). The set over which we optimize f is then simply formed by combining all such candidate sets, that is, $\mathcal{X} = \cup_c(\mathcal{S}_c, c)$. The motivation for this is that, certain rules (s) will be so rare for certain classes (c), that the corresponding rule (s, c) is not a valuable candidate rule to consider in the decision set. This helps us shrink the size of the input itemset, in practice.

B. Unconstrained Submodular Maximization

Maximization of submodular functions is a known NP-Hard problem. It has however been shown that it admits a $1/2$-approximate algorithm [27]. However, the randomized SLS algorithm, proposed in [27] achieves a $2/5$-approximation. IDS leverages this algorithm to maximize its objective function. In more recent work [35], a $1/2$-approximate algorithm for the unconstrained submodular maximization problem has been proposed, *we shall refer to this algorithm as USM*. This achieves a tight approximation guarantee matching the known hardness of this problem of $1/2$ [27].

USM has another important advantage over SLS, for our problem. It is a linear time randomized algorithm (linear in k, the number of rules in the input itemset), compared to SLS, which is $O(k^2)$. An interesting thing to note about USM is that it finds the (approximately) optimal set *incrementally*– a rule once added, is never removed from the set, a property that SLS lacks.

C. Frequent Itemset Counts and FP-Trees

Finally, we speed up the reward function calculation. We create data structures to aid us in optimizing the most computationally intensive reward functions, namely overlap between rules (within class and across classes, components of f_3 and f_4), correct cover for the rule set and incorrect cover for each rule in the rule set (components of f_6 and f_7). We explain the formation of these data structures – frequent itemset counts and FP-Trees in the following discussion.

Support on the transaction database is the outcome of any FIM algorithm. Hence, we simply store the support counts for frequent itemsets of S_c in a hash map (with the itemsets being the keys, to allow for an $\mathcal{O}(1)$ access). We name the data structures storing the class-wise frequent itemset counts as $\{F_c\}$. Further, we also perform FIM on the complete (global) transaction database (ignoring the class information), and store the counts in another data structure G (again, as a hash map). We use an absolute support threshold for the global mining operation. In practice, we run FIM on $\mathcal{S}_c$ with a threshold of $\epsilon \cdot |\mathcal{S}_c|$, the minimum of these counts across all classes is halved to get the threshold for the global FIM. Hence, any itemset which is present in any one of $\{F_c\}$ will also be present in G. Moreover, G will have some extra itemsets, which are not present in any of $\{F_c\}$.

Frequent Pattern Trees (FP-Trees) are special suffix trees introduced in the FP-Growth algorithm [32], the leading FIM algorithm today. It is a compressed representation of frequent itemsets, providing both a vertical and horizontal representation [28]. In their original form, FP-Trees can be utilized to efficiently retrieve the supports for any itemset (given a threshold).

Our algorithm requires maintaining two sets, $\mathcal{A}$ and $\mathcal{B}$. For each class c, we create two FP-Trees for each of $\mathcal{A}$ and $\mathcal{B}$. We name these T_{Ac},T^*_{Ac},T_{Bc} and T^*_{Bc}. The tree T_{Ac} contains frequent patterns (and corresponding support counts) from all data points of class c, which are covered by at least one rule in A. The tree T^*_{Ac} contains the frequent patterns (and corresponding support counts) from all the data points of class c, which are covered by no rule in A. Note that T_{Ac} and T^*_{Ac} form complements – any data point of class c is present in exactly one of these two trees. The trees T_{Bc} and T^*_{Bc} are defined similarly, again defined as complements of each other. Except for T_{Bc}, the other trees are all vanilla FP-Trees, that is, without any modifications to the original formulation given by [32] . T_{Bc} on the other hand, is an augmented data structure that we propose. We name this class of trees *Count FP-Trees*. A Count FP-Tree can be created for any reference set of itemsets (a given set of itemsets, which can be possibly dynamic) and a given transaction database. Just like a normal FP-Tree, it is a suffix tree storing the support of each frequent pattern. However, unlike a normal FP-Tree, every node also stores the number of itemsets (from the reference set) contained by the frequent pattern. Note that we are talking about the count of itemsets, and not count of transactions, i.e. the support. This additional count for the reference set can be initialized at any time – independent of the process of construction of the tree. This implies that any FP-Tree can be converted into a Count FP-Tree, *ex post*.

To construct our suite of FP-Trees, we start with empty trees. We first take all the transactions in class c and add them to the trees T^*_{Ac} and T^*_{Bc}. Now, we iterate over all rules in $\mathcal{S}_c$ and move the subtrees rooted at the frequent pattern nodes corresponding to these rules, to T_{Bc}. At the end of this operation, all the trees adhere to their definition (from the previous paragraph). At this time, we also initialize the reference counts for T_{Bc}, using $\mathcal{B}$ as the reference set.

It is worth mentioning how deletion of rules from the

reference set $\mathcal{B}$ will affect the Count FP-Tree T_{Bc}. Upon deletion of r from $\mathcal{B}$, we find the node in T_{Bc} whose frequent pattern matches with r. We then decrease the reference counts for all nodes in the sub-tree rooted at this node. If the reference count for any node falls to zero, we delete the sub-tree rooted at that node (since no rule in $\mathcal{B}$ covers any transaction indicated by the frequent pattern nodes in this sub-tree).

D. Overview of the Algorithm

The pseudocode of the RAPID is presented in Algorithm 1. Based on USM [35], it starts with two sets. Set $\mathcal{A}$ starts as the null set, and has elements added to it. Set $\mathcal{B}$ starts as $\mathcal{S}$, and has elements removed from it. Depending on whether $\mathcal{A}$ or $\mathcal{B}$ is being updated, all the appropriate FP-trees need to be updated. All elements in $\mathcal{S}$ are evaluated exactly once, and either added to a set or removed from a set.

Algorithm 1 PSEUDOCODE FOR RAPID

Input: Objective f, domain $\mathcal{X} = \cup_c(\mathcal{S}_c, c)$, FP-trees T_{Ac}, T^*_{Ac}, T_{Bc}, and T^*_{Bc} (for all $c \in \mathcal{C}$)
$\mathcal{A} \leftarrow \phi$
$\mathcal{B} \leftarrow \mathcal{X}$
for $r = (s, c) \in \mathcal{X}$ **do**
  $a \leftarrow \max(f(\mathcal{A} \cup \{r\}) - f(\mathcal{A}), 0)$
  $b \leftarrow \max(f(\mathcal{B} \setminus \{r\}) - f(\mathcal{B}), 0)$
  $\gamma \leftarrow \mathit{Uniform}(0, 1)$
  if $\gamma < \frac{a}{a+b}$ **then** ▷ if a and b are both 0, always do this step
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{r\}$
    $\mathit{TreeAdd}(T_{Ac}, r)$
    $\mathit{TreeDelete}(T^*_{Ac}, r)$
  else
    $\mathcal{B} \leftarrow \mathcal{B} \setminus \{r\}$
    $\mathit{TreeDelete}(T_{Bc}, r)$
    $\mathit{TreeAdd}(T^*_{Bc}, r)$
  end if
end for
return $\mathcal{A}$ (or equivalently $\mathcal{B}$).

As previously mentioned, our core set optimization problem involves using the USM algorithm [35], a linear time algorithm ($\Theta(|\mathcal{S}|) = \Theta(k)$). In each iteration of Algorithm 1, there are two key parts – determining whether the current rule being considered is to be added to the set $\mathcal{A}$ or to be deleted from the set $\mathcal{B}$, and then, actually adding or deleting the rule. For the former, we improve over [5] by calculating only the *difference* in the reward functions (we refer to these as Δf_i), incremental effects of adding a single rule to a set. For example, the terms Δf_1 and Δf_2 can be obtained in $\Theta(1)$ time by just retrieving the length of the current rule. The difference terms Δf_3 and Δf_4 entail finding the overlap of the current rule with all the other rules (in $\mathcal{A}$ or $\mathcal{B}$) of the same class and other classes, respectively. Consider the current rule $r = (s, c)$ and a candidate rule $r' = (s', c')$, the overlap of r and r' is simply the support of the itemset $s \cap s'$ (constructed by taking the intersection of all predicates in s and s'). The itemset $s \cap s'$ can have zero support, in case of invalid categorical attribute combinations like {Country = US, Gender = Male} and {Country = US, Gender = Female}. In case the support is non-zero, we leverage the data structure G. Since, we took a finer threshold (ϵ_G) in the construction of G, there is a significant likelihood that $s \cap s'$ is present in G. In that case, we simply retrieve the threshold from G, in $\Theta(1)$ time. In case the combined rule is not present in G, we take the global mining support as the overlap value, since that is a weak upper bound on the overlap. Since retrieval of support from the set G is a $\Theta(1)$ operation, and we perform this retrieval $\Theta(k)$ times, the calculation of Δf_3 and Δf_4 can be done in $\Theta(k)$ time.

Computing the difference Δf_5 for A involves determining whether r covers a previously unrepresented class in the dataset, and similarly Δf_5 for B involves determining whether r is the only rule covering some class in B – both are $\Theta(1)$ operations. Computing Δf_6 involves calculating the size of the incorrect cover of r. We use the data structures G and $\{F_c\}$ to obtain the sizes of the cover and correct cover for r (since by construction of G and $\{F_c\}$, r is present in both). The size of incorrect cover is simply the difference of both these values. Since retrieval from G and $\{F_c\}$ can both be done in $\Theta(1)$ time, Δf_6 is obtained in $\Theta(1)$ time. Lastly, Δf_7 involves calculating the changes in size of the correct cover of rule set $\mathcal{A}$ and $\mathcal{B}$), on adding and deleting rule r from them, respectively. We utilize our FP-Trees T^*_{Ac} and T_{Bc} for this purpose. For $\mathcal{A}$, Δf_7 will simply be the number of transactions in the correct cover of r, which were previously not covered by any rule in $\mathcal{A}$. Hence, we can simply obtain the support of transactions covered by r in T^*_{Ac}. To get Δf_7 for $\mathcal{B}$, we will simulate the rule deletion operation on T_{Bc} (as described in Section IV-C), to get a count of transactions which will be removed from T_{Bc} on deletion of r from $\mathcal{B}$. Since both of these calculations involve traversing the FP-Trees, they are $\Theta(M)$ operations (where M is the depth of the tree). In all, the time complexity of the objective function computation is $\Theta(k + M)$.

After obtaining the values of the objective functions, the set optimization algorithm either adds the rule r to A, or deletes it from B. In both the cases, the FP-Trees are the only data structures that we need to modify. In the former case, we remove the sub-tree rooted at the frequent pattern node of T^*_{Ac} and add it to T_{Ac}. In the latter case, we follow the rule deletion on T_{Bc} (as described in the previous subsection). Both of these operations take $\Theta(M)$ for each such r. Since there are k such rules (and by formulation of the set optimization algorithm, each rule is considered only once), over the complete algorithm, the set operations will take $\Theta(M \times k)$ time.

E. Time Complexity of RAPID

The overall time complexity of RAPID involves time spent in k iterations of Algorithm 1, along with time spent in frequent itemset mining and construction of FP-trees. The loop involves $\Theta(k^2 + M \cdot k + M \cdot N)$ time. Time spent

on mining frequent itemsets & the FP-tree construction is $\Theta(|\mathcal{C}| \cdot T_{fim})$, but for most practical purposes $|\mathcal{C}|$ (number of class labels) is a bounded constant. Ignoring this constant in our asymptotic analysis, the time spent on these two operations is T_{fim}. Therefore, the overall time complexity for RAPID is

$$T_{overall} = \Theta(|\mathcal{R}|^2 + M \cdot |\mathcal{R}| + M \cdot N) + T_{fim}. \quad (10)$$

F. Advantages of RAPID over IDS

RAPID has three major advantages over IDS.

Time Complexity: The first and motivating advantage of RAPID over IDS is its time complexity, as can been seen in Equations (9) and (10). Even if we assume small values for M (number of features) and $|\mathcal{R}_{max}|$ (largest $\mathcal{R}$ in all iterations), IDS is still a $\Theta(k^2 \cdot N \cdot \log N)$ operation. Using small values of ϵ can lead to large values of k, running into thousand of rules, and most big data applications will require running ML models on millions of data points. This limits IDS to applications in datasets of modest sizes (as is also seen in our experiments in Section V). On the other hand, the time complexity of RAPID is dominated by the greater of k^2 and $N \cdot M$, which is much more manageable. Note that RAPID involves multiple runs of the FIM algorithms (as many as $|C|$), but modern implementations of FIM, like FP-Growth [28] are highly optimized, and make up a negligible portion of the running time of the entire algorithm.

Better Guarantees: RAPID leverages the USM algorithm [35], this is a 1/2-approximate maximization algorithm. IDS on the other hand is based on SLS, which is a 2/5-approximate algorithm. Since f is positive, RAPID is likely to identify decision sets which have higher values of the objective than IDS. We demonstrate this qualitatively and quantitatively in Section V.

Extensive Hyperparameter Search: Finally, $\lambda_1, \lambda_2, \ldots, \lambda_7$ are hyperparameters that need to be specified prior to invoking either IDS or RAPID. The authors in [5] suggest using a coordinate ascent method to find parameters that produced a decision set with the highest AUC (on the validation set). The area of hyperparameter selection has been extensively studied, and recent work has focused on Bayesian methods for hyperparameter optimization [36]. An important thing to note about these methods is that they repeatedly call the primary routine during the hyperparameter search. Since RAPID runs much faster in practice, it is possible to use extensive hyperparameter optimization approaches with it. We propose using the Tree-structured Parzen Estimator Approach (TPE) [37] as implemented in [38] to find the values of $\lambda_1, \lambda_2, \ldots, \lambda_7$.

V. Experiments and Observations

In this section, we compare RAPID with baselines on public datasets. The comparison is performed against the following alternatives. The first and primary comparator of RAPID is IDS [5]. We took the implementation of IDS as shared by the authors on their Git repository [30], without any modifications. This is the closest possible representation of the proposal in [5]. Next, we selected a number of machine learning approaches which help us contrast the accuracy of RAPID with these. There are Logistic Regression (LR) [39], Decision Tree (DT) [18], Random Forest (RF) [40], Gradient Boosting Machine (GBM) [41], [42] and a Neural Network (NN) [43].

The choice of competing methods are motivated as follows. RF, GBM and NN are black-box predictors that often observed to be among the best off-the-shelf classifiers [3]. LR is a well understood and popular classification method. LR has good interpretability properties, in that, the feature coefficient estimates represent the log-odds ratios of the presence of a certain binary feature. But the classification rule of LR is a linear function of all features in prediction. Such a rule does not easily lend itself to easy interpretation. Additionally, LR for a multi-class classification problem involves performing multiple one-vs-all classification models. A binary DT provides a way to describe all the terminal nodes of the tree as intersections of predicates. But a predicate in the DT may be a `not` rule, leading to reduced interpretation of the rules when multiple negations happen. Further, as argued in [5] and [44], sets of decision rules are the most interpretable of all machine learning models.

We analyze the performance of our algorithm on three datasets. On the smallest dataset, we perform a relative comparison of IDS and RAPID to show that RAPID continues to maintain the good interpretability properties of IDS. We next show that the running time of RAPID in practice is significantly better than that of IDS. Finally, we show that RAPID continues to be close to the accuracies that can be achieved by the four baselines LR, DT, RF and GBM. All experiments are performed on a Linux machine with 61 GB of memory and running an Intel Xeon E5-2686 v4 8 Core Broadwell Processor with base frequency of 2.3 GHz. All numbers, unless specified, are based on an average over 10 runs.

A. Datasets

The datasets used in [5] are not public, this means that we are unable to test our proposal on these datasets. We thus test our algorithms on some public datasets described below [45]. For the baselines of LR, DT, RF and GBM, the categorical input features are converted to one-hot encoding before applying these methods.

Titanic dataset: This contains data for 1761 Titanic passengers and has been downloaded from a public GitHub repository [30]. Each row represents one passenger. Each passenger is characterized using three categorical features namely passenger category, age category and gender. The outcome variable denotes whether the person survived the sinking of the Titanic. The classification task here involves predicting which passengers survive the mishap.

US Census dataset: The data was collected as part of the 1990 census [45]. Each row is associated with an individual and contains features such as age, income-level, occupation, and so on. The dataset was originally proposed for clustering task, but we artificially construct a prediction task from it. A

TABLE IV: The ratio of reward values for RAPID relative to IDS. Higher value are preferred since we are maximizing the objective. RAPID attains higher values than IDS across different support thresholds, including in the objective function.

Ratio	$Support = 10$	$Support = 20$
$f1_{RAPID}/f1_{IDS}$	1.51 ± 0.08	1.42 ± 0.14
$f2_{RAPID}/f2_{IDS}$	1.31 ± 0.04	1.22 ± 0.06
$f3_{RAPID}/f3_{IDS}$	1.25 ± 0.01	1.02 ± 0.04
$f4_{RAPID}/f4_{IDS}$	1.25 ± 0.01	1.02 ± 0.04
$f5_{RAPID}/f5_{IDS}$	1.00 ± 0.00	0.80 ± 0.26
$f6_{RAPID}/f6_{IDS}$	1.18 ± 0.01	1.08 ± 0.02
$f7_{RAPID}/f7_{IDS}$	0.87 ± 0.03	0.81 ± 0.12
f_{RAPID}/f_{IDS}	1.25 ± 0.01	1.02 ± 0.04

TABLE V: Interpretability metrics of IDS and RAPID for classification task on Titanic dataset.

Model	*Fraction Overlap* ↓	*Fraction Uncovered* ↓	*Average Rule Length* ↓	*Number of Rules* ↓	*Fraction of Classes*(∼ 1)
RAPID(10% supp)	0.08 ± 0.04	0.02 ± 0.01	1.71 ± 0.30	6.80 ± 2.10	1.00 ± 0.00
RAPID(20% supp)	0.11 ± 0.10	0.08 ± 0.11	1.63 ± 0.28	4.80 ± 1.93	1.00 ± 0.00
IDS(10% supp)	0.13 ± 0.05	0.01 ± 0.02	1.82 ± 0.23	10.5 ± 2.46	1.00 ± 0.00
IDS(20% supp)	0.22 ± 0.06	0.05 ± 0.06	1.86 ± 0.15	8.50 ± 2.17	1.00 ± 0.00

specific column is chosen to be intended prediction using the remaining columns, here we try to predict whether a person is married or not. This dataset has $2,458,285$ data points, and in the interest of having a challenging classification task, we only use 7 features.

Mushroom dataset: The dataset is originally meant for binary classification to determine if the species of mushroom is poisonous or edible [45]. We formulated a multi-class classification problem by creating a task to predict whether the stalk surface above ring is fibrous, scaly, silky or smooth, the output label has to be one of these four surface textures. There are $8,124$ data-points and again to make the classification task non-trivial we conducted experiments on 4 features out of 22 available features.

B. Interpretability of Models

Before looking at the quantitative properties of the interpretability of the proposed model, let us look at the types of decision rules generated by RAPID. Figure 2 is a decision set of rules generated to predict survival of Titanic passengers. RAPID generates 5 rules to predict whether a passenger will survive or die on the Titanic based on three categorical features. The average rule size is 1.8 and it has an accuracy of 77%.

```
If Age_Category == child and Gender == female,                then survived
If Passenger_Category == 2nd_class and Age_Category == adult, then died
If Passenger_Category == crew and Gender == male,             then died
If Passenger_Category == 3rd_class,                           then died
If Passenger_Category == 2nd_class and Gender == female,      then survived
```

Fig. 2: An example output of RAPID run on the Titanic dataset.

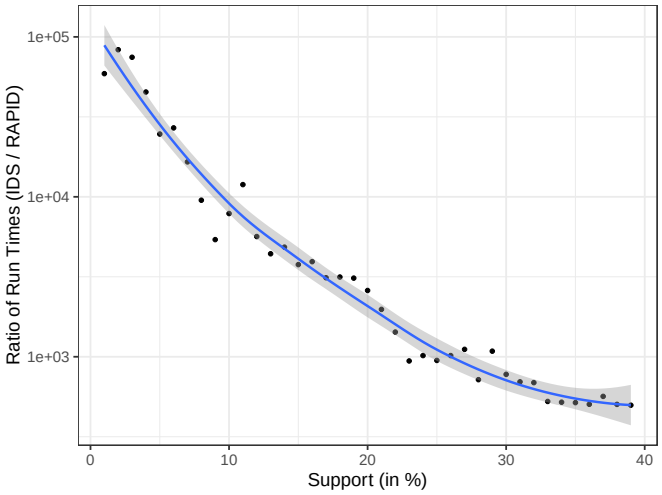

Fig. 3: Run time comparison, IDS vs RAPID on the Titanic dataset. The x-axis is the support threshold used in FIM, and the y-axis marks the corresponding ratio of run times of IDS and RAPID (in log scale). RAPID is four orders of magnitude faster than IDS.

Next, let us look at the quantitative aspects of the interpretability of IDS and RAPID. We first look at a number of parameters which are important for interpretability. First, we contrast the reward function values for IDS and RAPID in Table IV. Note that both algorithms are run with the same hyperparameters. We see that RAPID performs better on all the interpretability metrics ($f_1, \cdots, f_5$), and only slightly worse on one of the two accuracy metrics (f_7). From the last row of Table IV, we see that the relative value of the combined objective function (equation (1)) is on an average 25% more with a support of 10% and 2% more with a support of 20%. This observation is a natural implication of USM (used in RAPID) having better approximation guarantees than SLS (used in IDS).

Five metrics for measuring interpretability are defined and presented in [5]. These are (1) Fraction Overlap (extent of overlap between pairs of rules in a decision set); (2) Fraction Uncovered (data points which are not covered by any rule in decision set); (3) Average Rule Length; (4) Number of Rules; (5) Fraction of Classes (fraction of the class labels in the data predicted by at least one rule in decision set). For the first four of these, a lower value is better; the last metric should be close to 1. Table V presents these metrics (along with standard deviations) for IDS and RAPID for two support levels on the Titanic dataset. The hyperparameters for RAPID and IDS were chosen using TPE and coordinate ascent, respectively. As we can see, the first four terms are similar across the two methods, after accounting for the error bounds. The last metric is 1 for both methods.

C. Run Time Experimentation

On the Titanic dataset, we perform run time comparisons for a range of support values. These are presented in Figure 3 (based on a single run per support value). At a support

threshold of 20% on the Titanic dataset, there are only 24 candidate decision rules from which to select the decision set, even then RAPID is about $2,700$ times faster than IDS. For support thresholds between 1 and 10%, we see that RAPID is more than $10,000$ times faster than IDS. Lower threshold for the support is important to enable us to detect rare but high accuracy rules which may help reliably detect a small, but pure subset of the data, leading to better recall in rare classes.

TABLE VI: Run time of RAPID, IDS, LR, RF and GBM, in seconds. All times are averaged over 10 runs (presented along with the standard deviation).

	Titanic	Census - Sample Proportions			Mushroom
Model		1/100	1/10	1	
RAPID(10% supp)	0.10 ± 0.01	1.40 ± 0.03	8.87 ± 0.06	85.84 ± 0.42	0.62 ± 0.17
RAPID(20% supp)	0.09 ± 0.01	0.90 ± 0.01	7.83 ± 0.04	79.13 ± 0.27	0.56 ± 0.09
IDS(10% supp)	525.03 ± 90.03	TO^*	TO^*	TO^*	TO^*
IDS(20% supp)	147.82 ± 8.00	TO^*	TO^*	TO^*	TO^*
LR	0.01 ± 0.00	0.03 ± 0.01	0.42 ± 0.01	4.45 ± 0.17	0.09 ± 0.03
DT	0.01 ± 0.00	0.01 ± 0.00	0.06 ± 0.00	1.61 ± 0.13	0.02 ± 0.01
RF	0.03 ± 0.01	0.08 ± 0.01	0.50 ± 0.01	15.40 ± 0.52	0.20 ± 0.02
GBM	0.11 ± 0.01	0.59 ± 0.01	6.97 ± 0.16	110.26 ± 1.47	0.88 ± 0.05
NN	0.96 ± 0.17	9.26 ± 2.11	84.55 ± 18.88	404.16 ± 97.76	12.78 ± 1.62

*timeout – exceeded execution time-limit of 1 hour per run

In Table VI we perform multiple runs of these methods for Titanic, Mushroom and different samples from Census (with sampling ratios of $1/100$, $1/10$ and 1) datasets. We see that IDS times out (with a reasonable time-limit of one hour) for all samples of the Census dataset, while RAPID has run times which are comparable to even the baseline machine learning models. For the full Census dataset, RAPID is faster than GBM and NN, and takes about 5x as much time as RF. RAPID can be used to analyze millions of data points within minutes, and promises to be applicable in many realistic settings. Note that the classifiers LR, DT, RF, GBM and NN were used with default configuration settings as implemented in the Python library `scikit-learn` [46]. The NN architecture had four hidden layers with 8, 16, 16 and 8 nodes, respectively.

It is prudent to ask if RAPID is trading running time for memory requirement, given that it creates four FP-Trees for each class label. In our experiments, for the largest dataset, RAPID had a memory footprint of around 2GB for data which takes approximately 640MB of disk space. Thus showing that the memory requirements for the FP-Trees are manageable, that is, without orders of magnitude increase in memory requirements we gain four orders of magnitude in time.

D. Classification Accuracy

Next, we compare the classification performance of RAPID with IDS, LR, DT, RF, GBM and NN. Tables VII and VIII demonstrate the Area under the ROC curve and F1 score metrics for different classifiers on the prediction tasks within the three datasets. Since the classification task on Mushroom dataset is multi-class, we report micro-averaged results on running one-vs-all classifications for all classes. Similar to [5], we used the precision of a rule $r = (s, c)$ (on the class c) as the probability that a given data point is assigned class c by the rule r. The scores reported are averaged over 10 replicates, and

TABLE VII: AUC-ROC metric of RAPID, IDS, LR, RF and GBM on various datasets. All values averaged over 10 runs (presented with the standard deviation).

	Titanic	Census - Sample Proportions			Mushroom
Model		1/100	1/10	1	
RAPID(10% supp)	0.76 ± 0.03	0.78 ± 0.01	0.78 ± 0.01	0.79 ± 0.01	0.87 ± 0.02
RAPID(20% supp)	0.74 ± 0.04	0.75 ± 0.01	0.76 ± 0.01	0.76 ± 0.02	0.86 ± 0.01
IDS(10% supp)	0.76 ± 0.03	TO^*	TO^*	TO^*	TO^*
IDS(20% supp)	0.73 ± 0.02	TO^*	TO^*	TO^*	TO^*
LR	0.77 ± 0.01	0.76 ± 0.01	0.78 ± 0.01	0.79 ± 0.01	0.88 ± 0.00
DT	0.78 ± 0.01	0.78 ± 0.01	0.79 ± 0.01	0.79 ± 0.01	0.89 ± 0.00
RF	0.78 ± 0.01	0.78 ± 0.01	0.80 ± 0.01	0.81 ± 0.01	0.89 ± 0.00
GBM	0.78 ± 0.01	0.79 ± 0.01	0.80 ± 0.01	0.80 ± 0.01	0.89 ± 0.00
NN	0.77 ± 0.02	0.79 ± 0.01	0.80 ± 0.01	0.80 ± 0.01	0.88 ± 0.01

*timeout – exceeded execution time-limit of 1 hour per run

TABLE VIII: F1-score of RAPID, IDS, LR, RF and GBM on various datasets. All values averaged over 10 runs and presented along with standard deviation.

	Titanic	Census - Sample Proportions			Mushroom
Model		1/100	1/10	1	
RAPID(10% supp)	0.86 ± 0.02	0.73 ± 0.01	0.72 ± 0.01	0.72 ± 0.01	0.64 ± 0.02
RAPID(20% supp)	0.85 ± 0.03	0.72 ± 0.01	0.72 ± 0.01	0.72 ± 0.01	0.64 ± 0.01
IDS(10% supp)	0.84 ± 0.01	TO^*	TO^*	$TO*$	TO^*
IDS(20% supp)	0.82 ± 0.01	TO^*	TO^*	TO^*	TO^*
LR	0.87 ± 0.01	0.74 ± 0.01	0.72 ± 0.02	0.70 ± 0.01	0.64 ± 0.01
DT	0.87 ± 0.01	0.73 ± 0.01	0.72 ± 0.02	0.72 ± 0.01	0.64 ± 0.01
RF	0.87 ± 0.01	0.73 ± 0.01	0.72 ± 0.01	0.72 ± 0.01	0.64 ± 0.01
GBM	0.88 ± 0.01	0.74 ± 0.01	0.73 ± 0.01	0.72 ± 0.01	0.64 ± 0.01
NN	0.87 ± 0.02	0.74 ± 0.02	0.73 ± 0.01	0.73 ± 0.01	0.64 ± 0.01

*timeout – exceeded execution time-limit of 1 hour per run

presented along with the standard deviation. In terms of both metrics, the performance of RAPID is marginally superior to IDS, and close to LR, DT, RF, GBM and NN, on the Titanic dataset. We note that IDS timed out on Census and Mushroom datasets. However, as per [5], since our results are comparable to (albeit marginally inferior) those of RF, GBM and NN, RAPID can be expected to be superior, on the margin, to IDS on these two datasets as well. This finding is to be expected since RAPID is sacrificing accuracy to ensure that the model is also interpretable, consistent with [5].

VI. Conclusion

In this work, we propose the RAPID algorithm, which is a scalable approach to building interpretable ML models. The state of the art algorithm, IDS, has a number of appealing properties – it presents a flexible objective function that incorporates both explainability (or interpretability) and accuracy, and operates naturally on categorical data (which is common to many applications). However, the proposed algorithm in IDS fails to scale to anything beyond datasets of modest sizes. Since most common problems require building machine learning models on millions to billions of data points, IDS is limited in its applicability. RAPID on the other hand has all the good properties of IDS, while also easily scaling to millions of data points. RAPID also has better guarantees compared to IDS, and more extensive hyperparameter search can be performed, further boosting accuracy. There are multiple lines of future work that can be explored. RAPID can be extended to regression, by changing the objective functions, while still

leveraging the proposed novel data structures. Another promising avenue of research can be to improve the maximization algorithm by leveraging specific properties of the objective function.

REFERENCES

[1] G. Shmueli, "To explain or to predict?" *Statistical science*, vol. 25, no. 3, pp. 289–310, 2010.

[2] J. M. Hofman, A. Sharma, and D. J. Watts, "Prediction and explanation in social systems," *Science*, vol. 355, no. 6324, pp. 486–488, 2017.

[3] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3133–3181, 2014.

[4] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," *arXiv preprint arXiv:1702.08608*, 2017.

[5] H. Lakkaraju, S. H. Bach, and J. Leskovec, "Interpretable decision sets: A joint framework for description and prediction," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1675–1684.

[6] Q.-s. Zhang and S.-C. Zhu, "Visual interpretability for deep learning: A survey," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 1, pp. 27–39, 2018.

[7] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *ICCV*, 2017, pp. 618–626.

[8] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *ICML*, 2017.

[9] Q. Zhang, Y. Nian Wu, and S.-C. Zhu, "Interpretable convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8827–8836.

[10] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models," *ACM computing surveys (CSUR)*, vol. 51, no. 5, p. 93, 2018.

[11] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you?: Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 2016, pp. 1135–1144.

[12] ——, "Anchors: High-precision model-agnostic explanations," in *AAAI Conference on Artificial Intelligence*, 2018.

[13] T. Laugel, M.-J. Lesot, C. Marsala, X. Renard, and M. Detyniecki, "Comparison-based inverse classification for interpretability in machine learning," in *Information Processing and Management of Uncertainty in Knowledge-Based Systems. Theory and Foundations*, J. Medina, M. Ojeda-Aciego, J. L. Verdegay, D. A. Pelta, I. P. Cabrera, B. Bouchon-Meunier, and R. R. Yager, Eds. Cham: Springer International Publishing, 2018, pp. 100–111.

[14] S. Wachter, B. Mittelstadt, and C. Russell, "Counterfactual explanations without opening the black box: Automated decisions and the GPDR," *Harv. JL & Tech.*, vol. 31, p. 841, 2017.

[15] B. Kim, R. Khanna, and O. Koyejo, "Examples are not enough, learn to criticize! Criticism for interpretability," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. USA: Curran Associates Inc., 2016, pp. 2288–2296.

[16] J. Duan, Z. Zeng, A. Oprea, and S. Vasudevan, "Automated generation and selection of interpretable features for enterprise security," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 1258–1265.

[17] W. W. Cohen, "Fast effective rule induction," in *Machine learning proceedings 1995*. Elsevier, 1995, pp. 115–123.

[18] L. Breiman, *Classification and regression trees*. Routledge, 2017.

[19] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[20] B. Letham, C. Rudin, T. H. McCormick, D. Madigan *et al.*, "Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model," *The Annals of Applied Statistics*, vol. 9, no. 3, pp. 1350–1371, 2015.

[21] D. Bertsimas, A. Chang, and C. Rudin, "Orc: ordered rules for classification a discrete optimization approach to associative classification," 2012.

[22] R. L. Rivest, "Learning decision lists," *Machine learning*, vol. 2, no. 3, pp. 229–246, 1987.

[23] R. U. Kiran, A. Kotni, P. K. Reddy, M. Toyoda, S. Bhalla, and M. Kitsuregawa, "Efficient discovery of weighted frequent itemsets in very large transactional databases: A re-visit," in *2018 IEEE International Conference on Big Data (Big Data)*, Dec 2018, pp. 723–732.

[24] S. Buffett, "Candidate list maintenance in high utility sequential pattern mining," in *2018 IEEE International Conference on Big Data (Big Data)*, Dec 2018, pp. 644–652.

[25] P. Clark and T. Niblett, "The CN2 induction algorithm," *Machine learning*, vol. 3, no. 4, pp. 261–283, 1989.

[26] B. L. W. H. Y. Ma and B. Liu, "Integrating classification and association rule mining," in *Proceedings of the fourth international conference on knowledge discovery and data mining*, 1998.

[27] U. Feige, V. S. Mirrokni, and J. Vondrak, "Maximizing non-monotone submodular functions," *SIAM Journal on Computing*, vol. 40, no. 4, pp. 1133–1153, 2011.

[28] C. Borgelt, "Frequent item set mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 6, pp. 437–456, 2012.

[29] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '93. New York, NY, USA: ACM, 1993, pp. 207–216.

[30] H. Lakkaraju. (2017) Interpretable Decision Sets. Online, Accessed 18-August-2019. [Online]. Available: https://github.com/lvhimabindu/interpretable_decision_sets

[31] U. Feige, V. S. Mirrokni, and J. Vondrák, "Maximizing non-monotone submodular functions," *SIAM J. Comput.*, vol. 40, no. 4, pp. 1133–1153, Jul. 2011.

[32] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '00. New York, NY, USA: ACM, 2000, pp. 1–12.

[33] R. Sinha, D. Singal, P. Maneriker, K. Chawla, Y. Shrivastava, D. Pai, and A. R. Sinha, "Forecasting granular audience size for online advertising," in *Proceedings of the ADKDD'18*. ACM, 2018.

[34] C. Borgelt, "An Implementation of the FP-growth Algorithm," in *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*. ACM, 2005, pp. 1–5.

[35] N. Buchbinder, M. Feldman, J. Seffi, and R. Schwartz, "A tight linear time (1/2)-approximation for unconstrained submodular maximization," *SIAM Journal on Computing*, vol. 44, no. 5, pp. 1384–1402, 2015.

[36] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown, "Towards an empirical foundation for assessing bayesian optimization of hyperparameters," in *NIPS workshop on Bayesian Optimization in Theory and Practice*, vol. 10, 2013, p. 3.

[37] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, 2011, pp. 2546–2554.

[38] J. Bergstra, D. Yamins, and D. D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," in *Proceedings of the 12th Python in Science Conference*. Citeseer, 2013, pp. 13–20.

[39] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied logistic regression*. John Wiley & Sons, 2013, vol. 398.

[40] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[41] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[42] ——, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[43] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[44] C. Molnar, *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/, 2018, https://christophm.github.io/interpretable-ml-book/.

[45] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.