Geometric Heuristics for Transfer Learning in Decision Trees

Siddhesh Chaubal
The University of Texas at Austin
USA
spchaubal@gmail.com

Mateusz Rzepecki
University of Wrocław
Poland
mateuszrzepecki99@gmail.com

Patrick K. Nicholson
Nokia Bell Labs
Ireland
pat.nicholson@gmail.com

Guangyuan Piao
Maynooth University
Ireland
guangyuan.piao@mu.ie

Alessandra Sala
Nokia Bell Labs
Ireland
alessandra.sala@gmail.com

ABSTRACT

Motivated by a network fault detection problem, we study how recall can be boosted in a decision tree classifier, without sacrificing too much precision. This problem is relevant and novel in the context of *transfer learning* (TL), in which few target domain training samples are available. We define a geometric optimization problem for boosting the recall of a decision tree classifier, and show it is NP-hard. To solve it efficiently, we propose several near-linear time heuristics, and experimentally validate these heuristics in the context of TL. Our evaluation includes **7** public datasets, as well as **6** network fault datasets, and we compare our heuristics with several existing TL algorithms, as well as exact *mixed integer linear programming* (MILP) solutions to our optimization problem. We find that our heuristics boost recall in a manner similar to optimal MILP solutions, yet require several orders of magnitude *less* compute time. In many cases the F_1 score of our approach is competitive, and often better, than other TL algorithms. Moreover, our approach can be used as a building block to apply transfer learning to more powerful ensemble methods, such as random forests.

CCS CONCEPTS

• **Computing methodologies → Classification and regression trees**; **Transfer learning**.

KEYWORDS

Transfer learning, Decision trees, Random forests, Classification

ACM Reference Format:
Siddhesh Chaubal, Mateusz Rzepecki, Patrick K. Nicholson, Guangyuan Piao, and Alessandra Sala. 2021. Geometric Heuristics for Transfer Learning in Decision Trees. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482259

1 INTRODUCTION

With the rising use of machine learning (ML) to automate tasks in a wide variety of new industries, a pain point is often the ability to *transfer* a model—either trained using supervised methods, or hand-crafted by domain experts—to a new, and possibly different, setting. As a concrete and running example, consider an ML model trained to detect and assist the diagnosis of faults in a live communication network. If the model is trained using data collected from a network operator A, called the *source domain*, it may not be directly transferable to a different operator, B, or *target domain*. This is due to the large amount of configuration parameters, customized network topologies, and other bespoke customization of the networks. Even if the model is trained with data from a wide variety of operators, there is no way, a priori, to guarantee that this source training data is general enough without having a data scientist in-the-loop to validate the model in the target domain.

This need represents a major scalability problem for many startups and companies that wish to incorporate ML into their technology stack. A further issue for startups that train ML models based on customer (target domain) data is that stringent regulations may rule out the possibility of simultaneously accessing the source data and target data. For example, a customer may be legally unable to transmit their labeled target domain data to an external company. Instead, in such cases, ML models must be trained or adjusted in the customer's premises. Moreover, it is unlikely that the external company providing the source model would want to share their labeled source data.

Towards addressing these issues, we present novel techniques to expedite the transfer of ML models. Thus, we focus on techniques for the case of *homogeneous transfer learning* [34] that have the following important properties:

(1) The techniques should work in the case of *small data*, or, in other words, the case when the amount of labeled data in the target domain is limited.
(2) The techniques should *only require access to the source model and target data.*
(3) Finally, the techniques should also apply to hand-crafted models, e.g., configuration files created by domain experts. Such models can be easily converted to decision trees, and therefore our primary focus is transfer learning methods that are applicable to decision trees.

Our Contributions: We present a new post-processing method for binary classification decision trees, based on local expansion of orthogonal hyper-rectangles via geometric optimization. The goal of our method is to improve the recall of the positive class in the target domain, while minimizing loss of precision. At the core of many applications are scenarios where recall has higher importance than precision: e.g., detecting credit card fraud, or diagnosing cancer. In our networking application, we take an automated remedial action when a fault is detected. Taking this remedial action is costly, but has lower cost than allowing the fault to persist, which may negatively impact mobile phone subscribers.

Geometric optimization has been explored in the context of probably approximately correct learning [25], and other contexts discussed in Section 2. However, our method is the first such use in the context of homogeneous transfer learning. Our method can be applied to any existing decision tree, and provides an alternative mechanism for boosting recall, compared to adjusting the classification threshold, that can achieve superior trade-offs. Decision trees are a fundamental algorithm in many data mining applications [26], and are also the building blocks in many widely used ensemble methods such as *random forests* [10] and *gradient boosted trees* [12]. Thus, our methods are also applicable on top of these more powerful techniques.

At a deeper level, we define the *optimal expansion problem* for decision tree classifiers in Section 3. We show that this problem is NP-hard, but present a fast heuristic for expansion in Section 3, and provide a method for automatically tuning the hyper-parameters of our approach. In Section 6 we perform a comparison of our method to several state-of-the-art transfer learning algorithms, as well as an MILP approach that generates an optimal solution to our expansion problem. Since our approach can be run as a post-processing method after any existing transfer learning algorithm, we apply our geometric heuristics on top of these other approaches, and also define a procedure for selecting which algorithm to apply. We evaluate our approach and the state-of-the-art algorithms on **7** public datasets, as well as **6** network fault detection datasets. These experiments show how the various transfer learning approaches perform in different regimes of data sparsity. Overall we find that our approach is very effective at boosting recall, and for many datasets also achieves the highest F_1-scores. Our algorithm compares favorably to an optimal solution in terms of classification performance, while requiring several orders of magnitude less compute time than an MILP solver. Finally, we compare our method to one which lowers the classification threshold for the decision tree, and show it provides favorable trade-offs that are not available via this alternative method of boosting recall.

2 PRIOR WORK & LIMITATIONS

Transfer learning has been studied in many settings, and we refer the reader to surveys [27, 34] for more details. We follow [34], separating the related work into the categories of *homogeneous* and *heterogeneous* transfer learning.

Homogeneous transfer learning is the setting in which source and target domains share a common feature space. Most relevant to our setting is the work by [31] for homogenous transfer learning on decision trees (and random forests). They present two algorithms: 1) *STRUT*, or structural transfer, which preserves the structure of the source model but uses a bicriteria optimization of two information

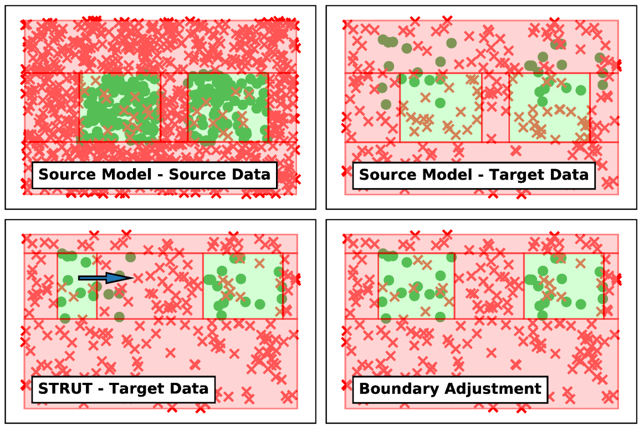


Figure 1: Top left: source tree on source data (two classes; red 'x' and green 'o'). Areas with red (resp. green) background indicate that points in that region are classified as red (resp. green). Top right: source tree on target data with additional noise present. Bottom left: STRUT applied to source tree using target data. STRUT achieves high precision for green points, but the right boundary of the left rectangle can be locally adjusted to boost recall, as indicated by the arrow. Bottom right: locally expanded rectangle.

theoretic measures – information gain and *divergence gain* – to adjust node thresholds, and; 2) *SER*, or structured expansion and reduction, that performs certain expansion and reduction steps on the source model based on the target data. During preliminary investigation for our motivating use case, we found that STRUT worked reasonably well in terms of precision. However, a major deficiency of the algorithm is that it is highly sensitive to *changes in noise levels* between source and target domains, and so the adjusted tree boundary often has poor recall. We provide an illustrative toy example showing this issue in Figure 1. Note that lowering the classification threshold in the bottom left panel results in low precision: a lower threshold turns the middle red area between the two green rectangles green, even though it contains many red points.

Another approach, similar to STRUT, has been described in [2]. There is also work not restricted to a specific classification model, such as that in [22], which generalizes the expansion step of SER to other models. Frustratingly Easy Domain Adaptation [14], or *FEDA*, utilizes a simple yet powerful method of *feature augmentation*, that can be applied to any classification model. Other works [23, 29] explore heterogeneous transfer learning in decision trees.

It is also worth noting that treating a rule-based classifier (or a decision tree) as a union of hyper-rectangles and applying geometric transformations on them has been explored in many other contexts. Salzberg [30] extends the model of exemplar-based learning, wherein hyper-rectangles are stored as "exemplars" and a sample point is classified based on distance from these exemplars. Maass [25] examined the problem in the context of PAC learning, and there have also been studies on learning empty regions in the dataset, by Liu et al. [24] and Edmonds et al. [16]. Increasing the recall for a primary class in the case of imbalanced datasets was explored by Grzymala-Busse et al. [18, 19], and by Stefanowski and Vanderpooten [32]. However, such methods have not been applied in the context of transfer learning.

Finally we note that some recent works attempt to find optimal decision trees using various techniques [9, 21, 33]. However, these works are not concerned with the transfer learning setting, and the objective function they are attempting to maximize is not focused on recall. Nonetheless, we compare an MILP formulation of our optimal expansion problem to our heuristics.

3 GEOMETRIC EXPANSION

In this section we formalize our algorithms for expanding hyper-rectangles, and provide efficient heuristics. We ignore categorical features in this section, but our methods operate in the presence of, without acting upon, such features.

We consider binary classification tasks, with the goal of increasing recall for class 1, noting that by symmetry we can apply the same procedure to class 0. Let T denote a decision tree trained on the source dataset, and, for brevity, denote the target dataset with n samples and d features as $\{(x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,d}), y_i), i \in \{1, \ldots, n\}\}$. Since we only deal with target data—recall that our algorithms require no access to the source data—we require no additional notation to distinguish the source and target data.

Consider any leaf node u in a decision tree T. For an input $(x'_1, \ldots, x'_d)$ that is contained in u, the path from the root of T to u corresponds to a list of threshold conditions and indicators (C_1, I_1), ..., (C_k, I_k), where condition C_i has the form "$x'_{j_i} < t_i$?" for some $j_i \in \{1, \ldots, d\}$ and I_i indicates whether C_i is true or false. Thus, (C_1, I_1), ..., (C_k, I_k) induces a d-dimensional axis-aligned hyper-rectangle in the feature space. In what follows, for brevity, we refer to such axis-aligned hyper-rectangles simply as rectangles. Since a feature can appear more than once in the path to u, it may be the case that the rectangle is unbounded in $d - k'$ dimensions, where $1 \leq k' \leq k$. For a given decision tree T, let $\mathcal{R}_1, ..., \mathcal{R}_z$ denote the set of disjoint rectangles whose union corresponds to the set of points that are classified as 1 by T.

Intuition. Our algorithm proceeds by iteratively, and locally, expanding each rectangle $\mathcal{R}$, so that the *precision of* $\mathcal{R}$ does not drop by more than a pre-specified amount $\delta \geq 0$, on the target data. Explicitly, the precision of $\mathcal{R}$ is defined as $\text{pr}(\mathcal{R}) = \frac{\sum_{(x_i, y_i) \in \mathcal{R}} y_i}{\sum_{(x_i, y_i) \in \mathcal{R}} 1}$. Since the target data is sparse, our heuristics assume (and exploit any) locality in examples of class 1. Since expanding $\mathcal{R}$ can only cause recall to increase, this method—much like lowering the prediction probability (i.e. classification threshold) for class 1—will result in a monotone increase in recall.

Optimal Expansion Problem. Consider an n sample dataset $\{(x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,d}), y_i), i \in \{1, \ldots, n\}\}$ in d-dimensional feature space, where each $y_i \in \{0, 1\}$. Given a rectangle $\mathcal{R}$ containing at least one data point, and $\delta \geq 0$, return a rectangle $\mathcal{R}^\star$ such that $\mathcal{R} \subseteq \mathcal{R}^\star$, $\sum_{(x_i, y_i) \in \mathcal{R}^\star} y_i$ is maximized, and $\text{pr}(\mathcal{R}^\star) \geq (1 - \delta)\text{pr}(\mathcal{R})$.

Theorem 1. *The optimal expansion problem is NP-hard.*

Proof. Suppose we are given an instance of Maximum Bichromatic Empty Rectangle (MBER) [5], which is NP-hard. The decision problem input is n red *and* blue points, and integer k, and output is "yes" if there exists a rectangle containing at least k blue points, but no red points. We reduce from the MBER problem to our optimal expansion problem as follows. During a preprocessing phase, all blue points that overlap with red ones are removed, as they are never part of a solution to the MBER problem. We assume at least one blue point remains after this pruning, otherwise the solution to the MBER instance is trivial. For each remaining blue point, we create a rectangle that surrounds only that point. For the created rectangle $\mathcal{R}$, assume we have an oracle that can answer the decision version of the optimal expansion problem (i.e., determine whether there is an expansion of $\mathcal{R}$ containing at least k points in the positive class). We query this oracle, setting $\delta = 0$, and treating red points as those in the negative class, and blue points as those in the positive class. Since the input rectangle $\mathcal{R}$ has $\text{pr}(\mathcal{R}) = 1$, and $\delta = 0$, a "yes" answer to the decision problem implies there exists a rectangle $\mathcal{R}^\star$ with $\text{pr}(\mathcal{R}^\star) = 1$ (i.e., no red points) and containing at least k positive (i.e., blue) points. Clearly, if any decision version of this optimal expansion problem returns "yes", then the answer to the MBER instance is "yes", otherwise the answer is "no". Since, i) MBER is NP-hard, ii) our reduction requires time polynomial in n, and, iii) the optimal expansion decision problem is clearly in NP via a linear-time verification step, we have shown that the optimal expansion problem is also NP-hard. □

We remark that there are strong connections to a host of related geometric problems [3, 6, 7, 15]. We note that these related problems admit a polynomial time solution when d is not part of the input (i.e., a fixed constant value), but this is not a reasonable assumption for our machine learning setting.

Heuristics for Geometric Expansion. Since we have shown the optimal expansion problem is NP-hard, we instead focus on heuristic methods to provide solutions quickly, albeit without guaranteeing solution optimality. We next describe our heuristic method for expanding rectangles, and how to compute it in terms of a set of abstract data structure operations.

Suppose we are given an arbitrary rectangle $\mathcal{R}$, with "left" boundary $\text{lb}(\mathcal{R}, j)$ (i.e., the minimum), and "right" boundary $\text{rb}(\mathcal{R}, j)$ (i.e., the maximum), with respect to feature j. Define the width with respect to feature j as $w(\mathcal{R}, j) = \text{rb}(\mathcal{R}, j) - \text{lb}(\mathcal{R}, j)$. Note that $\text{lb}(\mathcal{R}, j)$ can be $-\infty$ and $\text{rb}(\mathcal{R}, j)$ can be ∞. Suppose $\text{lb}(\mathcal{R}, j) = -\infty$, but $\text{rb}(\mathcal{R}, j)$ is finite (the alternative case is symmetric). We set $w(\mathcal{R}, j) = \text{rb}(\mathcal{R}, j) - \gamma$, where γ is the minimum value of feature j among points in $\mathcal{R}$. If both boundaries are infinite, then $w(\mathcal{R}, j) = \infty$. Our heuristics will iteratively expand $\mathcal{R}$ by iteratively adjusting these boundaries of $\mathcal{R}$.

Our prototypical geometric heuristic for expanding a single rectangle is formalized in Algorithm 1. Input parameters to the algorithm are: i) DS, a data structure that supports search operations on the dataset; ii) a parameter $\delta \geq 0$, controlling the allowed drop in precision on the target data; iii) a value $\beta > 0$ controlling the rate of expansion; iv) a value $\text{iter}_{\max}$ indicating the maximum number of iterations, and; v) the rectangle $\mathcal{R}$ that we wish to expand.

The algorithm first calls DS.Initialize (line 2), preparing the internal representation of the data structure to support efficient expansion of the input rectangle $\mathcal{R}$. The heuristic then iteratively selects the next feature, j, and attempts to expand rectangle $\mathcal{R}$ with respect to feature j. An expansion factor Δ is computed, which

Algorithm 1 BasicHeuristic(DS, δ, β, iter$_{\max}$, $\mathcal{R}$)

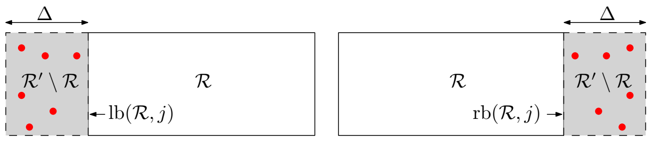
```
Require: Data structure DS, rectangle $\mathcal{R}$, parameters $\delta$, $\beta$, iter$_{\max}$
 1: $\mathcal{R}_0 \leftarrow \mathcal{R}$
 2: DS.Initialize($\mathcal{R}$)
 3: for $j \in \{1, \ldots, d\}$ do
 4:     $\Delta \leftarrow \beta \times w(\mathcal{R}, j)$
 5:     if $\Delta = 0$ or $\Delta = \infty$ then
 6:         continue
 7:     for iter $\in \{1, \ldots, \text{iter}_{\max}\}$ do
 8:         if $\text{lb}(\mathcal{R}, j) - \Delta \geq \min_i x_{i,j}$ then
 9:             $S \leftarrow$ DS.collect($j, -1, \Delta$)
10:             if $(1-\delta) \cdot \text{pr}(\mathcal{R}_0) \leq \text{pr}(\mathcal{R} \cup S)$ then
11:                 $\mathcal{R} \leftarrow$ DS.Commit()
12:             else
13:                 DS.Rollback()
14:         if $\text{rb}(\mathcal{R}, j) + \Delta \leq \max_i x_{i,j}$ then
15:             $S \leftarrow$ DS.Collect($j, 1, \Delta$)
16:             if $(1-\delta) \cdot \text{pr}(\mathcal{R}_0) \leq \text{pr}(\mathcal{R} \cup S)$ then
17:                 $\mathcal{R} \leftarrow$ DS.Commit()
18:             else
19:                 DS.Rollback()
20: return $\mathcal{R}$
```

determines the rate at which $\mathcal{R}$ will expand with respect to feature j, based on β and the width of $\mathcal{R}$. The algorithm iteratively expands both the left and right boundaries of $\mathcal{R}$ with respect to feature j by Δ, stopping only when either the precision of $\mathcal{R}$ drops by more than δ compared to the original rectangle $\mathcal{R}_0$, or iter$_{\max}$ is exceeded.

During the process, there are a few crucial operations executed on the DS. The first is called DS.Collect(j, sign, Δ) on line 9. Let $\mathcal{R}'$ be the rectangle induced by extending the $\text{lb}(\mathcal{R}, j)$ (resp. $\text{rb}(\mathcal{R}, j)$) to the left (resp. right) by Δ, if sign $= -1$ (resp. sign $= 1$). Then, DS.Collect(j, sign, Δ) returns the set of points in $\mathcal{R}' \setminus \mathcal{R}$ (i.e., the new points added by moving the boundary). This is illustrated for a case where $d = 2$ in Figure 2. Since adjusting left and right boundaries of $\mathcal{R}$ may result in too large a precision drop, DS provides the ability to reverse an expansion (called a Rollback). Alternatively, after computing the precision of $\mathcal{R} \cup S$, another operation, called Commit, finalizes the boundary expansion.

Theorem 2. *There is a data structure DS that occupies $O(nd)$ space, supports the operations Collect, Commit, and Rollback, and can be used to execute the for loop (lines 3-19) in Algorithm 1 in $O(nd + \text{iter}_{\max})$ time after an initial pre-processing phase (i.e., Initialize has completed on line 2). This pre-processing phase on line 2 requires $O(nd \lg n)$ time and $O(nd)$ space.*

Proof. The data structure DS maintains d sorted lists, as well as several additional structures. Let $\mathcal{L}_j$, for $j \in \{1, \ldots, d\}$, be the lists of feature values for the j-th dimension, sorted in ascending order. Each index $\mathcal{L}_j[k]$ will store more than just these feature values, so we denote the feature values as $\mathcal{L}_j[k]$.val. We re-index the data set into a rank index $\mathcal{I}$ such $\mathcal{I}[i]$ represents x_i, and stores the list of ranks $(k_1, ..., k_d)$ such that $(\mathcal{L}_1[k_1].\text{val}, ..., \mathcal{L}_d[k_d].\text{val}) = x_i$. Next we describe the additional fields associated with each list element $\mathcal{L}_j[k]$: i) a back pointer to its associated index in $\mathcal{I}$, such



Δ
$\mathcal{R}' \setminus \mathcal{R}$
$\mathcal{R}$
$\text{lb}(\mathcal{R}, j)$
Δ
$\mathcal{R}$
$\text{rb}(\mathcal{R}, j)$
$\mathcal{R}' \setminus \mathcal{R}$

Figure 2: Left: extending the left boundary of $\mathcal{R}$ by Δ: see lines 8-13 of Algorithm 1. Red points are returned as S by the function DS.Collect. Right: symmetric case, extending the right boundary of $\mathcal{R}$ by Δ, performed on lines 14-19.

that $\mathcal{I}[\mathcal{L}_j[k].\text{back}][j] = k$, and ii) a flag $\mathcal{L}_j[k]$.active indicating whether $\mathcal{L}_j[k]$ is *active* in the expanding current rectangle. Overall, although there are several additional fields for each data point, the space occupied is linear, $O(nd)$, and it can be constructed in the time required to sort each list of feature values, or $O(nd \lg n)$ time.

Consider rectangle $\mathcal{R}$ input to Initialize. Let $\{a_j, a_j + 1, \ldots, b_j\}$ be the indices of the range of feature values spanned by $\mathcal{R}$ in the j-th dimension. Formally, we have $\text{lb}(\mathcal{R}, j) \leq \mathcal{L}_j[a_j].\text{val} \leq ... \leq \mathcal{L}_j[b_j].\text{val} \leq \text{rb}(\mathcal{R}, j)$. So, all the data points contained in $\mathcal{R}$ are associated with feature values in $\mathcal{L}_j$ between indices a_j and b_j, but the converse is not true. Moreover, we can identify a_j, b_j for all j in $O(d \lg n)$ time. We maintain pointers to a_j and b_j, to keep track of the current rectangle $\mathcal{R}$.

We maintain the following invariants on the active flags: all feature values are active in $\mathcal{L}_1$. All feature values that are active in $\mathcal{L}_j$, for $j > 1$, represent data points that are contained in the current rectangle $\mathcal{R}$ with respect to feature dimensions $\{1, ..., j-1\}$. So, by definition, the feature values that are active in $\mathcal{L}_d$ and are in the range $\{a_d, a_d + 1, \ldots, b_d\}$ represent points that are in rectangle $\mathcal{R}$.

We have the following recursive scheme to iteratively expand $\mathcal{R}$: this describes how the Collect function operates. Suppose we expand dimension j in the positive direction, without loss of generality. Now $\text{rb}(\mathcal{R}, j) = \gamma$. We increment b_j, stopping when $\mathcal{L}_j[b_j].\text{val} \leq \gamma$ but $\mathcal{L}_j[b_j + 1].\text{val} > \gamma$. Let b_j' be this new value of b_j. For any feature values that we encounter in $\mathcal{L}_j[b_j], ..., \mathcal{L}_j[b_j']$ that were active in $\mathcal{L}_j$, we find the position of their associated data point's feature values in $\mathcal{L}_{j+1}$ using the back pointers and index $\mathcal{I}$. We then make their feature values active in $\mathcal{L}_{j+1}$. If they are contained in $\{a_{j+1}, \ldots, b_{j+1}\}$, then we repeat the process for dimensions $j+2, ...,$ etc. Finally, if $j = d$ and the active points are in the range $\{a_d, \ldots, b_d\}$, then we add these points to the set S.

Finally, we comment on the Rollback and Commit operations. Conceptually, Collect is reversible, provided we had all the old values of the pointers a_j and b_j. Thus, we can easily reverse the operation. Similarly, Commit need not actually do anything, other than erase the information necessary to perform a Rollback.

It is clear that the Collect operations can only touch each data point at most $O(d)$ times, since a data point can only be activated once in each dimension. Moreover, a point that is *de-activated* by a Rollback will never become active later with respect to that dimension. This means we will never propagate the point to next active list, so overall each point requires $O(d)$ time to process, since it was activated in each dimension only once. Thus, Rollback is at most as expensive as `Collect`, so the total running time after Initialize is $O(nd + \text{iter}_{\max})$. □

We remark that there are existing data structures in the computational geometry literature that support so-called *orthogonal range queries*, such as optimized range trees [11] or kd-trees [8], which *could* be leveraged by Algorithm 1. However, our dynamic programming solution exploits the structure of the queries we make in order to avoid a multiplicative extra cost associated with $\text{iter}_{\max}$, which is especially relevant when d is large.[1] Moreover, our algorithm is very simple to implement and cache-friendly as it boils down to simultaneous scanning of sorted lists with very few random accesses.

Initially, we record the precision of each rectangle in the tree T. As a pre-processing step, we then discard all points contained in all the rectangles $\mathcal{R}_1, ..., \mathcal{R}_z$. We apply Algorithm 1 to each rectangle $\mathcal{R}_1, ..., \mathcal{R}_z$ which satisfies $pr(\mathcal{R}_i) \geq pr(T)$. That is, we only mark rectangles that have precision higher than the overall precision (of class 1) of the tree T for expansion. After expanding $\mathcal{R}_i$, we discard all points contained in the expanded rectangle before processing $\mathcal{R}_{i+1}$. This is required since, after expanding the rectangles, they may no longer be mutually disjoint. Let us denote by $T^\star$, the rule-based classifier obtained after this process. It can be proved using simple calculations(which we omit for lack of space) that $pr(T^\star) \geq (1-\delta) \cdot pr(T)$, which guarantees that the precision does not decrease by more than a (1- δ) factor on the available target data. It is important to note that this does not ensure that the precision of the original tree will only drop by a $(1-\delta)$ factor on any new (e.g., holdout) data from the target domain.

This process of applying Algorithm 1 to rectangles iteratively illustrates only one heuristic, whereas many choices are possible. We found that several variants exhibit similar performance, with a slight edge given to a heuristic that orders rectangles in decreasing order of the number of target samples covered, and then expands them uniformly. Thus, in later evaluation we applied this heuristic.

Setting hyperparameters. Algorithm 1 depends on three hyperparameters - β, δ and $\text{iter}_{\max}$. We found that the dependence of the recall and precision on parameters δ and $\text{iter}_{\max}$ was monotone: higher values typically yield higher recall and lower precision if all other parameters are fixed. We also found that smaller values of β are advantageous as they result in more gradual expansion of the rectangles, avoiding early termination of the algorithm. For all our experiments, we set $\beta = 0.01$ (corresponding to a 1% increase in the width of the rectangle) and then tune $\text{iter}_{\max}$ automatically using a grid-search approach that maximizes F_1-score on sampled subsets of the target data. Thus, from a user perspective, they need only specify δ, the multiplicative precision loss parameter, in order to use our algorithm. For our experiments we used $\delta = 0.1$.

4 MIXED INTEGER LINEAR PROGRAMMING

We now describe a mixed integer linear programming (MILP) approach to find the exact solution to the optimal expansion problem. For a given decision tree T, recall that $\mathcal{R} = \{\mathcal{R}_1, ..., \mathcal{R}_z\}$ denotes the set of disjoint axis-aligned rectangles corresponding to the leaves labeled 1 in T which satisfy $\text{pr}(\mathcal{R}_i) \geq \text{pr}(T)$. We denote the corresponding output (i.e., expanded) rectangles by $\mathcal{R}'_1, \mathcal{R}'_2, \ldots, \mathcal{R}'_z$.

[1]An implementation using kd-trees would result in an additive $\text{iter}_{\max} n^{1-1/d} d$ term appearing in the running time, which is inferior to our bound if $\text{iter}_{\max} \in \omega(n^{1/d})$.

Similar to our heuristics, we conceptually delete all the points contained in any rectangle in the set $\mathcal{R}$ from the dataset by marking them.

Variables: For each rectangle $\mathcal{R}_i$ and each feature $j \in d$, we have continuous-valued variables $z^l_{i,j}$ representing the left boundary of $\mathcal{R}'_i$ (i.e. new left boundary of $\mathcal{R}_i$ after expansion), in the dimension corresponding to feature j and similarly, variables $z^r_{i,j}$ for the corresponding right boundaries of $\mathcal{R}'_i$.

To ensure that each boundary only expands and never contracts, we add the following constraints:

$$\min\left\{\text{lb}(\mathcal{R}_i, j), \min_{k\in\{1,\ldots,n\}} x_{k,j}\right\} \leq z^l_{i,j} \leq \text{lb}(\mathcal{R}_i, j) \text{ and}$$

$$\max\left\{\text{rb}(\mathcal{R}_i, j), \max_{k\in\{1,\ldots,n\}} x_{k,j}\right\} \geq z^r_{i,j} \geq \text{rb}(\mathcal{R}_i, j) \ .$$

For each rectangle $\mathcal{R}_i$ and data point (x_j, y_j), we have variables $b_{i,j}$: these would be set to 1 if point j is inside rectangle $\mathcal{R}'_i$, and is set to 0 otherwise. To achieve this, we add constraints: $b_{i,j} = \bigwedge_{k\in\{1,\ldots,d\}} d_{j,k}$, where variable $d_{j,k} = 1$ iff $x_{j,k} \in [\text{lb}(\mathcal{R}_i, k), \text{rb}(\mathcal{R}_i, k)]$: this can be encoded via a pair of indicator constraints. Further, for each data point (x_j, y_j), we have binary variables $c_j = \bigvee_{\mathcal{R}_i\in\mathcal{R}} b_{i,j}$ where the logical-OR is over all rectangles $\mathcal{R}_i$. In other words, variable c_j is 1 if point j is included in some rectangle from the set $\mathcal{R}$, and is 0 otherwise. For each rectangle $\mathcal{R}_i$ and each data point (x_j, y_j), we have binary variables $a_{i,j}$. Variable $a_{i,j}$ is set to 0 if data point x_j is outside rectangle $\mathcal{R}'_i$; it is set to 0 or 1 otherwise. Additionally, for each data point (x_j, y_j), $a_{i,j}$ is set to 1 for exactly one rectangle $\mathcal{R}'_i$ it is in unless it is not covered by any rectangle. Thus, we observe that: $c_j = \sum_{\mathcal{R}_i\in\mathcal{R}} a_{i,j}$ for $j \in \{1, \ldots, n\}$.

Constraints to limit precision drop: For each rectangle $\mathcal{R}_i$, we require: $\text{pr}(\mathcal{R}'_i) \geq (1-\delta)\text{pr}(\mathcal{R}_i)$ which translates to:

$$\frac{\sum_{y_j=1} a_{i,j} + \sum_{(x_j,y_j)\in\mathcal{R}_i} y_j}{\sum_{j\in\{1,\ldots,n\}} a_{i,j} + \sum_{(x_j,y_j)\in\mathcal{R}_i} 1} \geq (1-\delta)C(T) \ , \tag{1}$$

where $C(T)$ is a constant that depends only on T. Thus, this is a set of linear constraints in terms of variables $a_{i,j}$.

Objective function: Maximize recall: $\sum_{y_j=1} c_j$.

5 EXPERIMENTAL SETUP

All our experiments are performed using the Strlet library of [31], which builds on Weka [20], but uses their own implementation of Weka's J48 (C4.5 with normalized information gain) algorithm for learning source models, with pruning based on the hyperparameters of maximum depth, minimum impurity decrease and minimum samples in a leaf. Each dataset is partitioned into source and target data. Further, we then hold out half of the target data for testing purposes, and use the remaining half for training. For each fraction of target data (1%-15%), we train the model 20 times using this percentage of the training data, sampled randomly and stratified. This exhibits the performance of the heuristics with relatively small amounts of data from the target domain. In our implementation, we use 5-fold cross validation on the source data to tune the aforementioned hyperparameters. We also experimented with our approach applied on top of random forests (to each constituent tree) as in Segev *et al.* [31]. The results were similar to those for basic decision trees, and we include some of these comparisons for reference.

We used the following datasets, whose details are presented in Table 1. All datasets except "Network" are publicly available at UCI machine learning repository [4]. We selected these public datasets in line with the previous work [31], adapting them where possible to a binary classification setting. For ease of reproducibility, we have uploaded any modified public datasets to a public repository [1].

Table 1: Dataset information: d is the number of features, Src size is the number of samples in the source data and Src imbalance ratio (Tgt imbalance ratio) gives the fraction of samples in source data (respectively target data) from Class 1.

Dataset	d	Src size	Src imbalance ratio	Tgt imbalance ratio
CreditAU	14	468	0.44	0.45
Wine	11	4898	0.66	0.53
Letter	16	10822	0.055	0.099
Digits	64	5620	0.099	0.1
Landmine	9	8535	0.05	0.07
CreditDE	19	690	0.27	0.35
Invert	784	2000	0.1	0.1
Network 1	197	1508	0.07	0.04
Network 2	197	1508	0.07	0.10
Network 3	197	1508	0.07	0.10
Network 4	197	4643	0.06	0.06
Network 5	197	4643	0.06	0.04
Network 6	197	4643	0.06	0.10

CreditAU [28]: Banking dataset where samples represent credit card applications. The data is split into source and target domains based on whether the attribute "A1" is 1 or 0 respectively.

Wine [13]: Wine quality score dataset for white and red wines. White wines are used as the source domain and red wines as target. All samples with a quality-score of 6 or more have class 1 (the "fine" wines), and those with a score less than 6 are class 0.

Letter [17]: Letter recognition dataset with 26 classes and 16 features for classification. Source domain consists of the subset of the dataset, partitioned as in [31]. Letters "b" and "d" belong to class 1 and all others belong to class 0.

Digits: Images of hand-written digits. As in [31], the source domain task is identifying digit "6" while the target domain is identifying the digit "9".

Landmine: Dataset collected from 29 mine fields, represented by 9 features and a binary label. The source domain consists of 15 foliaged fields and the target domain consists of 14 barren fields.

CreditDE: German credit risk dataset with two classes (i.e., good or bad). We use the subset of male customers as source domain and female customers as target domain.

Invert: MNIST digit database where the target images have inverted colors as in [31].

Network: Labeled time-series metrics collected from several networks (different domains) with labeled faults. Normal behavior is class 0, and observed faults are class 1. There are **6** different transfer datasets of this type.

Baselines and State-of-the-art Competing Algorithms: Now we describe the TL algorithms with which we compare our heuristics:

SrcOnly: The baseline approach that trains a model only with the source domain data without using any target domain data, but with hyperparameters tuned on the source data.

TgtOnly: Another baseline approach which trains a model only with the target domain data without transfer learning, and the same hyperparameters as the source model.

STRUT [31]: The STRUT (structure transfer) algorithm transfers the structure of the decision tree learned from the source domain and chooses new thresholds for each tree node based on the target domain data.

SER [31]: The SER (structure expansion/reduction) algorithm can refine the decision tree learned from the source domain by specializing the rules of decision tree with the expansion transformation or generalizing the rules with the reduction transformation. We use the implementations of STRUT and SER from the code used in [31] (can be obtained from http://tiny.cc/kgz5dz).

Our approach, denoted **GeoX**, is to apply our geometric heuristics to baseline **X**, where **X** can be any of the above-mentioned baselines. We also compare our approach against these three additional baselines:

Lowering the classification threshold: the classification threshold of the decision tree constructed by SER is set to values ranging from 0.01 to 0.5.

FEDA [14]: Frustratingly Easy Domain Adaptation (FEDA) is an *instance transfer* approach in which all source training examples are available during the adaptation to the target domain. As pointed out in [31], this is an unfair comparison as our approach uses model transfer, which learns without access to source examples. Nevertheless, we included FEDA to understand the limitations of our model transfer approach.

MILP: as described in section 4 we used MILP to solve the Optimal Expansion problem, implemented in Gurobi (with Java bindings)[2]. In our experiments, we compare heuristics GeoSER to this MILP based expansion applied after running SER, which we denote as **MILPSER**.

Finally, we implement a tuning algorithm called **Auto** that chooses the best performing TL algorithm among TgtOnly, STRUT and SER via cross-validation on the available target data, using F_1 as the scoring metric, and then applies geometric heuristics on top of the selected transfer algorithm. Note that Auto does not have access to the test data while choosing the TL algorithm, and so may not outperform the other algorithms in terms of F_1-score, due to factors such as overfitting on the the target domain data.

6 RESULTS

Our goal is to increase the recall of the model for class 1, while minimizing the loss in terms of precision. To ensure that, we examine the recall of class 1, precision of class 1, and F_1-score of the model. The latter metric is the harmonic mean of precision and recall (i.e., $2(\text{precision} \times \text{recall})/(\text{precision} + \text{recall})$) which summarizes the

[2]https://www.gurobi.com/

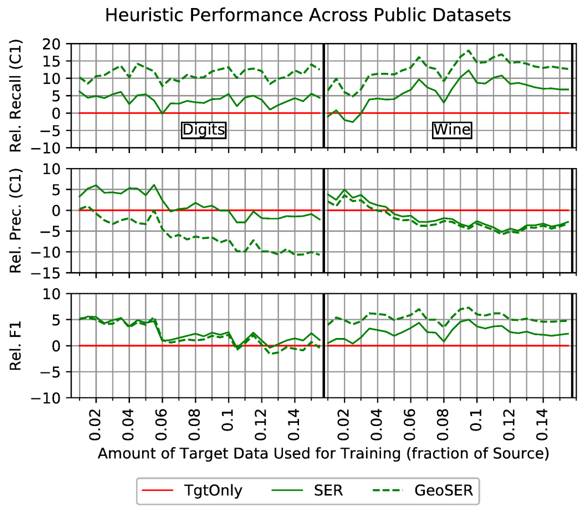


Figure 3: `Digits` and `Wine` datasets: The top chart shows the recall of class 1 (y-axis) for each heuristic, relative to the `TgtOnly` baseline, when the amount of target data available, as a fraction of the total source data, is varied (x-axis). Middle chart shows the relative precision of class 1 (y-axis) with the same x-axis. Finally, the bottom chart shows the *relative* difference between the F_1-score of `TgtOnly` and the other approaches (y-axis) as target data amount varies.

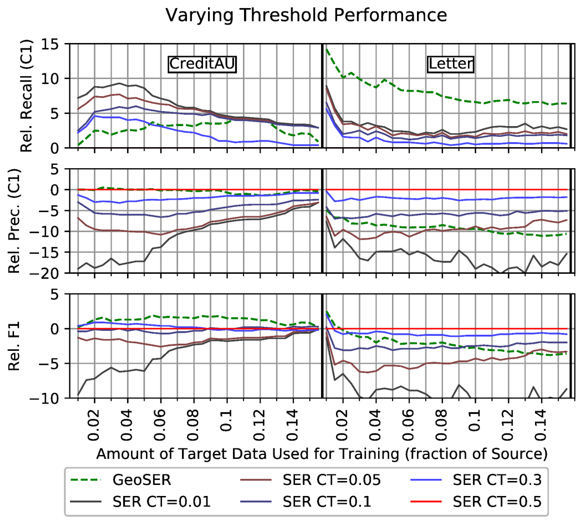


Figure 4: Effect of varying the classification threshold (CT) compared to our heuristic. Top: relative recall comparison to SER with default classification threshold (0.5). Middle: precision comparison. Bottom: relative F_1-score.

performance of the binary classifier for class 1, giving equal importance to recall and precision. Generalized scores like F_β would give more weight to recall for $\beta > 1$, and cast our results in a more favorable light. However, setting an appropriate β for each dataset can be confusing in a multi-dataset comparison, and the choice of β hard to justify for the public datasets. Thus, we opt to show F_1 to give a uniform evaluation of our heuristics. Ideally the heuristics should increase, or at least not greatly reduce, the F_1-score, while improving recall.[3] We evaluate the following contexts: i) as we vary the fraction of target data available; ii) compared to another method of boosting the recall; iii) across the 13 datasets described; iv) compared to an MILP approach, and v) on top of random forests.

6.1 Varying the fraction of target data

Figure 3 shows the results of the `TgtOnly`, `SER` and `GeoSER` algorithms on two of the public datasets (`Digits` and `Wine`). In the figure we show the results for each of the metrics (recall, precision, and F_1-score) with respect to `TgtOnly`, and for different amounts of data used from the target domain. Results above the `TgtOnly` line indicate that the model transfer approaches are providing superior performance compared to training only on the available target data. The figure shows that whenever the amount of target data is below a 0.1 fraction of the source data, both `SER` and `GeoSER` have better F_1 scores than `TgtOnly`. This indicates firstly that there is some potential gain by using these transfer learning methods when target data is scarce. Secondly, we see that there is a distinct trade-off in terms of recall and precision: as expected, `SER` tends to have higher precision, and lower recall than `GeoSER`. Thirdly, this trade-off is relatively stable as the fraction of target data varies. In particular, we observe that the gain in recall (resp. loss in precision) is 7.7 (resp. 6.9) on average for `Digits` dataset, and 7.1 (resp. 1.0) on average for the `Wine` dataset, and the actual trade-off is close to this as the fraction of target data is varied. Finally, we remark that by adjusting δ to a smaller value e.g., 0.05 we get similar behavior though with less of a difference between recall and precision.

6.2 Varying the classification threshold

Reducing the classification threshold is a natural way to boost recall at the cost of precision. In Figure 4, we present two datasets that are representative of the two types of trends we observed: other datasets showed qualitatively similar trends.

In the `CreditAU` dataset, `GeoSER` gives competitive recall values when the fraction of target data used is above 0.1. For lower fractions, setting the classification threshold to lower than 0.3 yields higher recall. However, there is a drop in the F_1-score for these settings of the threshold, whereas `GeoSER` has higher F_1-score.

On the other hand, in the `Letter` dataset, we find that `GeoSER` gives higher recall even compared to the heuristic which sets the classification threshold to as low as 0.01, with less loss in precision. We conclude that the trade-offs offered by our heuristic differ from those offered by adjusting the classification threshold.

6.3 Comparison across all datasets

Table 2 records the recall and F_1-scores, for a target dataset that contains 5% of the samples as the source dataset, for each algorithm. In terms of recall, we observe from Table 2 that for each dataset, the

[3]Since our expansion method cannot decrease recall, the change in the F_β score after applying our method would only be greater, for $\beta > 1$, than the change with respect to the F_1 score.

Table 2: Recall and F_1-score values for class 1 when the target data size is 5% of the source data size. Except `Auto` and `FEDA`, every other heuristic `X` has 2 sub-columns, the first corresponding to `X` itself (e.g. `STRUT`) whereas the second containing values for `GeoX` (e.g. `GeoSTRUT`). The highest values of recall and F_1-score appear in bold. For instance, for the dataset `Letter`, `GeoSER` gives the highest recall of 84.3, whereas the highest F_1-score of 75.4 is exhibited by `FEDA`.

	Recall										F_1-score									
Dataset	Src Only		Tgt Only		STRUT		SER		Auto	FEDA	Src Only		Tgt Only		STRUT		SER		Auto	FEDA
	X	GeoX	X	GeoX	X	GeoX	X	GeoX			X	GeoX	X	GeoX	X	GeoX	X	GeoX		
CreditAU	80.4	87.2	70.4	77.4	84.0	86.1	83.1	86.4	**87.5**	83.3	83.7	86.6	74.7	78.3	83.5	84.4	85.0	**86.6**	86.0	83.8
Wine	36.1	65.1	66.6	81.8	69.4	80.5	70.6	77.7	**83.1**	69.4	47.0	67.1	69.1	73.3	69.6	72.8	71.0	74.0	**74.1**	70.7
Letter	50.3	72.0	75.0	80.0	60.9	67.5	74.8	**84.3**	77.7	72.5	52.5	61.1	75.3	72.3	67.5	69.0	71.4	69.7	70.5	**75.4**
Digits	0.0	0.0	63.7	71.4	35.0	38.8	69.1	**76.8**	73.5	0.0	0.0	0.0	64.3	64.6	38.6	40.3	**68.7**	68.3	66.1	0.0
Landmine	2.9	10.3	25.9	31.7	22.1	27.9	23.7	**34.8**	32.0	12.9	3.7	5.3	24.5	22.3	**25.0**	23.9	23.0	22.3	22.9	17.8
CreditDE	37.0	45.1	42.0	**46.1**	35.2	39.5	28.5	29.0	35.8	37.0	39.6	42.7	44.4	**46.2**	39.9	42.3	35.9	35.7	40.0	43.2
Invert	0.0	0.0	67.7	77.5	49.5	54.8	29.3	38.5	**80.0**	62.6	0.0	0.0	**62.2**	57.0	46.2	40.2	31.1	30.9	60.2	61.9
Network 1	43.4	**52.6**	34.0	38.2	29.5	38.7	19.3	34.5	35.0	34.1	39.1	39.7	37.5	38.2	36.3	35.7	24.7	29.3	33.4	**40.8**
Network 2	63.8	64.4	52.1	55.7	55.8	**64.6**	50.9	55.3	61.2	51.0	43.8	42.7	55.6	55.2	**60.7**	59.9	53.6	51.0	56.4	53.2
Network 3	42.0	42.4	52.6	57.9	53.8	59.1	51.4	59.0	**61.1**	60.6	36.5	36.3	58.0	57.0	**66.8**	65.4	52.8	51.5	62.5	58.7
Network 4	73.8	**84.3**	67.4	73.4	36.1	58.7	73.1	83.9	76.0	72.7	50.0	49.2	68.5	66.0	39.5	52.2	69.1	59.1	62.3	**71.5**
Network 5	67.1	**73.2**	45.3	51.3	33.6	48.6	54.1	63.8	56.4	61.3	47.0	46.3	50.7	49.7	40.3	44.1	50.7	45.5	46.6	**59.2**
Network 6	75.4	75.4	70.8	75.6	64.4	74.4	78.4	**82.6**	80.4	62.1	34.3	34.3	71.7	68.3	70.8	69.4	**78.9**	76.4	74.3	69.2

`GeoX` heuristics are top performers, but that the algorithm `X` varies depending on the dataset. On the public datasets the `Auto` heuristic has the best recall in 3 cases, and is close to the top score in 2 others (`Digits` and `Landmine`). Since these are the mean scores over 20 runs, with a different 5% of the target data used for training during each run, `Auto` can exceed the scores of other algorithms.

For F_1-score, we observed that `SrcOnly` always performs worse than `TgtOnly` after the fraction of target data exceeds 0.1 for both the public and network datasets. Moreover, with notable exceptions (`Letter` and `Invert`), there is a TL heuristic that beats `TgtOnly` for most fractions of target data in terms of F_1-score on the public datasets. This shows positive transfer is possible for most of the public datasets. For the network datasets, we find `GeoSrcOnly` achieves high recall scores on 4 of the 6 datasets, but never wins in terms of F_1. For these datasets, `FEDA` is consistently among the top in terms of F_1-score, but `STRUT` and `SER` also perform well. We note that `GeoSTRUT` often beats `STRUT` in terms of F_1, while `Auto` is consistently competitive in terms of recall and F_1.

Table 3 records the increase in recall after applying the geometric heuristic to each TL algorithm. Note that for every baseline decision tree, applying our geometric heuristic only increases the recall. Furthermore, for most public datasets, the F_1-score increases for both `TgtOnly` and `STRUT`, while `SER` does not drop by more than 1.7. For the network datasets, the F_1-scores are somewhat lower, especially for `GeoSER`, but `Auto` has consistently good performance in terms of recall and F_1. Thus, we conclude that there can be general benefits to applying these techniques on top of TL algorithms, especially when recall is of more importance than precision. Moreover, automatic selection of the TL algorithm using an approach like `Auto` is prudent, since it is unknown which algorithm will work best.

Finally, Tables 4 and 5 show standard errors of the heuristics for both recall and precision, respectively, over the 20 runs. The small standard errors are indicative of scores tightly centered around the mean, implying robustness of our methodology.

Table 3: Change in recall, precision and F_1-score of baseline `X` compared with `GeoX` when the target size is 5% of the source size. For instance, the first entry of the table indicates that for the CreditAU dataset, averaged over 20 runs, (recall using `GeoTgtOnly`) - (recall using `TgtOnly`) = 7.0

	ΔRecall			ΔPrecision			ΔF_1-score		
Dataset	TgtOnly	STRUT	SER	TgtOnly	STRUT	SER	TgtOnly	STRUT	SER
CreditAU	7.0	2.1	3.3	-1.8	-0.4	-0.1	3.6	0.9	1.6
Wine	15.2	11.1	7.1	-6.0	-3.8	-1.0	4.2	3.2	3.0
Letter	5.0	6.6	9.5	-9.7	-5.6	-8.9	-3.0	1.5	-1.7
Digits	7.7	3.8	7.7	-6.0	-1.7	-6.9	0.3	1.7	-0.4
Landmine	5.8	5.8	11.1	-6.1	-8.8	-6.2	-2.2	-1.1	-0.7
CreditDE	4.1	4.3	0.5	-0.4	-0.5	-2.3	1.8	2.4	-0.2
Invert	9.8	5.3	9.2	-12.1	-13.2	-10.6	-5.2	-6.0	-0.2
Network 1	4.2	9.2	15.2	-4.0	-15.6	-14.9	0.7	-0.6	4.6
Network 2	3.6	8.8	4.4	-6.1	-11.7	-10.0	-0.4	-0.8	-2.6
Network 3	5.3	5.3	7.6	-8.6	-16.2	-9.4	-1.0	-1.4	-1.3
Network 4	6.0	22.6	10.8	-10.3	-2.2	-20.0	-2.5	12.7	-10.0
Network 5	6.0	15.0	9.7	-9.9	-13.1	-12.6	-1.0	3.8	-5.2
Network 6	4.8	10.0	4.2	-10.6	-14.2	-8.0	-3.4	-1.4	-2.5

6.4 Comparison to MILP optimal solution

Table 6 compares `MILPSER` and `GeoSER` with respect to recall, precision and F_1 for a target data set that is 5% of the source data size. Overall, `MILPSER` has recall similar to `GeoSER` in most cases. Note that it is not guaranteed by the optimality of the rectangle, that `MILPSER` has higher recall than `GeoSER`. `Invert` was the only dataset where the `MILPSER` greatly improved the recall (by 11.2) compared to `GeoSER`. However, for a larger fraction of target data (10%) the discrepancy for `Invert` was not as large (only 2.1), while

Table 4: Standard Errors for Recall over 20 runs.

Dataset	SrcOnly	GeoSrcOnly	TgtOnly	GeoTgtOnly	STRUT	GeoSTRUT	SER	GeoSER
CreditAU	0.0	0.01	0.04	0.04	0.02	0.02	0.01	0.01
Wine	0.0	0.01	0.02	0.02	0.02	0.01	0.02	0.02
Letter	0.0	0.01	0.01	0.01	0.02	0.02	0.01	0.01
Digits	0.0	0.0	0.02	0.02	0.04	0.04	0.02	0.02
Landmine	0.0	0.03	0.01	0.01	0.01	0.01	0.01	0.01
CreditDE	0.0	0.03	0.02	0.03	0.02	0.02	0.01	0.01
Invert	0.0	0.0	0.04	0.03	0.08	0.09	0.02	0.03

Table 5: Standard Errors for Precision over 20 runs.

Dataset	SrcOnly	GeoSrcOnly	TgtOnly	GeoTgtOnly	STRUT	GeoSTRUT	SER	GeoSER
CreditAU	0.0	0.01	0.03	0.02	0.01	0.01	0.0	0.0
Wine	0.0	0.0	0.01	0.01	0.01	0.01	0.01	0.01
Letter	0.0	0.01	0.01	0.01	0.02	0.02	0.01	0.01
Digits	0.0	0.0	0.02	0.01	0.04	0.04	0.01	0.01
Landmine	0.0	0.0	0.01	0.01	0.01	0.01	0.01	0.0
CreditDE	0.0	0.0	0.02	0.02	0.01	0.01	0.02	0.02
Invert	0.0	0.0	0.02	0.02	0.04	0.06	0.02	0.01

the recall on the other datasets were similar, indicating these differences are unstable as the amount of target data varies.

In Table 7 we show the average running times (over 20 runs) of these two heuristics, along with the multiplicative speedup obtained by using GeoSER over MILPSER. In many cases, GeoSER offers a large speedup over MILPSER, often greater than a factor of 100 when the target dataset contains a few hundred points. In the extreme case of the Wine dataset, the MILP solver did not terminate within 8 hours, so we can only estimate the speedup. Importantly, the Wine dataset is relatively balanced, meaning that there are several hundred class 1 samples in the target dataset. Wine is unique in this sense, as the other datasets are highly imbalanced. This shows an issue of scalability with the MILPSER solution. Finally, there are a few exceptional cases where MILPSER is faster than GeoSER. However, in these cases the running times of both approaches are between 50-200 milliseconds. Overall, based on these results, we conclude that GeoSER consistently outperforms MILPSER in terms of running time, while providing similar recall-precision trade-offs.

Table 6: MILP results (target size is 5% of source).

	Recall			Precision			F_1-score		
Dataset	SER	MILP SER	Geo SER	SER	MILP SER	Geo SER	SER	MILP SER	Geo SER
CreditAU	83.1	88.2	86.4	87.1	85.5	87.0	85.0	86.8	86.6
Wine	70.6	?	77.7	72.3	?	71.3	71.0	?	74.0
Letter	74.8	83.2	84.3	68.5	60.5	59.6	71.4	69.9	69.7
Digits	69.1	74.5	76.8	68.9	60.5	62.0	68.7	66.5	68.3
Landmine	23.7	33.4	34.8	22.7	16.4	16.5	23.0	22.0	22.3
CreditDE	28.5	33.1	29.0	49.5	46	47.2	35.9	38.1	35.7
Invert	29.3	49.7	38.5	38.1	33.1	27.5	31.1	38.4	30.9

Table 7: Comparing the average running time of MILPSER with that of GeoSER in seconds. The Speedup column gives the multiplicative speedup of GeoSER over MILPSER. For example, for the Letter dataset, when the target size is 10% of source size, GeoSER is 178.94 times faster than MILPSER.

	Tgt size = 5% of Src size			Tgt size = 10% of Src size		
Dataset	MILP SER	Geo SER	Speedup	MILP SER	Geo SER	Speedup
CreditAU	0.062	0.054	1.14	0.13	0.10	1.3
Wine	> 8 hrs	0.23	>125819	> 8 hrs	0.32	>90851
Letter	24.27	0.49	49.53	216.52	1.21	178.94
Digits	0.45	0.10	4.5	2.81	0.32	8.78
Landmine	13.94	0.28	49.78	161.3	0.81	199.13
CreditDE	0.069	0.19	0.36	0.062	0.126	0.49
Invert	0.75	0.69	1.08	1.62	1.38	1.17

6.5 Random forests

Table 8 shows results analogous to Table 3, for random forests. With random forests, in most cases there is a greater increase in recall, but also a greater drop in precision. However, the recall gain more often than not can be considered to compensate for the precision loss: the change in F_1-score is typically positive. Moreover, the drop in F_1-score never exceeds 1.7% for any dataset, in constrast to a drop of 6% when using only decision trees, which occurs for the Invert dataset with STRUT.

Table 8: Random forest (30 trees; target size is 5% of source).

	ΔRecall			ΔPrecision			ΔF_1-score		
Dataset	TgtOnly	STRUT	SER	TgtOnly	STRUT	SER	TgtOnly	STRUT	SER
CreditAU	4.0	3.0	2.7	-1.5	-1.6	-0.4	1.4	0.7	1.1
Wine	17.8	10.7	14.8	-11.7	-5.7	-9.4	-0.2	1.9	0.3
Letter	7.2	3.5	12.1	-9.1	-6.5	-13.1	0.3	0.4	1.0
Digits	12.4	7.2	16.7	-4.6	-1.3	-5.1	6.9	11.0	11.3
Landmine	3.8	4.2	7.3	-15.7	-18.5	-35.8	-1.5	1.8	-1.7
CreditDE	9.7	7.2	5.3	-3.1	-2.2	-2.0	5.4	4.8	5.3
Invert	13.4	21.8	19.2	-17.1	-2.2	-21.4	-1.6	17.7	-0.1

7 CONCLUSION

We presented novel geometric heuristics to improve the recall of decision tree classifiers. These heuristics can be applied in the context of homogeneous transfer learning when we have access to only a small number of target data samples. Overall, our heuristics boost recall, while achieving competitive F_1-scores compared to state-of-the-art algorithms.

Boosting alternative objective functions, such as precision, or attempting to simultaneously improve metrics for both positive and negative classes, are interesting avenues for future work. We remark that our algorithms can be adapted to shrink the rectangles with similar running times, but that was not our aim as the focus was on improving recall. However, simultaneously expanding and shrinking the rectangles would require new algorithmic ideas.

REFERENCES

[1] 2020. https://github.com/TLDatasets/Datasets

[2] Reem Al-Otaibi, Ricardo B. C. Prudêncio, Meelis Kull, and Peter Flach. 2015. Versatile Decision Trees for Learning Over Multiple Contexts. In *Machine Learning and Knowledge Discovery in Databases*. Springer International Publishing, Cham, 184–199.

[3] Bogdan Armaselu and Ovidiu Daescu. 2017. Maximum Area Rectangle Separating Red and Blue Points. *CoRR* abs/1706.03268 (2017). arXiv:1706.03268

[4] Arthur Asuncion and David Newman. 2007. UCI machine learning repository.

[5] Jonathan Backer and J. Mark Keil. 2010. The Mono- and Bichromatic Empty Rectangle and Square Problems in All Dimensions. In *LATIN 2010: Theoretical Informatics*, Alejandro López-Ortiz (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 14–25.

[6] Arturs Backurs, Nishanth Dikkala, and Christos Tzamos. 2016. Tight Hardness Results for Maximum Weight Rectangles. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy (LIPIcs, Vol. 55)*, Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 81:1–81:13.

[7] Jérémy Barbay, Timothy M. Chan, Gonzalo Navarro, and Pablo Pérez-Lantero. 2014. Maximum-weight planar boxes in O(n^2) time (and better). *Inf. Process. Lett.* 114, 8 (2014), 437–445.

[8] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (1975), 509–517.

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

[10] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.

[11] Timothy M. Chan, Kasper Green Larsen, and Mihai Patrascu. 2011. Orthogonal range searching on the RAM, revisited. In *Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, June 13-15, 2011*, Ferran Hurtado and Marc J. van Kreveld (Eds.). ACM, 1–10.

[12] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. 785–794.

[13] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. 2009. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems* 47, 4 (2009), 547–553.

[14] Hal Daumé III. 2007. Frustratingly Easy Domain Adaptation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. Association for Computational Linguistics, Prague, Czech Republic, 256–263.

[15] Jonathan Eckstein, Peter L. Hammer, Ying Liu, Mikhail Nediak, and Bruno Simeone. 2002. The Maximum Box Problem and its Application to Data Analysis. *Comp. Opt. and Appl.* 23, 3 (2002), 285–298.

[16] Jeff Edmonds, Jarek Gryz, Dongming Liang, and Renée J Miller. 2003. Mining for empty spaces in large data sets. *Theoretical Computer Science* 296, 3 (2003), 435–452.

[17] Peter W Frey and David J Slate. 1991. Letter recognition using Holland-style adaptive classifiers. *Machine learning* 6, 2 (1991), 161–182.

[18] Jerzy W Grzymala-Busse, Linda K Goodwin, Witold J Grzymala-Busse, and Xinqun Zheng. 2004. An approach to imbalanced data sets based on changing rule strength. In *Rough-neural computing*. Springer, 543–553.

[19] Jerzy W Grzymala-Busse, Jerzy Stefanowski, and Szymon Wilk. 2005. A comparison of two approaches to data mining from imbalanced data. *Journal of Intelligent Manufacturing* 16, 6 (2005), 565–573.

[20] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter* 11, 1 (2009), 10–18.

[21] Xiyang Hu, Cynthia Rudin, and Margo Seltzer. 2019. Optimal sparse decision trees. In *Advances in Neural Information Processing Systems*. 7265–7273.

[22] Sebastian Kauschke and Johannes Fürnkranz. 2018. Batchwise Patching of Classifiers. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[23] Jun Won Lee and Christophe G. Giraud-Carrier. 2007. Transfer Learning in Decision Trees. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2007, Celebrating 20 years of neural networks, Orlando, Florida, USA, August 12-17, 2007*. 726–731.

[24] Bing Liu, Liang-Ping Ku, and Wynne Hsu. 1997. Discovering interesting holes in data. In *IJCAI (2)*. 930–935.

[25] Wolfgang Maass. 1994. Efficient agnostic pac-learning with simple hypothesis. In *Proceedings of the seventh annual conference on Computational learning theory*. ACM, 67–75.

[26] Sreerama K. Murthy. 1998. Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey. *Data Min. Knowl. Discov.* 2, 4 (1998), 345–389.

[27] S. J. Pan and Q. Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (Oct 2010), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

[28] J. Ross Quinlan. 1987. Simplifying decision trees. *International journal of man-machine studies* 27, 3 (1987), 221–234.

[29] Erik Rodner and Joachim Denzler. 2009. Learning with few examples by transferring feature relevance. In *Joint Pattern Recognition Symposium*. Springer, 252–261.

[30] Steven Salzberg. 1991. A nearest hyperrectangle learning method. *Machine Learning* 6, 3 (01 May 1991), 251–276. https://doi.org/10.1007/BF00114779

[31] N. Segev, M. Harel, S. Mannor, K. Crammer, and R. El-Yaniv. 2017. Learn on Source, Refine on Target: A Model Transfer Learning Framework with Random Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39, 9 (Sep. 2017), 1811–1824. https://doi.org/10.1109/TPAMI.2016.2618118

[32] Jerzy Stefanowski and Daniel Vanderpooten. 2001. Induction of decision rules in classification and discovery-oriented perspectives. *International Journal of Intelligent Systems* 16, 1 (2001), 13–27.

[33] Sicco Verwer and Y. Zhang. 2019. Learning optimal classification trees using a binary linear program formulation. *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)* AAAI-19 (2019), 1625,1632.

[34] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. 2016. A survey of transfer learning. *Journal of Big data* 3, 1 (2016), 9.