Optimal Action Extraction for Random Forests and Boosted Trees

Zhicheng Cui
Washington University in St. Louis, USA
z.cui@wustl.edu

Wenlin Chen
Washington University in St. Louis, USA
wenlinchen@wustl.edu

Yujie He
Washington University in St. Louis, USA
yujie.he@wustl.edu

Yixin Chen
Washington University in St. Louis, USA
chen@cse.wustl.edu

ABSTRACT

Additive tree models (ATMs) are widely used for data mining and machine learning. Important examples of ATMs include random forest, adaboost (with decision trees as weak learners), and gradient boosted trees, and they are often referred to as the best off-the-shelf classifiers. Though capable of attaining high accuracy, ATMs are not well interpretable in the sense that they do not provide actionable knowledge for a given instance. This greatly limits the potential of ATMs on many applications such as medical prediction and business intelligence, where practitioners need suggestions on actions that can lead to desirable outcomes with minimum costs.

To address this problem, we present a novel framework to post-process any ATM classifier to extract an optimal actionable plan that can change a given input to a desired class with a minimum cost. In particular, we prove the NP-hardness of the optimal action extraction problem for ATMs and formulate this problem in an integer linear programming formulation which can be efficiently solved by existing packages. We also empirically demonstrate the effectiveness of the proposed framework by conducting comprehensive experiments on challenging real-world datasets.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications-Data Mining; J.3 [**Computer Applications**]: Life and Medical Sciences

General Terms

Theory

Keywords

Actionable knowledge, additive tree model, decision tree, random forest, integer linear programming

1. INTRODUCTION

Machine learning and data mining techniques have witnessed great success, a great deal of which can be attributed to the use of additive tree models (ATMs). ATMs are ensemble models built on top of multiple decision/regression trees. A large scope of widespread models can be viewed as special cases of ATMs, such as random forest, adaboost with trees, and gradient boosting trees.

In addition to superior classification/regression performance, ATM enjoys many appealing properties that other machine learning models lack [16], including the support for multi-class classification, nonlinear classification capability, and natural handling of missing values and data of mixed type (categorical and numerical features). Often referred to as the best off-the-shelf classifiers [16], ATMs have been widely deployed into many industrial products such as Kinect [26] and face detection in camera [29], and are the must-try methods for lots of data mining competitions such as web search ranking [25].

Despite these advantages, ATMs often face a key hurdle in many practical applications: the lack of *actionability.* Given an input instance and a desired output, the actionability of a model is the ability to identify a set of changes to the input features that transforms the model prediction of this input to the desired output. Automatic extraction of *actionable knowledge* rather than resorting to domain experts is badly needed in many real-world applications such as business intelligence. For example, an insurance company gives each of their customers a credit score by their underlying model. For customers with low credit scores, the insurance company may want to provide to those customers personal suggestions on how to improve their credit score to a desired level.

Take a recent clinical application as another example. There has been an emerging trend of using machine learning models as the workhorse for the early warning systems to predict sudden deterioration (e.g., septic shock, stroke, respiratory arrest) for hospitalized patients based on their vital signs (such as blood pressure, heart rate, oxygen saturation and etc) [1, 24] with interpretability in mind [9, 8]. A model with actionability can be readily *translated into intervening* *actions* to help avert potential clinical deterioration when the model predicts an undesired outcome. For example, if the model not only predicts that a patient is likely to have respiratory arrest in the next 48 hours, but also provides actionable knowledge such as increasing oxygen saturation, then the doctors can exploit the suggested actions to reduce the likelihood of sudden deterioration.

As most research in data mining and machine learning has focused on the accuracy, efficiency, and robustness of different techniques, only limited efforts have been made to extract actionable knowledge from advanced machine learning models. Some earlier works on actionable knowledge have focused on the development of effective interestingness metrics [17, 5]. Liu et al. proposed methods for pruning and summarizing the learnt rules, as well as matching rules by similarity [21, 22]. Cao et al. proposed domain-driven data mining, a paradigm shift from a research-centered discipline to a practical tool for actionable knowledge [6, 7]. Yang et al. have studied extracting actionable knowledge for a group of input instances based on a single decision tree and proposed a greedy algorithm to solve the problem with an approximation bound [31].

For certain models with a continuous closed-form function such as neural networks, the direction of feature change can be extracted by a gradient-based method [27]. However, extracting actionable knowledge from ATMs is more difficult since the tree models are discrete in nature and we cannot directly compute the gradients. The problem becomes even harder when it is compounded with the fact that changing different features may have different costs. For example, the cost of making a change to gender should be considered enormous or unrealistic. In fact, we will show that extracting actionable knowledge from ATMs is a NP-hard problem.

In this paper, we propose a novel approach to post-process any ATM classifier to extract an *optimal* actionable plan that can change a given input to a desired output with a minimum cost. In particular, we model the problem as an integer linear programming (ILP) problem which is efficient to solve with state-of-the-art solvers such as CPLEX [18]. Any approximation method for the ILP problem will get the same approximation ratio for our problem of extracting actionable knowledge. This is akin to the spirit of other reduction methods in machine learning [32, 10] which reduce a new problem to another well-established problem to exploit its extensive research and development.

In a nutshell, we make the following contributions in this paper:

- We formally define the problem of optimally action extraction (OAE) based on a general formulation of ATMs that covers a broad spectrum of machine learning models.
- We prove the NP-hardness of the OAE problem on ATMs.
- We propose a novel method that translates the OAE problem into a closed-form integer linear programming (ILP) problem which can be efficiently solved by off-the-shelf ILP solvers.
- We empirically demonstrate the effectiveness of our method on various datasets with excellent performance.

The rest of the paper is organized as follows. In Section 2, we formally define ATMs and several important models. In Section 3, we formalize the problem of optimal action extraction (OAE) and prove its NP-hardness. In Section 4, we develop the integer linear programming (ILP) formulation for OAE and prove its correctness. Related works are surveyed in Section 5. We present experimental results on several real-world benchmark datasets on Section 6 and conclude the paper in Section 7.

2. ADDITIVE TREE MODELS

In this section, we first introduce a general formulation for *additive tree models* that our approach can be applied. Our formulation is very general and encompasses several most important and popular models for classification and regression, including random forests, adaboost, and gradient boosting trees. As a result, the proposed action extraction method has very wide applicability.

2.1 General model

An **additive tree model (ATM)** is an ensemble of T decision trees where T is the number of trees. Let $\mathbf{x} = (x_1, \cdots, x_D) \in \mathcal{D}_1 \times \cdots \mathcal{D}_D$ be a D dimensional feature vector, where each x_i, $i = 1, \cdots, D$, can be either categorical or numerical.

Each decision tree outputs a real value; let the output from tree t be $f_t(\mathbf{x})$. For both classification and regression, the output F of the additive tree model is a weighted sum of all the tree outputs as follows:

$$F(\mathbf{x}) = \sum_{t=1}^{T} w_t f_t(\mathbf{x}), \qquad (1)$$

where $w_t \in \mathcal{R}$ is the weight of tree t. The formulation in (1) is very general and include some popular models as special cases. Next, we describe several additive tree models that are widely used in real-world applications.

2.2 Random forests

Random forest is arguably the most popular and powerful off-the-shelf classifier [4]. It can cope with regression and multi-class classification on both categorical and numerical datasets with superior accuracy. In essence, random forest is a bagging model [3] of trees where each tree is trained independently on a group of randomly sampled instances with randomly selected features.

Suppose we are given a dataset $\{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$ where N is the number of training instances, $\mathbf{x}^{(i)}$ and $y^{(i)}$ are feature vector and label of the i-th sample, respectively . The training of a random forest is as follows:

For $t = 1, \cdots, T$,

1. Sample n_{try} instances from the dataset with replacement.
2. Train an unpruned decision or regression tree f_t on the sampled instances with the following modification: at each node, choose the best split among m_{try} features randomly selected rather than among all features.

Both n_{try} and m_{try} and predefined constants. The final random forest model is the average of the outputs from all the trees, *i.e.* $F(\mathbf{x}) = \sum_{t=1} \frac{1}{T} f_t(\mathbf{x})$. For ease of presentation, we only consider binary classification for now. It is straightforward to extend our work to multi-class classification.

We can see that the random forest is a special case of (1) with $w_t = \frac{1}{T}$. To use random forests for classification, $f_t(\mathbf{x}) \in \{0, 1\}$ for all $t = 1, \cdots, T$, and $F(\mathbf{x})$ represents the probability of the label being 1. From the perspective of bias and variance decomposition, each tree in the random forest has low training bias but high variance since the tree is not pruned. The use of bagging over those independent low-bias trees greatly reduces the variance of the resulting model, leading to superior generalization ability.

2.3 Boosted trees

Unlike a random forest that combines low-bias unpruned trees, the boosting method ensembles multiple weak learners to make a strong final model [13]. For boosting, each weak learner is typically a shallow tree with high bias and low variance.

The boosting method trains the additive model sequentially in a forward stage-wise manner. Suppose $F_t(\cdot)$ is the resulting model up to stage t. The model of the next stage is

$$F_{t+1}(\mathbf{x}) \leftarrow F_t(\mathbf{x}) + \alpha_t f_t(\mathbf{x}),$$

where $f_t(\cdot)$ is the weak learner obtained at stage t and α_t is the weight of this weak learner. The final model turns out to be a weighted sum of all trees:

$$F(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t f_t(\mathbf{x}),$$

which is a special case of the general additive tree model in (1) with $w_t = \alpha_t$.

There are two common ways to train the weak learners, leading to two different models: adaboost and gradient boosting trees.

Adaboost.

When training a new weak learner, adaboost focuses more on instances that previous weak learners cannot correctly classify [13]. At stage t, the weight α_t and the tree model $f_t(\cdot)$ are jointly optimized to minimize the loss function L of the resulting model F_{t+1}.

$$\underset{\alpha_t, f_t}{\text{minimize}} \quad \sum_{i=1}^{N} L(y^{(i)}, F_t(\mathbf{x}^{(i)}) + \alpha_t f_t(\mathbf{x}^{(i)})) \tag{2}$$

where L is a loss function measuring the difference between the true label $y^{(i)}$ and the model $F_{t+1}(\mathbf{x}^{(i)})$. In particular, when L is the exponential loss ($L(a, b) = e^{-ab}$), there is a nice closed-form solution for α_t, and f_t can be learned by training on the *weighted* instances.

Gradient boosting.

Gradient boosting is an even more general technique for boosting [14]. It counts each addition to the current model $F_t(\cdot)$ as a gradient update of (2) in the function space of $F_t(\cdot)$, where α_t is the learning rate and $f_t(\cdot)$ is the negative gradient of minimizing the loss function L. Specifically, $f_t(\cdot)$ is trained so that

$$f_t(\mathbf{x}^{(i)}) \approx -\frac{\partial L(y^{(i)}, F_t(\mathbf{x}^{(i)}))}{\partial F_t(\mathbf{x}^{(i)})}$$

which is equivalent to training a regression tree on the original instances with new labels defined by the negative gradient. The learning rate α_t can be set as a small constant or determined by a line search.

3. THE OPTIMAL ACTION EXTRACTION PROBLEM

In this section, we formally define the problem of optimally extracting actionable knowledge from additive tree models defined in (1), which as we show above is very general and encompasses random forests, adaboost, and gradient boosted trees as special cases.

Definition 1. **The optimal action extraction (OAE) problem**. Given a feature vector $\mathbf{x}^c$ and an additive tree model in (1) with output function F, the problem is to find a feature vector $\mathbf{x}$ such that $F(\mathbf{x})$ reaches a target value and the change from $\mathbf{x}^c$ to $\mathbf{x}$ incurs a minimum cost as measured by a cost function $\ell(x^c, x)$. Formally, it is:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \ell(\mathbf{x}^c, \mathbf{x}), \\ \text{subject to} \quad & F(\mathbf{x}) \geq z, \end{aligned} \tag{3}$$

where $z \in \mathcal{R}$ is the target output. $\ell(\mathbf{x}^c, \mathbf{x})$ is the cost function measuring the cost of changing from $\mathbf{x}^c$ to $\mathbf{x}$. For example, $\ell(\mathbf{x}^c, \mathbf{x}) = \|\mathbf{x} - \mathbf{x}^c\|_0$ evaluates the number of features changed.

We now show that the OAE problem is in general a *NP-hard* problem. We prove it by reducing from the DNF-MAXSAT problem [23] defined below.

Definition 2. **The DNF-MAXSAT problem** is defined over a boolean formula in disjunctive normal form. Suppose a set of boolean variables $\{v_1, v_2, \cdots, v_D\}$ where $v_i \in \{0, 1\}$, and a set of T clauses where each clause $\phi_i, i = 1, \cdots, T$ has the form $\phi_i = \wedge_j l_{ij}$, where l_{ij} is the j^{th} literal in the i^{th} clause and is either a positive or a negative expression of a variable. If clause i is satisfied, we have $\phi_i = 1$; otherwise $\phi_i = 0$. The problem is to determine if there exists an assignment of all the boolean variables such that there are at least m clauses satisfied, *i.e.* $\sum_i^T \phi_i \geq m$.

THEOREM 1. *The OAE problem in* (3) *is NP-hard.*

PROOF. It is well known that the DNF-MAXSAT problem is NP-hard [15]. We now reduce the DNF-MAXSAT problems to the OAE problem in (3).

Given any DNF-MAXSAT problems with T clauses, we construct an additive tree model (and in particular, a random forest) with D binary features and T binary trees where each tree represents a clause. To construct tree i, we iterate through all literals in clause ϕ_i and execute the following steps:

1. Construct the root node and denote it as q. Let $j \leftarrow 1$.
2. Suppose l_{ij} is a literal involving boolean variable v_k, then node q is a decision node for feature k. If $l_{ij} = v_k$, construct two children for node q where the right child is a leaf node labeled as 0, and assign q's left child to q. If $l_{ij} = \neg v_k$, a similar operation will be applied with the left child being a leaf node and right child assigned to q.
3. Let $j \leftarrow j + 1$. If $j = n_i + 1$, make node q a leaf node with label 1; otherwise, go to Step 2.

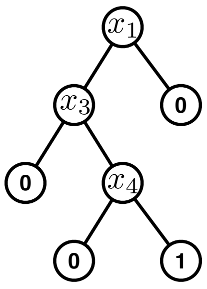

Figure 1: An illustration of constructing the tree for clause $v_1 \wedge \neg v_3 \wedge \neg v_4$.

An example of constructing the tree for clause $v_1 \wedge \neg v_3 \wedge \neg v_4$ is shown in Figure 1. Following the above procedure, we construct a random forest for regression. In particular, each tree only has one leaf node labeled as 1. The output of tree i is 1 if and only if all literals in clause i are made true. Let all $w_t = \frac{1}{T}$, the output of this constructed random forest is exactly $\frac{1}{T}\sum_{i=1}^{T}\phi_i$. Finally, the DNF-MAXSAT problem reduces to the OAE problem in (3) with $\ell(\mathbf{x}^c, \mathbf{x}) = 0$ and $z = m/T$. $\square$

Given that the OAE problem is NP-hard, we do not expect to have any efficient algorithm for optimally solving it. Note that when there is only one decision tree, the OAE problem is in fact easy. We can enumerate all the paths leading up to leaves with desired outputs and choose one with the minimum cost. However, such enumeration does not work for ATMs since the outputs from different trees are inter-connected due to the features they share. Changing the value of a feature may impact the outputs of multiple trees and the final weighted-sum output. Next, we present our solution which formulates the OAE problem as an integer linear programming problem and utilizes highly-optimized optimization solvers.

4. AN INTEGER LINEAR PROGRAMMING APPROACH

In this section, we describe our integer linear programming approach for solving the OAE problem (3). Specifically, we convert the constraint $F(\mathbf{x}) \geq z$ to a more approachable format so that efficient off-the-shelf optimization solvers can be directly applied to it.

The formulation consists of four parts: 1) tree output formulation, 2) feature value formulation, 3) decision logic formulation, and 4) objective function. We will discuss each part separately before putting them together.

4.1 Tree output

We note that (3) is not in closed form because $f_t(\mathbf{x})$ in (1), representing the output from a tree, is not in a mathematical formula. To address this problem, suppose m_t is the number of leaf nodes in the t^{th} tree.

Tree output variables. For each leaf node $k = 1, \cdots, m_t$, we use a binary variable $\phi_{t,k} \in \{0, 1\}$ to denote whether a given instance $\mathbf{x}$ reaches it.

Tree output constraints. Due to the property of the tree structure, each instance reaches exactly one leaf node.

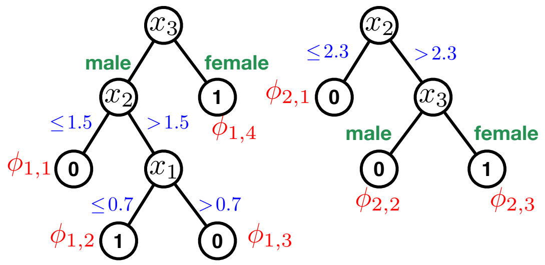

Figure 2: An illustration of random forest containing three features where the the third feature is categorical and the other two are numerical. The leaf nodes with label 1 are the target leaf nodes.

Therefore, we have,

$$\sum_{k=1}^{m_t} \phi_{t,k} = 1, \quad \text{for } t = 1, \cdots, T, \tag{4}$$

where T is the number of trees in the additive tree model. This way, $f_t(\mathbf{x})$ can be expressed as:

$$f_t(\mathbf{x}) = \sum_{k=1}^{m_t} h_{t,k}\phi_{t,k}. \tag{5}$$

where $h_{t,k} \in \mathcal{R}$ is the value of leaf node k in tree t.

Note that although (4) (5) and (1) form an optimization problem in (3), with $\phi_{t,k}$ and $\mathbf{x}$ being the variables, it is incorrect as it does not consider the correlation between $\phi_{t,k}$ among different trees. For example, Figure 2 shows a small random forest with two trees. In this case, it is impossible to have both $\phi_{1,1} = 1$ and $\phi_{2,3} = 1$. If $\phi_{1,1} = 1$, x_3 should be "male". If $\phi_{2,3} = 1$, x_3 should be "female" instead, which is conflicting. Therefore, we need to impose more constraints so that the solution respects the property of the additive tree model.

4.2 Feature value

Definition 3. **(Feature partitions.)** Given an additive tree model, each feature x_i, $i = 1, \cdots, D$, is split into a number of partitions.

- If x_i is categorical with n categories, then x_i naturally has n partitions.
- If x_i is numerical, we assume each tree node branches in the form of $x_i \geq b$ where $b \in \mathcal{R}$ is a **splitting point**. If there are n splitting points for x_i in all the trees in the additive tree model, the feature x_i is naturally split into $n + 1$ partitions.

In the following, let n_i be the number of partitions for feature x_i.

Feature value variables. Given an instance $\mathbf{x}$, we use a binary variable $v_{ij} \in \{0, 1\}$ to denote whether x_i is in the j^{th} partition of dimension i. $v_{i,j} = 1$ if and only if x_i is in the j^{th} partition.

Feature value constraints. Since an instance could only reside in exactly one partition for each feature, we know that v_{ij} satisfies the following property:

$$\sum_{j=1}^{n_i} v_{ij} = 1 \tag{6}$$

Example 1. As shown in Figure 2, the feature x_2 occurs twice in the non-leaf nodes of the random forest. The split points are 1.5 and 2.3, respectively, leading to three partitions for this feature, *i.e.* $(-\infty, 1.5]$, $(1.5, 2.3]$ and $(2.3, +\infty)$. Given an instance $\mathbf{x} = (x_1, \cdots, x_D)$, if $1.5 < x_2 \leq 2.3$ (say $x_2 = 1.9$), then $v_{2,2} = 1$ and $v_{2,1} = v_{2,3} = 0$. $\square$

4.3 Decision logic

Now we need to link $v_{i,j}$ with $\phi_{t,k}$. Given a leaf node k in tree t, suppose $\pi_{t,k}$ is the set of all its ancestor nodes in the tree. For any node $p \in \pi_{t,k}$, suppose p branches on feature i, we define $S_{k,p}$ to be the set containing all predicates $v_{i,j}$ satisfying that $v_{ij} = 1$ leads to the branch towards leaf node k.

Example 2. Continuing with Figure 2, the second leaf node of the first tree (marked with $\phi_{1,2}$) has three ancestors, *i.e.* $\pi_{1,2} = \{x_3, x_2, x_1\}$[1]. For node x_2, the corresponding $S_{2,x_2} = \{v_{2,2}, v_{2,3}\}$, since both $v_{2,2} = 1$ (*i.e.* $x_2 \in (1.5, 2.3]$) and $v_{2,3} = 1$ (*i.e.* $x_2 \in (2.3, +\infty)$) are in the branch where the second leaf node lies. Likewise, $S_{2,x_3} = \{v_{3,1}\}$ where $v_{3,1} = 1$ stands for $x_3 =$ "male". $\square$

Making use of the tree structure, we have the following properties:

- If $\phi_{t,k} = 1$, meaning that the instance $\mathbf{x}$ lies in the leaf node k in tree t, then there is always one of the predicates v_{ij} in $S_{k,p}$ being 1 for any node $p \in \pi_{t,k}$.
- If $\phi_{t,k} = 0$, then there exists at least one node p such that all the predicates in $S_{k,p}$ are 0. For example, to have $\phi_{1,2} = 1$, either $v_{2,2}$ or $v_{2,3}$ should be 1. To have $\phi_{1,2} = 0$, there should be at least one node that does not branch towards the leaf node k.

Putting the above properties in a mathematical form, we have that

$$\begin{aligned} \phi_{t,k} = 1 &\iff \sum_{v \in S_{k,p}} v = 1, \quad \forall p \in \pi_{t,k} \\ \phi_{t,k} = 0 &\iff \sum_{v \in S_{k,p}} v = 0, \quad \exists p \in \pi_{t,k} \end{aligned} \tag{7}$$

(7) is still not in a closed form and we need to further reduce it. To simplify (7), we first prove the following results.

LEMMA 1. *Given that* $0 \leq u_i \leq 1$, $\frac{1}{U}\sum_{i=1}^{U} u_i = 1$ *implies* $u_i = 1$ *for* $i = 1, \cdots, U$.

THEOREM 2. *If* (6) *holds, we have that*

$$\frac{1}{|\pi_{t,k}|} \sum_{p \in \pi_{t,k}} \sum_{v \in S_{k,p}} v \leq 1 \tag{8}$$

where $|\pi_{t,k}|$ *is the cardinality of* $\pi_{t,k}$, i.e. *the number of ancestor nodes of the leaf node* k. *In addition, there is only one leaf node satisfying* (8) *with equality.*

PROOF. First, we prove (8) holds. Suppose that node $p \in \pi_{t,k}$ branches at feature $r(p)$. Combined with (6), we have that

$$\sum_{v \in S_{k,p}} v \leq \sum_{j=1}^{n_{r(p)}} v_{r(p),j} = 1, \quad \forall p \in \pi_{t,k} \tag{9}$$

which leads to (8).

Next, we prove that only one leaf node satisfies (8) with equality. We prove this by contradiction. Suppose there are two leaf nodes satisfying (8) with equality. Let node a and b be these two leaf nodes. Hence, we have that

$$\frac{1}{|\pi_{t,k}|} \sum_{p \in \pi_{t,k}} \sum_{v \in S_{k,p}} v = 1, \quad \text{for } k = a, b.$$

According to (9) and Lemma 1, we have that $\sum_{v \in S_{a,p}} v = 1$ and $\sum_{v \in S_{b,p}} v = 1$ for any node p in π_a and π_b.

Let node q be the lowest common ancestor of a and b and it branches at feature i, *i.e.* $r(a) = r(b) = i$. We know that $S_{a,q} \cap S_{b,q} = \emptyset$, since they belong to different branches of q. Hence, we have:

$$\sum_{i=1}^{n_i} v_{i,j} \geq \sum_{v \in S_{a,q} \cup S_{b,q}} v = \sum_{v \in S_{a,q}} v + \sum_{v \in S_{b,q}} v = 2, \tag{10}$$

which contradicts (6). $\square$

According to (8), we can simplify (7) into the following equivalent equation:

$$\phi_{t,k} = \left\lfloor \frac{1}{|\pi_{t,k}|} \sum_{p \in \pi_{t,k}} \sum_{v \in S_{k,p}} v \right\rfloor, \forall t, k \tag{11}$$

Eq. (11) implies the following three properties:

1. From Theorem 2, we know that the term within the floor operation is equal to or less than 1. Therefore, $\phi_{t,k} \leq 1$, $\forall t, k$.
2. From Lemma 1, we also have that $\phi_{t,k}$ in (11) is 1 if and only if $\sum_{v \in S_{k,p}} v = 1$, $\forall p \in \pi_{t,k}$.
3. From Theorem 2, we have that there is only one leaf node k in tree t satisfying that $\phi_{t,k} = 1$.

The presence of the floor operator poses great difficulty to optimization. To address it, we replace the equality constraint by an inequality constraint so that the floor operator can be removed. This gives rise to the following **decision logic constraints**:

$$\phi_{t,k} \leq \frac{1}{|\pi_{t,k}|} \sum_{p \in \pi_{t,k}} \sum_{v \in S_{k,p}} v, \forall t, k \tag{12}$$

Next, we show that (12) is equivalent to (11) when the tree output constraints in (4) and feature value constraints in (6) are all satisfied.

THEOREM 3. *When* (4) *and* (6) *hold,* (12) *is equivalent to* (11).

PROOF. It is easy to show that (11) implies (12) due to the property of the floor operator. Now we show that (12) implies (11). There are two cases:

a) If

$$\frac{1}{|\pi_{t,k}|} \sum_{p \in \pi_{t,k}} \sum_{v \in S_{k,p}} v < 1,$$

[1]With a slight abuse of notations, we now use x_i to represent the node for simplicity of presentation, because each feature only occurs once in the first tree of Figure 2.

from (12) we have that $\phi_{t,k} < 1$. Since $\phi_{t,k} \in \{0,1\}$, we must have $\phi_{t,k} = 0$.

b) If

$$\frac{1}{|\pi_{t,k}|} \sum_{p \in \pi_{t,k}} \sum_{v \in S_{k,p}} v = 1,$$

according to Theorem 2, there is only one leaf node k' in tree t that satisfies this property. Since all $\phi_{t,k} \leq 1$ and only $\phi_{t,k'}$ is possible to be 1, if $\phi_{t,k'} = 0$, we have $\sum_{k=1}^{m_t} \phi_{t,k} = 0$, which contradicts (4). Therefore, $\phi_{t,k'} = 1$.

In both cases, (12) implies (11). □

Based on the above analysis, we have shown that (12) correctly models the decision logic of additive trees.

4.4 Objective function

Now we consider the objective function $\ell(\mathbf{x}^c, \mathbf{x})$ in (3). The objective plays a key role for users to specify their preferences over which kind of changes to make. For business operations, for example, each change may involve certain investment or operational cost and we want to achieve our goal with the minimum expense. For clinical prediction, the doctors may want the algorithm to suggest as few vital signs as possible, in order to understand and identify the key factor for intervention.

Let $\mathbf{v}$ be the vector containing all feature value variables $v_{i,j}$. Since $\mathbf{v}$ represents the partition each dimension of $\mathbf{x}$ should fall in, we see that all $\mathbf{x}$'s that correspond to the same $\mathbf{v}$ have the same outputs from the trees and the ATM. Therefore, we see that we can replace the cost function $\ell(\mathbf{x}^c, \mathbf{x})$ in (3) by $\ell(\mathbf{x}^c, \mathbf{v})$, which models the minimum cost of moving a given sample $\mathbf{x}^c$ to the partitions specified in $\mathbf{v}$.

A typical cost function is defined over a **cost matrix** $\mathbf{C}(\mathbf{x}^c)$, in which $C_{i,j}(\mathbf{x}^c) \in \mathcal{R}$ is the cost of moving the i^{th} feature of $\mathbf{x}^c$ to the partition $v_{i,j}$. The objective function can be expressed as follows.

$$\ell(\mathbf{x}^c, \mathbf{v}) = \sum_{i=1}^{D} \sum_{j=1}^{n_i} v_{i,j} C_{i,j}(\mathbf{x}^c) \tag{13}$$

which is a linear function of $v_{i,j}$. In principle, $C_{i,j}(\mathbf{x}^c)$ is user-defined, offering total flexibility in modeling the cost of changes in practice. For example, if a feature i cannot be easily changed (such as gender or age), we can assign $C_{i,j}(\mathbf{x}^c) = \infty$ for all $j \neq i$.

For each numerical feature i, we can define a general form for $C_{i,j}(\mathbf{x}^c)$. Let $b_{i,j}$ and $e_{i,j}$ be the beginning and end values of partition $v_{i,j}$, respectively. For example, if $v_{i,j} = (1, 5, 2.3]$, $b_{i,j} = 1.5$ and $e_{i,j} = 2.3$. A general form of the cost matrix is as the following.

$$C_{i,j}(\mathbf{x}^c) = \begin{cases} 0, & x_i^c \text{ is in } v_{i,j} \\ \min\{(x_i^c - b_{i,j})^p, (x_i^c - e_{i,j})^p\}, & \text{otherwise} \end{cases} \tag{14}$$

where x_i^c is the i^{th} feature value of $\mathbf{x}^c$, and $p \in \mathcal{R}$ is a constant.

With the cost matrix in (14), our cost function in (13) is general enough to accommodate most common needs in practice. In fact, the above cost function represents an ℓ_p **norm**. For example, when $p = 0$, the cost matrix defines the ℓ_0 norm:

$$C_{i,j}(\mathbf{x}^c) = \begin{cases} 0, & x_i^c \text{ is in } v_{i,j} \\ 1, & \text{otherwise} \end{cases} \tag{15}$$

which makes our OAE problem minimize the number of changed features, *i.e. Hamming distance.* When $p = 1$, it defines an ℓ_1 norm and the OAE problem minimizes the *Manhattan distance.* When $p = 2$, it defines an ℓ_2 norm and the OAE problem minimizes the *Euclidean distance.*

4.5 Overall optimization formulation

Combining all the four parts above, we now can fully express the OAE problem in (3) in a closed-form optimization problem by integrating the objective in (13) and constraints in (1), (4), (5), (6), and (12). The overall problem is:

$$\begin{aligned} \underset{v_{i,j}, \phi_{t,k} \in \{0,1\}}{\text{minimize}} \quad & \sum_{i=1}^{D} \sum_{j=1}^{n_i} v_{i,j} C_{i,j}(\mathbf{x}^c) \\ \text{subject to:} \quad & \sum_{t=1}^{T} w_t \sum_{k=1}^{m_t} h_{t,k} \phi_{t,k} \geq z \\ & \sum_{j=1}^{n_i} v_{ij} = 1, \quad i = 1, \cdots, D \\ & \phi_{t,k} \leq \frac{1}{|\pi_{t,k}|} \sum_{p \in \pi_{t,k}} \sum_{v \in S_{k,p}} v, \quad \forall t, k \\ & \sum_{k=1}^{m_t} \phi_{t,k} = 1, \quad t = 1, \cdots, T \end{aligned} \tag{16}$$

Since all the objective and constraints are linear functions over the binary variables $\mathbf{v}$ and ϕ, (16) is an **integer linear programming (ILP)** problem. ILPs have been extensively studied and can be efficiently solved by powerful off-the-shelf solvers such as IBM ILOG CPLEX. Note that CPLEX supports a distributed parallel algorithm that can naturally leverage parallel and multi-core computers.

4.6 Discussions

We give some more discussions on the OAE problem and the proposed method.

Multi-class classification. When it comes to classification, the defined OAE problem in (3) can only be applied when the class label is binary. For multi-class datasets, we can still train ATMs with the original labels. When solving the OAE problem, we can use the one-versus-all trick to reduce multinomial labels to binary ones. Specifically, we set the desired class to the positive label and all other classes to negative label. This way, both the model of ATMs in (1)and the definition in (3) remain unchanged, and our method can still be applied.

Efficiency. The proposed method involves solving an ILP problem as shown in (16). Its efficiency highly depends on the dimensionality of the dataset since the number of design variables in (16) increases as the dimensionality gets higher. Therefore, our method is not suitable for datasets with very high dimensionality such as text and image. In fact, for ultra-high dimensional data, it typically does not make much sense to identify changes in the original feature space (e.g., change a few pixels in an image).

Individualization. Solving the OAE problem not only provides an action plan, but also helps identify individualized feature importance for each instance. Most feature selection algorithms are based on a given training dataset and classifier, but are not customized for an individual instance. In contrast, the output from OAE can be viewed as a result of individualized feature selection, as it identifies the few features that can most efficiently change the prediction output of a particular instance. Such individualized feature selection may find many applications, such as personalized healthcare and targeted marketing.

5. RELATED WORK

Actionable knowledge discovery has been studied mostly in the domain of business and marketing. Research on this subject is still very limited in the data mining and machine learning community.

Some earlier works on actionable knowledge have focused on the development of effective interestingness metrics. Hilderman et al. proposed an two-step process for ranking the interestingness of discovered patterns [17]. Chi-square test is used in the first step and objective measures of interestingness are used in the second step. Cao et al. highlighted both technical interestingness and domain-specific expectations and proposed a two-way framework to measure knowledge actionablility [5]. They developed a ranking mechanism which balances the technical and business interests.

Another line of work on actionable knowledge discovery develops post-analysis techniques. Liu et al. tried to discover actionable knowledge by pruning and summarizing the learnt rules, as well as matching rules by similarity [21, 22]. Cao et al. proposed domain-driven data mining, a paradigm shift from a research-centered discipline to a practical tool for actionable knowledge [6, 7]. Ubiquitous intelligence must be involved and meta-synthesized into the mining process. They proposed several types of frameworks capable of handling different problems and applications.

Techniques have also been proposed to postprocess decision trees to extract actionable knowledge [31, 19, 30, 12]. For example, Yang et al. considers the problem of suggesting actions that maximize the expected profit based a decision tree model [31]. Their work is significantly different from ours. Yang's work considered finding the best actions for a *group* of instances based on a single tree, and proposed a greedy algorithm that approximately solves the problem. We consider finding the best actions for a single given instance based on an ensemble of trees, and proposed a mathematical programming algorithm that exactly solves the problem.

Another related topic is the problem of negotiable features where actionable knowledge extraction is based on changing negotiable features to achieve desired results [2]. However, they require those features to be monotonic and sensitive. Actionable changes are also closely relevant to interpretability and cost-sensitive learning. For example, in order to leverage the interpretability of one single decision tree and the superior accuracy performance of random forest, Domingos [11] proposed to rebuild a decision tree based on the randomly generated examples from the random forest. Cost-sensitive learning [28] also seeks to minimize the feature cost. However, the primary goal is to maximize classification accuracies which is quite different than our scenario.

6. EXPERIMENTAL RESULTS

In this section, we conduct extensive experiments on several benchmark datasets to evaluate the proposed methods on the OAE problem. For ease of presentation, we refer to our method as the ILP method.

Dataset	N	D	C
heart	270	13	2
liver disorders	345	6	2
breast cancer	683	10	2
australian	690	14	2
ionosphere	351	34	2
a1a	1000	123	2
mushrooms	8124	112	2
dna	3386	180	3
glass	214	9	6
vowel	990	10	11

Table 1: The number of instances (N), number of features (D), and number of classes (C) of all testing datasets

6.1 Baseline methods

For comparison, we consider the following baseline methods for solving the OAE problem. These baseline methods are reasonable and in some cases highly competitive. None of them can guarantee optimality as our ILP method does. We do not report results on some exhaustive search methods that can guarantee optimality since they are prohibitively expensive and can only solve tiny cases.

1. **Iterative testing.** This method iterates through all training instances available, denoted as $\mathbf{x}^{(i)}$, and compute the prediction $F(\mathbf{x}^{(i)})$ as well as the cost $\ell(\mathbf{x}^c, \mathbf{x}^{(i)})$. The final solution has the minimum cost among all instances with the desired output, as follows:

$$\min_i \ell(\mathbf{x}^c, \mathbf{x}^{(i)}), \quad s.t.: \; F(\mathbf{x}^{(i)}) \geq z$$

2. **Greedy algorithm.** Starting from $\mathbf{x}^c$, this method greedily changes the feature that maximizes $F(\mathbf{x})$. Mathematically, it starts from $\mathbf{x}^0 = \mathbf{x}^c$, and solves the following problem in the i^{th} iteration:

$$\max_{\mathbf{x}^i} F(\mathbf{x}^i) - F(\mathbf{x}^{i-1}), \quad s.t.: \; \|\mathbf{x}^i - \mathbf{x}^{i-1}\|_0 = 1. \quad (17)$$

To solve (17) and find $\mathbf{x}^i$, the greedy algorithm first pick a feature, denoted as d, and fixes all other features of $\mathbf{x}^{i-1}$. Then it iterates through all partitions of feature d and finds one that maximizes the marginal utility $F(\mathbf{x}^i) - F(\mathbf{x}^{i-1})$. Namely, each iteration changes one feature to a value that maximizes the gain. The greedy algorithms keeps iterating until $F(\mathbf{x}) \geq z$ is met or there is no more features to change. The goal of this method is simply finding a feasible solution without considering the cost.

3. **Cost-aware greedy algorithm.** This method is similar to the greedy algorithm except that it takes the cost into account by changing the utility to a cost-aware utility. In particular, (17) is changed to the following and the rest of the greedy process remains the same:

$$\max_{\mathbf{x}} \frac{F(\mathbf{x}^i) - F(\mathbf{x}^{i-1})}{\ell(\mathbf{x}^i, \mathbf{x}^{i-1})}, \quad s.t. \; \|\mathbf{x}^i - \mathbf{x}^{i-1}\|_0 = 1 \quad (18)$$

Dataset	Greedy hit rate
heart	0.83
liver disorders	1.00
breast cancer	0.52
australian	0.91
ionosphere	0.69
a1a	0.71
mushrooms	0.24
dna	0.58
glass	1.00
vowel	1.00

Table 2: Probability rate that the greedy algorithm finds a feasible solution on the test datasets.

From (18), we can see that with this method a feature change with a high cost $\ell(\mathbf{x}^c, \mathbf{x})$ is less likely to be chosen.

6.2 Experimental setup

We test all the methods on nine benchmark datasets from the UCI repository[2] and the LibSVM website[3]. The information of datasets are listed in Table 1. For compactness, we only show the experimental results on random forest for classification since we found the basic trend is consistent among all ATMs. The number of trees in the random forest is 200. We split each dataset into training and testing sets. The random forest is built on the training set. Implementation-wise, we adopt the R package implemented by Liaw et al., which is the current state-of-the-art random forest package [20]. For the OAE problem, we choose a weighted Euclidean distance as the loss function,

$$\ell(\mathbf{x}^c, \mathbf{x}) = \sum_{i=1}^{D} \beta_i (x_i^c - x_i)^2 \tag{19}$$

where β_i is the cost weight on feature i. For each dataset, we randomly generate ten sets of feature cost β_i in the range between 1 and 100. For each set of cost, we sample ten testing instances at random and solve the OAE problem with desired outputs being the other classes.

In summary, for each dataset, for each method, we solve the OAE problem under $100(C-1)$ number of different settings where C is the number of unique labels in a dataset. The reported results are the average of all tested settings. All experiments are run on a desktop computer with 2.5GHz CPU and 16G memory.

6.3 Finding feasible solutions

Our ILP method is provably guaranteed to find a feasible solution to the OAE problem (*i.e.* $F(\mathbf{x}) > z$) unless there's no feasible solution for the problem. The iterative testing algorithm is also guaranteed to find a feasible solution as long as the dataset contains at least one instance with the desired output. However, since the greedy algorithm and cost-aware greedy algorithm are only heuristics for searching a feasible solution, they do not always find one in the end. In fact, they fail to find a feasible solution in many cases.

In Table 2, we list the probability that the greedy algorithm manages to find a feasible solution for each tested dataset. The number in Table 2 shows the percentage of runs that the greedy algorithm can find a feasible solution. We can see that this rate varies dramatically among all the datasets. For datasets like glass, the greedy algorithm always finds a feasible solution while in mushroom dataset it fails in most cases.

[2]https://archive.ics.uci.edu/ml/datasets.html

[3]http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

6.4 Comprehensive results

Table 3 shows a comprehensive comparison in terms of the running time and the solution quality measured by the cost $\ell(\mathbf{x}^c, \mathbf{x})$. The reported results are the average performance of different settings as stated before. We omit the settings where either the greedy algorithm or cost-aware greedy algorithm fails to find a feasible solution, since it results in an infinite loss.

From Table 3, we make the following observations: 1) Our ILP method always attains the minimum cost and largely outperforms all other methods in terms of quality, which is under our expectation since the ILP method is guaranteed to find the optimal solution. As a result, the ILP method also takes more time than all other methods. 2) The cost-aware greedy algorithm outperforms the greedy algorithm in most cases in terms of solution quality. This is because the greedy algorithm does not take costs into consideration when searching for a feasible solution. In terms of running time, these two methods are comparable. 3) Though the iterative testing algorithm is the most efficient among all tested methods, it has the worst performance in terms of the solution quality.

We show a scatter plot of the performance of all methods under all settings in Figure 3. The x-axis stands for relative cost, which is the ratio of the cost obtained by each method over the optimal cost (found by the ILP method). We can see that all results by the ILP method lies in the spike whose relative cost is 1. For these baseline methods, although they are in general faster, their solution quality can be extremely (up to 10^3 times) poor, making their solution not useful. As we have discussed before, in many cases, extracting the optimal actions is a means for individualized feature selection. Non-optimal methods with poor quality are not useful in this case.

7. CONCLUSIONS

As data mining and machine learning are increasingly used by researchers and practitioners in making decisions with critical consequences (e.g. in medicine or business), it is insufficient for decision makers to only have prediction results. Rather, it is absolutely crucial that, for any input instance, the users are able to identify changes to the input features that transform the model prediction to the desired output. Although much efforts have been developed to improve the accuracy and efficiency of learning models, the important notion of *actionability* is still not well studied for many important models.

In this paper, we proposed a novel method to support actionability for additive tree models (ATMs). We started from a mathematical formulation of ATMs, which is general and accommodates some of the most powerful classification and regression models including random forests, adaboost, and gradient boosted trees. We then formally defined the optimal action extraction (OAE) problem, which is to find the set of actions that can change an input instance's status to a desired one with the minimum cost. We proved that

Dataset	Iterative testing		Greedy		Cost-aware Greedy		ILP method	
	Time(sec)	Cost	Time(sec)	Cost	Time(sec)	Cost	Time(sec)	Cost
a1a	0.12	116.46	3.38	39.48	4.75	32.74	7.35	**9.24**
liver disorders	0.00	11.64	2.21	8.21	2.57	4.02	31.22	**1.10**
australian	0.00	41.76	2.99	12.30	3.13	6.51	108.31	**3.42**
breast cancer	0.00	74.19	0.16	41.69	0.22	45.57	31.80	**18.35**
dna	0.08	1112.22	4.23	34.45	5.00	34.47	9.84	**24.10**
glass	0.00	16.40	1.38	21.63	1.36	11.82	57.67	**1.18**
heart	0.00	53.30	1.12	32.52	1.10	22.67	5.71	**2.51**
ionosphere	0.01	195.67	15.01	53.53	21.20	40.87	48.89	**6.98**
mushrooms	0.27	199.68	0.91	55.51	1.26	68.69	3.49	**32.53**
vowel	0.02	34.44	7.65	22.55	7.57	10.86	68.72	**1.24**

Table 3: Solution time and action costs of various methods. All results are averaged over 100(C-1) runs. Optimal costs are marked in bold. See text for details of experimental setup.

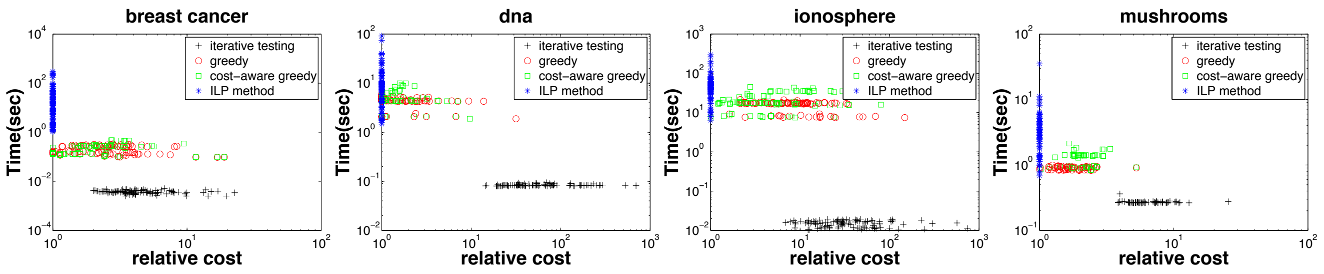

Figure 3: Results of running time and relative costs of all the methods under different settings.

the OAE problem on ATMs is NP-hard. We then proposed an integer linear programming (ILP) formulation to model the OAE problem and proved its correctness. The ILP formulation allows us to leverage on the extensive research and development on ILP solvers such as IBM ILOG CPLEX. Extensive experimental results showed that the proposed method is optimal, efficient, and reliable. It significantly outperforms other baseline methods in terms of the solution quality.

Our experiments were conducted on a desktop computer. Our method can be easily made much faster since CPLEX supports distributed parallel optimization and can naturally utilized multiple computers. Given its optimality, efficiency, and robustness, we believe that the proposed work will become a popular method for extracting actionable knowledge on a large scope of real-world applications such as marketing and clinical prediction.

8. ACKNOWLEDGMENTS

This work was partially supported by CCF-1215302, IIS-1343896, and DBI-1356669 grants from the National Science Foundation of the USA, a Microsoft Research New Faculty Fellowship, a grant from the Barnes-Jewish Hospital Foundation, and a URSA grant from the Washington University.

9. REFERENCES

[1] T. Bailey, Y. Chen, Y. Mao, C. Lu, G. Hackmann, S. T. Micek, K. Heard, K. Faulkner, and M. H. Kollef. A trial of a real-time alert for clinical deterioration in patients hospitalized on general medical wards. *Journal of Hospital Medicine*, 8:236–242, 2013.

[2] A. Bella, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Using negotiable features for prescription problems. *Computing*, 91(2):135–168, 2011.

[3] L. Breiman. Bagging predictors. *Machine Learning*, 26:123–140, 1996.

[4] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[5] L. Cao, D. Luo, and C. Zhang. Knowledge actionability: Satisfying technical and business interestingness. *Int. J. Bus. Intell. Data Min.*, 2(4):496–514, Dec. 2007.

[6] L. Cao, C. Zhang, Q. Yang, D. Bell, M. Viachos, B. Taneri, E. Keogh, P. Yu, N. Zhong, M. Ashrafi, D. Taniar, E. Dubossarsky, and W. Graco. Data driven actionable knowledge discovery. *IEEE Intelligent Systems*, 22(4):77–88, 2007.

[7] L. Cao, Y. Zhao, H. Zhang, D. Luo, and C. Z. nad E. K. Park. Flexible frameworks for actionable knowledge discovery. *IEEE Trans. Knowledge and Data Engineering*, 22(9):1299–1312, 2009.

[8] W. Chen, Y. Chen, Y. Mao, and B. Guo. Density-based logistic regression. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 140–148. ACM, 2013.

[9] W. Chen, Y. Chen, and K. Q. Weinberger. Fast flux discriminant for large-scale sparse nonlinear classification. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 621–630, 2014.

[10] W. Chen, Y. Chen, and K. Q. Weinberger. Filtered search for submodular maximization with controllable approximation bounds. In *Proc. The 18th*

International Conference on Artificial Intelligence and Statistics (AISTATS), 2015.

[11] P. Domingos. Knowledge discovery via multiple models. *Intelligent Data Analysis*, 2(3):187–202, 1998.

[12] J. Du, Y. Hu, C. X. Ling, M. Fan, and M. Liu. Efficient action extraction with many-to-many relationship between actions and features. In *Logic, Rationality, and Interaction*, pages 384–385. Springer, 2011.

[13] Y. Freund and R. E. Schapire. A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

[14] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29:1189–1232.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[16] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, and R. Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.

[17] R. J. Hilderman and H. J. Hamilton. Applying objective interestingness measures in data mining systems. In *Proceedings of the 4th European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 432–439, 2000.

[18] IBM. V12. 1: User manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

[19] M. Karim and R. Rahman. Decision tree and naive bayes algorithm for classification and generation of actionable knowledge for direct marketing. *Journal of Software Engineering and Applications*, 6:196–206, 2013.

[20] A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[21] B. Liu and W. Hsu. Post-analysis of learned rules. In *Proc. AAAI*, pages 828–834, 1996.

[22] B. Liu, W. Hsu, and Y. Ma. Pruning and summarizing the discovered associations. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pages 125–134, 1999.

[23] A. Manindra and T. Thomas. Satisfiability problems. *Technical Report*, 2000.

[24] Y. Mao, W. Chen, Y. Chen, C. Lu, M. Kollef, and T. Bailey. An integrated data mining approach to real-time clinical monitoring and deterioration warning. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2012.

[25] A. Mohan, Z. Chen, and K. Weinberger. Web-search ranking with initialized gradient boosted regression trees. In *Journal of Machine Learning Research, Workshop and Conference Proceedings*, volume 14, pages 77–89, 2011.

[26] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.

[27] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[28] P. D. Turney. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of artificial intelligence research*, pages 369–409, 1995.

[29] P. Viola and M. J. Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[30] Q. Yang, J. Yin, C. Ling, and R. Pan. Extracting actionable knowledge from decision trees. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1):43–56, 2007.

[31] Q. Yang, J. Yin, C. X. Ling, and T. Chen. Postprocessing decision trees to extract actionable knowledge. In *Proc. ICDM*, 2003.

[32] Q. Zhou, W. Chen, S. Song, J. R. Gardner, K. Q. Weinberger, and Y. Chen. A reduction of the elastic net to support vector machines with an application to gpu computing. In *Proc. AAAI Conference on Artificial Intelligence*, 2015.