Stochastic Gradient Trees

Henry Gouk 1  Bernhard Pfahringer 1  Eibe Frank 1

Abstract
We present an online algorithm that induces decision trees using gradient information as the source of supervision. In contrast to previous approaches to gradient-based tree learning, we do not require soft splits or construction of a new tree for every update. In experiments, our method performs comparably to standard incremental classification trees and outperforms state of the art incremental regression trees. We also show how the method can be used to construct a novel type of neural network layer suited to learning representations from tabular data and find that it increases accuracy of multiclass and multilabel classification.

1. Introduction
Deep learning has received a lot of attention in a wide number of application domains—particularly computer vision and natural language processing (LeCun et al., 2015). To a significant extent, this is due to the convenient framework supplied by automatic differentiation coupled with gradient-based optimisation methods. This allows different inductive biases to be combined with very little technical effort. For example, a convolutional network can be trained to perform many different tasks by simply changing the loss function. Similar remarks can be made for recurrent neural networks trained on text data. However, on tabular data, neural networks are usually outperformed by decision tree induction methods, which in standard implementations that employ “hard” splits, do not fit into the gradient-based optimisation framework that makes deep learning so ubiquitous. In this work, we aim to bridge the gap between decision tree induction and gradient-based representation learning, thus providing a principled way to backpropagate from a neural network into a conventional decision tree with hard splits. We hypothesise that this will result in a class of models that have an inductive bias suited to modelling tabular data, but with the flexibility of reusing this same inductive bias for a variety of tasks.

The online nature of the optimisation algorithms used for training neural networks mandates that any potential decision tree component of the model must also be learned in an incremental manner to enable the joint optimisation desideratum to be met. To this end, we develop an incremental algorithm for constructing decision trees that uses gradient information as a supervision signal. We adapt the Taylor series approximation method used in gradient boosting techniques but reformulate it in such a way that one can train a single decision tree to optimise a given loss function, rather than requiring an ensemble of trees.

Previous approaches to incremental training of decision trees are primarily based on the Hoeffding tree algorithm of Domingos & Hulten (2000), which uses a hypothesis test based on the Hoeffding concentration inequality to determine whether a split should be made. We show how this type of hypothesis test is cumbersome to perform when learning from gradients, and instead make use of one-sample t-tests. Experiments in Section 5.1 show that our method performs favourably when compared to state of the art incremental data mining methods.

After introducing the incremental decision tree learner that can be trained with gradients, we show how an ensemble of these trees can be used to construct a novel neural network layer. This tree-based layer can be used as a feature extractor for tabular data, similar in concept to how a block of convolutional layers is often used to extract features from images. We compare neural networks with this novel feature extractor to conventional neural networks and common decision tree ensemble methods. The results of these experiments show that the network with the tree layer outperforms a conventional neural network on a variety of datasets, and approaches the performance of state of the art decision tree ensembles.

2. Related Work
Hoeffding trees (Domingos & Hulten, 2000) are commonly used decision trees for incremental learning. In each leaf node, they maintain a co-occurrence histogram between feature values and classes and apply the Hoeffding concentration inequality to determine whether there is enough
We adapt this to modify a single tree instead of an ensemble.

There is also work on learning representations using deep neural networks with a soft decision tree that has a predefined number of leaves.  Various extensions for Hoeffding trees exist: algorithms for adapting to concept drift (Bifet & Gavaldà, 2009), specialised ensembling approaches (Bifet et al., 2010b; Pfahringer et al., 2007), and hybrid approaches that build simpler models in each leaf node (Gama et al., 2003). Recent work has also investigated alternative strategies for selecting splits, such as splitting on the first attribute that proves to be a statistically significant improvement over the current tree, rather than trying to find the best possible split before modifying the tree (Manapragada et al., 2018). We also employ this strategy but do not attempt to revisit the split if it becomes apparent that another attribute would have provided better gain in predictive performance. There are also extensions that can perform regression: FIMT-DD (Ikonomovska et al., 2011a), which can build model trees, and ORTO (Ikonomovska et al., 2011b), which takes advantage of option nodes.

The batch decision tree learners most related to our work are those based on gradient boosting [Friedman, 2001]. Conventional gradient tree boosting expands an ensemble of trees using a sequential update rule to find the tree that maximally improves the current ensemble. The algorithm in [Friedman, 2001] uses the derivative of a loss function, such as cross entropy or squared error, to determine how much the new tree should change the output of the ensemble, implementing a form of gradient descent in function space. We take inspiration from XGBoost (Chen & Guestrin, 2016), which instead uses a second order approximation to the loss function (thus performing Newton steps in function space). XGBoost learns directly from the gradient and Hessian of the loss function. We adapt this to modify a single tree instead of an ensemble and also take ideas from Hoeffding trees to enable incremental learning. To deal with numeric attributes, we apply the approach used in LightGBM (Ke et al., 2017), which discretizes numeric attributes into ordinal quantities (Ranka & Singh, 1998; Frank & Witten, 1999) to speed up learning.

There is also work on learning representations using decision trees. Kontschieder et al. (2015) replace the fully connected layers often found at the end of convolutional networks with a soft decision tree that has a predefined structure. Each binary decision node defines a differentiable oblique split (Murthy et al., 1994), giving a probability distribution over which child node an instance should be assigned to. The splits are jointly optimised with the convolutional feature extractor, resulting in a model that performs comparably to a conventional convolutional network. Yang et al. (2018) focus on learning interpretable decision trees by training soft decision trees using backpropagation. They use a soft binning function to learn how features should be discretized. The depth of the tree is determined by the number of features in the dataset so the method does not scale to datasets with many features. This is mitigated by training an ensemble where each tree is supplied with a subset of the attributes. This approach yields interpretable models on small tabular datasets, providing accuracy similar to conventional networks. Zhou & Feng (2017) show that an ensemble of ensembles of conventional decision trees can be used to learn useful representations for classification, but their method cannot be generalised to other tasks and requires a very large number of decision models to be competitive with XGBoost.

3. Stochastic Gradient Trees

In supervised incremental learning, data is of the form \((\vec{x}_t, y_t) \in \mathcal{S}\), a new pair arrives at every time step, \(t\), and the aim is to predict the value of \(y_t\) given \(\vec{x}_t\). Algorithms for this setting must be incremental and enable prediction at any time step—they cannot wait until all instances have arrived and then train a model. In this section, we describe our method for incrementally constructing decision trees that can be trained to optimise arbitrary twice-differentiable loss functions. We refer to such models as stochastic gradient trees (SGTs). The first key ingredient is a technique for evaluating splits and computing leaf node predictions using only gradient information, which we adapt from the gradient boosting literature (Friedman, 2001; Chen & Guestrin, 2016). Secondly, to enable loss functions that have unbounded gradients, we employ standard one-sample \(t\)-tests rather than hypothesis tests based on the Hoeffding inequality.

3.1. Leveraging Gradient Information

We assume a loss function, \(l(y, \hat{y})\), that measures how well our predictions, \(\hat{y}\), match the true values, \(y\). Training should minimize its expected value, as estimated from the available data at a particular time step, \(t\). Assuming i.i.d. data,

\[
E[l(y, \hat{y})] \approx \frac{1}{t} \sum_{i=1}^{t} l(y_i, \hat{y}_i). \tag{1}
\]

The predictions \(\hat{y}_i\) are obtained from the SGT, \(f_t\). At each time step, we aim to find a modification, \(u\), to the tree that takes a step towards minimising the expected loss:

\[
f_{t+1} = f_t + \arg \min_u [\mathcal{L}_t(u) + \Omega(u)], \tag{2}
\]

where

\[
\mathcal{L}_t(u) = \sum_{i=1}^{t} l(y_i, f_t(\vec{x}_i) + u(\vec{x}_i)), \tag{3}
\]

and

\[
\Omega(u) = \gamma |Q_u| + \frac{\lambda}{2} \sum_{j \in Q_u} v_j^2, \tag{4}
\]
The $\Omega$ term is a regularizer. $Q_u$ is the set of new leaf nodes associated with $u$, and $v_u$ maps these new leaf nodes to the difference between their predictions and the prediction made by their parent. The first term in $\Omega$ imposes a cost for each new node added to the tree, and the second term can be interpreted as a prior that encourages the leaf prediction values to be small. In our experiments, we set $\lambda$ to 0.1 and $\gamma$ to 1. Because $f_t$ is a decision tree, $u$ will be a function that represents a possible split to one of its leaf nodes, or an update to the prediction made by a leaf: the addition of $f_t$ and $u$ is actually the act of splitting a node in $f_t$, or changing the value predicted by an existing leaf node. In the case of Hoeffding trees, and also SGTs, only the leaf that contains $\vec{x}$ will be considered for splitting at time $t$, and information from all previous instances that have arrived in that leaf will be used to determine the quality of potential splits. The algorithm also has the option to leave the tree unmodified if there is insufficient evidence to determine the best split.

There are two obstacles to incrementally training a tree using an arbitrary loss function. Firstly, the splitting criterion must be designed to be consistent with the loss to be minimised. Secondly, the leaf nodes’ predictions must be chosen in a manner that is consistent with the loss. Both problems can be overcome by adopting a trick used in gradient boosting (Friedman 2001) that expands an ensemble of trees by applying a Taylor expansion of the loss function around the current state of the ensemble. We only consider modification of a single tree, therefore the empirical expectation of the loss function can be approximated using a Taylor expansion around the unmodified tree at time $t$:

$$L_t(u) \approx \sum_{i=1}^{t} l(y_i, f_t(\vec{x}_i)) + g_i u(\vec{x}_i) + \frac{1}{2} h_i u^2(\vec{x}_i),$$

where $g_i$ and $h_i$ are the first and second derivatives, respectively, of $l$ with respect to each $\vec{y}_i$ produced so far. Optimisation can be further simplified by eliminating the constant first term inside the summation, resulting in

$$\Delta L_t(u) = \sum_{i=1}^{t} [g_i u(\vec{x}_i) + \frac{1}{2} h_i u^2(\vec{x}_i)],$$

which now describes the change in loss due to the split, $u$.

This function is evaluated for each possible split to find the one that yields the maximum reduction in loss. As in the Hoeffding tree algorithm, at time $t$, we only attempt to split the leaf node into which $\vec{x}$ falls, and we consider splitting on each attribute. For each potential split, we need to decide what values should be assigned to the corresponding leaf nodes. Note that we also consider the option of not performing a split at all, and only updating the prediction made by the existing leaf node.

We introduce some notation to explain our procedure. Firstly, we define what a potential split looks like:

$$u(\vec{x}) = \begin{cases} v_u(q_u(\vec{x})), & \text{if } \vec{x} \in \text{Domain}(q_u) \\ 0, & \text{otherwise} \end{cases}$$

where $q_u$ maps an instance in the current leaf node to a leaf node that would be created if the split were performed. We denote the codomain of $q_u$—the set of leaf nodes that would result from this split—as $Q_u$. We define the set $I_u^t$ as the set of indices of the instances that would reach the new leaf node, $j$. The objective can then be rewritten as

$$\Delta L_t(u) = \sum_{j \in Q_u} \sum_{i \in I_u^t} [g_i v_u(j) + \frac{1}{2} h_i v_u^2(j)],$$

which can be rearranged to

$$\Delta L_t(u) = \sum_{j \in Q_u} [\left( \sum_{i \in I_u^t} g_i \right) v_u(j) + \frac{1}{2} \left( \sum_{i \in I_u^t} h_i \right) v_u^2(j)],$$

then setting the derivative to zero,

$$0 = \left( \sum_{i \in I_u^t} g_i \right) + \left( \lambda + \sum_{i \in I_u^t} h_i \right) v_u(j),$$

and solving for $v_u(j)$, yielding

$$v_u(j) = -\frac{\sum_{i \in I_u^t} g_i}{\lambda + \sum_{i \in I_u^t} h_i}.$$
3.2. Splitting on Numeric Attributes

When splitting on nominal attributes, we create a branch for each value of the attribute, yielding a multi-way split. We deal with numeric attributes by discretizing them using simple equal width binning, similar to approaches used in recent gradient boosting techniques [Ke et al., 2017]. A sample of instances from the incoming data is used to estimate the minimum and maximum values of each numeric attribute—if these are not already known in advance. Any future values that do not lie in the estimated range are clipped. The number of bins and the number of instances used to estimate the range of attribute values are user-provided hyperparameters. In our experiments, we set them to 64 and 1,000, respectively. Given a discretized attribute, we consider all possible binary splits that can be made based on the bin boundaries, thus treating it as ordinal [Frank & Witten, 1999].

3.3. Determining when to Split

Equation 9 estimates the quality of a split but does not indicate whether a split should be made. Hoeffding trees use the Hoeffding concentration inequality to make this decision. It states that, with some probability 1 − δ,

$$E[X] > X - \epsilon,$$

where $X$ is the sample mean of a sequence of random variables $X_i$, $R$ is the range of values each $X_i$ can take, and $n$ is the sample size used to calculate $X$. Suppose the best split considered at time $t$ is $u^t$. Let $L = \frac{1}{n} \sum L_i(u^t)$ be the mean change in loss if the split were applied, as measured on a sample of $n \leq t$ instances. Thus, if $-L > \epsilon$, we know, with $1 - \delta$ confidence, that applying this split will result in a reduction of loss on future instances.

In order to apply the Hoeffding bound, we must know the range, $R$, of values that can be taken by the $n$ terms, $\Delta l_i$, in $\Delta L$. In our application, this would require proving upper and lower bounds of the first and second derivatives of the loss function, and constraining the output of the tree to lie within some prespecified range, thus preventing rapid experimentation with different loss functions for novel tasks—one of the properties that has made deep learning on image and language data so widespread. To circumvent this problem, we instead use Student’s $t$-test to determine whether a split should be made. The $t$ statistic is computed by

$$t = \frac{L - \frac{1}{n} \sum L_i}{s/\sqrt{n}},$$

where $s$ is the sample standard deviation of $L$ and, under the null hypothesis, $\frac{1}{n} \sum L_i$ is assumed to be zero—i.e., it is assumed that the split does not result in a change in loss. A $p$ value can be computed using the inverse cumulative distribution function of the $t$ distribution and, if $p$ is less than $\delta$, the split can be applied.

This test assumes that $\bar{L}$ follows a normal distribution. Although it cannot be assumed that each $L_i$ will be normally distributed, due to the central limit theorem, we are justified in assuming $\bar{L}$ will be normally distributed for sufficiently large $n$. Computing $s$ requires estimating the sample variance of $L_i$, which is made easier by initially considering each of the new leaf nodes, $j \in Q_u$, in isolation:

$$\text{Var}(L_i) = \text{Var}(G_i v_u(j) + \frac{1}{2} H_i v_u^2(j)),$$

where $G_i$ and $H_i$ are the random variables representing the gradient and Hessian values, respectively. We intentionally treat $v_u(j)$ as a constant, even though this ignores the correlation between the prediction update values and the gradient and Hessian values. Empirically, this does not appear to matter, and it eliminates the need to compute the variance of a quotient of random variables—an expression for which there is no distribution-free solution.

Equation 16 cannot be computed incrementally because the $v_u(j)$ are not known until all the data has been seen. It is also infeasible to store all of the gradient and Hessian pairs because this could lead to unbounded memory usage. Instead, the equation can be rearranged using some fundamental properties of variances to yield

$$\text{Var}(L_i) = v_u^2 \text{Var}(G_i) + \frac{1}{4} v_u^4 \text{Var}(H_i) + v_u^3 \text{Cov}(G_i, H_i),$$

where we have dropped the “(j)” for compactness. The variances and covariances associated with each feature value can be incrementally estimated using the method of Welford [1962] and one of the algorithms in Bennett et al. (2009), respectively. When considering splits, the sample statistics associated with each feature value—and therefore each new leaf node—can be aggregated using the concurrent estimation algorithms given by Bennett et al. (2009), resulting in the sample variance, $s^2$.

The process used to determine whether enough evidence has been collected to justify a split would be prohibitively expensive to carry out every time a new instance arrives. In practice, we follow the common trend in online decision tree induction and only check whether enough evidence exists to perform a split when the number of instances that have fallen into a leaf node is a multiple of some user specified parameter. As with many incremental decision tree induction implementations, this value is set to 200 by default.
4. Ensembles of Stochastic Gradient Trees

One of the advantages of deep networks is their ability to output a vector of predictions. This is useful for tasks such as multiclass classification, multi-label classification, multitarget regression, and learning embeddings. SGTs, as formulated so far, output only a single value and are thus less flexible. However, multiple SGTs, combined in an ensemble, can be trained to produce a vector of outputs. The first type of ensemble we consider is a committee of trees where each component in the output vector is produced by an SGT. The second approach backpropagates from a neural network into a committee of SGTs, decoupling the number of trees from the number of output components.

4.1. Training Committees of SGTs

To produce multiple outputs, a different SGT can be trained to estimate each component of the prediction vector, yielding a simple method for multiclass classification. For an \( m \)-class classification problem, \( m - 1 \) SGTs can be trained to produce margin values that are fed into a softmax activation function composed with the cross entropy loss function— the \( m \)th margin value can be hard-coded to zero. Similarly, a committee of \( m \) SGTs can be trained to produce a label vector for multi-label classification, where more (or less) than one of the \( m \) binary labels can be associated with a single instance.

While conceptually simple, this method has some drawbacks. Firstly, if there is a large number of outputs, the computational resources required to construct the model could make training a committee of SGTs impractical. Moreover, for non-convex loss functions, care must be taken to ensure the trees are not initialised in such a way that they lie at a saddle point on the loss surface. This is a very real concern when training embedding models using objectives such as the contrastive loss (Hadsell et al., 2006) or a triplet loss (Chechik et al., 2010). This second issue is not a problem for neural networks, due to their random initialisation.

4.2. Training SGT Networks

Our solution to the problems associated with training a committee of SGTs is to use an ensemble of trees as a layer in a neural network. This allows the output dimensionality to be dictated by the number of units in the final layer of the network, rather than the number of trees that will be trained. Typical neural networks are composed of a series of linear transformations interleaved with nonlinear activation functions. The conventional wisdom in the deep learning community is that a large number of layers will result in higher accuracy. We propose using an ensemble of SGTs as the first hidden layer and conventional fully connected layers for the rest of the network, with all but the output layer also making use of the rectified linear unit activation function. We refer to such networks as SGT networks.

SGTs require second order information to evaluate splits and compute leaf prediction values. Therefore, we must backpropagate these second derivatives through the fully connected layers along with the gradient information usually produced by backpropagation. More formally, let \( \phi_{sgt}(\vec{x}) \) be a function that produces a vector, where each component is determined by a different SGT. We can write down the equation for an SGT network, \( f \), as

\[
f(\vec{x}) = (\phi_{fc}^{(m)} \circ \phi_{relu}^{(m-1)} \cdots \phi_{relu} \circ \phi_{fc}^{(1)} \circ \phi_{sgt})(\vec{x}),
\]

where each \( \phi_{fc}^{(i)} \) is a fully connected layer with a randomly initialised weight matrix, \( W^{(i)} \), and bias vector, \( \vec{b}^{(i)} \).

Suppose now that the loss function, \( l \), accepts a vector of predictions, rather than a scalar prediction, and denote the input of layer \( i \) as \( \vec{z}^{(i)} \). We must compute the first and second derivatives of the loss function with respect to each of the SGTs in the first layer of the network. It is important to note that we do not need to compute the full Hessian matrix for the neural network parameters or activations—we need only compute elements that lie on the diagonal and are associated with the activations of each layer. The first derivatives can be computed using repeated application of the chain rule (i.e., backpropagation). Faà di Bruno’s formula (Arbogast, 1800; Faà di Bruno, 1855) can be used in place of the chain rule when dealing with higher order derivatives, resulting in rules for backpropagating second derivatives through fully connected layers,

\[
\frac{\partial^2 l}{\partial \vec{z}^{(i)}_j} = \sum_{k=1}^{d_{t+1}} \frac{\partial^2 l}{\partial \vec{z}^{(i+1)}_j} W^{(i)}_{kj},
\]

and rectified linear units,

\[
\frac{\partial^2 l}{\partial \vec{z}^{(i)}_j} = \mathbb{I}(\vec{z}^{(i)}_j > 0) \frac{\partial^2 l}{\partial \vec{z}^{(i+1)}_j}.
\]

In these equations \( \vec{z}^{(i)}_j \) is the output of \( \phi^{(i)} \) during a forward propagation, \( d_i \) represents the number of components in \( \vec{z}^{(i)}_j \), and \( \mathbb{I} \) is the indicator function that evaluates to one when the expression in parentheses is true. The first and second derivatives of \( l \) with respect to the output of \( \phi_{sgt} \) can be used to train the committee of SGTs, and the first order derivatives calculated by the conventional backpropagation process can be used for training the fully connected layers using any of the usual optimisation techniques. In our experiments, we use the Adam optimiser (Kingma & Ba, 2014) to learn the weights in the fully connected layers.

The weights in the fully connected layers are initialised using the method of Glorot & Bengio (2010). We also randomly initialise the trees in the SGT layer: a randomly
generated split is applied to each tree in the layer before training begins. This split is generated by sampling a random feature, and in the case of numeric attributes, a random boundary between quantiles; the resulting leaf nodes are assigned random prediction values.

5. Experiments

In this section, we investigate the performance of the SGT on incremental learning tasks, and also demonstrate how well the SGT layer improves neural network accuracy on tabular data in the batch setting. We implemented our approaches in Java, making use of the MOA framework [Bifet et al., 2010b] for the experiments with incremental learning, WEKA [Hall et al., 2009] for the batch multiclass classification experiments, and MEKA [Read et al., 2016] for the multi-label classification experiments. The implementation is available online.\footnote{https://github.com/henrygouk/stochastic-gradient-trees}

5.1. Incremental Learning

To gain insight into how well SGTs perform in an incremental learning setting, we evaluate their performance on a collection of large classification and regression datasets. For each dataset, we report the mean across 10 runs, where the data is randomly shuffled for each run. Absolute error is used to evaluate performance on regression problems and classification error is used otherwise. The standard Hoeffding tree algorithm (VFDT) and the faster extension of Manapragada et al. (2018) (EFDT) are used as baselines for the experiments, and the FIMT-DD [Ikonomovska et al., 2011a] and ORTO [Ikonomovska et al., 2011b] methods are used as points of reference for regression. The squared error loss function was used for training the regression trees, and the loss function, composed with a softmax, for the classification tasks. Learning curves for these experiments are given in Figure[1]. Statistics on overall performance, model size, and runtime for models trained on the classification and regression datasets are given in Table[1]. From this table and the figure, we can see that SGTs perform similarly to state of the art methods on classification problems, uniformly outperforming VFDT and exhibiting two wins and two losses each compared to EFDT, and exhibit an advantage over existing incremental regression tree methods. On most problems, SGTs also result in smaller model sizes, and take a similar length of time to train compared to the baseline methods. An exception to this is in the multiclass classification scenario, where the model size and training time grows proportionally to the number of classes because a different tree is trained for each class.

5.2. Batch Learning

The SGT layer presented in Section[4.2] is intended to be used as a feature extractor for tabular data. The first experiment we conduct to characterise the performance of this novel layer is a comparison with standard multiclass classification methods. The datasets used in these experiments were collected from OpenML [Vanschoren et al., 2013] by taking the 10 largest datasets containing a mixture of nominal and numeric attributes in the “Categorical Columns Analysis” study, making sure not to include duplicate datasets labelled with different names. The SGT network trained on these datasets contains an SGT layer with 100 trees and two fully connected hidden layers, each with 100 units. The neural network without the SGT layer contains three fully connected hidden layers with 100 units each. Adam [Kingma & Ba, 2014] is used to optimise the neural networks and a validation set consisting of 10% of the training data is used to determine how many epochs should be used for training. We have included random forest and XGBoost baselines to give an indication of how well state of the art tree based methods perform on tabular data. Each of these tree ensemble approaches is trained with 100 trees and default parameters otherwise. The results of performing 10-fold cross validation are given in Table[2] where statistically significant differences have been tested with a corrected t-test [Nadeau & Bengio, 2000]. They show that the network with the SGT layer exhibits superior performance on tabular data compared to a conventional neural network. The SGT network also comes close to the performance of random forests, but it is outperformed by the gradient boosted decision tree models produced by XGBoost.

Neural networks are commonly applied in settings where model outputs have a more complicated structure than a single category or numeric value. For example, convolutional networks are often applied to segmentation problems where a segmentation mask is generated. A analogous task on tabular data is multi-label prediction, where a set of labels must be assigned to each input instance. To demonstrate that the representations learned by SGT networks are well suited to such structured prediction problems, we compare our method to a standard neural network and two common multi-label classification methods. We use the same SGT network and neural network as was used for the multiclass classification experiments, but we instead use the logistic activation function and a sum of binary cross entropy loss functions. The two multi-label classification methods we use are the binary relevance method and the ensemble of classifier chains technique of Read et al. [2011]. Both of these are problem transformation methods that create an ensemble of classifiers in order to produce the set of predicted labels. The binary relevance approach simply trains a classifier for each potential label, whereas the classifier chain
Table 1. Mean error, model size, and runtime of the trees produced by the classification methods (top) and regression methods (bottom) on 10 random shuffles of each dataset.

| Dataset       | Classification Error | Model size (Nodes) | Runtime (seconds) |
|---------------|----------------------|--------------------|-------------------|
|               | SGT      | VFDT     | EFDT   | SGT      | VFDT     | EFDT   | SGT      | VFDT     | EFDT   |
| Higgs         | 30.10    | 30.25    | 31.46  | 1,933.2  | 8,081.4  | 38,535.1| 384.13   | 115.68   | 512.32 |
| HEPMASS       | 14.66    | 14.81    | 15.22  | 1,289.0  | 7,256.0  | 24,760.2| 290.46   | 108.11   | 425.49 |
| KDD'99        | 0.27     | 0.73     | 0.07   | 446.3    | 160.0    | 913.8   | 466.26   | 33.15    | 97.61  |
| Covertype     | 26.94    | 32.54    | 22.05  | 620.8    | 91.8     | 3,261.4 | 32.56    | 3.04     | 47.23  |

| Dataset       | Mean Absolute Error | Model size (Nodes) | Runtime (seconds) |
|---------------|---------------------|--------------------|-------------------|
|               | SGT      | FIMT-DD  | ORTO  | SGT      | FIMT-DD  | ORTO  | SGT      | FIMT-DD  | ORTO  |
| Airline       | 20.86    | 21.14    | 20.41 | 39,287.0 | 30,060.4 | 33,584.8| 22.76    | 22.43    | 50.16  |
| AWS Prices    | 0.27     | 0.58     | 0.58  | 3,316.3  | 165,660.6| 177,929.0| 79.96    | 80.89    | 42.66  |
| Zurich        | 61.59    | 65.80    | 65.78 | 763.1    | 21,756.2 | 23,769.0| 33.18    | 29.71    | 22.16  |
| MSD Year      | 7.20     | 13.52    | 21.76 | 235.8    | 2,264.4  | 2,466.8 | 58.23    | 70.75    | 37.17  |

Figure 1. Learning curves for the incremental classification and regression problems.
Table 2. 10-fold cross validation results for a collection of datasets found on OpenML [Vanschoren et al., 2013].

| Dataset       | SGT Network | Neural Network | Random Forest | XGBoost |
|---------------|-------------|----------------|---------------|---------|
| adult         | 86.69±0.48  | 85.03±0.36     | 85.13±0.69    | 87.27±0.51 |
| bank-marketing| 89.10±0.93  | 89.60±1.45     | 89.67±0.87    | 89.80±0.81 |
| churn         | 95.18±1.33  | 92.84±1.27     | 95.10±1.11    | 95.40±0.79 |
| electricity   | 89.00±0.34  | 84.19±0.69     | 92.47±0.64    | 91.89±0.37 |
| eye-movements | 65.20±1.18  | 57.53±2.12     | 70.60±1.88    | 73.85±1.64 |
| internet-usage| 88.27±0.71  | 86.10±1.61     | 87.84±0.83    | 89.69±1.19 |
| nomao         | 96.27±0.56  | 95.87±0.59     | 96.95±0.34    | 97.31±0.35 |
| phishing      | 96.47±0.34  | 96.60±0.61     | 97.26±0.40    | 97.11±0.35 |
| speed-dating  | 100.00±0.00 | 98.87±0.38     | 92.87±0.75    | 100.00±0.00|
| thyroid       | 98.62±0.62  | 97.30±0.98     | 98.49±0.75    | 99.05±0.52 |

○, • statistically significant improvement or degradation, respectively, at 95% confidence.

Table 3. Results of 10-fold cross validation on multi-label datasets.

| Dataset | Average Precision | Macro F1 | Accuracy |
|---------|--------------------|----------|----------|
|         | BR     | ECC     | NN      | SGT     | BR     | ECC     | NN      | SGT     | BR     | ECC     | NN      | SGT     |
| CAL500  | 0.345  | 0.427   | 0.497   | 0.504   | 0.411  | 0.444   | 0.313   | 0.460   | 0.263  | 0.291   | 0.189   | 0.305   |
| music   | 0.658  | 0.778   | 0.810   | 0.793   | 0.562  | 0.661   | 0.577   | 0.634   | 0.465  | 0.566   | 0.503   | 0.552   |
| scene   | 0.690  | 0.833   | 0.873   | 0.851   | 0.583  | 0.730   | 0.702   | 0.725   | 0.548  | 0.693   | 0.686   | 0.705   |
| yeast   | 0.582  | 0.723   | 0.772   | 0.727   | 0.561  | 0.637   | 0.631   | 0.607   | 0.436  | 0.523   | 0.582   | 0.500   |
| birds   | 0.713  | 0.775   | 0.650   | 0.735   | 0.188  | 0.255   | 0.078   | 0.153   | 0.512  | 0.598   | 0.481   | 0.550   |
| Avg. rank | 3.8    | 2.6     | 1.8     | 1.8     | 3.4    | 1.2     | 3.2     | 2.2     | 3.6    | 1.6     | 3       | 1.8     |

The results of running 10-fold cross validation on a collection of multi-label datasets is given in Table 3 [Wu & Zhou, 2017] show that many multi-label classification evaluation measures fit into one of two groups: those that favour instance-wise performance, and those that favour label-wise performance. As such, we report both average precision (label-wise) and macro F1 (instance-wise). We also report the Jaccard index (also known as multi-label accuracy), because it is another common measure and was not considered by [Wu & Zhou, 2017]. Looking at the average rank of each method, we can see that the SGT network consistently performs better than, or similarly to, the conventional neural network. The binary relevance method performs the worst, and ensembles of classifier chains perform slightly better than the SGT network.

6. Conclusion

In this paper, we have presented an incremental algorithm for constructing decision trees that exclusively uses gradient information as the source of supervision. This algorithm is inspired by Hoeffding trees but uses the t-test instead of the Hoeffding inequality because the former does not require upper bounds on the gradient and Hessian of the loss function. We have also shown how to use a neural network as the source of gradient information required for supervision, resulting in a new network layer that can be used for extracting features from tabular data. The results given in Section 5 demonstrate that our technique is competitive with standard online classification trees, and outperforms state of the art incremental regression trees. It is also shown that the SGT neural network layer provides a clear advantage over conventional neural networks when learning from tabular data.

We anticipate that our method will allow a more diverse range of incremental learning tasks to be addressed using decision tree models. Our reformulation of the Taylor expansion-based loss function given in Section 3.1 can also be adapted to work in the batch setting, which has the potential to result in smaller ensemble sizes when training gradient boosted decision tree models. The SGT layer presented in Section 4.2 should also enable construction of neural network architectures for complex multi-relational tabular data mining problems, such as click prediction and designing content-based recommendation systems.
Stochastic Gradient Trees

References

Arbogast, L. F. A. *Du Calcul Des Dérivations*. Levrault, Strasbourg, 1800.

Bennett, J., Grout, R., Pebay, P., Roe, D., and Thompson, D. Numerically stable, single-pass, parallel statistics algorithms. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pp. 1–8, August 2009.

Bifet, A. and Gavalda, R. Adaptive Learning from Evolving Data Streams. In *Advances in Intelligent Data Analysis VIII*, Lecture Notes in Computer Science, pp. 249–260. Springer Berlin Heidelberg, 2009.

Bifet, A., Frank, E., Holmes, G., and Pfahringer, B. Accurate Ensembles for Data Streams: Combining Restricted Hoeffding Trees using Stacking. In *JMLR Workshop and Conference Proceedings* 13, pp. 16, Tokyo, Japan, 2010a.

Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. MOA: Massive Online Analysis. *Journal of Machine Learning Research*, 11(May):1601–1604, 2010b.

Chechik, G., Sharma, V., Shalit, U., and Bengio, S. Large Scale Online Learning of Image Similarity through Ranking. *Journal of Machine Learning Research*, 11:1109–1135, 2010.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, March 2010.

Hadsell, R., Chopra, S., and LeCun, Y. Dimensionality Reduction by Learning an Invariant Mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*, volume 2, pp. 1735–1742, June 2006.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsrl.*, 11(1):10–18, November 2009.

Ikonomovska, E., Gama, J. a., and Džeroski, S. Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 23(1):128–168, July 2011a.

Ikonomovska, E., Gama, J. a., Ženko, B., and Džeroski, S. Speeding Up Hoeffding-Based Regression Trees with Options. In *28th International Conference on Machine Learning*, pp. 8, Bellevue, WA, USA, 2011b.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems 31*, pp. 3146–3154, 2017.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kontschieder, P., Fiterau, M., Criminisi, A., and Bulo, S. R. Deep Neural Decision Forests. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1467–1475, Santiago, Chile, December 2015. IEEE.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521(7553):436–444, May 2015.

Manapragada, C., Webb, G., and Salehi, M. Extremely Fast Decision Tree. In *Knowledge Discovery in Databases 2018*, London, United Kingdom, August 2018. ACM.

Murthy, S. K., Kasif, S., and Salzberg, S. A System for Induction of Oblique Decision Trees. *Journal of Artificial Intelligence Research*, 2:1–32, August 1994.

Nadeau, C. and Bengio, Y. Inference for the Generalization Error. In *Advances in Neural Information Processing Systems*, pp. 307–313, 2000.

Pfahringer, B., Holmes, G., and Kirkby, R. New Options for Hoeffding Trees. In *AI 2007: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pp. 90–99. Springer Berlin Heidelberg, 2007.
Ranka, S. and Singh, V. CLOUDS: A decision tree classifier for large datasets. In Proceedings of the 4th Knowledge Discovery and Data Mining Conference, volume 2, 1998.

Read, J., Pfahringer, B., Holmes, G., and Frank, E. Classifier chains for multi-label classification. Machine learning, 85(3):333–359, 2011.

Read, J., Reutemann, P., Pfahringer, B., and Holmes, G. MEKA: A Multi-label/Multi-target Extension to WEKA. JMLR, 17(21):1–5, 2016.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. OpenML: Networked Science in Machine Learning. SIGKDD Explorations, 15(2):49–60, 2013.

Welford, B. P. Note on a Method for Calculating Corrected Sums of Squares and Products. Technometrics, 4(3):419–420, 1962.

Wu, X.-Z. and Zhou, Z.-H. A Unified View of Multi-Label Performance Measures. In International Conference on Machine Learning, pp. 9, Sydney, Australia, 2017.

Yang, Y., Morillo, I. G., and Hospedales, T. M. Deep Neural Decision Trees. arXiv:1806.06988 [cs, stat], June 2018.

Zhou, Z.-H. and Feng, J. Deep Forest: Towards An Alternative to Deep Neural Networks. In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, pp. 3553–3559, Melbourne, Australia, August 2017. International Joint Conferences on Artificial Intelligence Organization.