Gradient Tree Boosting for Regression Transfer

Anonymous authors
Paper under double-blind review

Abstract

Many real-world modeling problems are hindered by limited data availability. In such cases, *transfer learning* leverages related source domains to improve predictions in a target domain of interest. We extend the classical gradient tree boosting paradigm to a regression transfer algorithm by modeling the weak learner as a sum of two regression trees. The trees are fitted on source data and target data, respectively, and jointly optimized for the target data. We derive optimal coefficients for the model update under the least-squares, the least-absolute-deviation, and the Huber loss functions. We benchmark our approach against the widely used XGBoost algorithm in several transfer scenarios, achieving superior performance in seven out of eight cases.

1 Introduction

Transfer learning aims to improve a learner in a *target* domain by transferring information from similar but different *source* domains (Weiss et al., 2016). Typically, the target domain data is limited, so that external information from source datasets can be used to improve the performance of the learner.

Deep learning has become the dominant approach to tackle complex machine learning tasks, with impressive results in areas such as large language modeling, computer vision, and image generation. However, for tabular datasets, gradient boosting methods are often competitive in terms of performance (McElfresh et al., 2023; Grinsztajn et al., 2022). Moreover, they generally require shorter training times and offer greater interpretability than neural networks, making them a viable option for tasks involving tabular datasets.

Unlike neural networks, which enable transfer learning by fine-tuning learned representations, gradient boosting does not naturally support such adaptation. Consequently, gradient boosting techniques for transfer learning have traditionally been instance-based, operating on the union of the source and target datasets while achieving adaptation by reweighting the importance of source and target data samples. These approaches include TrAdaBoost (Dai et al., 2007), a modification of AdaBoost (Freund and Schapire, 1997) that was introduced for transfer learning in classification tasks. TrAdaBoost was adapted in Pardoe and Stone (2010) to handle regression settings, leading to algorithms such as TrAdaBoost.R2 and two-stage TrAdaBoost.R2. More recent gradient boosting algorithms, such as XGBoost (Chen and Guestrin, 2016), have also been adapted to transfer learning scenarios (Fang et al., 2019; Liu et al., 2020).

Our contribution is a model-based approach for regression transfer using gradient tree boosting. The base learner is modeled as a sum of two regression trees, one fitted on the target dataset and one fitted on the source dataset. We optimize this base learner toward the target dataset with the idea that structural information from the source tree can be used to support transfer learning.

We start by reviewing the classical gradient tree boosting paradigm (for regression tasks) as outlined in (Friedman, 2001) (Section 2). We then extend the theory to include a sum of two regression trees as the base learner, deriving optimal coefficients for the model update at each iteration for the least-squares, least-absolute-deviation, and Huber losses (Section 3). In Section 4, we perform a simulation study for our algorithm under these three loss functions. In Section 5, we benchmark our algorithm in eight transfer scenarios on real data. Finally, in Section 6 we discuss the key findings and outline future work.

2 Preliminaries

Gradient boosting performs numerical optimization in function space by sequentially fitting weak learners to minimize a differentiable loss function (Friedman, 2001; Mason et al., 1999). For a loss function L, we try to find the function $F(\mathbf{x})$ that minimizes the joint expectation given by

$$\mathbb{E}_{y,\mathbf{x}}\left[L(y,F(\mathbf{x}))\right] = \mathbb{E}_{\mathbf{x}}\left[\mathbb{E}_{y}\left[L(y,F(\mathbf{x}))\mid\mathbf{x}\right]\right] = \mathbb{E}_{y}\left[L(y,F(\mathbf{x}))\mid\mathbf{x}\right].$$

Following the numerical optimization paradigm, the optimal solution is taken to be

$$F^{*}(\mathbf{x}) = \sum_{m=0}^{M} f_m(\mathbf{x}),$$

where f_0 is an initial guess, f_m is an improvement step, or *boost*, computed according to the optimization method, and M denotes the number of boosts.

For steepest-descent, $f_m(\mathbf{x}) = -\rho_m g_m(\mathbf{x})$ where $-g_m(\mathbf{x})$ is the negative gradient evaluated at $F_{m-1}(\mathbf{x}) = \sum_{i=0}^{m-1} f_i(\mathbf{x})$:

$$\begin{aligned} -g_m(\mathbf{x}) &= -\left[\frac{\partial \mathbb{E}_{y}\left[L(y,F(\mathbf{x}))\mid\mathbf{x}\right]}{\partial F(\mathbf{x})}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \\ &= -\mathbb{E}_{y}\left[\frac{\partial\left[L(y,F(\mathbf{x}))\right]}{\partial F(\mathbf{x})}\mid\mathbf{x}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \end{aligned}$$

and ρ_m is an optimal step size, found via

$$\rho_m = \arg\min_{\rho} \mathbb{E}_{y}\left[L(y,F_{m-1}(\mathbf{x}) - \rho g_m(\mathbf{x})\right].$$

For finite data samples $\{\mathbf{x}_i, y_i\}_1^N$, we may consider updates using the data-based negative gradient given by

$$-g_m(\mathbf{x}_i) = -\left[\frac{\partial L(y_i,F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}.$$

However, the data-based gradient does not generalize to new, unseen data. A solution to this is to fit a parameterized function $h(\mathbf{x};\mathbf{a}_m)$, often referred to as *base learner*, to the data-based negative gradient using a least-squares approximation. The function $F_m(\mathbf{x})$ can then be updated every iteration using this "generalized" version of the data-based negative gradient as

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x};\mathbf{a}_m).$$

A J-terminal node regression tree partitions the input space into finitely many disjoint regions $\{R_j\}_1^J$ and assigns a constant b_j to each region (typically the mean of the responses in that region). When the base learner is a J-terminal node regression tree, it can be expressed as

$$h(\mathbf{x};\mathbf{a}_m) = h(\mathbf{x};\{b_{jm},R_{jm}\}_1^J) = \sum_{j=1}^{J} b_{jm}\mathbf{1}(\mathbf{x}\in R_{jm}).$$

The update is then given by

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m \sum_{j=1}^{J} b_{jm}\mathbf{1}(\mathbf{x}\in R_{jm}). \tag{1}$$

In this case, the fit can be improved by considering individual optimal coefficients $\{\gamma_{jm}\}_1^J$ instead of an optimal global step size ρ_m, rewriting equation 1 as

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jm}\mathbf{1}(\mathbf{x}\in R_{jm}), \tag{2}$$

where $\gamma_{jm} = \rho_m b_{jm}$. These optimal coefficients are given by

$$\{\gamma_{jm}\}_1^J = \arg\min_{\{\gamma_j\}_1^J} \sum_{i=1}^N L\left(y_i, F_{m-1}\left(\mathbf{x}_i\right) + \sum_{j=1}^J \gamma_j \mathbf{1}(\mathbf{x}_i \in R_{jm})\right).$$

Since we have disjoint regions, for each γ_{jm}, this reduces to

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{jm}} L\left(y_i, F_{m-1}\left(\mathbf{x}_i\right) + \gamma\right).$$

The region-wise optimal coefficients are easily derived in closed form for the least-squares loss and for the least-absolute-deviation loss; for the Huber loss, they can be approximated using standard iterative methods (Friedman, 2001).

To avoid overfitting, the fitted tree is often regularized with a *shrinkage parameter*, or *learning rate* ν. Putting all of this together, we get the algorithm for gradient tree boosting, outlined in Algorithm 1.

Algorithm 1 TreeBoost

1: $F_0(\mathbf{x}) = c_0 = \arg\min_c \sum_{i=1}^N L(y_i, c)$
2: **for** $m = 1, \ldots, M$ **do**
3: $\quad \tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x}) = F_{m-1}(\mathbf{x})}, \quad i = 1, \ldots, N$
4: $\quad \{R_{jm}\}_1^J = J$-node regression tree $\left(\{\mathbf{x}_i, \tilde{y}_i\}_1^N\right)$
5: $\quad \gamma_{jm} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{jm}} L\left(y_i, F_{m-1}\left(\mathbf{x}_i\right) + \gamma\right), \quad j = 1, \ldots, J$
6: $\quad F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \sum_{j=1}^J \gamma_{jm} \mathbf{1}(\mathbf{x} \in R_{jm})$
7: **end for**

3 Sum of trees as base learner

We consider our base learner to be the sum of $S \geq 2$ regression trees, where each tree is either constructed from a source dataset, or the target dataset. However, to simplify the exposition in the following section, we have chosen to study the case of $S = 2$ (*i.e.* a single-source dataset). The generalized version for $S \geq 2$ is covered in Appendix C. The reasoning for the case of $S \geq 2$ mirrors the arguments in this section.

To enable transfer learning, we consider the sum of two regression trees as the base learner

$$h(\mathbf{x}; \{b_{jm}, R_{jm}\}_1^J, \{\hat{b}_{km}, \hat{R}_{km}\}_1^K) = \sum_{j=1}^J b_{jm} \mathbf{1}(\mathbf{x} \in R_{jm}) + \sum_{k=1}^K \hat{b}_{km} \mathbf{1}(\mathbf{x} \in \hat{R}_{km}),$$

where $\{b_{jm}, R_{jm}\}_1^J$ are constructed from the target dataset and $\{\hat{b}_{km}, \hat{R}_{km}\}_1^K$ from the source dataset. Analogous to equation 1, and equation 2, the update at iteration m can be improved using individual optimal coefficients. In other words, we consider updates according to

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^J \gamma_{jm} \mathbf{1}(\mathbf{x} \in R_{jm}) + \sum_{k=1}^K \hat{\gamma}_{km} \mathbf{1}(\mathbf{x} \in \hat{R}_{km}),$$

where $\gamma_{jm} = \rho_m b_{jm}$ and $\hat{\gamma}_{km} = \rho_m \hat{b}_{km}$. For each iteration m, the individual optimal coefficients are given by

$$\{\{\gamma_{jm}\}_1^J, \{\hat{\gamma}_{km}\}_1^K\} = \arg\min_{\{\{\gamma_j\}_1^J, \{\hat{\gamma}_k\}_1^K\}} \sum_{i=1}^N L\left(y_i, F_{m-1}(\mathbf{x}_i) + \sum_{j=1}^J \gamma_j \mathbf{1}(\mathbf{x}_i \in R_{jm}) + \sum_{k=1}^K \hat{\gamma}_k \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km})\right). \quad (3)$$

We further consider an additional tunable *weight parameter* α_m, determining the contribution from each tree. This yields the following update for F_m:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + (1-\alpha_m)\sum_{j=1}^{J}\gamma_{jm}\mathbf{1}(\mathbf{x}\in R_{jm}) + \alpha_m\sum_{k=1}^{K}\hat{\gamma}_{km}\mathbf{1}(\mathbf{x}\in \hat{R}_{km}), \tag{4}$$

where $\alpha_m \in [0,1]$ may depend on the iteration m. This allows both for static and decaying α_m, implying that we can control the learning rates for the two terms of the base learner.

Combining all of this, we get our transfer learning algorithm, outlined in Algorithm 2.

Algorithm 2 TransferTreeBoost

Target data $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$, source data $\{\hat{\mathbf{x}}_i, \hat{y}_i\}_{i=1}^{\hat{N}}$
$F_0(\mathbf{x}) = c_0 = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$
for $m = 1, \ldots, M$ **do**
  $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \quad i = 1, \ldots, N$
  $\tilde{\hat{y}}_i = -\left[\frac{\partial L(\hat{y}_i, F(\hat{\mathbf{x}}_i))}{\partial F(\hat{\mathbf{x}}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \quad i = 1, \ldots, \hat{N}$
  $\{R_{jm}\}_1^J = J$-node regression tree $\left(\{\mathbf{x}_i, \tilde{y}_i\}_1^N\right)$
  $\{\hat{R}_{km}\}_1^K = K$-node regression tree $\left(\{\hat{\mathbf{x}}_i, \tilde{\hat{y}}_i\}_1^{\hat{N}}\right)$

$$\{\{\gamma_{jm}\}_1^J, \{\hat{\gamma}_{km}\}_1^K\} = \underset{\{\{\gamma_j\}_1^J, \{\hat{\gamma}_k\}_1^K\}}{\arg\min} \sum_{i=1}^{N} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \sum_{j=1}^{J}\gamma_j\mathbf{1}(\mathbf{x}_i\in R_{jm}) + \sum_{k=1}^{K}\hat{\gamma}_k\mathbf{1}(\mathbf{x}_i\in \hat{R}_{km})\right)$$

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu\Big((1-\alpha_m)\sum_{j=1}^{J}\gamma_{jm}\,\mathbf{1}(\mathbf{x}\in R_{jm}) + \alpha_m\sum_{k=1}^{K}\hat{\gamma}_{km}\,\mathbf{1}(\mathbf{x}\in \hat{R}_{km})\Big)$$

end for

In (Mason et al., 1999) it was proven that, under mild assumptions, an equivalent formulation of Algorithm 1 will converge. In particular, the assumptions are that the loss function L is convex and has Lipschitz continuous derivative, that the learning rate ν is sufficiently small, and lastly that the base learner $h(\cdot, \mathbf{a}_m)$ maximizes $-\langle \nabla L(F_{m-1}), h(\cdot, \mathbf{a}_m)\rangle$. The following result indicates that our algorithm possesses the same convergence property as it is a relaxation of the optimization formulation in Algorithm 1.

Theorem 1. *TransferTreeBoost (Algorithm 2) admits a convergence rate bound that is superior or equal to that of standard TreeBoost (Algorithm 1).*

A proof of Theorem 1 is given in Appendix A. Moreover, we show in Appendix B that the sum of two regression trees of J and K regions can equivalently be described as a new regression over $J \cdot K$ intersected regions. This shows that the sum of two trees is not merely a heuristic construct, but can be described as a regression tree itself. However, this formulation does not allow for individual weighting of each tree, as formulated in equation 4.

3.1 Deriving optimal coefficients

We will consider three approaches: least-squares (LS) regression, least-absolute-deviation (LAD) regression, and M-regression with the Huber loss. We will refer to these methods as `LSTransferTreeBoost`, `LADTransferTreeBoost`, and `MTransferTreeBoost`, respectively. The region-wise optimal coefficients in equation 3 cannot be derived in closed form in any of these cases. However, we will provide recipes for how they can be estimated.

3.1.1 LS regression

Let B denote our objective, such that $\arg\min B$ corresponds to RHS of equation 3, *i.e.*

$$B \coloneqq \sum_{i=1}^{N} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \sum_{j=1}^{J} \gamma_j \mathbf{1}(\mathbf{x}_i \in R_{jm}) + \sum_{k=1}^{K} \hat{\gamma}_k \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km})\right)$$

which, by symmetry, can be written as the following two double sums

$$\begin{aligned} B &= \sum_{j=1}^{J} \sum_{\mathbf{x}_i \in R_{jm}} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \gamma_j + \sum_{k=1}^{K} \hat{\gamma}_k \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km})\right) \\ &= \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in \hat{R}_{km}} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \sum_{j=1}^{J} \gamma_j \mathbf{1}(\mathbf{x}_i \in R_{jm}) + \hat{\gamma}_k\right). \end{aligned}$$

By differentiating B w.r.t. γ_j and $\hat{\gamma}_k$ for each j and k, respectively, we get

$$\frac{\partial B}{\partial \gamma_j} = \sum_{\mathbf{x}_i \in R_{jm}} \frac{\partial L}{\partial \gamma_j}, \quad \text{and} \quad \frac{\partial B}{\partial \hat{\gamma}_k} = \sum_{\mathbf{x}_i \in \hat{R}_{km}} \frac{\partial L}{\partial \hat{\gamma}_k}.$$

If L is the least-squares loss, we need the following conditions to hold for optimal values of $\{\{\gamma_{jm}\}_1^J, \{\hat{\gamma}_{km}\}_1^K\}$:

$$\frac{\partial B}{\partial \gamma_j} = \sum_{\mathbf{x}_i \in R_{jm}} \left(y_i - F_{m-1}(\mathbf{x}_i) - \gamma_j - \sum_{k=1}^{K} \hat{\gamma}_k \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km})\right) = 0,$$

and

$$\frac{\partial B}{\partial \hat{\gamma}_k} = \sum_{\mathbf{x}_i \in \hat{R}_{km}} \left(y_i - F_{m-1}(\mathbf{x}_i) - \hat{\gamma}_k - \sum_{j=1}^{J} \gamma_j \mathbf{1}(\mathbf{x}_i \in R_{jm})\right) = 0.$$

Let $|R_{jm}|$ and $|\hat{R}_{km}|$ denote the amount of datapoints contained in regions R_{jm} and $\hat{R}_{km}$, respectively, and let $N_{R_{jm} \cap \hat{R}_{km}} = N_{\hat{R}_{km} \cap R_{jm}}$ denote the amount of datapoints in the intersection of R_{jm} and $\hat{R}_{km}$. Furthermore, put

$$\mathbf{R} \coloneqq \begin{bmatrix} |R_{1m}| & 0 & \cdots & 0 \\ 0 & |R_{2m}| & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & |R_{Jm}| \end{bmatrix}, \; \widehat{\mathbf{R}} \coloneqq \begin{bmatrix} |\hat{R}_{1m}| & 0 & \cdots & 0 \\ 0 & |\hat{R}_{2m}| & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & |\hat{R}_{Km}| \end{bmatrix},$$

$$\mathbf{N} \coloneqq \begin{bmatrix} N_{R_{1m} \cap \hat{R}_{1m}} & N_{R_{1m} \cap \hat{R}_{2m}} & \cdots & N_{R_{1m} \cap \hat{R}_{Km}} \\ N_{R_{2m} \cap \hat{R}_{1m}} & N_{R_{2m} \cap \hat{R}_{2m}} & \cdots & N_{R_{2m} \cap \hat{R}_{Km}} \\ \vdots & \vdots & \ddots & \vdots \\ N_{R_{Jm} \cap \hat{R}_{1m}} & N_{R_{Jm} \cap \hat{R}_{2m}} & \cdots & N_{R_{Jm} \cap \hat{R}_{Km}} \end{bmatrix}, \text{ and } \quad \mathbf{r} \coloneqq \begin{bmatrix} \sum_{\mathbf{x}_i \in R_{1m}} y_i - F_{m-1}(\mathbf{x}) \\ \sum_{\mathbf{x}_i \in R_{2m}} y_i - F_{m-1}(\mathbf{x}) \\ \vdots \\ \sum_{\mathbf{x}_i \in R_{Jm}} y_i - F_{m-1}(\mathbf{x}) \\ \sum_{\mathbf{x}_i \in \hat{R}_{1m}} y_i - F_{m-1}(\mathbf{x}) \\ \sum_{\mathbf{x}_i \in \hat{R}_{2m}} y_i - F_{m-1}(\mathbf{x}) \\ \vdots \\ \sum_{\mathbf{x}_i \in \hat{R}_{Km}} y_i - F_{m-1}(\mathbf{x}) \end{bmatrix}.$$

Now, combine $\mathbf{N}$, $\mathbf{R}$, and $\widehat{\mathbf{R}}$ into a block matrix

$$\mathbf{M} \coloneqq \begin{bmatrix} \mathbf{R} & \mathbf{N} \\ \mathbf{N}^\intercal & \widehat{\mathbf{R}} \end{bmatrix}. \tag{5}$$

The optimal coefficients can thus be found by solving the system of equations given by

$$\mathbf{M}\begin{bmatrix}\boldsymbol{\gamma}\\ \hat{\boldsymbol{\gamma}}\end{bmatrix} = \mathbf{r}. \tag{6}$$

The following proposition shows that $\mathbf{M}$ is singular. Consequently, we will rely on least-squares approximations to obtain estimates. The singularity of $\mathbf{M}$ arises because the union of regions in both the target and source trees covers the same ambient space, which creates a linear dependence in the block matrix. An alternative solution, which may yield unique solutions, is to introduce a small ridge regularization.

Proposition 1. *Let* $\mathbf{M}$ *be defined as in equation 5. Then* $\mathbf{M}$ *is singular.*

A proof is given in Appendix A.

3.1.2 LAD regression

Let the loss L in equation 3 be the least-absolute-deviation loss. Then the optimal coefficients at iteration m are given by

$$\{\{\gamma_{jm}\}_1^J, \{\hat{\gamma}_{km}\}_1^K\} = \underset{\{\{\gamma_j\}_1^J, \{\hat{\gamma}_k\}_1^K\}}{\arg\min} \sum_{i=1}^{N} \Big| y_i - F_{m-1}(\mathbf{x}_i) - \sum_{j=1}^{J} \gamma_j \mathbf{1}(\mathbf{x}_i \in R_{jm}) - \sum_{k=1}^{K} \hat{\gamma}_k \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km}) \Big|.$$

Now, introduce auxiliary slack variables defined by

$$t_i \coloneqq \Big| y_i - F_{m-1}(\mathbf{x}_i) - \sum_{j=1}^{J} \gamma_j \mathbf{1}(\mathbf{x}_i \in R_{jm}) - \sum_{k=1}^{K} \hat{\gamma}_k \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km}) \Big|, \quad i = 1, \ldots, N. \tag{7}$$

Then we can formulate the search for the optimal coefficients as a linear programming (LP) problem as follows

$$\begin{aligned} &\min \sum t_i \\ &\text{s.t.} \quad t_i \geq \Big| y_i - F_{m-1}(\mathbf{x}_i) - \sum_{j=1}^{J} \gamma_j \mathbf{1}(\mathbf{x}_i \in R_{jm}) - \sum_{k=1}^{K} \hat{\gamma}_k \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km}) \Big|, \quad i = 1, \ldots, N. \end{aligned} \tag{8}$$

This is an optimization problem with $N + J + K$ unknown variables and $2N$ linear constraints. Let $\mathbf{I}_N$ be the identity matrix of size N, $\mathbf{c}_N$ be the identity vector of length N, $\mathbf{t}^\intercal = \begin{bmatrix} t_1 t_2 \ldots t_N \end{bmatrix}$ be the transpose of the vector to the N slack variables, and $(\mathbf{t}\gamma\hat{\gamma})^\intercal = \begin{bmatrix} \mathbf{t}^\intercal \gamma_1 \gamma_2 \ldots \gamma_J \hat{\gamma}_1 \hat{\gamma}_2 \ldots \hat{\gamma}_K \end{bmatrix}$ be the transpose of the vector consisting of all decision variables (of length $N + J + K$). Furthermore, put

$$\boldsymbol{\Gamma} \coloneqq \begin{bmatrix} \mathbf{1}(\mathbf{x}_1 \in R_{1m}) & \cdots & \mathbf{1}(\mathbf{x}_1 \in R_{Jm}) & \mathbf{1}(\mathbf{x}_1 \in \hat{R}_{1m}) & \cdots & \mathbf{1}(\mathbf{x}_1 \in \hat{R}_{Km}) \\ \mathbf{1}(\mathbf{x}_2 \in R_{1m}) & \cdots & \mathbf{1}(\mathbf{x}_2 \in R_{Jm}) & \mathbf{1}(\mathbf{x}_2 \in \hat{R}_{1m}) & \cdots & \mathbf{1}(\mathbf{x}_2 \in \hat{R}_{Km}) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{1}(\mathbf{x}_N \in R_{1m}) & \cdots & \mathbf{1}(\mathbf{x}_N \in R_{Jm}) & \mathbf{1}(\mathbf{x}_N \in \hat{R}_{1m}) & \cdots & \mathbf{1}(\mathbf{x}_N \in \hat{R}_{Km}) \end{bmatrix},$$

and

$$\mathbf{b} \coloneqq \begin{bmatrix} y_1 - F_{m-1}(\mathbf{x}_1) \\ y_2 - F_{m-1}(\mathbf{x}_2) \\ \vdots \\ y_N - F_{m-1}(\mathbf{x}_N) \end{bmatrix}.$$

We can write equation 8 in matrix form as

$$\begin{aligned} &\min \quad \mathbf{c}^\intercal \mathbf{t} \\ &\text{s.t.} \begin{bmatrix} -\mathbf{I}_N & \boldsymbol{\Gamma} \\ -\mathbf{I}_N & -\boldsymbol{\Gamma} \end{bmatrix} (\mathbf{t}\gamma\hat{\gamma}) \leq \begin{bmatrix} \mathbf{b} \\ -\mathbf{b} \end{bmatrix}. \end{aligned} \tag{9}$$

Thus, we conclude that the optimal coefficients are given solving the linear optimization in equation 9.

3.1.3 M-regression with Huber loss

M-regression techniques aim to maintain high efficiency for normally distributed errors while remaining robust to outliers (Kowalskie, 2025). One common approach is to use the Huber loss function, defined by

$$L(y,F) = \begin{cases} \frac{1}{2}(y-F)^2 & \text{if } |y-F| \le \delta, \\ \delta\left(|y-F| - \frac{\delta}{2}\right) & \text{if } |y-F| > \delta \end{cases}.$$

The threshold δ serves as a cutoff between inliers and outliers, and is often selected based on a quantile of the residual distribution. In gradient boosting frameworks, it can be re-estimated at each iteration from the current residuals. For the Huber loss function, the optimal coefficients are given by

$$\begin{aligned}
&\{\{\gamma_{jm}\}_1^J, \{\hat{\gamma}_{km}\}_1^K\} = \\
&\underset{\{\{\gamma_j\}_1^J, \{\hat{\gamma}_k\}_1^K\}}{\arg\min} \left[\sum_{\mathbf{x}_i : |y_i - F_{m-1}(\mathbf{x}_i)| \le \delta_m} \frac{1}{2} \left(y_i - F_{m-1}(\mathbf{x}_i) - \sum_{j=1}^{J} \gamma_j \, \mathbf{1}(\mathbf{x}_i \in R_{jm}) - \sum_{k=1}^{K} \hat{\gamma}_k \, \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km}) \right)^2 \right. \\
&\left. + \sum_{\mathbf{x}_i : |y_i - F_{m-1}(\mathbf{x}_i)| > \delta_m} \delta_m \left(|y_i - F_{m-1}(\mathbf{x}_i) - \sum_{j=1}^{J} \gamma_j \, \mathbf{1}(\mathbf{x}_i \in R_{jm}) - \sum_{k=1}^{K} \hat{\gamma}_k \, \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km})| - \frac{\delta_m}{2} \right) \right].
\end{aligned}$$

Now, introduce N slack variables defined as in equation 7. Then the search for the optimal coefficients $\{\{\gamma_{jm}\}_1^J, \{\hat{\gamma}_{km}\}_1^K\}$ can be formulated as a quadratic programming (QP) problem according to

$$\begin{aligned}
\min \quad & \sum_{t_i : |y_i - F_{m-1}(\mathbf{x}_i)| \le \delta_m} \frac{1}{2} t_i^2 + \sum_{t_i : |y_i - F_{m-1}(\mathbf{x}_i)| > \delta_m} \delta_m t_i \\
\text{s.t.} \quad & t_i \ge \left| y_i - F_{m-1}(\mathbf{x}_i) - \sum_{j=1}^{J} \gamma_j \mathbf{1}(\mathbf{x}_i \in R_{jm}) - \sum_{k=1}^{K} \hat{\gamma}_k \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km}) \right|, \quad i = 1, \ldots, N.
\end{aligned}$$

Similarly to LAD regression, we obtain an optimization problem with $N + J + K$ unknowns and $2N$ linear constraints. The constraints for Huber become the same as for LAD, but the objective is split into a quadratic part and a linear part.

Let $\mathbf{Q}_N$ be the identity matrix of size N, but put the diagonal element to zero if the corresponding ith slack variable does not contribute to the quadratic part. Similarly, let $\mathbf{c}_N$ be the identity vector of length N, but put the ith element to zero if the corresponding slack variable is not in the linear part of the objective. Then we get the QP problem formulated in matrix form as

$$\begin{aligned}
\min \quad & \frac{1}{2} \mathbf{t}^\intercal \mathbf{Q}_N \mathbf{t} + \mathbf{c}_N^\intercal \mathbf{t} \\
\text{s.t.} \quad & \begin{bmatrix} -\mathbf{I}_N & \boldsymbol{\Gamma} \\ -\mathbf{I}_N & -\boldsymbol{\Gamma} \end{bmatrix} (\mathbf{t}\gamma\hat{\gamma}) \le \begin{bmatrix} \mathbf{b} \\ -\mathbf{b} \end{bmatrix}.
\end{aligned} \tag{10}$$

Remark 1. *Note that the matrix $\mathbf{Q}_N$ is positive semi-definite, which together with linear constraints guarantees convexity of the problem.*

3.2 Implementation

Our implementation is available at `github.com (placeholder)`. The base learner was implemented using `DecisionTreeRegressor()` (on both the source and target data) in `scikit-learn` (Pedregosa et al., 2011). Implementing the coefficient updates for each method is straightforward given their matrix representations. For LSTransferTreeBoost (system of equations defined in equation 6), we use Numpy's built-in least-squares solver. For LADTransferTreeBoost (LP problem defined in equation 9), and for MTransferTreeBoost (QP problem defined in equation 10), we use the default solver configurations in `Scipy` (Virtanen et al., 2020) and `CVXPY` (Diamond and Boyd, 2016), respectively.

4 Simulation study

Assessing our three regression approaches with respect to dataset size, error distribution, and degree of domain shift is most conveniently achieved using simulated datasets, where these properties can be explicitly controlled.

4.1 Dataset

We use the *Friedman #1* dataset, introduced in (Friedman, 1991). The dataset is generated using 10 predictor variables $x_1, \ldots, x_{10}$ drawn from a uniform distribution over the interval $[0, 1]$, with a response generated from the first 5 predictor variables according to

$$y = 10 \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \epsilon, \tag{11}$$

where ϵ is generated according to some error distribution.

For transfer learning purposes, we also construct an *altered* version of the Friedman #1 dataset, generated according to the prescription in (Pardoe and Stone, 2010). In particular, two types of transformations are introduced to the features, namely *feature scaling* and *feature shift*, obtaining transformed features given by

$$\hat{x}_i = b_i x_i + c_i, \quad i = 1, \ldots, 5.$$

Furthermore, the response variable y is modified with additional scaling and an altered response is obtained according to

$$\hat{y} = a_1 10 \sin(\pi \hat{x}_1 \hat{x}_2) + a_2 20(\hat{x}_3 - 0.5)^2 + a_3 10\hat{x}_4 + a_4 5\hat{x}_5 + \hat{\epsilon}, \tag{12}$$

where $\hat{\epsilon}$ is the error distribution for the altered version of the response. The fixed parameters a_i, b_i were drawn from a normal distribution $\mathcal{N}(1, 0.1d)$, whereas c_i were drawn from $\mathcal{N}(0, 0.05d)$, where the positive integer d can be thought of as a "disturbance level", controlling the strength of the domain difference between the source and target datasets.

The target and source datasets $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$, $\{\hat{\mathbf{x}}_i, \hat{y}_i\}_{i=1}^{\hat{N}}$ can be generated from equations 11 and 12, respectively. This way, we can assess performance across train sizes, error distributions, and the closeness of the joint distributions (this "closeness" being determined by the disturbance level d).

4.2 Experimental setup

We considered three types of distributions of the errors ϵ and $\hat{\epsilon}$: the normal distribution (well-behaved tails), the slash distribution (very heavy tails), and the $t(2)$-distribution (moderately heavy tails). The errors were adjusted to give a 3/1 signal-to-noise ratio (SNR). We added errors to the target training dataset and the source dataset, but not to the validation or test sets.

The source dataset size was set to a fixed amount of 1000 datapoints, whereas the target dataset consisted of 100, 300, or 500 datapoints. An additional validation set of 1000 datapoints was generated to be used for early stopping. Finally, each hyperparameter configuration was evaluated on an unbiased test set consisting of another 1000 datapoints. This procedure was repeated for disturbance levels $d = 3, 6, 9$, across 5 distinct random seeds. The model fitting process was interrupted after 5 consecutive boosting rounds (epochs) of a non-improving validation set RMSE.

We consider the following hyperparameters:

- Shrinkage parameter ν: We let this be fixed as $\nu = 0.1$.
- Maximum source/target tree depth: We consider maximum depths of source/target trees between 1 and 3. All combinations of these are considered (*e.g.*, source tree depth = 1, target tree depth = 3 is a possible combination).

- Weight parameter α_m: We let this be defined by the exponential decay scheduler $\alpha_m = m_0 e^{-km}$, with hyperparameters $m_0 \in \{0.1, 0.5, 0.9\}$, and $k \in \{0, 0.01, 0.05\}$. Note that setting $k = 0$ results in static α_m-values equal to m_0, whereas $k > 0$ gives a decaying schedule.

- Threshold δ_m (only for MTransferTreeBoost): Although this parameter might be optimized through experimentation, we set it to be the 90% quantile of the current residual distribution.

4.3 Results

We present the RMSE values for the five best hyperparameter configurations for each triplet (target instances, error distribution, d) and for each regression technique. These configurations were selected by averaging performance across the five seeds and retaining those with the lowest test RMSE. These results are shown in Figure 1.

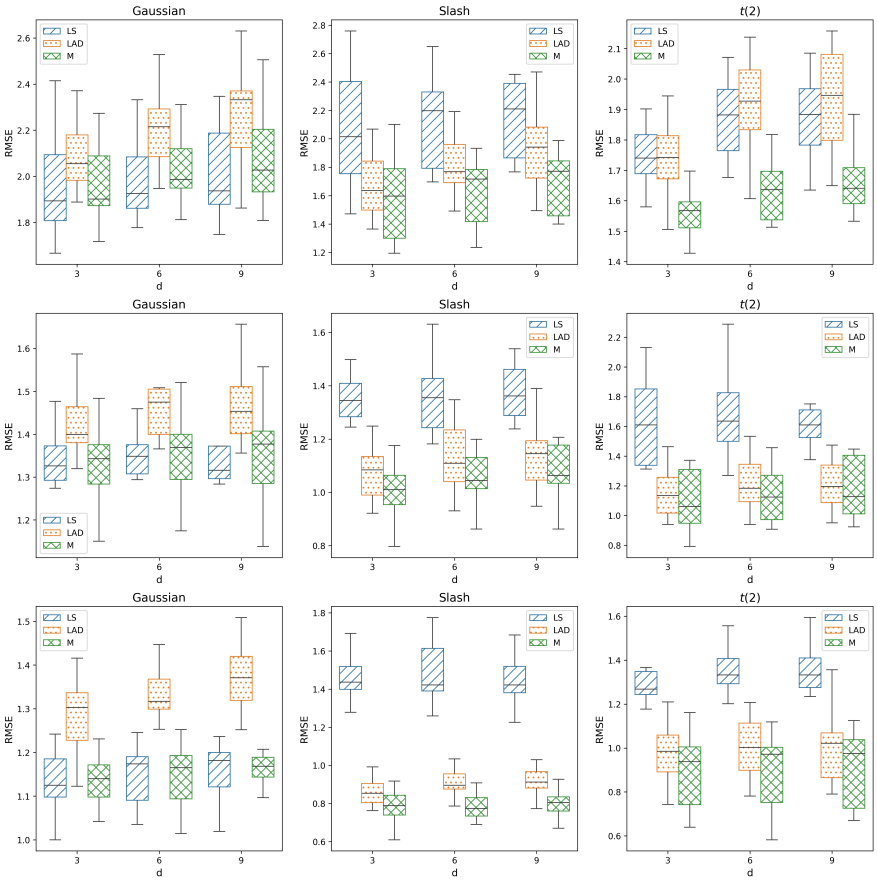

Figure 1: Test RMSE values on the Friedman #1 dataset using the best five hyperparameter settings for each triplet (target instances, error distribution, d), for LSTransferTreeBoost (LS), LADTransferTreeBoost (LAD), and MTransferTreeBoost (M). The top, middle, and bottom rows correspond to 100, 300, and 500 target instances, respectively.

Performance-wise, MTransferTreeBoost is the clear winner: it achieves superior results compared to LSTransferTreeBoost and LADTransferTreeBoost for both slash and $t(2)$ errors, and performs on par with LSTransferTreeBoost under normally distributed errors.

5 Benchmarking

5.1 Datasets

Monte Carlo simulation studies are convenient for assessing model performance under controlled conditions. However, transfer learning algorithms must be investigated in real-world transfer scenarios. Therefore, we evaluate our approaches on datasets with meaningful source–target splits and on two real-world transfer applications in forestry.

5.1.1 UCI datasets

We select six regression datasets from the UCI machine learning repository[1]. Because these datasets are from only one domain, the source and target data have to be constructed artificially using splitting techniques. This is a common strategy for creating source and target domains when evaluating transfer methods (Dai et al., 2007; Pardoe and Stone, 2010; Segev et al., 2016).

Following Pardoe and Stone (2010), we create source–target splits based on a continuous feature with a moderate degree of correlation (approximately 0.4) with the response variable. For each dataset, we identify the continuous feature whose absolute correlation with the response is closest to 0.4. Instances are then ordered according to this feature and divided into four equally sized subsets. The first or the last subset is randomly chosen to constitute the target data, and the remaining three quarters are used as source data. Finally, the splitting feature is removed from both source and target datasets. The resulting datasets are summarized in Table 1.

Table 1: UCI datasets after source-target splitting.

Dataset	Target N	Source N	Features
InfraRed	254	764	32
Concrete strength	257	773	7
Auto MPG	98	294	6
Real estate valuation	103	311	5
Airfoil self-noise	378	1125	4
Forest fires	130	387	7

5.1.2 (Individual) stem volume estimation

Estimating the stem volume of individual trees is essential for supporting accurate and detailed forest management. Advances in sensing technologies, such as digital imagery and terrestrial laser scanning, make it possible to extract geometric information from the entire stem of individual trees. However, obtaining complete stem measurements typically requires terrestrial data collection, often involving operators walking through the forest and capturing detailed recordings, which is a time-consuming process. Thus, estimating the total stem volume using only the lower part of the stem is an interesting task.

The Swedish stem bank (Björklund and Moberg, 1999; Moberg, 2001) contains measurements of individual trees. The data were collected from 198 pine trees and 114 spruce trees sampled across Sweden. The trees were cut into logs and scanned using imaging technology, providing detailed information about the stems. For example, stem diameters are available at 10 cm intervals, enabling accurate estimation of stem volume.

We use data from the Swedish stem bank to estimate total stem volume based on measurements from the lower part of the stem. A total of 23 predictor variables, corresponding to stem diameters measured between 0.3 m and 2.5 m, were used to predict the entire stem volume. Pine trees were used as the source dataset, while spruce trees served as the target dataset.

[1] https://archive-beta.ics.uci.edu/

5.1.3 (Average) stem diameter estimation

National airborne laser scanning (ALS) and national field inventory (NFI) data are commonly used to model forest variables at the national level. A common approach is to summarize ALS point-cloud data into statistical metrics and relate these metrics to field-measured variables, which are typically aggregated as means or totals over NFI plots.

We consider a Swedish source dataset and a Latvian target dataset. A total of 30 ALS-derived metrics were used as predictor variables, with mean stem diameter as the response variable. The Swedish dataset contains 8 412 observations, while the Latvian dataset contains 1 900 observations. Previous ALS-based forest inventory studies have suggested that around 500 field plots can be sufficient to build reliable models (Maltamo et al., 2009).

To evaluate our transfer learning approach, we randomly selected 300 target instances from the Latvian dataset and 2 000 source instances from the Swedish dataset prior to model training. This setup allows us to test the effectiveness of transfer learning under limited target data conditions.

5.2 Baseline model

A natural baseline for transfer learning is to ignore the source data and train only on the target data. For tabular regression problems, gradient tree boosting methods, such as XGBoost (Chen and Guestrin, 2016), consistently achieve state-of-the-art performance across a wide range of datasets, as demonstrated in (McElfresh et al., 2023; Grinsztajn et al., 2022).

We additionally evaluated instance-based transfer boosting approaches, including TrAdaBoost.R2 and two-stage TrAdaBoost.R2. Across all considered datasets, these methods were consistently outperformed by standard XGBoost, often by a substantial margin. This observation aligns with results in (Segev et al., 2016), indicating that instance-based transfer methods rarely outperform strong gradient boosting baselines for tasks on tabular data. Given their inferior performance and lack of consistent empirical support in the literature, we do not include these methods in the reported results.

Alternative deep learning–based transfer approaches have shown promising results on tabular data, particularly in classification settings (Levin et al., 2022). However, these methods differ fundamentally in their modeling assumptions, optimization objectives, and computational requirements, and fall outside the scope of boosting-based transfer learning considered in this work.

For these reasons, we use XGBoost trained solely on target data as the only benchmark in this study. As a strong and well-established model for tabular regression, it provides a clear and meaningful point of comparison for evaluating boosting-based transfer methods.

5.3 Experiments

We found that LSTransferTreeBoost performed on par with or better than MTransferTreeBoost in our experiments on real data. Consequently, we only present the results of LSTransferTreeBoost in this section.

For TransferTreeBoost, we use the same hyperparameter search settings as in Section 4.2. For XGBoost, we let the shrinkage parameter be constant ($\nu = 0.1$) and vary the maximum tree depth between 1 and 6. For all datasets, we employ a random 60/20/20 train/validation/test split on the target data, and we use early stopping after 5 consecutive non-improving iterations on the validation set. This process was repeated for 30 distinct random seeds. The results for the best hyperparameter configurations are reported in Table 2.

LSTransferTreeBoost achieved lower RMSE than XGBoost on seven out of the eight datasets and matched its performance on the remaining dataset (InfraRed). Excluding InfraRed, the relative improvement in average RMSE ranged from 1.16% to 12.37%, with a mean improvement of 4.77%.

The optimal hyperparameters for our method are presented in Appendix D. We did not observe any systematic relationship between the optimal hyperparameters and dataset properties, such as dataset size or number of predictors. Based on our experiments, it remains unclear whether any such relationship exists.

Table 2: Average RMSE values along with standard deviations.

Dataset / Method	TransferTreeBoost		XGBoost	
	RMSE	std	RMSE	std
InfraRed	0.232	0.031	0.232	0.035
Concrete strength	**3.877**	0.638	4.184	0.707
Auto MPG	**2.608**	0.570	2.738	0.538
Real estate valuation	**4.776**	0.941	5.027	0.937
Airfoil self-noise	**4.109**	0.302	4.157	0.316
Forest fires	**90.617**	77.659	91.556	77.028
Stem volume	**0.105**	0.023	0.120	0.026
Stem diameter	**7.576**	1.590	7.713	1.553

6 Discussion and future work

Our transfer learning algorithm, TransferTreeBoost, differs from most known boosting regression frameworks. Compared to instance-based approaches, it does not re-weight the importance of source data samples in every iteration. Instead, it jointly optimizes the individual coefficients of a sum of a source tree and target tree on the target dataset. Consequently, source information is transferred implicitly by updating this combined base learner at each iteration.

By updating only the target data, our algorithm retains the convergence properties of gradient boosting and can be shown to minimize the training error at least as effectively as standard TreeBoost.

Instance-based methods (because the importance of source samples is reweighted at each iteration) can handle multiple source datasets by simply combining them into a single large dataset. Our current implementation supports only one source dataset, which (unless the source datasets are very similar) limits its capabilities in multi-source settings. However, we show in Appendix C that optimal coefficients for our algorithm can also be derived in the multi-source setting. Further analysis of our algorithm in cases with multiple available source domains is an interesting direction for future work.

It also remains an open question whether systematic relationships between dataset characteristics and optimal hyperparameters exist beyond the settings considered here. A more comprehensive study of these relationships could potentially unravel such dependencies and support the development of functional default values.

The time complexity of our algorithm is not discussed in this work. We note that, under the least-absolute-deviation and Huber losses, the computational time to compute optimal coefficients depends explicitly on the number of target instances, since slack variables are introduced into the optimization. Although transfer learning often involves a small target dataset, making this larger computational time less problematic, the time complexity of our algorithm can be investigated in greater detail.

Our results suggest that gradient boosting provides a powerful and underexplored framework for transfer learning. In particular, model-based boosting transfer methods, such as TransferTreeBoost, retain theoretical guarantees while enabling direct knowledge transfer within the optimization process. We hope this work inspires future research along this avenue.

References

Lars Björklund and Lennart Moberg. *Modelling the inter-tree variation of knot properties for Pinus sylvestris in Sweden.* 1999.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Boosting for transfer learning. In *Proceedings of the 24th international conference on Machine learning*, pages 193–200, 2007.

Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Wenjing Fang, Chaochao Chen, Bowen Song, Li Wang, Jun Zhou, and Kenny Q Zhu. Adapted tree boosting for transfer learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 741–750. IEEE, 2019.

Yoav Freund and Robert 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.

Jerome H Friedman. Multivariate adaptive regression splines. *The annals of statistics*, 19(1):1–67, 1991.

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

Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35:507–520, 2022.

Anne Kowalskie. The impact of outliers on linear regression models: Detection and correction strategies. *OTS Canadian Journal*, 4(6):108–118, 2025.

Roman Levin, Valeriia Cherepanova, Avi Schwarzschild, Arpit Bansal, C Bayan Bruss, Tom Goldstein, Andrew Gordon Wilson, and Micah Goldblum. Transfer learning with deep tabular models. *arXiv preprint arXiv:2206.15306*, 2022.

Wei Liu, Wei David Liu, and Jianwei Gu. Predictive model for water absorption in sublayers using a joint distribution adaption based xgboost transfer learning method. *Journal of Petroleum Science and Engineering*, 188:106937, 2020.

M Maltamo, P Packalén, A Suvanto, KT Korhonen, L Mehtätalo, and P Hyvönen. Combining als and nfi training data for forest management planning: a case study in kuortane, western finland. *European Journal of Forest Research*, 128(3):305–317, 2009.

Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent. *Advances in neural information processing systems*, 12, 1999.

Duncan McElfresh, Sujay Khandagale, Jonathan Valverde, Vishak Prasad C, Ganesh Ramakrishnan, Micah Goldblum, and Colin White. When do neural nets outperform boosted trees on tabular data? *Advances in Neural Information Processing Systems*, 36:76336–76369, 2023.

Lennart Moberg. Models of internal knot properties for picea abies. *Forest ecology and management*, 147 (2-3):123–138, 2001.

David Pardoe and Peter Stone. Boosting for regression transfer. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 863–870, 2010.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

Noam Segev, Maayan Harel, Shie Mannor, Koby Crammer, and Ran El-Yaniv. Learn on source, refine on target: A model transfer learning framework with random forests. *IEEE transactions on pattern analysis and machine intelligence*, 39(9):1811–1824, 2016.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3:1–40, 2016.

A Proof of results from Section 3

Proof of Theorem 1. Throughout this proof we simplify notation by $h \coloneqq h(\cdot, \mathbf{a}_m)$. Furthermore, denote by $\mathcal{H}_T$ and $\mathcal{H}_S$ the spaces of regression trees fitted on target and source data respectively. In standard TreeBoost (Algorithm 1) the step h is selected among the J-terminal node regression trees, *i.e.* $h \in \mathcal{H}_T$. Albeit, in TransferTreeBoost, we select the base learner as a sum of trees one from $\mathcal{H}_T$ and one from $\mathcal{H}_S$. This joint space is defined as

$$\mathcal{H}_{T+S} = \{h \colon h = w_1 h_t + w_2 h_s, h_t \in \mathcal{H}_T, h_s \in \mathcal{H}_S, w_1, w_2 \in \mathbb{R}\}.$$

Since the K-terminal tree

$$\sum_{i=1}^{K} 0 \cdot \mathbf{1}(\mathbf{x} \in \hat{R}_{km}),$$

exists in $\mathcal{H}_S$, we conclude that $\mathcal{H}_T \subset \mathcal{H}_{T+S}$. Hence, the problem can be viewed as a relaxation of the original TreeBoost optimization. Following the functional gradient descent interpretation of boosting (Mason et al., 1999), the convergence rate is governed by the alignment between the negative gradient and the best available direction in the hypothesis space. Since $\mathcal{H}_T \subset \mathcal{H}_{T+S}$, TransferTreeBoost is guaranteed to find a direction with alignment greater than or equal to that of standard TreeBoost, thereby ensuring a superior bound on the geometric convergence rate. This completes the proof of the theorem. □

Proof of Proposition 1. Note that the sum of each row in $\mathbf{R}$ and $\mathbf{N}$ as well as $\widehat{\mathbf{R}}$ and $\mathbf{N}^\intercal$ are equal, since each $\hat{R}_{im}$ are disjoint and $\cup_i \hat{R}_{im}$ is the entire ambient space, and the sum of the ith row in $\mathbf{N}$ is equal to the number of points in the region R_{im}, which trivially equals $|R_{im}|$. Hence, consider the vector $\mathbf{u} = [\mathbf{1}_J, -\mathbf{1}_K]^\intercal$, where the two components are vectors of length J and K respectively. Then

$$\mathbf{M}\mathbf{u} = [\mathbf{R}\mathbf{1}_J - \mathbf{N}\mathbf{1}_K, \mathbf{N}^\intercal \mathbf{1}_J - \widehat{\mathbf{R}}\mathbf{1}_K]^\intercal = \mathbf{0}.$$

Since $\mathbf{u} \neq \mathbf{0}$ this implies that $\mathbf{M}$ is singular. This completes the proof of the proposition. □

B Alternative view of a sum of two regression trees

We show that the sum of two regression trees can be expressed as a single equivalent regression tree. Consequently, the optimal coefficients could be derived using the same methods applied to single base learners, implying that the result is effectively just another regression tree.

Lemma 1. *Let $\mathbf{T}_1$ and $\mathbf{T}_2$ be two regression trees over the same feature space X defined by regions and scalars $\{b_j, R_j\}_1^J$ and $\{\hat{b}_k, \hat{R}_k\}_1^K$, respectively. We define the sum of $\mathbf{T}_1$ and $\mathbf{T}_2$ as*

$$(\mathbf{T}_1 + \mathbf{T}_2)(\mathbf{x}) \coloneqq \sum_{j=1}^{J} b_j \mathbf{1}(\mathbf{x} \in R_j) + \sum_{k=1}^{K} \hat{b}_k \mathbf{1}(\mathbf{x} \in \hat{R}_k). \tag{13}$$

Then the corresponding sum is a regression tree.

Proof of Lemma 1. Let $\mathbf{x}$ be an arbitrary datapoint. Applying the sum of the two trees to this datapoint gives us a prediction of the form equation 13. Since $\{R_j\}_{j=1}^J$ and $\{\hat{R}_k\}_{k=1}^K$ form two partitions of the same feature space, it follows that $\mathbf{x}$ must lie in the intersection of two of these sets ($\mathbf{x} \in \tilde{R}_{jk} \coloneqq R_j \cap \hat{R}_k$ for some j, k). Thus, we can write

$$(\mathbf{T}_1 + \mathbf{T}_2)(\mathbf{x}) = \sum_{j,k} \tilde{b}_{jk} \mathbf{1}(\mathbf{x} \in \tilde{R}_{jk}), \tag{14}$$

where $\tilde{b}_{jk} = b_j + \hat{b}_k$. Since the regions $\tilde{R}_{jk}$ are disjoint and $\tilde{b}_{jk}$ is a scalar, it follows that the sum of the two trees is a new regression tree. □

Figure 2 provides an illustration of the resulting equivalent single tree given from the sum of two regression trees.

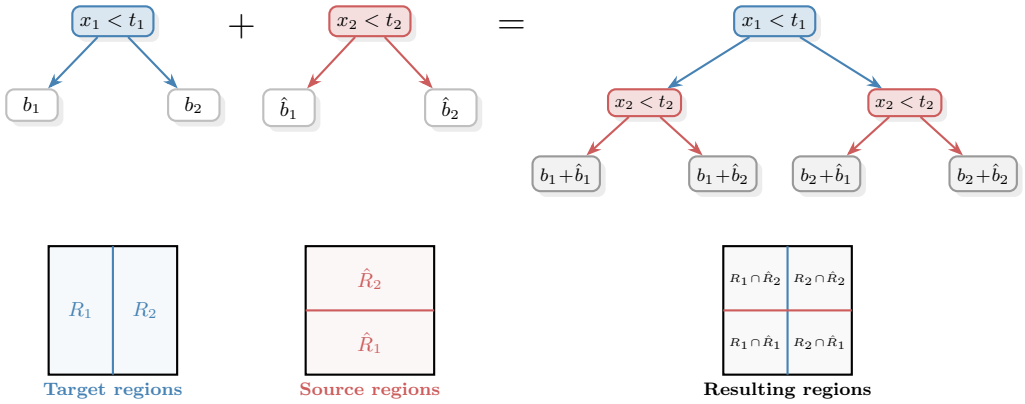

Figure 2: Visualization of Lemma 1. The partitions (below) illustrates how the regions overlap and make for a finer grid.

C Multiple source domains

In a multi-source domain setting with S source domains, the optimal coefficients at iteration m are given by

$$\left\{\{\gamma_{jm}\}_{j=1}^{J},\ \left\{\{\hat{\gamma}_{km}^{s}\}_{k=1}^{K(s)}\right\}_{s=1}^{S}\right\} = \\ \underset{\left\{\{\gamma_j\}_{j=1}^{J},\ \left\{\{\hat{\gamma}_k^s\}_{k=1}^{K(s)}\right\}_{s=1}^{S}\right\}}{\arg\min} \sum_{i=1}^{N} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \sum_{j=1}^{J}\gamma_j \mathbf{1}(\mathbf{x}_i \in R_{jm}) + \sum_{s=1}^{S}\sum_{k=1}^{K(s)} \hat{\gamma}_k^s \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km}^s)\right). \tag{15}$$

For notational convenience, let the target tree's regions and corresponding optimal coefficients be denoted by $\{\hat{R}_{km}^0\}_1^{K(0)}$ and $\{\hat{\gamma}_{km}^0\}_1^{K(0)}$, respectively, where $K(0) = J$. We can then write equation 15 as

$$\left\{\{\hat{\gamma}_{km}^{s}\}_{k=1}^{K(s)}\right\}_{s=0}^{S} = \underset{\left\{\{\hat{\gamma}_k^s\}_{k=1}^{K(s)}\right\}_{s=0}^{S}}{\arg\min} \sum_{i=1}^{N} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \sum_{s=0}^{S}\sum_{k=1}^{K(s)} \hat{\gamma}_k^s \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km}^s)\right). \tag{16}$$

C.1 LS regression

Denote by B the objective in equation 16. Now, let $s' \in \{0, 1, 2, \dots S\}$ be an arbitrary dataset. We can write B as the following sum

$$B = \sum_{k=1}^{K(s')} \sum_{x_i \in \hat{R}_{km}^{s'}} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \hat{\gamma}_k^{s'} + \sum_{s \neq s'}\sum_{k=1}^{K(s)} \hat{\gamma}_k^s \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km}^s)\right).$$

By differentiating B w.r.t. $\hat{\gamma}_k^{s'}$ for $k = 1, 2, \dots, K(s')$, we get

$$\partial \hat{\gamma}_k^{s'} = \sum_{\mathbf{x}_i \in \hat{R}_{km}} \frac{\partial L}{\partial \hat{\gamma}_k^{s'}}.$$

If L is the least-squares loss, the following condition must hold

$$\sum_{x_i \in \hat{R}_{km}^{s'}} \left(y_i - F_{m-1}(\mathbf{x}_i) - \hat{\gamma}_k^{s'} - \sum_{s \neq s'}\sum_{k=1}^{K(s)} \hat{\gamma}_k^s \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km}^s)\right) = 0,$$

which is equivalent to

$$\sum_{x_i \in \hat{R}^{s'}_{km}} \left(\hat{\gamma}^{s'}_k + \sum_{s \neq s'} \sum_{k=1}^{K(s)} \hat{\gamma}^s_k \mathbf{1}(\mathbf{x}_i \in \hat{R}^s_{km}) \right) = \sum_{x_i \in \hat{R}^{s'}_{km}} (y_i - F_{m-1}(\mathbf{x}_i)). \tag{17}$$

Let A and B be two arbitrary regions. Denote by $|A|$ the number of datapoints in region A, and further denote by $N_{A \cap B}$ the number datapoints in the intersection of the two regions A and B. Clearly,

$$\sum_{x_i \in \hat{R}^{s'}_{km}} \hat{\gamma}^{s'}_k = |\hat{R}^{s'}_{km}| \hat{\gamma}^{s'}_k .$$

Moreover, for a given region $\hat{R}^{s''}_{k''m}$ with $s'' \neq s'$, and corresponding coefficients $\hat{\gamma}^{s''}_{k''}$, we have that

$$\sum_{x_i \in \hat{R}^{s'}_{km}} \hat{\gamma}^{s''}_{k''} \mathbf{1}(\mathbf{x}_i \in \hat{R}^{s''}_{k''m}) = N_{\hat{R}^{s'}_{km} \cap \hat{R}^{s''}_{k''m}} \cdot \hat{\gamma}^{s''}_{k''}.$$

Thus, for each $k = 1, 2, \ldots, K(s')$, we can write equation 17 as

$$|\hat{R}^{s'}_{km}| \hat{\gamma}^{s'}_k + \sum_{s \neq s'} \sum_{k=1}^{K(s)} N_{\hat{R}^{s'}_{km} \cap \hat{R}^s_{km}} \cdot \hat{\gamma}^s_k = \sum_{x_i \in \hat{R}^{s'}_{km}} (y_i - F_{m-1}(\mathbf{x}_i)). \tag{18}$$

Now, define $\widehat{\mathbf{R}}^{s'} \in \mathbb{R}^{K(s') \times K(s')}$ and $\mathbf{r}^{s'} \in \mathbb{R}^{K(s')}$ as

$$\widehat{\mathbf{R}}^{s'} \coloneqq \begin{bmatrix} |\hat{R}^{s'}_{1m}| & 0 & \cdots & 0 \\ 0 & |\hat{R}^{s'}_{2m}| & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & |\hat{R}^{s'}_{K(s')m}| \end{bmatrix}, \quad \text{and } \mathbf{r}^{s'} \coloneqq \begin{bmatrix} \sum_{\mathbf{x}_i \in \hat{R}^{s'}_{1m}} (y_i - F_{m-1}(\mathbf{x}_i)) \\ \sum_{\mathbf{x}_i \in \hat{R}^{s'}_{2m}} (y_i - F_{m-1}(\mathbf{x}_i)) \\ \vdots \\ \sum_{\mathbf{x}_i \in \hat{R}^{s'}_{K(s')m}} (y_i - F_{m-1}(\mathbf{x}_i)) \end{bmatrix}.$$

Let $\mathbf{N}^{s'} \in \mathbb{R}^{K(s') \times \sum_{s \neq s'} K(s)}$ be the matrix whose kth row has the elements of all intersections in equation 18. Note that $\mathbf{N}^{s'}$ contains pairwise intersections, and can thus be constructed from S submatrices according to

$$\mathbf{N}^{s'} = \left[\mathbf{N}^{s'}_s \right]_{s \in \{0, \ldots, S\} \setminus \{s'\}},$$

where $\mathbf{N}^{s'}_s \in \mathbb{R}^{K(s') \times K(s)}$, and can be written as

$$\mathbf{N}^{s'}_s = \begin{bmatrix} N_{\hat{R}^{s'}_{1m} \cap \hat{R}^s_{1m}} & N_{\hat{R}^{s'}_{1m} \cap \hat{R}^s_{2m}} & \cdots & N_{\hat{R}^{s'}_{1m} \cap \hat{R}^s_{K(s)m}} \\ N_{\hat{R}^{s'}_{2m} \cap \hat{R}^s_{1m}} & N_{\hat{R}^{s'}_{2m} \cap \hat{R}^s_{2m}} & \cdots & N_{\hat{R}^{s'}_{2m} \cap \hat{R}^s_{K(s)m}} \\ \vdots & \vdots & \ddots & \vdots \\ N_{\hat{R}^{s'}_{K(s')m} \cap \hat{R}^s_{1m}} & N_{\hat{R}^{s'}_{K(s')m} \cap \hat{R}^s_{2m}} & \cdots & N_{\hat{R}^{s'}_{K(s')m} \cap \hat{R}^s_{K(s)m}} \end{bmatrix}.$$

Combining all of this, we get the following linear system of equations

$$\begin{bmatrix} \widehat{\mathbf{R}}^0 & \mathbf{N}^0_1 & \mathbf{N}^0_2 & \cdots & \mathbf{N}^0_S \\ \mathbf{N}^1_0 & \widehat{\mathbf{R}}^1 & \mathbf{N}^1_2 & \cdots & \mathbf{N}^1_S \\ \mathbf{N}^2_0 & \mathbf{N}^2_1 & \widehat{\mathbf{R}}^2 & \cdots & \mathbf{N}^2_S \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{N}^S_0 & \mathbf{N}^S_1 & \mathbf{N}^S_2 & \cdots & \widehat{\mathbf{R}}^S \end{bmatrix} \begin{bmatrix} \boldsymbol{\gamma}^0 \\ \boldsymbol{\gamma}^1 \\ \boldsymbol{\gamma}^2 \\ \vdots \\ \boldsymbol{\gamma}^S \end{bmatrix} = \begin{bmatrix} \mathbf{r}^0 \\ \mathbf{r}^1 \\ \mathbf{r}^2 \\ \vdots \\ \mathbf{r}^S \end{bmatrix},$$

or, by symmetry, the equivalent formulation

$$\begin{bmatrix} \widehat{\mathbf{R}}^0 & \mathbf{N}_1^0 & \mathbf{N}_2^0 & \cdots & \mathbf{N}_S^0 \\ \mathbf{N}_1^{0\intercal} & \widehat{\mathbf{R}}^1 & \mathbf{N}_2^1 & \cdots & \mathbf{N}_S^1 \\ \mathbf{N}_2^{0\intercal} & \mathbf{N}_2^{1\intercal} & \widehat{\mathbf{R}}^2 & \cdots & \mathbf{N}_S^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{N}_S^{0\intercal} & \mathbf{N}_S^{1\intercal} & \mathbf{N}_S^{2\intercal} & \cdots & \widehat{\mathbf{R}}^S \end{bmatrix} \begin{bmatrix} \boldsymbol{\gamma}^0 \\ \boldsymbol{\gamma}^1 \\ \boldsymbol{\gamma}^2 \\ \vdots \\ \boldsymbol{\gamma}^S \end{bmatrix} = \begin{bmatrix} \mathbf{r}^0 \\ \mathbf{r}^1 \\ \mathbf{r}^2 \\ \vdots \\ \mathbf{r}^S \end{bmatrix}.$$

Hence, we may put

$$\mathbf{M} \coloneqq \begin{bmatrix} \widehat{\mathbf{R}}^0 & \mathbf{N}_1^0 & \mathbf{N}_2^0 & \cdots & \mathbf{N}_S^0 \\ \mathbf{N}_1^{0\intercal} & \widehat{\mathbf{R}}^1 & \mathbf{N}_2^1 & \cdots & \mathbf{N}_S^1 \\ \mathbf{N}_2^{0\intercal} & \mathbf{N}_2^{1\intercal} & \widehat{\mathbf{R}}^2 & \cdots & \mathbf{N}_S^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{N}_S^{0\intercal} & \mathbf{N}_S^{1\intercal} & \mathbf{N}_S^{2\intercal} & \cdots & \widehat{\mathbf{R}}^S \end{bmatrix},$$

and analogous to Proposition 1, we have that $\mathbf{M}$ is singular as the vector

$$\mathbf{w} \coloneqq \frac{1}{S}\left[S \cdot \mathbf{1}_{K(0)}, -\mathbf{1}_{K(1)}, -\mathbf{1}_{K(2)}, \ldots, -\mathbf{1}_{K(S)}\right]^\intercal$$

serves as the vector $\mathbf{u}$ in the proof of Proposition 1, and the result of singularity follows by the same argument.

C.2 LAD regression and M-regression

Extending these approaches to multi-source settings is straight-forward and can be achieved in very similar ways. Consequently, we will only present the extension to multiple source domains for the least-absolute-deviation loss. Combining the reasoning for LAD regression with the arguments in Section 3.1.3, the extension to M-regression in multi-source settings follows directly.

For the least-absolute-deviation loss, equation 16 becomes

$$\left\{\{\hat{\gamma}_{km}^s\}_{k=1}^{K(s)}\right\}_{s=0}^{S} = \underset{\left\{\{\hat{\gamma}_k^s\}_{k=1}^{K(s)}\right\}_{s=0}^{S}}{\arg\min} \sum_{i=1}^{N} \left| y_i - F_{m-1}(\mathbf{x}_i) - \sum_{s=0}^{S}\sum_{k=1}^{K(s)} \hat{\gamma}_k^s \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km}^s) \right|.$$

Now, introduce auxiliary slack variables defined by

$$t_i \coloneqq \left| y_i - F_{m-1}(\mathbf{x}_i) - \sum_{s=0}^{S}\sum_{k=1}^{K(s)} \hat{\gamma}_k^s \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km}^s) \right|, \quad i = 1, \ldots, N.$$

Then we can formulate the search for the optimal coefficients as a linear programming (LP) problem as follows

$$\begin{aligned} &\min \sum t_i \\ &\text{s.t.} \quad t_i \geq \left| y_i - F_{m-1}(\mathbf{x}_i) - \sum_{s=0}^{S}\sum_{k=1}^{K(s)} \hat{\gamma}_k^s \mathbf{1}(\mathbf{x}_i \in \hat{R}_{km}^s) \right|, \quad i = 1, \ldots, N. \end{aligned} \tag{19}$$

This is an optimization problem with $N + \sum_{s=0}^{S} K(s)$ unknown variables and $2N$ linear constraints. Let $\mathbf{I}_N$ be the identity matrix of size N, $\mathbf{c}_N$ be the identity vector of length N, $\mathbf{t}^\intercal = \begin{bmatrix} t_1 t_2 \ldots t_N \end{bmatrix}$ be the transpose of the vector to the N slack variables, and $(\mathbf{t}\hat{\boldsymbol{\gamma}})^\intercal = \begin{bmatrix} \mathbf{t}^\intercal \hat{\gamma}_1^0 \hat{\gamma}_2^0 \ldots \hat{\gamma}_{K(0)}^0 \hat{\gamma}_1^1 \hat{\gamma}_2^1 \ldots \hat{\gamma}_{K(1)}^1 \ldots \hat{\gamma}_1^S \hat{\gamma}_2^S \ldots \hat{\gamma}_{K(S)}^S \end{bmatrix}$ be the

transpose of the vector consisting of all decision variables (of length $N + \sum_{s=0}^{S} K(s)$). Furthermore, put

$$\boldsymbol{\Gamma} \coloneqq \begin{bmatrix} \mathbf{1}(\mathbf{x}_1 \in \hat{R}^0_{1m}) & \cdots & \mathbf{1}(\mathbf{x}_1 \in \hat{R}^0_{K(0)m}) & \cdots & \mathbf{1}(\mathbf{x}_1 \in \hat{R}^S_{1m}) & \cdots & \mathbf{1}(\mathbf{x}_1 \in \hat{R}^S_{K(S)m}) \\ \mathbf{1}(\mathbf{x}_2 \in \hat{R}^0_{1m}) & \cdots & \mathbf{1}(\mathbf{x}_2 \in \hat{R}^0_{K(0)m}) & \cdots & \mathbf{1}(\mathbf{x}_2 \in \hat{R}^S_{1m}) & \cdots & \mathbf{1}(\mathbf{x}_2 \in \hat{R}^S_{K(S)m}) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{1}(\mathbf{x}_N \in \hat{R}^0_{1m}) & \cdots & \mathbf{1}(\mathbf{x}_N \in \hat{R}^0_{K(0)m}) & \cdots & \mathbf{1}(\mathbf{x}_N \in \hat{R}^S_{1m}) & \cdots & \mathbf{1}(\mathbf{x}_N \in \hat{R}^S_{K(S)m}) \end{bmatrix},$$

and

$$\mathbf{b} \coloneqq \begin{bmatrix} y_1 - F_{m-1}(\mathbf{x}_1) \\ y_2 - F_{m-1}(\mathbf{x}_2) \\ \vdots \\ y_N - F_{m-1}(\mathbf{x}_N) \end{bmatrix}.$$

We can write equation 19 in matrix form as

$$\begin{aligned} \min \quad & \mathbf{c}^\intercal \mathbf{t} \\ \text{s.t.} \quad & \begin{bmatrix} -\mathbf{I}_N & \boldsymbol{\Gamma} \\ -\mathbf{I}_N & -\boldsymbol{\Gamma} \end{bmatrix} (\mathbf{t}\hat{\boldsymbol{\gamma}})^\intercal \leq \begin{bmatrix} \mathbf{b} \\ -\mathbf{b} \end{bmatrix}. \end{aligned} \tag{20}$$

Thus, we conclude that the optimal coefficients are given solving the linear optimization in equation 20.

D Optimal hyperparameters

We present the optimal hyperparameters for each dataset in Section 5.3.

Table 3: Optimal hyperparameter configurations for LSTransferTreeBoost for each dataset. The shrinkage parameter was set to $\nu = 0.1$ in all experiments.

Dataset / Hyperparameter	Max. source tree depth	Max. target tree depth	m_0	k
InfraRed	1	1	0.5	0
Concrete strength	1	3	0.9	0.01
Auto MPG	2	1	0.5	0.01
Real estate valuation	1	2	0.5	0.01
Airfoil self-noise	2	2	0.5	0
Forest fires	1	1	0.5	0.01
Stem volume	2	2	0.5	0
Stem diameter	2	2	0.5	0