# Personal Insights for Altering Decisions of Tree-based Ensembles over Time

Naama Boer[*] Daniel Deutch[†] Nave Frost[‡] Tova Milo[§]
Tel Aviv University
{naamaboer[*], navefrost[‡]}@mail.tau.ac.il {danielde[†], milo[§]}@post.tau.ac.il

## ABSTRACT

Machine Learning models are prevalent in critical human-related decision making, such as resume filtering and loan applications. Refused individuals naturally ask what could change the decision, should they reapply. This question is hard for the model owner to answer: first, the model is typically complex and not easily interpretable; second, models may be updated periodically; and last, attributes of the individual seeking approval are apt to change in time. While each of these challenges have been extensively studied in isolation, their conjunction has not.

To this end, we propose a novel framework that allows users to devise a plan of action to individuals in presence of Machine Learning classification, where both the ML model and the user properties are expected to change over time. Our technical solution is currently confined to a particular yet important class of models, namely those of tree-based ensembles (Random Forests, Gradient Boosted trees). In this setting it uniquely combines state-of-the-art solutions for single model interpretation, domain adaptation techniques for predicting future models, and constraint databases to represent and query the space of possible actions. We devise efficient algorithms that leverage these foundations in a novel solution, and experimentally show that they are effective in proposing useful and actionable steps leading to the desired classification.

**PVLDB Reference Format:**
Naama Boer, Daniel Deutch, Nave Frost, Tova Milo. Personal Insights for Altering Decisions of Tree-based Ensembles over Time. *PVLDB*, 13(6): 798-811, 2020.
DOI: https://doi.org/10.14778/3380750.3380752

## 1. INTRODUCTION

Critical decision-making processes, such as credit score assignment, resume filtering and loan approvals, is supported by complex machine learning (ML) classifiers. A significant drawback to the utilization of such models is their opaqueness: ML models are typically complex and hard to understand. Indeed, much research has been devoted recently to the analysis of complex ML models. For example, some solutions [47, 12] identify the input's most influential features, and others [53] produce approximated interpretable models.

In this paper we focus on a novel class of analysis (to our knowledge, see discussion of related work in Section 7), that is (1) *personal*, i.e. aims at explaining a particular classification related to an individual (e.g. a loan applicant) rather than the entire model, and (2) *temporal*, in the sense that it accounts for the evolution of both the user characteristics and the classification models. Combined, these two aspects yield *concrete plans of action for users aspiring to change the decision*, supporting queries such as:

1. **No modification:** What is the closest time point (if any) at which reapplying without modifications to the applicant features is expected to be APPROVED?
2. **Minimal features set:** What is the smallest set of features whose modification can lead to APPROVAL? (also, how should they be modified, and over what timeframe?).
3. **Minimal overall modification:** What is the minimal overall modification (according to some distance measure) that leads to APPROVAL, and when?
4. **Maximal confidence:** Which modifications (and at which time point) would maximize the APPROVAL score [1]?
5. **Dominant feature:** Is there a single feature whose modification leads to APPROVAL in all future time points? (and how should it be modified at each point?)

Our solution is the first, to our knowledge, to support such queries with respect to ML classifications. Our approach is general, but the concrete technical solution is confined to a restricted yet important class of ML models: that of decision trees and their ensembles (including in particular Random Forests and Gradient Boosted Trees).

We next outline our main contributions, referring to the sections that follow for details.

*Model (Section 3).* After reviewing the necessary preliminaries in Section 2, we formally state our problem. At a high level, the idea is that given a set of time-points, at each time point we have (or estimate, see below) a model instance and a profile of the relevant individual (e.g. loan applicant). Then, we consider the "hypothetical" infinite database of all possible modifications to the input profile

[1]ML classification models often output a score reflecting their confidence in the binary classification.

*Proceedings of the VLDB Endowment,* Vol. 13, No. 6
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3380750.3380752

that alter the classification result, at each time point. Intuitively, this is the database of all relevant action items that could be proposed to the individual. Then, the semantics of queries is defined with respect to this database, so that query results correspond to combination of action items into an "action plan": e.g. gradually change a particular feature each year, or wait for five years before re-applying, etc.

*Exact (Inefficient) Solution (Section 4).* We then devise a first solution for the problem, whose architecture is depicted in Figure 3. Recall that at each point we need a model instance and the applicant's individual profile. Both are given as input for at present time. Expected changes in the profile are modeled through a function that we refer to as "temporal update function" capturing the semantics of temporal features. It may reflect e.g. changes in age over years, etc. Future model instances are often harder to predict, but they may be estimated using model adaptation techniques [3]. Then, the main technical novelty is in that we compactly represent the infinite set of classification-altering modifications in each time point in a finite form, using a Constraint Database [40, 52]. This construction is model-dependent and we develop it for tree-based ensembles. Its main idea is to use constraints to jointly capture *regions* of modifications that are of interest. Finally, query evaluation is performed with respect to the constraint database.

*Heuristic Improvement (Section 5).* The constraint database obtained in the exact solution is finite but its size may be exponential in the model's size. For complex models this is intractable, thus we provide a polynomial time inexact alternative. The solution is based on limiting the allowed modifications' size, combined with leveraging constraints that arise from the input database. Using these two types of restrictions, we devise a polynomial time algorithm for database generation at each time point, and then extend these databases to form a time-aware database. The main idea of our solution is to apply a beam search methodology that identifies regions able to alter a single tree classification at each step, ultimately identifying a polynomial number of regions in the locality of the applicant profile.

*Experimental Study (Section 6).* We have experimentally studied our solution in terms of its execution time and outputs quality. We have used three openly available datasets (peer-to-peer loans, home credit and Visa petitions), trained tree-based ensemble models and executed our solution with respect to them. We show that our algorithms generally incur reasonable execution times and are successful in providing insights. In particular our heuristic speeds up the solution by orders of magnitude, allowing its application to large-scale models, without significant deterioration of the quality of found solutions.

## 2. PRELIMINARIES

Next, we overview some basic notions used in our work. In particular, we will briefly review tree-based learning models, the notion of model adaptation, and constraints databases. For the convenience of the reader, Table 1 summarizes the different notations that will be used throughout the paper.

### 2.1 Tree-based Models

There is an active research on the development of learning models. The models differ from one another in their complexity and prediction power. We focus in this work on

**Table 1: Symbols Table**

| Symbol | Meaning |
|---|---|
| $x$ | Input vector |
| $d$ | Dimension of input space |
| $\mathcal{M}$ | Machine learning model instance |
| $\delta$ | Threshold score for positive classification |
| $\mathcal{T}$ | Decision tree instance |
| $\mathcal{E}$ | Tree-based ensemble instance |
| $N$ | Number of trees in the ensemble |
| $L$ | Number of leaves in a tree |
| $T$ | Number of time points |
| $\diamond_t$ | $\diamond$ at time $t$ (e.g. $\mathcal{T}_t$ is a tree at time $t$) |
| $\hat{\diamond}$ | Estimated $\diamond$ (e.g. $\hat{\mathcal{T}}$ is an estimated tree) |
| $\psi$ | Conjunction of linear inequalities |
| $\mathcal{D}$ | Positive Candidates Database (Definition 3.1) |
| $C$ | Additional constraints (Definition 3.7) |
| $\mathcal{R}$ | Positive regions (Definition 4.2) |
| $G$ | Database generator |
| $\varphi$ | Query |
| $\epsilon$ | Algorithm 2 improvement factor |
| $k$ | Algorithm 3 beam size |

tree-based models, which are the most popular non-linear Machine Learning models [32]. Next we will briefly describe this class of models, starting with the simple rather weak model of decision trees, then the more complex (and respectively more powerful) tree-based ensembles. For ease of presentation, we will focus on binary classification, but the framework can be easily generalized to multi-class problems.

*Decision Trees.* A *Decision Tree* $\mathcal{T}$ is a binary tree representing a function $\mathbb{R}^d \mapsto [0, 1]$. Each internal node is labeled by a condition over one of the $d$ coordinates of the input vector, and each of the leaves is associated with a numeric value in the range $[0, 1]$. Each input vector $x \in \mathbb{R}^d$ defines a root-to-leaf path in $\mathcal{T}$: for each internal node, if the condition is satisfied we choose the left branch and otherwise the right branch. We then define the *tree prediction* of $\mathcal{T}$ with respect to $x$, denoted by $\mathcal{T}(x)$, as the value associated with the leaf thus obtained.

Note that the tree prediction is a numeric value in $[0, 1]$; it should be understood as the estimated probability that the input $x$ is classified as $True$. A model is further associated with a *threshold* for accepting/rejecting an input.

EXAMPLE 2.1. *Figure 1 depicts a small decision tree for classifying loan applications. For instance, given the applicant's properties vector $x$, the tree prediction is 0.3 (as depicted by the highlighted path); Assuming a threshold of 0.5 for a positive classification, this applicant will be rejected by the model in Figure 1.*

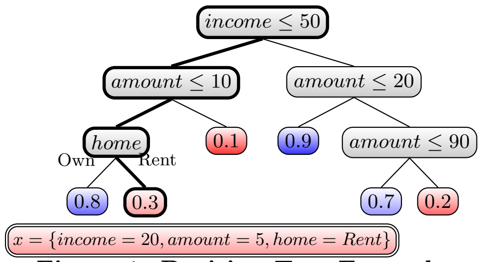


**Figure 1: Decision Tree Example**

The prediction of decision trees can be backtracked by traversing the tree from root to leaf and identifying the relevant conditions of the computation. Thus decision trees are considered to be interpretable models.

We next consider stronger models, which combine multiple decision trees in an ensemble.

***Tree-based Ensemble.*** Ensemble learning methods are used for many ML tasks [16, 8]; They aggregate multiple learning algorithms to gain better predictive capabilities. *Tree-based Ensemble* is a popular learning method which boosts the performance of the weak decision tree learners.

Definition 2.2. *A* Tree-based Ensemble $\mathcal{E}$ *is a set of* $n$ *decision trees* $\{\mathcal{T}_1, \ldots \mathcal{T}_N\}$. *Given an input vector* $x \in \mathbb{R}^d$, *the ensemble prediction is a weighted mean of all individual tree predictions. For simplicity we will assume the ensemble trees have equal weights, i.e.* $\mathcal{E}(x) = \frac{\sum_{i=1}^{N} \mathcal{T}_i(x)}{N}$.

As the number of trees grows, it is harder to understand the reason for the final model prediction: the result depends on the interaction between different trees. This interaction will also be the main challenge for our solution.

There are two popular types of tree-based ensemble learning methods - *Random Forest* [25] and *Gradient Boosted Trees* [17]. For our purposes, the ensemble approach selected is irrelevant, as the final model structure remains the same; our solution is general for tree-based ensembles.

## 2.2 Domain Adaptation

As explained in the introduction, considering only the modification that would alter the decision in the present model is insufficient; in contrast, in order to supply timely relevant explanations it is necessary to understand how the model will behave in the future. The field of *Domain Adaptation* studies the task of recognizing and applying knowledge learned in previous domains (sources), to novel ones (targets). In this context, the evolution of models over time has been investigated, taking the temporal aspect into consideration and studying a time-varying probability distribution from which sample sets at different time points are observed (e.g. [41]).

There are three main approaches for the task of domain adaptation: (1) Instance Weighting: reweight or select instances to reduce the discrepancy between source and target domains. (2) Representation Learning: change the feature representation to better represent shared characteristics between the two domains. (3) Adjusting: integrate some information about the target samples iteratively.

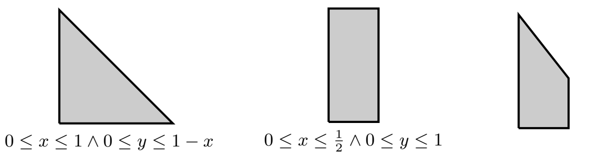
Our problem setting uniquely differs from most settings in which domain adaptation techniques are applied, since there is no access to any data from the target domain, i.e future data. This means approaches (2) and (3) are irrelevant by and large, hence we decided to focus more attention towards instance weighting. However, our solution is independent of the domain adaptation technique being used and we will use domain adaptation as a black box for our solution.

## 2.3 Constraint Databases

Constraint databases [33, 40, 18] are a generalization of relational databases, optimized for storing and querying possibly infinite-sized data points represented by a finite set of constraints. In general, constraints are expressed by quantifier-free first-order formulas over a fixed vocabulary.

Example 2.3. *Consider a constraint database with two relations R and S, where R contains the record* $\psi_1(x, y)$ *(depicted in Figure 2a), and S contains the record* $\psi_2(x, y)$ *(depicted in Figure 2b). Figure 2c depicts the intersection of both relations, i.e. the result of the following query:*

$$\varphi(x, y) := R(x, y) \wedge S(x, y) = 0 \le x \le \frac{1}{2} \wedge 0 \le y \le 1 - x.$$


$0 \le x \le 1 \wedge 0 \le y \le 1 - x$
$0 \le x \le \frac{1}{2} \wedge 0 \le y \le 1$

(a) $\psi_1(x, y)$ (b) $\psi_2(x, y)$ (c) $\varphi(x, y)$

**Figure 2: Constraint Database Example**

Constraints databases have been extensively studied; There has been much work on studying the expressive power and complexity of such systems (e.g. [33, 34, 2]), and on developing query evaluation strategies for queries over such DBs [35, 56, 24, 5].

## 3. PROBLEM DEFINITION

In this section we define the problem of deriving personal insights for altering model decisions over time. We start by defining the *Positive Candidates Database*, a "hypothetical" infinite database containing all vectors positively classified by the model.

Definition 3.1. *Given a model* $\mathcal{M}$, *the* Positive Candidates Database $\mathcal{D}_{\mathcal{M}}$ *is a hypothetical database with d attributes, each corresponding to a feature of the input space. The tuples of* $\mathcal{D}_{\mathcal{M}}$ *are all vectors positively classified by* $\mathcal{M}$, *i.e.* $\mathcal{D}_{\mathcal{M}} = \{x \in \mathbb{R}^d \mid \mathcal{M}(x) > \delta\}$. *For brevity, we will refer to* $\mathcal{D}_{\mathcal{M}}$ *as* $\mathcal{D}$ *when* $\mathcal{M}$ *is clear from the context, and to* $\mathcal{D}_{\mathcal{M}_t}$ *as* $\mathcal{D}_t$ *when a models sequence is clear from context.*

We extend our setting to be time-aware, in the sense that we don't consider a single model and input but rather a sequence thereof. We support queries over their positive candidate databases, in a quite expressive query language: First Order Logic augmented with a distance function over candidate vectors, which we denote $FO^{dist}$. We are now ready to define the *Decision Altering Problem.*

Problem 3.2 (Decision Altering Problem). *Given a sequence of tree-based models* $\mathcal{M}_0, \ldots, \mathcal{M}_T$, *input vectors* $x_0, \ldots, x_T$, *and a query* $\varphi$ *in* $FO^{dist}$, *return the result of* $\varphi$ *with respect to* $\mathcal{D}_0, \ldots, \mathcal{D}_T$.

Note that $\mathcal{D}_0, \ldots, \mathcal{D}_T$ may each be of infinite size and so may not be materialized in general.

Example 3.3. *We start by exemplifying the problem for a single time point. Recall the decision tree and rejected applicant from Example 2.1, she may ask what is the minimal modification she needs to make (in terms of* $l_2$ *distance) assuming she is not willing to become a home owner; It can be formulated as the following query:*

$$\begin{aligned}\varphi(x) = \{x_1 \in \mathcal{D} \mid & x_1[home] = Rent \wedge \forall x_2 \in \mathcal{D}. \\ & x_2[home] = Rent \rightarrow dist(x, x_1) \le dist(x, x_2)\}\end{aligned}$$

*where x is the vector representation of the rejected applicant depicted in Figure 1. Evaluating the above query will reveal that the minimal modification required for loan acceptance (while living in rent) is an increase of income to 50.*

Next, we will exemplify the problem over multiple time points.

Example 3.4. *Recall the dominant feature query presented in the introduction, asking whether an attribute a is a* *feature whose sole modification (possibly differently at each time point) can lead to approval in all the future time points; this may be captured by the following formal query:*

$$\begin{aligned}\varphi(x_0,\ldots,x_T) =& \exists x'_0 \in \mathcal{D}_0,\ldots,x'_T \in \mathcal{D}_T. \\ & \varphi_a(x_0,x'_0) \wedge \ldots \wedge \varphi_a(x_T,x'_T) \\ \varphi_a(x_1,x_2) =& (x_1[0]=x_2[0]) \wedge \ldots \wedge (x_1[a-1]=x_2[a-1]) \wedge \\ & (x_1[a+1]=x_2[a+1]) \wedge \ldots \wedge (x_1[d]=x_2[d]).\end{aligned}$$

In what follows, we discuss how we obtain the individual representation of the user in future time points. In the next section we will discuss our solution for the generation of the classifier models for future time points and the positive candidates database creation.

*Handling Temporal Attributes.* Recall that at the time of the query we do not know how user properties will change. Some of the attributes are known to change over time (e.g. age increases), while others are known to remain constant (e.g. gender does not change), or may change in an unpredictable manner. Our framework allows both the user and the administrator to provide guidelines on changes to the user properties over time. The administrator will define global guidelines which will hold to all the users in the system, while the users are able to provide guidelines regards to their own plans. We formulate those guidelines as a *Temporal Update Function* which defines the set of $T$ input vectors $\hat{x_0},\ldots,\hat{x_T}$ that will be used for the database generation.

Definition 3.5. *A* Temporal Update Function *operates on the original input vector $x$:*

$$f \colon \mathbb{R}^d \times \{1,\ldots,T\} \mapsto \mathbb{R}^d$$

*For each future time point, $f$ outputs a representation of the applicant in that time point. For brevity $\{f(x,t)\}_{t=1}^{T}$ will be denoted as $\hat{x_1},\ldots,\hat{x_T}$.*

Example 3.6. *To continue with our running example, we can expect the system administrator to specify that $f(x,t)[age] = x[age]+t$ which will result in $\hat{x_t}[age] = x_0[age]+t$. For simplicity, we assume that every feature absent from the guidelines formulated by either the administrator or applicant will stay constant. E.g. in case there were no guidelines on the income feature, then $\forall i \in \{1,\ldots T\} \colon f(x,i)[income] = x[income]$.*

We assume in our framework that the temporal update function is given as input, and leave a study of its (semi-) automatic inference for future work.

*Additional Constraints.* There are numerous cases in which some of the model's positive regions are irrelevant, or only partially relevant for the generation of positive candidates database. For example, a person's age can not decrease, a user may prefer not to change her address, etc. Thus, our framework enables specification of additional constraints by the system administrator and the applicant alike. Those constraints are conjuncted to the query. Constraints may refer to a single point in time or all of them.

Definition 3.7. Personal and Domain Constraints *$\{C\}_{t=0}^{T}$ is a set of formulas; each formula consists of conjunctions and disjunctions of boolean conditions over the d input features. The constraints are added as conjuncts to the query in Problem 3.2.*

Example 3.8. *Personal and Domain Constraints may help specifying what type of modifications are of interest. For example, the system administrator may state that changing the applicant's gender is irrelevant, while the user may specify that her income may increase by up to 10% each year; These will result in the following constraints:*

$$C_t = (x_t[gender] = x[gender]) \wedge (x_t[income] \leq 1.1^t \cdot x[income]).$$

## 4. EXACT SOLUTION

We introduce our temporal decision altering framework; its architecture is presented in Figure 3. The architecture consists of four main components. The first component is *Future Models Estimation*: since $M_1,\ldots,M_T$ are unknown in advance we will need to estimate them; the estimated models will be generated through domain adaptation over the historic training data. Next, the system will initiate the *Database Generation* phase; in this phase it will combine the estimated future models and applicant representations, and generate $\mathcal{D}_1,\ldots,\mathcal{D}_T$ which will populate a constraint database with the positive candidates. To focus on relevant candidates, we give both the user and system administrator the option of defining additional constraints on top of the generated ones; this restricts queries to only return relevant and actionable candidates. By pushing the administrator and the user constraints into the database generation phase, we enable re-usage of the generated database for multiple queries. Then, in the *Query Evaluation* phase, the query will be evaluated over the generated database, allowing the user to obtain insights for achieving the desired classification. Note that Figure 3 depicts an additional component for Time Expansion. For now it can be ignored; it will be revisited in Section 5.

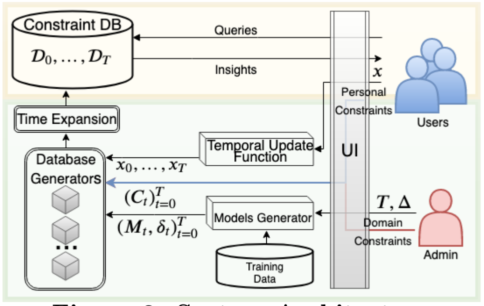


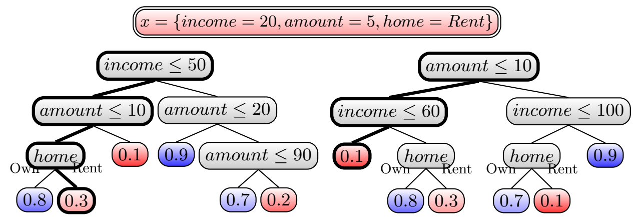
**Figure 3: System Architecture**

### 4.1 Future Models Estimation

Generating estimated future models requires labeled training data with timestamps, such that each point $x \in \mathbb{R}^d$ is associated with a label $y \in \{0,1\}$. Additionally, two parameters, $T$ and $\Delta$, determine the timespan handled by the system: $T$ is the number of time points considered and $\Delta$ sets the interval length between consecutive points. As noted earlier, our system may use any suitable domain adaptation technique as a black box.

Our goal is to create a sequence of pairs $(\hat{M_t},\delta_t)_{t=0}^{T}$, where $\hat{M_t}$ is the expected approximated model at future time $t$, and $\delta_t$ is its threshold. Concretely, our implementation is inspired by methods described in [41, 38, 39], that use the training data and past models to learn the time variations of the labels distribution, thus overcoming the lack of available data from the target (future) distribution.

In the implementation of this framework, we solved the problem under the class imbalance assumption [29, 30], stating that given some class label, the conditional distribution of $x$ is the same in the past and future distributions. Then we assigned instance-dependent weights to the loss function before training the estimated future models. Note that this part of the candidates generation process is performed only once and is independent of any specific user. In Section 6, we give a detailed description of the Models Estimation process used for evaluation.



(a) Present Decision Tree (b) Future Decision Tree

| Name | Time | Region |
|---|---|---|
| $\psi_1$ | 0 (Present) | $income \leq 50 \wedge amount \leq 10 \wedge home = Own$ |
| $\psi_2$ | 0 (Present) | $income > 50 \wedge amount \leq 90$ |
| $\psi_3$ | 1 (Future) | $income > 60 \wedge amount \leq 10 \wedge home = Own$ |
| $\psi_4$ | 1 (Future) | $income > 100 \wedge amount > 10$ |
| $\psi_5$ | 1 (Future) | $income \leq 100 \wedge amount > 10 \wedge home = Own$ |

(c) Positive Candidates Databases

**Figure 4: Decision tree adaptation with relevant databases**

EXAMPLE 4.1. *Reconsider the decision tree from Figure 1, appearing as the "present" tree in Figure 4a. Assume that the training data indicates an increase in incomes and in the rate of home owners; Thus the present model becomes obsolete, and needs to be modified. A re-trained decision tree for time "present + $\Delta$" appears in Figure 4b.*

## 4.2 Database Generation

The core of the temporal candidates generation phase is a set of candidates generator components. Each generator $G_t$ is responsible for outputting $\hat{\mathcal{D}}_t$, a positive candidate database fitted for the corresponding time point $t \in [T]$, that is obtained based on the predicted model $\hat{M}_t$ and the expected input representation $\hat{x_t}$. The generators are independent of each other, thus they can be executed in parallel. Consequently, from now on we will describe an algorithm for a single time point. The results (set of candidates per time point) are then stored in a *candidates* table, ready to be queried by the user. Since the number of candidates may be infinite, explicit generation of all candidates is impossible. Hence, we store a representation of the different *Positive Regions* of $\hat{M}_t$ in a constraint database which will be queried in the following step.

DEFINITION 4.2. *Given a model $\mathcal{M}$, we say that a set $\mathcal{R}_\mathcal{M} = \{\psi_1, \ldots, \psi_m\}$ where $\psi_i$ is a conjunction of boolean conditions over the $d$ input features, is a set of* Positive Regions *if it satisfies that $\mathcal{M}(x) > \delta$ iff $\exists \psi_i \in \mathcal{R}_\mathcal{M} \colon \psi_i(x) = True$.*

EXAMPLE 4.3. *Recall the trees from Example 4.1. The relevant Positive Regions are depicted in Figure 4c. Regions $\psi_1$ and $\psi_2$ were generated according to the model in Figure 4a; $\psi_3$, $\psi_4$, and $\psi_5$ according to the model in Figure 4b.*

The process of computing such a set $\mathcal{R}_\mathcal{M}$ is model dependent. As described in Section 2, a tree ensemble $\mathcal{E}$ consists of $N$ decision trees and it classifies a point based on a weighted sum of their scores. Next, we show that for tree-based ensembles the size of the minimal $\mathcal{R}_\mathcal{E}$ may grow exponentially in the number of ensemble trees. Given a tree-based ensemble $\mathcal{E}$, Algorithm 1 generates a constraint database $\mathcal{D}_\mathcal{E}$, such that $\mathcal{E}(x) > \delta$ iff $x$ satisfies exactly one of the constraints in $\mathcal{D}_\mathcal{E}$. The algorithm iterates over all ensemble leaves combinations that lead to positive label (Lines 2, 3). For each positive combination a conjunction of all the conditions is calculated (while loop in Line 5). Eventually, the positive regions are stored in a constraint database (Line 10). After the database generation, one can add an optimization "cleanup" step that deletes constraints that are unsatisfiable; though this is not essential for correctness, nor it affects the worst-case complexity bounds discussed next.

**Algorithm 1:** Database Generation for Tree Ensembles

```
input  : Tree-based Ensemble $\mathcal{E}$
output: Constraint database of all positive regions
1  $\mathcal{D} = \{\}$;
2  foreach $\langle l_1, \ldots l_N \rangle \in \mathcal{E}.\mathcal{T}_1.leaves \times \ldots \times \mathcal{E}.\mathcal{T}_N.leaves$ do
3      if $\frac{1}{N}\sum_{i=1}^{N} l_i.value() \geq \mathcal{E}.\delta$ then
4          $\mathcal{C} = \{\}$;
5          foreach $l \in \langle l_1, \ldots, l_n \rangle$ do
6              $parent = l.parent()$;
7              while $parent \neq Null$ do
8                  $\mathcal{C}.add(parent.condition())$;
9                  $parent = parent.parent()$;
10         $\mathcal{D}.add(\bigwedge_{c \in \mathcal{C}} c)$;
11 return $\mathcal{D}$;
```

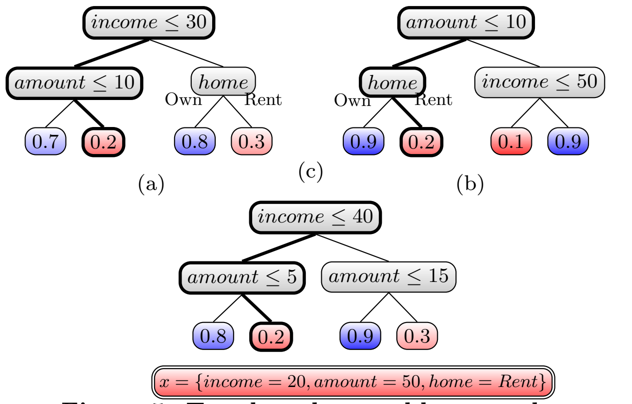


**Figure 5: Tree-based ensemble example.**

EXAMPLE 4.4. *Consider the tree ensemble depicted in Figure 5 with a threshold of 0.5. Let us consider the iteration step where the current leaves are all of the left-most leaves in their respective trees. This combination leads to an overall score of $\frac{0.7+0.9+0.8}{3} = 0.8 \geq \delta$, and thus a positive region is found; the conjunction of these 3 paths yields the following region, which will be stored in the database:*

$$(x[income] \leq 30) \wedge (x[amount] \leq 5) \wedge (x[home] = Own).$$

*Overall, for the ensemble depicted in Figure 5, with 3 trees having 4 leaves each, we obtain a database containing 28 positive regions.*

**Table 2: Formal definition in $FO^{dist}$ of queries presented in the introduction.**

| | Query |
|---|---|
| Q1 | $\{t_1 \mid x \in \mathcal{D}_{t_1} \wedge \forall t_2.\, t_2 < t_1 \rightarrow x \notin \mathcal{D}_{t_2}\}$ |
| Q2 | $\{(x_1, t) \mid x_1 \in \mathcal{D}_t \wedge\ \forall t_2 \forall x_2 \in \mathcal{D}_{t_2}.\, d_{l_0}(x_{t_1}, x_1) \leq d_{l_0}(x_{t_2}, x_2)\}$ |
| Q3 | $\{(x_1, t) \mid x_1 \in \mathcal{D}_t \wedge\ \forall t_2 \forall x_2 \in \mathcal{D}_{t_2}.\, d_{l_2}(x_{t_1}, x_1) \leq d_{l_2}(x_{t_2}, x_2)\}$ |
| Q4 | $\{(x_1, t) \mid x_1 \in \mathcal{D}_t \wedge\ \forall t_2 \forall x_2 \in \mathcal{D}_{t_2}.\, \mathcal{M}_t(x_1) \leq \mathcal{M}_{t_2}(x_2)\}$ |
| Q5 | `Dominant feature (See Example 3.4)` |

In general, the correctness proof of Algorithm 1 (omitted here) implies the following:

PROPOSITION 4.5 (UPPER BOUND). *All positive regions of a tree-based ensemble model can be stored in a constraint database $\mathcal{D}$ of size $O(L^N)$, where $L$ is the maximal number of leaves of a tree in the ensemble.*

Can this exponential blowup be avoided? Unless $P = NP$, the answer is negative (proof omitted for space constraints).

PROPOSITION 4.6 (LOWER BOUND). *Assuming $P \neq NP$, there exists a tree ensemble $\mathcal{E}$ with $N$ trees, whose positive regions set $\mathcal{R}_{\mathcal{E}}$ can not be stored in a constraint database $\mathcal{D}_{\mathcal{E}}$ whose size is polynomial in $N$.*

### 4.3 Query Evaluation

We have explained so far how the constraints database that represents the relevant positive candidates is generated. Next, users query the generated database using an engine such as [9, 22] in order to obtain insights.

EXAMPLE 4.7. *Recall the decision trees and databases over two time points presented in Figure 4, and let us consider the applicant depicted in the Figure (where the relevant root-to-leaf paths are highlighted). According to both models the application will be denied, receiving* 0.3 *by the present model and* 0.1 *by the future model. Table 2 shows multiple example queries. For example, assume the applicant wishes to check if "income" is a dominant feature (i.e. Q*5*). Evaluating Q*5 *over the above database will reveal that "income" is not a dominant feature. At the present year the applicant may increase her income to be above* \$50 *in order to get positive classification ($\psi_2$), however in the second year it is not sufficient to solely increase the income: the applicant would also be required to be a home owner ($\psi_3$) or to raise the loan amount ($\psi_4$).*

Complexity of query evaluation over constraints databases has been extensively studied through the years [33, 35, 56]; Our database contains conjunctions of inequality conditions, i.e. linear constraints. Thus, the evaluation of constraint queries can be done in polynomial data complexity [23].

## 5. PARTIAL REGIONS

In Section 4.2 we observed an exponential blowup of the constraint database size. We next propose a heuristic alternative to the exact *Database Generation* component, that does not generate all positive regions; instead, the approach focuses on the area in proximity of the input vector, and furthermore leverages additional constraints (i.e. administrator and user constraints) to avoid the insertion of irrelevant regions into the database. Note that, in contrast to the exact database generation, where the DB is generated once and can then serve multiple users, our heuristic database is generated per user. However, as illustrated in our experiments, the brief generation time makes up for theses multiple constructions (whereas the exact generation is non-feasible for practical size models).

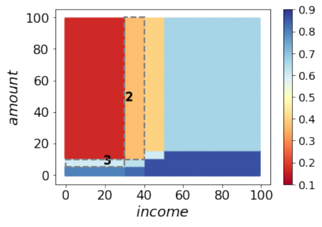
Our solution is then comprised of three algorithms, as follows. Algorithms 2 and 3 operate on a single time point, and can be executed in parallel over the $T$ time-points; Algorithm 4 aggregates the $T$ partial databases, and extends them by checking if a region that was obtained at time $t$ can be applied to the remaining time points.

First, Algorithm 2 generates only a pruned set of all positive regions, focusing only on those that alter the the decision of (at least) a single tree in the ensemble. It iterates over all the regions induced by a single ensemble tree (lines 3,4) and for each it finds the closest point in it to the input vector (line 5). This point is called a *seed*. For each seed, the algorithm calculates its score, i.e. its acceptance probability by the trees in the ensemble (line 6), and in case the seed score is substantially higher than the input vector score, the algorithm will compute the ensemble-region in which the seed lies (lines 7–8), i.e. for each ensemble tree identify the seed's root-to-leaf path and compute their conjunction. Next, the obtained region will be combined with the additional constraints (line 9), and in case the resulting region is non-empty it will be stored along with the seed and its score (lines 10–11). Eventually the algorithm will return all stored seeds and corresponding regions.

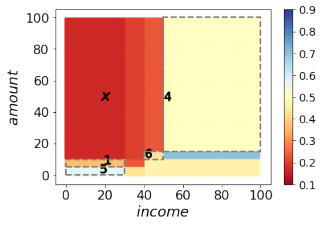
**Algorithm 2:** Seeds Generation

```
   input : Tree-based ensemble $\mathcal{E}$
           Input vector $x$
           Required Constraints $C$
           Metric $dist$
   output: Seeds along with their score and accepting region
1  $\mathcal{S} = \{\}$;
2  $score = \mathcal{E}.score(x)$;
3  foreach $\mathcal{T} \in \mathcal{E}.trees$ do
4      foreach $\mathcal{R} \in \mathcal{T}.regions$ do
5          $seed = \arg\min_{x' \in \mathcal{R}} dist(x, x')$;
6          $seedScore = \mathcal{E}.score(seed)$;
7          if $seedScore > score + \epsilon$ then
8              $AR = \mathcal{E}$ Regions accepting $seed$;
9              $\psi = AR.intesect() \wedge C$;
10             if $\psi \neq contradiction$ then
11                 $\mathcal{S}.add(\langle seed, seedScore, \psi \rangle))$;
12 return $\mathcal{S}$;
```

EXAMPLE 5.1. *Recall the tree-based ensemble and applicant depicted in Figure 5; the model's decision boundaries are presented in Figure 6. The applicant input vector $x$ is also presented in Figure 6b, and is outside all accepting regions; its model score is* 0.167. *Further assume the user defined a personal constraint stating she is not interested in becoming a home owner; the relevant positive regions are shown in Figure 6b. For each of the ensemble trees, Algorithm 2 will iterate over its positive regions, and will return as seeds the closest points in each region. The regions identified by Algorithm 2 are presented in Figure 6c and are marked with numbers in Figures 6a and 6b. The first two seeds were derived from tree (a), the third and fourth were derived from tree (b) and the last two seeds were derived from tree (c); i refers to income, a is amount, h is home, and dist is the $l_2$ distance between $x$ and the seed (where home is treated as a binary feature). As seeds 2 and 3 violate the constraint they will not be returned by the Algorithm.*



(a) *home* = *Own*



(b) *home* = *Rent*

| id | $i$ | $a$ | $h$ | score | dist | $\psi$ |
|---|---|---|---|---|---|---|
| 1 | 20 | 10 | Rent | 0.367 | 40 | $(i \leq 30) \wedge (5 < a \leq 10) \wedge (h = Rent)$ |
| ~~2~~ | ~~31~~ | ~~50~~ | ~~Own~~ | ~~0.367~~ | ~~11.05~~ | ~~$(30 < i \leq 40) \wedge (10 < a) \wedge (h = Own)$~~ |
| ~~3~~ | ~~20~~ | ~~10~~ | ~~Own~~ | ~~0.6~~ | ~~40.01~~ | ~~$(i \leq 30) \wedge (5 < a \leq 10) \wedge (h = Own)$~~ |
| 4 | 51 | 50 | Rent | 0.5 | 31 | $(50 < i) \wedge (15 < a) \wedge (h = Rent)$ |
| 5 | 20 | 5 | Rent | 0.567 | 45 | $(i \leq 30) \wedge (a \leq 5) \wedge (h = Rent)$ |
| 6 | 41 | 15 | Rent | 0.433 | 40.82 | $(40 < i \leq 50) \wedge (10 < a \leq 15) \wedge (h = Rent)$ |

(c) Output of Algorithm 2

**Figure 6: Tree-based Ensemble Decision Boundary**

Algorithm 2 considers only regions that require a modification of a single ensemble tree, and thus it may miss some positive regions. Thus, we introduce Algorithm 3 to refine its output and identify additional positive regions in the locality of the regions computed by Algorithm 2.

Algorithm 3 begins with the input vector as its only seed (line 1). In each iteration, seeds that are positively classified by the ensemble (i.e their score exceeds the model threshold, see line 8), are stored in the positive regions database (line 9). Otherwise, for each seed scored below the model threshold, Algorithm 2 is called with the seed as its input, generating a new set of seeds (line 11). In line 12 the algorithm will apply pruning of the obtained seeds based on their distance from $x$ and will continue the process until the list of seeds will be fully exploited, or until the database is no longer expanded.

**Algorithm 3:** Partial Database Generation for Trees Ensemble

```
input : Tree-based ensemble $\mathcal{E}$
        Input vector $x$
        Required Constraints $C$
        Integer $k$
        Metric $dist$
output: Partial database with respect to $x$ and $C$
1  $\mathcal{S} = \{\langle x, \mathcal{E}.score(x), x \text{ accepting regions}\rangle\}$;
2  $\mathcal{D} = \{\}$;
3  $s = -\infty$;
4  while $\mathcal{S} \neq \emptyset$ and $|\mathcal{D}| > s$ do
5      $s = |\mathcal{D}|$;
6      $\mathcal{S}_{\text{next}} = \{\}$;
7      foreach $seed \in \mathcal{S}$ do
8          if $seed.score \geq \mathcal{E}.\delta$ then
9              $\mathcal{D}.add(seed.region)$;
10         else
11             $\mathcal{S}_{\text{next}}.add(\text{SEEDS}(\mathcal{E}, seed.vector, C, dist))$;
12     $\mathcal{S} = \text{PRUNE}(\mathcal{S}_{\text{next}}, x, k, d)$;
13 return $\mathcal{D}$;
```

EXAMPLE 5.2. *Observe that out of the regions retrieved in Example 5.1 the fifth region is positive (seed number 5 is the only one with score* $> 0.5$*), hence it would be stored in the database. Out of the remaining valid seeds the forth one is closest to the input vector (has minimal distance); hence seed number 4 will remain after the pruning step. In the second iteration Algorithm 2 will be executed with seed 4 as its input vector. This execution will return the positive region* $(50 < i) \wedge (10 < a \leq 15) \wedge (h = Rent)$ *which will be stored in the database as well. After the second iteration all positive regions were identified by the algorithm.*

*Discussion.* Due to the complexity of tree-based ensembles, it is infeasible to build an exact database of positive candidates. The proposed solution is a heuristic aiming to generate a constraint database in the locality of the applicant's representation. To do so it applies beam search over the input space – in each step Algorithm 2 explore regions in the locality of its input vector, and Algorithm 3 prunes the list of identified regions. The parameters $k$ and $\epsilon$ control the operation of the solution, and balance between the execution time and the size of the generated database. The improvement factor, $\epsilon$, defines a lower bound for the confidence increase at each iteration. Small $\epsilon$ values will entail that more regions would be considered, and thus both the database size and execution time will increase, as opposed to large $\epsilon$ values which will cause the solution to overlook regions that are not significantly preferable over the previous regions. The beam size, $k$, defines how many seeds will remain after each iteration. Small $k$ values will result in massive pruning after each iteration, guiding the solution to focus on a small number of potential regions, whereas larger $k$ values will enable the solution to explore more regions, increasing the execution time. There are no theoretical guarantees on the quality of the solution; in extreme cases the search process may reach a negative region which is a local maximum, and as a result, it will return an empty database. In Section 6 we empirically analyze the solution quality, and show that for real-world data this rarely occurs.

*Time Sensitive Expansion.* A pitfall of our heuristic thus far is that it is executed separately for each time point. Consequently, queries involving intersections of regions across several time points (such as the "dominant feature" query) may return a sub-optimal answer. To illustrate, consider the following example.

To address this issue, we add a *Time Expansion* component which will consume the results of the parallel execution of $G_1, \ldots, G_T$, evaluate each of the obtained regions across all the time-points, and extend the database accordingly.

Algorithm 4 iterates over all of the obtained regions (line 2), and evaluates their interaction across all other time points. Each region will be divided into a grid across all its features (line 3), and each vector over the grid will be evaluated over the different $T$ models (line 4), in case some grid point found to be accepted by any of the models (line 5) we will add the relevant region to the appropriate database (lines 6–9). Eventually, the extended databases will be returned (line 10).

*Complexity.* Algorithm 2 iterates over $N$ ensemble trees, and each of their accepting regions, which is proportional to the number of tree leaves (denoted by $L$). For each accepting region, the algorithm first identifies the *seed* vector (can be done in $O(d)$ by applying minimal modifications to each feature that satisfies the region constraints). Once the *seed* was identified, the algorithm obtains $AR$ which are all of the regions that accept the *seed* vector, this can be achieved by a second iteration over the $N$ trees and obtain a prediction after at most $h = \log L$ steps. Thus the overall complexity of Algorithm 2 is $O(N^2 dL \log L)$.

**Algorithm 4:** Time Expansion

```
input  : Set of all saved positive regions $\mathcal{R}s$
         Expected models for all time points $(\hat{M}_t, \delta_t)_{t=0}^{T}$
output: Expansion of positive regions to all time points
1  $\mathcal{D}_1, \ldots, \mathcal{D}_T = \{\}$;
2  foreach $\mathcal{R} \in \mathcal{R}s$ do
3      foreach $x \in \mathcal{R}.grid()$ do
4          foreach $t \in [T]$ do
5              if $\hat{\mathcal{M}}_t(x) \geq \delta_t$ then
6                  $AR = \hat{\mathcal{M}}_t$ Regions accepting $x$;
7                  $\psi = AR.intesect() \wedge C$;
8                  if $\psi \neq contradiction$ then
9                      $\mathcal{D}_t.add(\psi)$;
10 return $\mathcal{D}_1, \ldots, \mathcal{D}_T$;
```

At each iteration of Algorithm 3 it holds at most $k$ seeds, and for each seed there is at most one call to Algorithm 2. Note that at each iteration of Algorithm 3, either we do not find a new set of seeds, or the scores of the newly retrieved seeds are greater than the previous seed score plus some constant $\epsilon$, thus there can be at most $\frac{1}{\epsilon}$ iterations. Overall the running time of Algorithm 3 is bounded by $O(\frac{k}{\epsilon}N^2 dL \log L)$. Finally, the positive regions obtained for each time-point are being tested on all other time-points by Algorithm 4, hence our solution's overall complexity is $O(T^2 \frac{k}{\epsilon}N^2 dL \log L)$. Note that unlike the procedure in Section 4, the complexity is polynomial in the ensemble size, and thus feasible even for large models.

## 6. EXPERIMENTS

We have implemented a system prototype [6] and conducted an experimental study to evaluate our proposed algorithms in terms of execution times and output quality. We start by describing the experimental settings and datasets, then give the results.

### 6.1 Experimental settings

*Datasets.* We have used three publicly accessible datasets: (1) the *Lending Club* (*LC*) dataset [43] describes loans issued by the Lending Club peer-to-peer lending company in the years 2007-2018, where the classification variable indicates whether the loan was fully paid or not; (2) the *Home Credit* (*HC*) dataset [26] describing loan applications made to the Home Credit company, where the classification variable indicates whether the loan was fully paid or not; (3) a dataset of *H-1B Visa Petitions* (*H1-B*) from the years 2011-2016, where the classification variable indicates whether the visa application was certified or not. *HC* does not include timestamps, yet many of its features are temporal (e.g *days_employed*, *days_birth*). We thus assumed that for this dataset the model stays fixed over time; changes still apply to applicant profiles. We have performed standard preprocessing (removing tuples with inconclusive classification variables and features with too much missing data, scaling etc.), resulting in 892544, 307511 and 690531 records and 25,25 and 7 features in *LC*, *HC* and *H1-B* correspondingly.

The experimental setting also includes a base configuration for a Random Forest classification model for each of the datasets, to be used throughout the evaluation process. We used grid search to select the best performing configuration of parameters *ntrees* (number of trees) and *max_depth* (maximal tree depth), in terms of the model performance on the validation set and its size and complexity. The selected configurations are (1) *max_depth* = 6, *ntrees* = 20 (*LC*), (2) *max_depth* = 6, *ntrees* = 15 (*HC*), and (3) *max_depth* = 9, *ntrees* = 5 (*H1-B*).

In all the conducted experiments we used a random sample of 100 negatively-classified applicants, i.e ones the evaluated model classifies as high-risk for defaulting on their loans or unworthy of a visa.

For applicants rejected at time $t_0$, unless stated otherwise, experiments were performed on single classification models trained on data gathered during that time point. Subsections 6.3 and 6.4 each contain an experiment that examines the effects of incorporating multiple time points, with a corresponding approximated classification model for each. Algorithm 2 gets a distance metric as input. We executed all experiments with two metrics $l_0$ and $l_2$.

*Queries.* Throughout this Section we evaluate our solution's performance on the queries presented in the introduction, that are later formalized in Section 6.4.

*Baselines.* To the best of our knowledge, ours is the first work to formulate Problem 3.2, and so no previous work could be used as a baseline. Algorithm 1 is used as a baseline for database generation, for small models (having roughly *ntrees* $\cdot$ *max_depth* $\sim$ 25. Due to the exponential blowup it is infeasible for large models). We have also designed a fairly simple heuristic baseline for partial database generation, inspired by $A^*$ algorithm. It is detailed next.

***A* baseline.*** We implemented the well known path-finding search algorithm for identifying decision-altering regions. We briefly describe the algorithm bellow. Full details can be found in the technical report [7]. For each feature $i \in [d]$ let $splits_i$ be the set of split values extracted from traversing the ensemble's trees. For an applicant $x$, We construct a graph $G$ who's nodes are $\{z \mid \forall i. z_i \in splits_i \cup \{x_i\}\}$, and edges connect nodes who differ in exactly one attribute. *A** prioritizes nodes according to a function $f(\cdot) = g(\cdot) + h(\cdot)$. $g$ is the distance from the temporal input $x_t$, while $h$ is a heuristic measuring the difference between the node's prediction score and the threshold model score $\delta_t$. The algorithm can be executed with different choices of a distance function $g$ each time, taking the union of results.

As a post-processing step, we convert every modification discovered by the execution of *A** to its corresponding positive region (by executing lines 8-9 of Algorithm 2).

*Hyper-parameter Tuning.* Last, we need to set the beam width $k$ and improvement factor $\epsilon$ used in our solution. To tune the value of $k$ we have experimented with different $k$ values from 2 to 10, and chose to fix $k$ to be 10,6,6 for *H1-B*, *LC* and *HC* respectively, since they produced the largest database with a manageable execution time overhead (on average 2.52, 22.72, 12.35 seconds for *H1-B*, *LC* and *HC* respectively). As the execution times were reasonable we fixed $\epsilon$ to be 0 for all datasets, thus ensuring the the consideration of all potential improving regions. These $k$ and $\epsilon$ values will be used for the remainder of this Section. We start by examining the execution time of the partial database generation (6.2). Then study the quality of the generated database (6.3) and queries answers over it (6.4). Eventually, we present insights generated for real-world users (6.5).

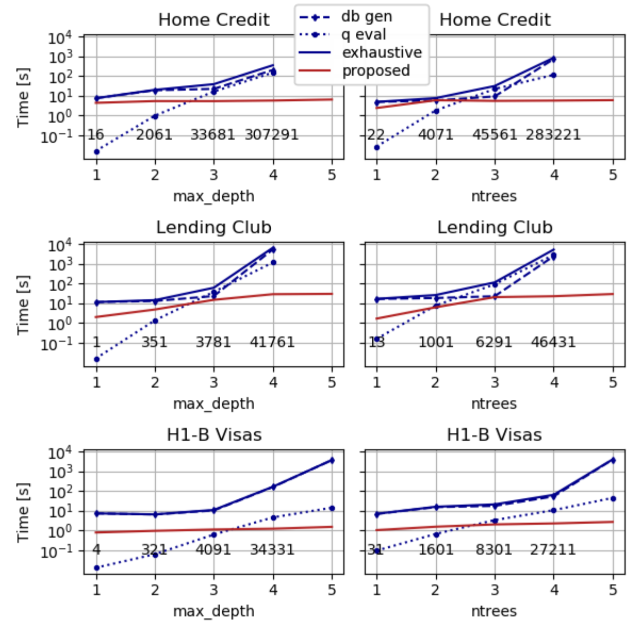


**Figure 7: Exploring the affect of varying the models size on the database generation runtimes, comparing the exhaustive solution to our proposed approach. The fixed parameter (*ntrees*/*max_depth*) in each setting is set to 5.**

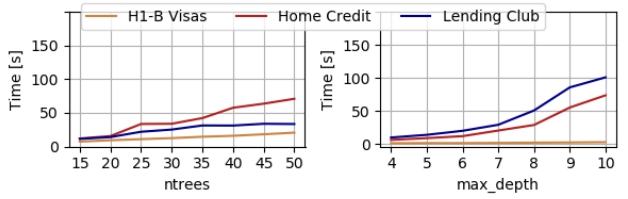


**Figure 8: Exploring the impact of the model's size on execution times. The fixed parameter in each chart is set to its value in the base configuration detailed in Section 6.1.**

## 6.2 Execution Time

In this part of the evaluation we study the execution times of our framework in different settings.

*Comparison with the exact solution.* We compare the execution time of our partial database generation solution (Algorithms 2 - 4) to the exact costly solution (Algorithm 1), for small models where the latter is still feasible. Figure 7 depicts the execution time of our solution in the red plot and the exact alternative in the solid blue plot, as a function of the model depth and number of trees. The size of the complete database generated by Algorithm 1 is presented in each figure; the size of the partial database in all settings did not exceed 10. Observe the moderate growth of the execution time for our heuristic solution, in contrast to the exponential growth of the exact one. We also present a breakdown of the exact solution to the database generation step (blue dashed line) and query evaluation (blue dotted line). Both grow exponentially, indicating that a perceived solution of running exact database generation once and repeatedly querying it would still be infeasible. In contrast, the execution time of the query evaluation step in our solution is negligible: it never exceeded 2 seconds in our experiments, and on average took only 0.3154 seconds.

*Execution Times for Practical-Size Models.* We then consider larger models, where the exact solution is no longer feasible. The results are depicted in Figure 8. We observe a reasonably moderate growth here as well, where the execution time is more sensitive to an increase in $max\_depth$ compared to $ntrees$. This is consistent with our complexity analysis in Section 5. Note that $HC$ exhibited the largest execution times in the left chart, since its base configuration has $max\_depth = 9$, the largest among the datasets, as well as its features amount $d$. Similarly $LC$ exhibited the longest runtimes on the right chart, since its base configuration has $ntrees = 20$, which is the largest among the datasets.

## 6.3 Database Generation Quality

In this Subsection we study the quality of the *Database Generation* component (Section 4.2). We first measure the percentage of applicants for which our framework successfully identified at least one classification-altering region (i.e. at least one tuple). We term this rate "coverage".

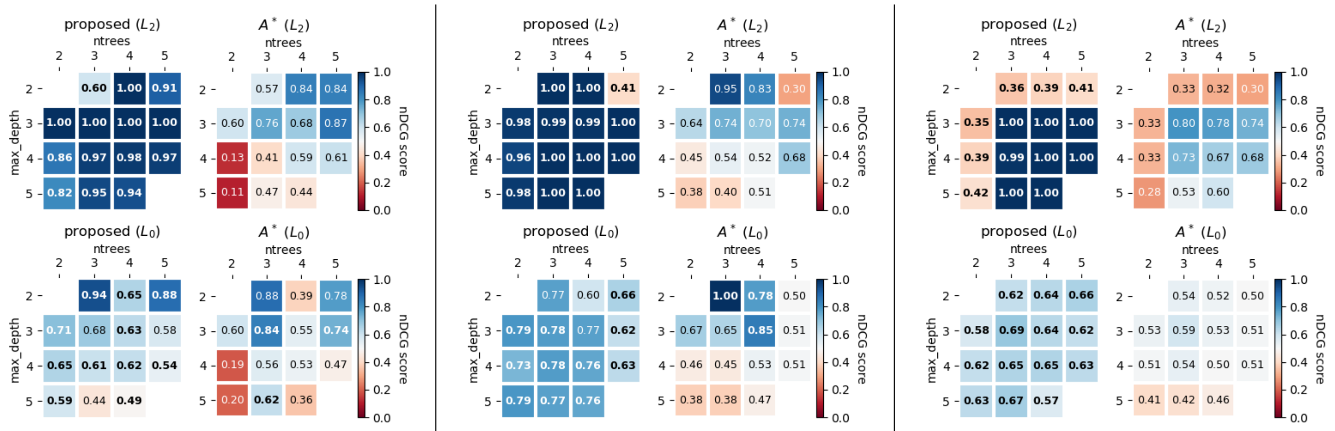
*Coverage and Interpretability.* As mentioned in Section 5, the success of our approach in identifying decision-altering regions is not guaranteed in general; however in practice we observed high coverage for all datasets tested: 100% coverage in *HC* and *H1-B*, whereas *LC* had coverage of 81%.

We further looked at the number of identified regions (obtained database size). The average size and standard deviation obtained for each of the datasets were: avg 8.89, std 3.16 (*HC*); avg 9.03, std 0.68 (*H1*); avg 6.28, std 3.99 (*LC*). We note that we have observed low correlation between the applicant score and the number of identified regions, implying that our solution is equally effective for the different users (maximal Pearson Correlation of |0.193| over all datasets).

Decision-altering regions that require a large number of modifications (even if many of them are "small"), are generally less interpretable for humans. This viewpoint on intepretability is intuitive and common in the literature, i.e. [53, 58]. Table 3 shows additional statistics on the partial database regions' $l_0$ distance. A smaller $l_0$ distance means the user is presented with classification altering options requiring a modification to only a few items in their profile. On average, our plans will include a modification to only 3.15, 2.24 and 1.89 features (for *LC*, *H1-B* and *HC* accordingly). The Max-Min column in the table specifies the maximal $l_0$ distance a user from our test sample was presented with, when asking to minimize that distance (i.e, w.r.t Q2).

*Distance Evaluation.* We next compare the database $\hat{\mathcal{D}}_t$ generated by Algorithm 3 for a fixed time point $t$, to the optimal one $\mathcal{D}_t$ generated by Algorithm 1. As a baseline, we also compare the output of $A*$ to $\mathcal{D}_t$.

We use the well known normalized cumulative gain score as a quality metric. $nDCG$ is commonly used for evaluation in information retrieval, web search engines and recommender systems [60]. A perfect algorithm that returns the top-ranked $r$ tuples w.r.t the relevance function of interest, will have an $nDCG_r$ score 1.0. Intuitively, a high $nDCG$ score implies that the regions in the partial database are comparable to the best regions of the complete database, in terms of the relevance functions of interest. The relevance



(a) Lending Club (b) Home Credit (c) H1-B Visas

**Figure 9: heatmaps depicting the *nDCG* scores recorded in the distance evaluation experiment.**

functions considered for this evaluation are based on two distance metrics $l_2$, $l_0$, thus rating regions that are closer to the input (and require smaller modifications) as more relevant.

Each experiment was performed on 14 small models, having *max_depth* and *ntrees* ranging between 2 to 5, and w.r.t 2 relevance functions and 3 datasets. Thus, a total of $14 \cdot 2 \cdot 3 = 84$ results were recorded for each of the tested approaches. The results are depicted in Figure 9, where for each dataset, four heatmaps were plotted. On 80 out of 84 scores, Algorithm 3 (i.e, "proposed") outperformed $A*$. For the 4 models in which $A*$ achieved better scores, the difference was minor ($< 0.16$). The average *nDCG* scores of the proposed algorithm are 0.68, 0.78 and 0.84 (for *H1-B*, *LC* and *HC*), as opposed to 0.51, 0.55, 0.59 for $A^*$ respectively. See the full version of this paper for more details [7].

*Adaptation Effectiveness.* In experiments presented thus far, we focused on a single time point for which the classification model $M_t$ is known. When Algorithms 2-4 are applied to the same model used at the time of re-evaluation, each point in the partial database is guaranteed to induce the desired outcome. In this experiment we change the setting to include multiple time-points, for which the classification models may be unknown. Such a setting demonstrates the full capabilities of our solution to produce actionable and effective insights in a temporal framework. The temporal framework suggested in this work is composed of two independent components: the temporal update function, and the model estimation component. In some scenarios the model estimation component is irrelevant, either because the decision making model is expected to remain constant throughout the relevant time-period, or due to the unavailability of timestamps of the historic training data records (like in *HC* dataset). When this is the case, the temporal framework still applies to the inputs (i.e, the user profile). However, when the decision making model is likely to change, the model estimation component is activated as well.

Next, we define the notion of effectiveness: an effective modification is one that leads to the desired classification at the time of re-evaluation. We measure the precision, namely the fraction of regions containing effective suggestions among all database regions.

We used time-series analysis to predict the target distribution for future time points. The predicted distributions were then used for assigning appropriate weights to the loss

**Table 3: The average, median and max-min $l_0$ distances of regions in the partial database.**

| | Average | Median | Max-Min |
|---|---|---|---|
| Lending Club | 3.15 | 3 | 5 |
| H1B Visas | 2.24 | 2 | 2 |
| Home Credit | 1.89 | 2 | 2 |

**Table 4: precision scores of *H1-B* and *LC* estimated models.**

| | Year | Instance Reweighting | No Adaptation |
|---|---|---|---|
| Lending Club | $t_0+1$ (2017) | **0.832** | 0.656 |
| | $t_0+2$ (2018) | **0.80** | 0.746 |
| H1B Visas | $t_0+1$ (2015) | **1.0** | 0.992 |
| | $t_0+2$ (2016) | **0.992** | 0.985 |

function, and generating approximated future models using the training data known at the time of application. This standard technique is called "Instance Reweighting"; See the full online version of this paper for a detailed explanation [7]. We assume that classifiers are updated on a yearly basis. For each of the datasets we selected a time point $t_0$, and a sample of rejected applicants that applied during $t_0$. We created adapted models for 2 future time-points and executed Algorithms 2–4 on them to produce partial regions databases. As a baseline, we used the known model at the time of the application to generate modifications for the future time-points (i.e, "no adaptation"). The experiment was performed for *LC* ($t_0 = 2016$) and *H1-B* ($t_0 = 2014$), for which all records include timestamps (as opposed to *HC*). In table 6 we present a comparison of f1 scores of the adapted and non-adapted models, all evaluated on the test set.

The precision scores are presented in Table 4. We can see that even the simple adaptation technique applied was consistently preferable over not adapting the known model at all. The improvement was especially significant in the *LC* dataset, where the adapted models showed an improvement of 17.6%, 5.4% for 2017, 2018 respectively.

*Tightness.* An overly strict adapted model will likely achieve high precision, since altering its classification will

**Table 5: nDCG tightness scores of the *H1-B Visas* and *LC*.**

| | Year | $l_2$ | $l_0$ |
|---|---|---|---|
| Lending Club | $t_0+1$ (2017) | 0.895 | 0.879 |
| | $t_0+2$ (2018) | 0.904 | 0.688 |
| H1-B Visas | $t_0+1$ (2015) | 0.869 | 0.883 |
| | $t_0+2$ (2016) | 0.725 | 0.962 |

**Table 6: A comparison of f1 scores calculated on the test sets of the estimated models and the reused models.**

| | Lending Club | | H1-B Visas | |
|---|---|---|---|---|
| | $t_0+1$ (2017) | $t_0+2$ (2018) | $t_0+1$ (2015) | $t_0+2$ (2016) |
| Instance Reweighting | **0.497** | **0.522** | **0.898** | **0.691** |
| No Adaptation | 0.496 | 0.521 | 0.665 | 0.670 |

require larger modifications in general. For example, an overly strict model in the case of the *H1-B* dataset may require *prevailing_wage* $\sim$ \$500,000, when in fact the mean prevailing wage among all certified applicants in the dataset is approximately \$72,000. Unfortunately, due to the prohibiting complexity of the full database generation, it cannot be used as a baseline for models of realistic size. As an alternative, we study the tightness of partial databases generated by Algorithm 3, by comparing databases derived from adapted models to ones derived from actual models.

We used the same adapted models and time points from the *Adaptation Effectiveness* experiment, and calculated *nDCG* scores as detailed in the *Distance Evaluation* experiment, compared to actual future models. This experiment aims to isolate and quantify the effect of the adaptation process on the quality of positive regions identified by our solution, in terms of the relevance functions $l_0$ and $l_2$.

The *nDCG* scores are plotted in Table 5. For that purpose, a score of 1.00 means that the retrieved regions derived from the adapted and actual future models were equally relevant for all sampled applicants. As one can expect, positive regions calculated on the actual future models were tighter, i.e closer to the inputs in general, in all the settings we tested. However, the *nDCG* scores shown in Table 5 demonstrate that the partial databases driven by the reweighted adapted models contain results that are comparable in terms of their $l_2$, $l_0$ distances, suggesting that the decision boundaries of the adapted models are not very far off from those trained on data from the future target distributions.

## 6.4 Query Evaluation Quality

In this Subsection we incorporate the *Query Evaluation* component, and aim to evaluate the quality of our solution's end-to-end queries' outputs. All experiments were performed over the queries presented in Section 4.3.

*Comparison with exact solution.* In this experiment we measure the approximation ratio (ratio between input-output distances according to the metric of interest) of outputs generated by our framework to exact outputs generated by the *Exhaustive* baseline on small models. We executed queries 2-4 (suitable for a single time-point setting) on 14 small models for each of the datasets.

We calculated the approximation ratio of the outputs for every rejected applicant in our sample. The averaged ratios are depicted in Figure 10. For example, if $Q3$'s optimal output requires a modification that has $l_2$ distance 13.64 from the input, while our solution has $l_2$ distance 15, the ratio is 1.09. The average scores measured for each dataset were 1.04 (*H1-B*), 1.31 (*HC*), and 1.38 (*LC*).

*Evaluating queries on Practical Models.* In this experiment we study the quality of queries' outputs evaluated on practical-sized models, comparing our framework and $A^*$. We hold this comparison on both actual and adapted future models, in a multiple time-point setting. For details regarding the selected time-points, applicants and adaptation process see experiment *Adaptation Effectiveness*.

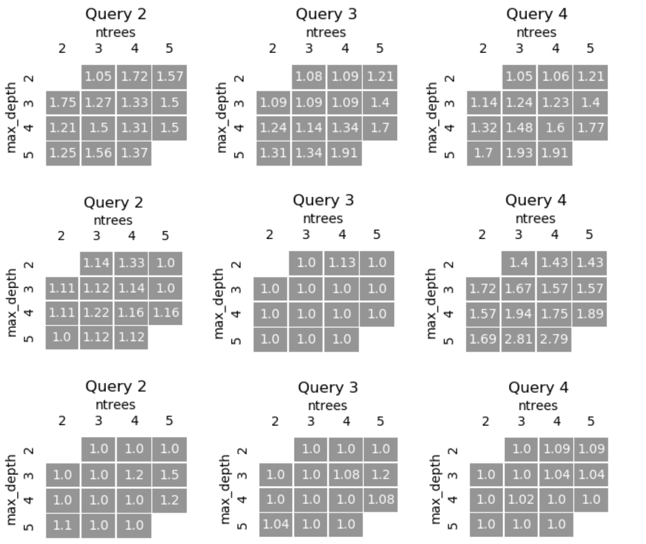


**Figure 10: Query Evaluation outputs comparison of our solution vs. *Exhaustive*. Top: *LC*, Middle: *HC*, Bottom: *H1-B*. The numbers signify the average approximation ratio.**

In Table 7, the average ratios between our solution's outputs and $A^*$'s outputs are presented: a value $< 1$ means our solution outperformed $A^*$, and vice versa. Note that the results of Q1 were omitted from Table 7: since $A^*$'s execution always starts from the temporal input $x_t$, it is guaranteed to return an optimal output for this query, and the two approaches coincide in this case. For $Q5$ we summed the amount of applicants for which some dominant feature was identified, and calculated the ratio between the sum obtained by our solution and by $A^*$. We evaluated the 5 queries presented in Subsection 6.4 for each dataset, in a total of $5 \cdot 5 = 25$ experiments. In 22 out of 25 experiments our solution achieved equal or superior final outputs.

**Table 7: The average ratio between the $A^*$'s output and our framework's output. Queries for which our solution achieved equal or superior scores are noted in bold.**

| | | Q2 | Q3 | Q4 | Q5 |
|---|---|---|---|---|---|
| Lending Club | Exact models | **0.9** | **0.66** | **0.97** | **1.0** |
| | Estimated models | 1.13 | **0.9** | **0.68** | **1.0** |
| H1-B Visas | Exact models | **1.0** | **1.0** | **0.99** | **0** |
| | Estimated models | **1.0** | **1.0** | **0.99** | **0.15** |
| Home Credit | Exact models | **1.0** | 1.05 | **0.64** | 1.55 |

## 6.5 Real User Cases

In this Subsection we demonstrate the use of our solution to derive insights and plans for real-life users. For each of the datasets, Table 8 depicts results for a representative sample of user profiles who were rejected by the model. For each user we present the output of queries Q2, Q3, and Q4 suggesting different modifications that would enable their acceptance in the following year. Even when the datasets contain tens of features (25 in LC and HC), and the model is composed of many different regions, it can be seen that the generated suggestions require a small amount of modifications to the users' profiles. For example, the first suggestion

Table 8: Real user profiles along with plans for positive classification

Home Credit

| | Credit | Education | Family Status | Income Type | Org Type | Region Population | Confidence | $l_2$ | $l_0$ |
|---|---|---|---|---|---|---|---|---|---|
| **User profile** | 269.5 | Secondary | Civil Marriage | State servant | Government | 0.00496 | 89.2% | - | - |
| **Q2** | 269.5 | Secondary | Civil Marriage | Working | Government | 0.00496 | 94% | 1 | **1** |
| **Q3** | 269.5 | Secondary | Civil Marriage | State servant | Government | 0.031885 | 89.6% | **0.027** | **1** |
| **Q4** | 269.5 | Higher Education | Civil Marriage | State servant | Government | 0.00496 | **94.3%** | 1 | **1** |
| **User profile** | 270 | Secondary | Married | Working | Self-employed | 0.019101 | 87.7% | - | - |
| **Q2,Q3** | 270 | Higher Education | Married | Working | Self-employed | 0.019101 | 92.4% | **1** | **1** |
| **Q4** | 893.7 | Higher Education | Married | Working | Self-employed | 0.019101 | **94.7%** | 623.7 | 2 |

Landing Club

| | Annual Income | Debt to Income | Installment | Loan Amount | Mortgage Accounts | Confidence | $l_2$ | $l_0$ |
|---|---|---|---|---|---|---|---|---|
| **User profile** | 60.648 | 28.87 | 289.1 | 8,400 | 0 | 49.7% | - | - |
| **Q2** | 60.648 | 21.43 | 289.1 | 8,400 | 0 | 78.2% | 7.44 | **1** |
| **Q3** | 60.648 | 28.87 | 289.1 | 8,400 | 4 | 78.1% | **4** | **1** |
| **Q4** | 60.648 | 13.625 | 165.2 | 8,400 | 2 | **81.1%** | 124.8 | 3 |
| **User profile** | 65 | 18.52 | 534.72 | 15,000 | 0 | 48.6% | - | - |
| **Q2,Q3** | 65 | 18.52 | 534.72 | 15,000 | 2 | 79.5% | **2** | **1** |
| **Q4** | 65 | 18.52 | 318.03 | 8,921 | 2 | **80.4%** | 6,082 | 3 |

H1-B Visas

| | Employer | Full Time | Title | Wage | Site | Confidence | $l_2$ | $l_0$ |
|---|---|---|---|---|---|---|---|---|
| **User profile** | Amazon | Y | SDE II | \$102$K$ | Seattle | 49% | - | - |
| **Q2, Q3, Q4** | Amazon | Y | SDE III | \$102$K$ | Seattle | **100%** | **1** | **1** |

in *LC* dataset, advises the user to decrease the "Debt to Income" ratio from 28.87 to 21.43, which is expected to induce an increase of the model's acceptance confidence from 49.7% to 78.2% in the following year.

## 7. RELATED WORK

In this section we review related work in three areas: ML explanations, domain adaptations and constraint databases.

**Machine Learning Explanations.** In recent years the problem of *interpretable Machine Learning* [48, 57, 15, 51] has been extensively studied. One approach in this respect is *explaining the model as a whole* [27, 28]. A second approach is to focus on the reasoning underlying *individual classifications*. Such local explanations can also be divided into two classes: one quantifying *Feature Importance* and the other focuses on *Counterfactual Explanations*; in this work we follow the latter. Feature importance methods quantify the influence of the sample attributes on the prediction [53, 12, 47, 54, 1]. In [46, 45], a solution to evaluate Shapley values for tree-based models was developed. Solutions that quantify feature importance mostly promote understanding of the model and its decisions, but they are not well suited for obtaining an actionable plan for altering the classification. In contrast, our solution is more in line with works on counterfactual explanations. Generally speaking, a counterfactual explanation of a prediction describes a small change to the feature values that modifies the prediction to a predefined output. Recent works, such as [58, 42, 14, 13], have suggested various methods to identify counterfactual examples. A first prototype of our solution system was demonstrated in [6]. The short demo paper accompanying the demonstration presented an earlier version of the system, and gave a high level overview of the systems capabilities and user interface whereas the present paper details the model and algorithms underlying our solution as well their experimental evaluation. To our knowledge, all prior work other than [6] aimed at finding a single counterfactual explanation; ours is the first work to generate a database of multiple explanations. Additionally, prior works on ML explanations handle only a single model; in contrast, we focus on evolving models and on explanations that will be relevant for future time points.

**Domain Adaptation.** The field of domain adaptation studies the task of utilizing the knowledge learned from source distribution to be applied on a new target distribution. Methods for domain adaptation based on Instance Reweighing were studied in [44, 11]; since those technique require knowledge of the target distribution, [10] suggested a method for its estimation. The effect of representation change for domain adaptation was analyzed in [4]. In recent years there has been active study of domain adaptation using deep neural networks [59]; it has been used in numerous settings such as image classification [37, 55], natural language processing [21], and medical diagnosis [31]. We use a domain adaptation module as a black box in our solution, and thus these efforts are in a sense orthogonal to our work.

**Constraint Databases.** Constraints databases were first introduced in [33]. A large body of theoretical work was devoted to the expressiveness and complexity of FO queries over constraints databases (e.g. [34, 35, 56]), as well as systems that were developed [9] and [22]. [23] studied the properties of linear and polynomial constraints used in our framework. The application of constraint databases in the spatial context led to numerous theoretical results (e.g. [49, 20]). In recent years, constraint query languages have been studied in the domain of spatio-temporal and moving objects [19, 36]. Temporal query languages such as Linear temporal logic (LTL) [50] also may be used in this context as they allow formulation of queries about future states of a temporal system.

Here again, a constraint database is used as a black box in our solution. To our knowledge, ours is the first usage of this field to specify the decision boundaries of ML model.

## 8. CONCLUSION

We have proposed a novel approach for querying ML classifier results, that accounts for temporal changes in both the classifier and its input. Query results may be used as actionable recommendations for individuals affected by the classification. We have developed efficient algorithms for the problem and experimentally shown their effectiveness. As future work, we aim at extending our framework to support further ML models including in particular Neural Networks.

*Acknowledgements.* This research has been funded by the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (Grant agreement No. 804302), the Israeli Science Foundation (ISF) Grant No. 978/17 and 639/17, the US-Israel Binational Science foundation, the Israel Innovation Authority - MDM, Len Blavatnik and the Blavatnik Family foundation. The contribution of Nave Frost is part of a Ph.D. thesis research conducted at Tel Aviv University.

## 9. REFERENCES

[1] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[2] M. Baudinet, M. Niezette, and P. Wolper. On the representation of infinite temporal data and queries. In *Proc. of the 10th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 280–290. Denver, 1991.

[3] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.

[4] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira. Analysis of representations for domain adaptation. In *Advances in neural information processing systems*, pages 137–144, 2007.

[5] M. Benedikt, G. Dong, L. Libkin, and L. Wong. Relational expressive power of constraint query languages. *Journal of the ACM (JACM)*, 45(1):1–34, 1998.

[6] N. Boer, D. Deutch, N. Frost, and T. Milo. Just in time: Personal temporal insights for altering model decisions. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1988–1991. IEEE, 2019.

[7] N. Boer, D. Deutch, N. Frost, and T. Milo. Personal insights for altering decisions of tree-based ensembles over time (technical report). `http://bit.ly/2YCceoP`, 2019.

[8] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[9] J.-H. Byon and P. Z. Revesz. Disco: A constraint database system with sets. In *ESPRIT WG CONTESSA Workshop on Constraint Databases and Applications*, pages 68–83. Springer, 1995.

[10] Y. S. Chan and H. T. Ng. Word sense disambiguation with distribution estimation. In *IJCAI*, volume 5, pages 1010–5, 2005.

[11] Y. S. Chan and H. T. Ng. Estimating class priors in domain adaptation for word sense disambiguation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 89–96. Association for Computational Linguistics, 2006.

[12] A. Datta, S. Sen, and Y. Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 598–617. IEEE, 2016.

[13] D. Deutch and N. Frost. CEC: Constraints based explanation for classifications. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1879–1882. ACM, 2018.

[14] D. Deutch and N. Frost. Constraints-based explanations of classifications. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 530–541. IEEE, 2019.

[15] F. Doshi-Velez and B. Kim. Towards a rigorous science of interpretable machine learning. 2017.

[16] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

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

[18] F. Geerts. Constraint databases. In *Encyclopedia of Database Systems*, pages 585–586. Springer New York, 2018.

[19] F. Geerts, S. Haesevoets, and B. Kuijpers. A theory of spatio-temporal database queries. In *International Workshop on Database Programming Languages*, pages 198–212. Springer, 2001.

[20] F. Geerts and B. Kuijpers. Real algebraic geometry and constraint databases. In *Handbook of Spatial Logics*, pages 799–856. Springer, 2007.

[21] X. Glorot, A. Bordes, and Y. Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 513–520, 2011.

[22] S. Grumbach, P. Rigaux, M. Scholl, and L. Segoufin. Dedale, a spatial constraint database. In *International Workshop on Database Programming Languages*, pages 38–59. Springer, 1997.

[23] S. Grumbach and J. Su. Queries with arithmetical constraints. *Theoretical Computer Science*, 173(1):151–181, 1997.

[24] S. Grumbach, J. Su, and C. Tollu. Linear constraint databases. 1995.

[25] T. K. Ho. Random decision forests. In *Document analysis and recognition, 1995., proceedings of the third international conference on*, pages 278–282, 1995.

[26] Home credit data. `https://www.kaggle.com/c/home-credit-default-risk/data`.

[27] J. Huysmans, B. Baesens, and J. Vanthienen. Using rule extraction to improve the comprehensibility of predictive models. 2006.

[28] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51:141–154, 2011.

[29] N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

[30] J. Jiang. A literature survey on domain adaptation of statistical classifiers. 3:1–12, 2008. `http://sifaka.cs.uiuc.edu/jiang4/domainadaptation/survey`.

[31] M. Kachuee, S. Fazeli, and M. Sarrafzadeh. Ecg heartbeat classification: A deep transferable representation. In *2018 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 443–444. IEEE, 2018.

[32] Kaggle. The state of ml and data science 2017. `https://www.kaggle.com/surveys/2017`, 2017.

[33] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, 1995.

[34] M. Koubarakis. Complexity results for first-order theories of temporal constraints. In *Principles of*

*Knowledge Representation and Reasoning*, pages 379–390. Elsevier, 1994.

[35] M. Koubarakis. The complexity of query evaluation in indefinite temporal constraint databases. *Theoretical Computer Science*, 171(1-2):25–60, 1997.

[36] B. Kuijpers and W. Othman. Trajectory databases: Data models, uncertainty and complete query languages. 2010.

[37] B. Kulis, K. Saenko, and T. Darrell. What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *CVPR 2011*, pages 1785–1792. IEEE, 2011.

[38] A. Kumagai and T. Iwata. Learning future classifiers without additional data. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[39] A. Kumagai and T. Iwata. Learning non-linear dynamics of decision boundaries for maintaining classification performance. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[40] G. Kuper, L. Libkin, and J. Paredaens. *Constraint databases*. Springer Science & Business Media, 2013.

[41] C. H. Lampert. Predicting the future behavior of a time-varying probability distribution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 942–950, 2015.

[42] T. Laugel, M.-J. Lesot, C. Marsala, X. Renard, and M. Detyniecki. Inverse classification for comparison-based interpretability in machine learning. *arXiv preprint arXiv:1712.08443*, 2017.

[43] Lending club data. `https://www.lendingclub.com/info/download-data.action`.

[44] Y. Lin, Y. Lee, and G. Wahba. Support vector machines for classification in nonstandard situations. *Machine learning*, 46(1-3):191–202, 2002.

[45] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee. Explainable ai for trees: From local explanations to global understanding. *arXiv preprint arXiv:1905.04610*, 2019.

[46] S. M. Lundberg, G. G. Erion, and S.-I. Lee. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*, 2018.

[47] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774, 2017.

[48] C. Molnar. *Interpretable Machine Learning*. 2019. `https://christophm.github.io/interpretable-ml-book/`.

[49] J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a theory of spatial database queries. In *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 279–288. ACM, 1994.

[50] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. IEEE, 1977.

[51] F. Poursabzi-Sangdeh, D. G. Goldstein, J. M. Hofman, J. W. Vaughan, and H. Wallach. Manipulating and measuring model interpretability. In *NIPS 2017 Transparent and Interpretable Machine Learning in Safety Critical Environments Workshop*, 2017.

[52] P. Z. Revesz. Constraint databases: A survey. In *International Workshop on Semantics in Databases*, pages 209–246. Springer, 1995.

[53] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. pages 1135–1144, 2016.

[54] A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. *arXiv preprint arXiv:1704.02685*, 2017.

[55] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko. Simultaneous deep transfer across domains and tasks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4068–4076, 2015.

[56] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *Journal of Computer and System Sciences*, 54(1):113–135, 1997.

[57] A. Vellido, J. D. Martín-Guerrero, and P. J. Lisboa. Making machine learning models interpretable. In *ESANN*, volume 12, pages 163–172, 2012.

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

[59] M. Wang and W. Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, 2018.

[60] Y. Wang, L. Wang, Y. Li, D. He, W. Chen, and T.-Y. Liu. A theoretical analysis of ndcg ranking measures. In *Proceedings of the 26th annual conference on learning theory (COLT 2013)*, volume 8, page 6, 2013.