A Gradient-Based Split Criterion for Highly Accurate and Transparent Model Trees

Klaus Broelemann and **Gjergji Kasneci**

SCHUFA Holding AG, Wiesbaden, Germany

{Klaus.Broelemann,Gjergji.Kasneci}@schufa.de

Abstract

Machine learning algorithms aim at minimizing the number of false decisions and increasing the accuracy of predictions. However, the high predictive power of advanced algorithms comes at the costs of transparency. State-of-the-art methods, such as neural networks and ensemble methods, result in highly complex models with little transparency.

We propose shallow model trees as a way to combine simple and highly transparent predictive models for higher predictive power without losing the transparency of the original models. We present a novel split criterion for model trees that allows for significantly higher predictive power than state-of-the-art model trees while maintaining the same level of simplicity. This novel approach finds split points which allow the underlying simple models to make better predictions on the corresponding data. In addition, we introduce multiple mechanisms to increase the transparency of the resulting trees.

1 Introduction

With recent advances in Machine Learning and increasing computational power, automated predictive and decision making systems become widely used and influence peoples lives in many aspects. Applications include personal assistance, advertisement, scoring solutions, fraud prevention and recommendation systems.

Predictive models are influencing our daily lives more and more, thus creating an increasing interest in understandable and transparent predictive models. This interest is opposed by the interest in highly accurate systems. Complex systems, such as neural networks or ensemble models, have shown superior predictive power for a wide range of applications and datasets. This high predictive power comes at the costs of transparency:

"The best explanation of a simple model is the model itself; it perfectly represents itself and is easy to understand. For complex models, such as ensemble methods or deep networks, we cannot use the original model as its own best explanation because it is not easy to understand." [Lundberg and Lee, 2017]

As a consequence, the choice of a predictive model for a specific task includes a trade-off between simplicity and transparency on one side and complexity and higher predictive power on the other side. If the task is mainly driven by the quality of the prediction, complex state-of-the-art models are best suited. Yet, many applications require a certain level of transparency. This can be driven by regulatory constraints, but also by the user's wish for transparency. A recent study shows that model developers often feel the pressure to use transparent models [Veale *et al.*, 2018], even if more accurate models are available. This leads to simple models, such as linear or logistic regression, where more complex models would have much higher predictive power. In other cases, transparency is not required, but optional. An economic pressure towards models with higher accuracy may lead to low transparency in many important applications. For all these reasons, improving the accuracy of simple models can help to preserve transparency and improve predictions.

In this work, we present shallow model trees as a way to gain high predictive power with a highly transparent model. A model tree is similar to a decision tree. It consists of a set of rules that form a tree structure. Each leaf contains a simple predictive model. Thus, a model tree is a collection of simple models combined with a set of rules to select a model for each input sample. Shallow model trees consist only of few models and rules. Our experiments show that even a model stump, which uses two simple models, can significantly improve the predictive power, when compared to using only a singe simple model.

Figure 1 shows a schematic explanation of a model stump. The strength of the influence of features is used to explain a simple linear model. The model stump consists of two such models. Both are connected by single rule in the root node.

1.1 Our Contribution

The main contributions of this work are:

1. Shallow model trees as a way to combine transparency with high predictive power.

2. A novel split criterion for model trees that allows for significantly higher accuracy when compared to state-of-the-art model trees.

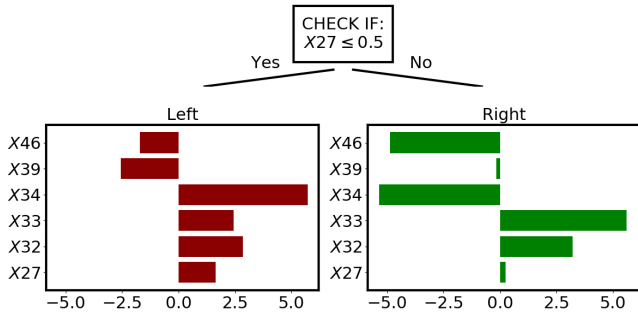3. The theoretical foundation of the split criterion.

Figure 1: Example explanation of a model stump. The explanation consists of a simple rule in the root node and the influence strength of features in both leaf node models (depicted by the bars). This example is taken from the preprocessed Bankruptcy dataset (see evaluation section) and the features such as $X27$ refer to this dataset.

4. The mechanism of renormalization that further improves predictive power of the presented model trees.

The rest of the paper is organized as follows: In Section 2, we give a brief overview of related work. Section 3 introduces our novel method for creating model trees and gives the theoretical background. Section 4 extends this method to further improve the predictive power. In Section 5, we show the effectiveness of our method on various datasets for classification and regression problems. Finally, Section 6 summarizes this works and gives an outlook and further research.

2 Related Work

2.1 Transparency and Explainability

The recent developments of more and more complex predictive models causes an increasing interest in transparency and explainability of predictions. Du et al. present a survey of different explainability approaches [Du *et al.*, 2018]. They distinguish between model-wise and prediction-wise explainability. The former one aims at interpreting a predictive model as a whole, while the latter one explains single preditions. In their comprehensive survey on black box explanation methods, Guidotti et al. distinguish between four different explanation problems [Guidotti *et al.*, 2018]: *model explanation* aims at model-wise explanation and *outcome explanation* aims at prediction-wise explanation (analogous to Du et al.). Additionally, they identify the *black box inspection*, aiming at gaining insides about the unknown interna of a blck box model, and the *transparent design* problem, aiming at finding predictive methods that are explainable by themselves. In this setting, our work mainly addresses the *transparent design* problem.

Mimic learning is one way to explain a predictive model as a whole. The idea is to approximate a complex model by a simpler, transparent model. Earlier works include approaches that use decision trees as simple models for mimic learning [Bastani *et al.*, 2017; Vandewiele *et al.*, 2016; Craven and Shavlik, 1996; Schmitz *et al.*, 1999]. Dancey et al. used logistic model trees for mimic learning [Dancey *et al.*, 2007] of neural networks. Although our algorithm can be used in the same way for mimic learning of complex models, this

is out of scope for this work and not covered by our experiments. Another approach for model-wise explainability is to measure the feature importance [Altmann *et al.*, 2010; Hastie *et al.*, 2009].

Prediction-wise explainability does not provide an overall explanation, but concentrates on single samples. One approach towards that goal is to use gradients or Taylor approximations at the given sample point [Kasneci and Gottron, 2016; Bach *et al.*, 2015]. Other approaches distort the input to measure the effects of different features on the input [Fong and Vedaldi, 2017; Robnik-Šikonja and Kononenko, 2008; Zintgraf *et al.*, 2017]

While most of the above work explains existing models, there is also a body of research on explainability by model design. One can argue that mimic learning falls into this category, depending on whether the newly created simple model is used for explanation purposes only or whether it serves as a replacement for the more complex model. Other approaches include constraints to derive simple models [Freitas, 2014] and regularized representations [Zhang *et al.*, 2018]. By using shallow model trees, this work also deals with explainability by model design. In difference to other approaches, our approach does not add constraints and restrictions to a complex model (except the preference of shallow trees), but rather uses a simple model, which is already highly explainable, and increases its predictive power to make it more attractive for real-world applications.

For a more comprehensive survey on explainability and transparency, we refer to [Guidotti *et al.*, 2018; Du *et al.*, 2018].

2.2 Tree-Based Prediction

Tree-Based models are a widely used class of predictors. A basic form are decision trees [Safavian and Landgrebe, 1991; Quinlan, 1993; Hastie *et al.*, 2009], which can be used for both, regression and classification. Decision trees are highly transparent and explainable as long as they are restricted in depth.

A way to increase the predictive power of decision trees is to use ensembles of trees, such as random forests [Breiman, 2001] or gradient boosted trees [Chen and Guestrin, 2016]. Again, the increase of predictive power comes at the costs of explainability.

Another class of tree-based predictors are model trees. These are similar to decision trees with the difference that each node contains a simple predictive model instead of a fixed value. The M5 model tree algorithm [Quinlan and others, 1992; Wang and Witten, 1997] for regression is constructed by using decision tree construction rules and is followed by the training of linear models in the nodes. The logistic model tree [Landwehr *et al.*, 2005] warm-starts the training of child-nodes, using the trained model of the parent nodes.

Another approach for training model trees is to train for each potential split a model in each child node [Potts and Sammut, 2005] and to compute for each potential split the loss on the training set. This allows finding the split with respect to the node-models by using the loss as split criterion.

On the downside, we see high computational costs that allow to test only a small number of potential split points (such as five per feature dimension in case of [Potts and Sammut, 2005]). In difference to existing model tree approaches, our approach does not use existing decision tree constructions methods, but uses a method that is optimized for model trees, which results in improved predictive power. Unlike [Potts and Sammut, 2005], our criterion can be computed much more efficiently and allows testing considerably more split points.

Hierarchical mixtures of experts [Jordan and Jacobs, 1994] are a similar concept with probabilistic splits. As a consequence, the final prediction is a probabilistic mixture of all leaf models. Originally, each split depends on a combination of all features, which reduces the explainability. Wang et al. [Wang *et al.*, 2015] present an extension that aims at sparse representations to increases explainability. In difference to our approach, the split criterion depends on multiple features, which results in more complex and less transparent splits. Furthermore, the larger search space in combination with the EM algorithm leads to much higher computational costs. Another approach by Sokolovska et al. [Sokolovska *et al.*, 2018] avoids hierarchical structures by splitting features independently of each other. This leads to $\prod_{i=1}^{m}(n_i + 1)$ regions / linear models (here, n_i is the number of split points along feature dimension i), which is potentially much higher than in tree structures.

3 Model Tree Construction

The idea of model trees has already been around for several years. In this section, we introduce a novel way to construct model trees. This allows for significantly higher predictive power than state-of-the-art model trees without increasing the number of nodes.

In the following, we briefly introduce current model tree algorithms (in 3.1). We then introduce the idea of our method (in 3.2). Subsequently, we present the theoretical background of our method (in 3.3). In the end of this section, we present an efficient way to compute the novel split criterion (in 3.4).

3.1 Model Tree Algorithms

Model trees have many aspects in common with decision trees, with the major difference that leaf nodes do not contain prediction values, but predictive models instead. Analogously to ensemble methods, we refer to these predictive models as weak models.

Typically, a model tree is constructed by creating a decision tree and training weak models in the leafs [Quinlan and others, 1992]. Algorithm 1 shows a basic version of recursively constructing a model tree. Approaches that train one weak model for each split [Potts and Sammut, 2005] also fit into the framework of Algorithm 1, but provide a different (and computationally expensive) way to compute the gain. We refer to the literature [Hastie *et al.*, 2009] for further details, such as the definition of candidate splits and the gain computation, as well as improvements, such as pruning, smoothing or handling missing data.

Similar to [Landwehr *et al.*, 2005], in the depicted algorithm weak models are trained in all nodes of the tree. This is not required for model trees, but helpful for our novel method.

Algorithm 1 Training Model Trees

1: **function** BUILDNODE($X \in \mathbb{R}^{n \times m}$, $y \in \mathbb{R}^n$, $d \geq 0$)
2:      node← NewNode()
3:      node.model←TrainWeakModel(X, y)
4:      **if** $d > 0$ **then**
5:          $\mathcal{T} \leftarrow$ GetCandidateSplits(X)
6:          $S \leftarrow \arg\max_{S' \in \mathcal{T}} \text{Gain}(X, y, S')$
7:          node.split $\leftarrow S$
8:          node.left $\leftarrow$ BuildNode($X_S, y_S, d-1$)
9:          node.right $\leftarrow$ BuildNode($X_{\bar{S}}, y_{\bar{S}}, d-1$)
10:      **end if**
11: **end function**

3.2 Toward a Better Split Criterion for Model Trees

Current model tree construction uses the decision tree construction algorithms with the extension that weak models are trained in the nodes. Model trees typically use the same loss/gain measure as decision trees, e.g. squared error for regression and cross entropy / information gain for classification [Quinlan and others, 1992; Landwehr *et al.*, 2005].

These loss/gain measures are optimized for decision trees. Although such decision tree measures have been shown to yield good results for model trees, a model-tree-specific gain/loss measure can significantly improve the predictive power of the resulting trees. An example can be seen in Fig. 2. The cross-entropy measure results in a split that separates both classes in a best-possible way, but gets nearly no improvement from models in the leafs (Fig. 2(b)). A better split for model trees would take into account the weak models and makes the classes separable by the weak models instead of directly separating them in the best possible way (Fig. 2(c)).

Based on this observation, the major part of this work is a novel gain/loss measure that results in improved model tree predictions without the need to increase the number of nodes. Hence, our work improves the predictive power without changing the interpretability of a shallow model trees.

3.3 Gradient-Based Split Criterion

We greedily construct the model tree, similar to Algorithm 1. The major novelty is the way to measure the gain of a potential split of a node. In the following, we present the theoretical background of our method and derive the split criterion. A list of symbols and terms used can be found in Tab. 1.

Training a Weak Model

Let M_θ be the class of *weak models* with parameters θ. Our approach handles any weak model that can be trained by (stochastic) *gradient descent*. Nevertheless, with transparency in mind, we recommend to use simple models such as linear or logistic regression.

Given a sequence $(x_i, y_i)_{i=1,\ldots,n}$ of training samples, a loss function $\mathcal{L} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and a set $I \subseteq \{1,\ldots,n\}$ of samples that correspond to a certain node[1], then training a

---

[1]For the root node $I = \{1,\ldots,n\}$. Each child node only holds a subset of the parent nodes training data.

(a) Logistic Regression   (b) Model tree with cross entropy split   (c) Model tree with gradient based split
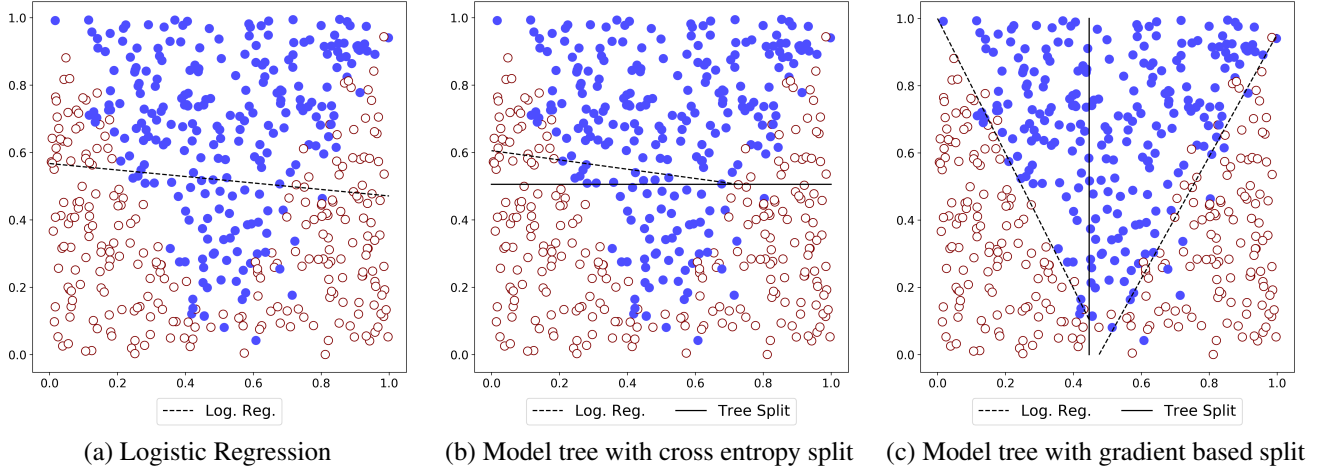
Figure 2: Comparison of different classification models for an example with two features. (a) Plain logistic regression. The other plots show model trees with one split and logistic regression models in the leafs. Splitting is done (b) by using cross entropy and (c) by using the proposed method. Solid lines are the tree splits. Dotted lines are the 50% class separators of logistic regression models.

Symbol	Description
$x_i \in \mathbb{R}^m$	Feature Vector of Sample i
y_i	Label of Sample i
$I, S \subseteq \{1, \ldots, n\}$	Index Sets of Samples
$M_\theta : \mathbb{R}^m \to \mathbb{R}$	Weak Model
$\hat{y}_i^\theta = M_\theta(x_i)$	Prediction of M_θ for sample i
$\mathcal{L} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$	Loss function
θ_S	Parameters minimizing the loss on S
G_S	Gain of a split into S and $\bar{S} = I \backslash S$.
d	Maximal depth of a (sub-)tree

Table 1: List of Symbols

weak model results in M_{θ_I} with

$$\mathcal{L}_\theta(I) = \sum\nolimits_{i \in I} \mathcal{L}(y_i, M_\theta(x_i)) \qquad (1)$$

$$\theta_I = \arg\min_\theta \mathcal{L}_\theta(I) \qquad (2)$$

Optimal Split

We represent a split of a node by the subset $S \subset I$ of samples that go into the left child node ($\bar{S} = I \backslash S$ is the complement that goes into the right child node). Generalizing from decision trees, the gain of the split S can then be defined as the reduction of the loss by using weak models in the child nodes instead of using one model in the parent node:

$$G_S = \mathcal{L}_{\theta_I}(I) - \mathcal{L}_{\theta_S}(S) - \mathcal{L}_{\theta_{\bar{S}}}(\bar{S}) \qquad (3)$$

Using this gain, the optimal split can be found by computing the gain for each possible split and taking the split with the maximal gain. This requires computing θ_S and $\theta_{\bar{S}}$ that is training weak models for all possible child nodes. This is computationally intractable to do for all potential split points. In [Potts and Sammut, 2005], this is done by limiting the number of candidate-splits to five per feature dimension. To overcome this limitation, we present in the following a method to approximate the gain.

Gain Approximation

When optimizing for θ_S, one can warm-start the gradient descent by using θ_I as an initial value [Landwehr *et al.*, 2005]. This will reduce the number of required gradient descent steps and speed up the learning. A rough estimation of θ_S is then already given by a single gradient descent step:

$$\theta_S \approx \theta_I - \lambda \nabla_{\theta_I} \frac{1}{|S|} \mathcal{L}_{\theta_I}(S) \qquad (4)$$

This estimation works better for simple models, which typically require less gradient descent steps then complex models.

This approximation can be computed efficiently (as we will describe in 3.4), but we still have to apply for each set S the model M_{θ_S} to all samples in S in order to compute the loss $L_{\theta_S}(S)$. Doing so would still be computationally intractable. For this reason, we simplify the computation of the loss using a Taylor approximation around θ_I:

$$\mathcal{L}_\theta(S) \approx \mathcal{L}_{\theta_I}(S) + (\theta - \theta_I)^T \cdot \nabla_{\theta_I} \mathcal{L}_{\theta_I}(S)$$

$$\overset{(4)}{\Rightarrow} \mathcal{L}_{\theta_s}(S) \approx \mathcal{L}_{\theta_I}(S) - \frac{\lambda}{|S|} \left\| \nabla_{\theta_I} \mathcal{L}_{\theta_I}(S) \right\|_2^2 \qquad (5)$$

Finally, we get an approximation of the loss:

$$\overset{(3)}{\Rightarrow} G_S \approx \mathcal{L}_{\theta_I}(I) - \mathcal{L}_{\theta_I}(S) + \frac{\lambda}{|S|} \left\| \nabla_{\theta_I} \mathcal{L}_{\theta_I}(S) \right\|_2^2$$

$$- \mathcal{L}_{\theta_I}(\bar{S}) + \frac{\lambda}{|\bar{S}|} \left\| \nabla_{\theta_I} \mathcal{L}_{\theta_I}(\bar{S}) \right\|_2^2$$

$$\overset{(1)}{=} \frac{\lambda}{|S|} \left\| \nabla_{\theta_I} \mathcal{L}_{\theta_I}(S) \right\|_2^2 + \frac{\lambda}{|\bar{S}|} \left\| \nabla_{\theta_I} \mathcal{L}_{\theta_I}(\bar{S}) \right\|_2^2 \qquad (6)$$

Using this approximation, the gain of all possible split points can be computed and the split with the maximal approximated gain can be used in the tree construction. Here, λ is just a scaling factor and can w.l.o.g. be set to 1. In the following, we will show an efficient way to this approximation.

Algorithm 2 Gradient-Based Split Finding

Require: Index Set I, Optimized Parameters θ_I, Training Data $(x_i, y_i)_{i \in I}$
1: $j_{Opt}, t_{Opt}, gain_{Opt} \leftarrow -1$
2: $g_i \leftarrow \nabla_{\theta_I} \mathcal{L}(y_i, M_{\theta_I}(x_i))$
3: $g_I \leftarrow \sum_{i \in I} g_i$
4: $g_S \leftarrow (0, 0, \ldots, 0)^T$
5: **for** feature $j = 1, \ldots, m$ **do**
6:     Sort g_i and x_i according to feature $x_{\cdot, j}$
7:     **for** Split Point t_k along feature j **do**
8:         $U \leftarrow \{i \in I | t_{k-1} < x_{i,j} \le t_k\}$
9:         $g_S \leftarrow g_S + \sum_{i \in U} g_i$
10:         $g_{\bar{S}} \leftarrow g_I - g_S$
11:         $gain \leftarrow \|g_S\|_2^2 + \|g_{\bar{S}}\|_2^2$
12:         **if** $gain > gain_{Opt}$ **then**
13:             $j_{opt} \leftarrow j; t_{opt} \leftarrow t_k; gain_{opt} \leftarrow gain$
14:         **end if**
15:     **end for**
16: **end for**
17: **return** $j_{Opt}, t_{Opt}, gain_{Opt}$

3.4 Implementation

The split measure (6) requires to compute the L2 norm of the gradients in both split sets S and $\bar{S}$. This has to be done for every candidate split S. Although feasible, the computation is expensive. The gain-computation for all splits is based on the same sample-wise gradients

$$g_i = \nabla_{\theta_I} \mathcal{L}(y_i, M_{\theta_I}(x_i)) \tag{7}$$

These only need to be computed once per node. For continuous features, every split candidate is given by a feature index j and a threshold t, which leads to $S = \{i \in I | x_{i,j} \le t\}$. By sorting along values of feature j, the split measure (6) can efficiently be updated. Algorithm 2 shows an efficient way to compute the split criterion by updating the sum of the gradients. This can then be used as an implementation for lines 5 and 6 in Algorithm 1.

This can even be improved further by using the fact that θ_I is optimized on I. This means that the gradient g_I is approximately zero (otherwise, θ_I is not optimized). As a consequence, $g_{\bar{S}} = -g_S$. Due to practical considerations, such as stopping criteria for stopping before reaching $g_I = 0$ or the usage of mini-batches, we do not directly set g_I to zero, but compute it as described in line 3 of Algorithm 2.

Note that our algorithm trains **exactly one model** for each node. It then uses gradients g_i of this trained model to approximate the gain and to find the optimal split point. With this split point, we train afterwards exactly one weak model for each child node. The training of child nodes can be warm-started by the learned parameters θ_I of the parent node's weak model.

4 Renormalization

The previous section introduced the gradient-based split criterion. Due to the approximations used, the split quality measure directly depends on the gradient. Unfortunately, the gradients are often not translation invariant, i.e. shifting the

feature values can change the optimal split point (in terms of criterion (4)). Furthermore, gradient descent often shows low performance and slow convergence when the input data is poorly scaled. Note that each split shrinks / restricts the domain of one feature for the child nodes. As a consequence, data that was normalized on the whole training set might lead to poorly normalized subsets in the leafs.

We renormalize the training data to work with normalized data in each node. We then get an improved, translation invariant version of approximation (4) and also speed up and improve the model training via gradient descent by providing well scaled input features. The renormalization only takes into account the training data from set S. As a consequence, the renormalization is done for each potential split individually.

A basic approach is to renormalize the features for S and $\bar{S}$, compute the gradients on these normalized features and compute the loss according to (5). Computing all gradients anew for each potential split is intractable. In addition, the weak model of the parent node is trained on differently normalized features, which complicates the gradient computation.

In the following, we present the mathematical background of our renormalization approach. The theory leads to a computationally more efficient way to compute the gradients from renormalized features. Subsequently, we present the explicit computation for linear models, such as linear and logistic regression, in combination with linear renormalizations, such as z-normalization and min-max-normalization.

4.1 Theory

Given a potential split $S \subset I$, the corresponding training features x_i can be normalized, e.g. by using a z-normalization. In the most general form, the normalization is a transformation $T_S : x_i \mapsto \tilde{x}_i$ from original feature space into a normalized feature space. We further restrict T_S to be compatible with the model, i.e. for each parameter θ, there must exist a parameter $\tilde{\theta}$ with

$$M_\theta(x_i) = M_{\tilde{\theta}}(\tilde{x}_i) \tag{8}$$

As a consequence, the transformed loss $\tilde{\mathcal{L}}$ can be written as:

$$\tilde{\mathcal{L}}_{\tilde{\theta}}(S) := \sum_{i \in S} \mathcal{L}(y_i, M_{\tilde{\theta}}(\tilde{x}_i)) = \mathcal{L}_\theta(S) \tag{9}$$

Analogously to (5), the loss can be approximated using gradients in the normalized feature space:

$$\mathcal{L}_{\theta_S}(S) = \tilde{\mathcal{L}}_{\tilde{\theta}_S}(S) \approx \tilde{\mathcal{L}}_{\tilde{\theta}_I}(S) - \frac{\lambda}{|S|} \left\| \nabla_{\tilde{\theta}_I} \tilde{\mathcal{L}}_{\tilde{\theta}_I}(S) \right\|_2^2$$

$$\overset{(9)}{=} \mathcal{L}_{\theta_I}(S) - \frac{\lambda}{|S|} \left\| \nabla_{\tilde{\theta}_I} \mathcal{L}_{\theta_I}(S) \right\|_2^2 \tag{10}$$

Note that there is a functional dependency between θ and $\tilde{\theta}$, which allows computing the last gradient in (10). For simple models, such as linear or logistic regression, Equation (10) often gives better approximations of the gain than (5). For such models, normalizing the features speeds up the gradient descent convergence and leads to better approximations after one step.

Still, normalizing and computing the gradients for each step individually is computationally infeasible. Rewriting the gradients can help to find a faster computation:

$$\nabla_{\tilde{\theta}_I} \mathcal{L}_{\theta_I}(S) = (\nabla_{\theta_I} \mathcal{L}_{\theta_I}(S))^T \cdot \frac{\partial \theta_I}{\partial \tilde{\theta}_I} \tag{11}$$

This means that only one computation of the gradient per node is required. It can then be corrected using the derivative $\frac{\partial \theta_I}{\partial \tilde{\theta}_I}$. This can be done in Algorithm 2 in line 11 by adding this sample-independent factor. Note that this derivative has to be computed for each potential split set anew. In the linear case, this can be done efficiently:

4.2 Linear Case

Linear models have the form

$$M_\theta(x) = f(w^T x + b), \qquad \theta = (w, b) \tag{12}$$

with a function $f : \mathbb{R} \to \mathbb{R}$. For linear regression, f is the identity, while for logistic regression f is the logistic function. Linear normalizations can be written as

$$\tilde{x} = T_S(x) = Ax + c \tag{13}$$

where A is a diagonal matrix, e.g. for z-normalization, A contains the inverse standard deviations of all features. Based on (8) and (12) follows:

$$x = A^{-1}\tilde{x} - A^{-1}c \qquad w = A\tilde{w} \qquad b = \tilde{b} + \tilde{w}^T c$$

The partial derivative can then be written as:

$$\frac{\partial \theta_I}{\partial \tilde{\theta}_I} = \begin{pmatrix} A & 0 \\ c^T & 1 \end{pmatrix} \tag{14}$$

For z-normalization, A and c depend on mean and the standard deviation. For min-max normalization, A and c depend on minimum and maximum. Both cases can efficiently be updated when computing the sum of the gradients (see also Algorithm 2 for updating computations).

It is worth noting that once a model with linear weak models is trained, one can create an equivalent model that operates solely on the unnormalized data by applying the inverse normalization steps on split points and model weights. This can be used speed up the predictions done with such a model, e.g. in real-time scenarios where each prediction has to be performed within few milliseconds.

4.3 Example

In order to demonstrate the improvement from renormalizing the data, we computed the split gain along the two axes of the previous example. Figure 3 shows three different methods to compute the gain. It can be seen that cross entropy finds the best split on the y-axis while the presented gradient-based method finds a better point on the x-axis. Still, the ideal point (in terms of the loss of the resulting model tree) can be found using the renormalization technique.

5 Evaluation

The previous sections introduced our novel method to create model trees. In this section, we will evaluate the effectiveness of this novel method on multiple datasets. The experiments cover both classification and regression.
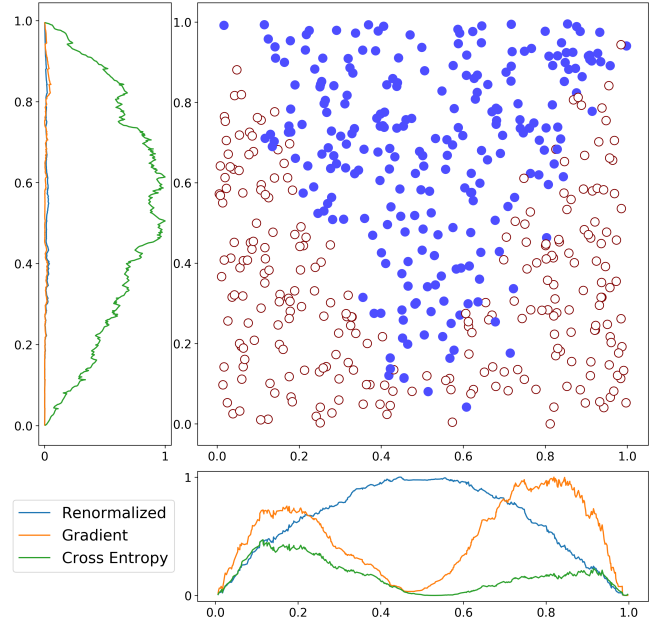

Figure 3: Different gain measures along the two axes. The gain-plots are normalized so that the maximal gain (across both feature dimensions) of a measure (i.e.. split measure value at the chosen split point) gets a value of 1. One can see, that the cross entropy measure leads to a horizontal split, while both gradient-base measures have the maximum at the first axis (i.e. a vertical split) with the renormalized version finding the best split for this example.

5.1 Datasets

For the experiments, we used the multiple datasets. These datasets include financial and medical datasets, where decisions can have a high impact and, hence, transparent predictions are required. All datasets come from public sources[2] [Yeh and Lien, 2009; Zikeba *et al.*, 2016] and cover both classification and regression tasks. The number of samples and attributes of these datasets are displayed in Tab. 3.

5.2 Methods

We compare the performance of different methods with each other. As a baseline, we use the weak models Lineare and Logistic Regression (Lin) as well as Decision Trees (DT). As a reference complex model, we use XGBoost (XGB) [Chen and Guestrin, 2016]. This model has been shown to be robust towards the choice of hyper parameters. Furthermore, we use model trees with state-of-the-art split criteria od decision trees (MT-DT). These are mean-square-error for regression and the entropy-based information gain for classification.

All these methods are compared with two versions of our model trees. The original version from Section 3 (MT-G)

---

[2] https://archive.ics.uci.edu/ml/datasets/Polish+companies+bankruptcy+data, https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic), https://archive.ics.uci.edu/ml/datasets/Census-Income+%28KDD%29, https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients, https://www.kaggle.com/heesoo37/120-years-of-olympic-history-athletes-and-results, https://www.kaggle.com/harlfoxem/housesalesprediction, https://www.kaggle.com/budincsevity/szeged-weather

Method	Linear	DT	XGB	MT-G			MT-GR			MT-DT		
Depth				1	2	3	1	2	3	1	2	3
(a) Bankruptcy	80.8	85.0	**93.2**	88.6	91.1	**91.2**	88.6	90.4	91.2	82.2	87.8	90.7
Breast Cancer	99.4	92.6	99.2	99.6	99.1	99.1	99.6	**99.7**	99.4	99.0	98.2	98.2
Census	**94.1**	89.6	**94.7**	**94.0**	93.8	93.7	93.9	93.7	93.3	93.9	93.9	93.7
Credit Card	71.7	73.8	**76.2**	71.7	74.9	74.8	74.4	75.3	**75.6**	72.1	74.8	75.1
Olympics	75.2	63.8	75.8	76.1	76.6	76.8	76.2	77.3	**78.3**	76.2	76.3	77.1
(b) House	77.2	78.3	87.9	82.2	83.4	83.8	83.9	86.3	**88.2**	77.9	82.6	83.7
Weather-Temp	62.1	85.7	**90.1**	69.7	78.6	84.0	80.4	87.3	**88.3**	64.9	70.8	79.4
Weather-Vis	24.2	57.9	**63.9**	43.2	49.7	55.5	46.2	54.5	**58.0**	43.6	51.9	56.0

Table 2: Evaluation results for (a) classification and (b) regression. Highlighted are results that are equaly to or better than the best model trees result.

Dataset	#Samples	#Attr. [Col.]		Preval.
Bankruptcy	43,405	64		4.82%
Breast Cancer	569	30		37.26%
Census	299,285	507		6.20%
Credit Card	30,000	24	[33]	22.12%
Olympics	271,116	8	[304]	14.67%
House	21,613	19		
Weather-Temp	96,453	10	[11]	
Weather-Vis	96,453	11	[12]	

Table 3: List of datasets. For each dataset the number of samples and attributes is listed. A second number for attributes indicates the number of columns after one-hot-encoding categorical attributes. For classification datasets the prevalence of class 1 is also denoted.

and the version using renormalization (MT-GR). With transparency and shallow trees in mind, we restrict all model trees to a fixed depth of 1, 2, or 3. In our experiments, we evaluate the effect of using different maximal depth limits.

5.3 Experiments

For our experiments, we performed 4-fold cross validation and averaged the 4 performance measurements.

Classification. We use the area under the roc-curve (auc) as measure for classification. The classification results are shown in Tab. 2 (a). It can be seen that the gradient-based model trees with renormalization provide the best results, often even a model stump (depth = 1) results in significantly higher auc than logistic regression. The exception is the census dataset that can sufficiently be modeled by logistic regression. Additional splits in models trees only create slight overfitting effects. In two cases, model trees even out-perform XGBoost.

Regression. We evaluate the regression experiments using the r^2-measure. The results are shown in Tab. 2 (b). Our results show that the renormalized version of our splitting criterion consistently creates the best results and each new level of depth increases the predictive power.

6 Conclusion

Starting with the motivation to find a transparent model with high predictive power, our work concentrated on shallow model trees. While state-of-the-art model trees are highly transparent, their predictive power bears potential for improvement. In this work, we presented a novel way to create model trees, which has a strong theoretical foundation, is efficient and results in higher predictive power than current model tree algorithms. Given a high requirement of transparency, our gradient-based model trees are powerful method for automated predictive and decision making systems.

Based on this work, we aim at further improvements of this method. This includes ways to enhance the explainability, e.g. through regularization terms that prefer round numbers as split points.

References

[Altmann *et al.*, 2010] André Altmann, Laura Toloşi, Oliver Sander, and Thomas Lengauer. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347, 2010.

[Bach *et al.*, 2015] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7), 2015.

[Bastani *et al.*, 2017] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpretability via model extraction. *arXiv preprint arXiv:1706.09773*, 2017.

[Breiman, 2001] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[Chen and Guestrin, 2016] 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. ACM, 2016.

[Craven and Shavlik, 1996] Mark Craven and Jude W Shavlik. Extracting tree-structured representations of trained networks. In *Advances in neural information processing systems*, pages 24–30, 1996.

[Dancey *et al.*, 2007] Darren Dancey, Zuhair A Bandar, and David McLean. Logistic model tree extraction from artificial neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(4):794–802, 2007.

[Du *et al.*, 2018] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. *arXiv preprint arXiv:1808.00033*, 2018.

[Fong and Vedaldi, 2017] Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3429–3437, 2017.

[Freitas, 2014] Alex A Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD explorations newsletter*, 15(1):1–10, 2014.

[Guidotti *et al.*, 2018] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):93, 2018.

[Hastie *et al.*, 2009] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer New York, 2009.

[Jordan and Jacobs, 1994] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

[Kasneci and Gottron, 2016] Gjergji Kasneci and Thomas Gottron. Licon: A linear weighting scheme for the contribution ofinput variables in deep artificial neural networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 45–54. ACM, 2016.

[Landwehr *et al.*, 2005] Niels Landwehr, Mark A. Hall, and Eibe Frank. Logistic model trees. *Machine Learning*, 59(1-2):161–205, 2005.

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

[Potts and Sammut, 2005] Duncan Potts and Claude Sammut. Incremental learning of linear model trees. *Machine Learning*, 61(1-3):5–48, 2005.

[Quinlan and others, 1992] John Ross Quinlan et al. Learning with continuous classes. In *5th Australian joint conference on artificial intelligence*, volume 92, pages 343–348. World Scientific, 1992.

[Quinlan, 1993] John Ross Quinlan. *C4. 5: programs for machine learning*. Morgan Kaufmann Publishers, 1993.

[Robnik-Šikonja and Kononenko, 2008] Marko Robnik-Šikonja and Igor Kononenko. Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering*, 20(5):589–600, May 2008.

[Safavian and Landgrebe, 1991] S. Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[Schmitz *et al.*, 1999] Gregor PJ Schmitz, Chris Aldrich, and Francois S Gouws. Ann-dt: an algorithm for extraction of decision trees from artificial neural networks. *IEEE Transactions on Neural Networks*, 10(6):1392–1401, 1999.

[Sokolovska *et al.*, 2018] Nataliya Sokolovska, Yann Chevaleyre, and Jean-Daniel Zucker. A provable algorithm for learning interpretable scoring systems. In *International Conference on Artificial Intelligence and Statistics*, pages 566–574, 2018.

[Vandewiele *et al.*, 2016] Gilles Vandewiele, Olivier Janssens, Femke Ongenae, Filip De Turck, and Sofie Van Hoecke. Genesim: genetic extraction of a single, interpretable model. *arXiv preprint arXiv:1611.05722*, 2016.

[Veale *et al.*, 2018] Michael Veale, Max Van Kleek, and Reuben Binns. Fairness and accountability design needs for algorithmic support in high-stakes public sector decision-making. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 440:1–440:14, New York, NY, USA, 2018. ACM.

[Wang and Witten, 1997] Yong Wang and Ian H. Witten. Induction of model trees for predicting continuous classes. In *Poster papers of the 9th European Conference on Machine Learning*. Springer, 1997.

[Wang *et al.*, 2015] Jialei Wang, Ryohei Fujimaki, and Yosuke Motohashi. Trading interpretability for accuracy: Oblique treed sparse additive models. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1245–1254, 2015.

[Yeh and Lien, 2009] I-Cheng Yeh and Che-hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2):2473–2480, 2009.

[Zhang *et al.*, 2018] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, page 5, 2018.

[Zikeba *et al.*, 2016] Maciej Zikeba, Sebastian K Tomczak, and Jakub M Tomczak. Ensemble boosted trees with synthetic features generation in application to bankruptcy prediction. *Expert Systems with Applications*, 2016.

[Zintgraf *et al.*, 2017] Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. *arXiv preprint arXiv:1702.04595*, 2017.