TREX: Tree-Ensemble Representer-Point Explanations

Jonathan Brophy [1] **Daniel Lowd** [1]

Abstract

How can we identify the training examples that contribute most to the prediction of a tree ensemble? In this paper, we introduce TREX, an explanation system that provides instance-attribution explanations for tree ensembles, such as random forests and gradient boosted trees. TREX builds on the representer point framework previously developed for explaining deep neural networks. Since tree ensembles are non-differentiable, we define a kernel that captures the structure of the specific tree ensemble. By using this kernel in kernel logistic regression or a support vector machine, TREX builds a surrogate model that approximates the original tree ensemble. The weights in the kernel expansion of the surrogate model are used to define the global or local importance of each training example.

Our experiments show that TREX's surrogate model accurately approximates the tree ensemble; its global importance weights are more effective in dataset debugging than the previous state-of-the-art; its explanations identify the most influential samples better than alternative methods under the remove and retrain evaluation framework; it runs orders of magnitude faster than alternative methods; and its local explanations can identify and explain errors due to dataset shift.

1. Introduction

Tree ensembles, including random forests (Breiman, 2001) and gradient boosted trees (Friedman, 2001), remain one of the most effective machine learning approaches to classification in many domains. In recent years, their popularity has only grown, as shown by the increasing number of gradient boosting frameworks, including XGBoost (Chen & Guestrin, 2016), LightGBM (Ke et al., 2017), and CatBoost (Prokhorenkova et al., 2018).

[1]Department of Computer Science, University of Oregon, Eugene, Oregon, USA. Correspondence to: Jonathan Brophy <jbrophy@cs.uoregon.edu>, Daniel Lowd <lowd@cs.uoregon.edu>.

As performant as tree ensembles are, their complexity and scale can make them inherently difficult to understand, and their outputs challenging to interpret. This can have a number of consequences, including lower trust and decreased use of these models, especially in problem domains where decisions can have major impacts (e.g. health care, autonomous vehicles, etc.). By understanding how these models make predictions at a deeper level, we can expose deficiencies in the model or the data they are trained on. This can lead to higher quality data, which can result in more performant models or models that do what practitioners would expect, ultimately increasing trust for the consumers of their decisions.

One approach to understanding model predictions are through *instance attribution* explanations, which identify examples in the training data that have the greatest impact on a model's prediction for a given query instance. This type of interpretability may not only deepen one's understanding and trust of the model, but may also aid in model and dataset debugging. Instance attribution methods can also be used in conjunction with other explanation approaches, such as feature-attribution methods (Lundberg & Lee, 2017; Ribeiro et al., 2016).

Recent work (Yeh et al., 2018) adapted the concept of representer theorems (Schölkopf et al., 2001) to introduce a representer point framework for explaining the predictions of deep learning classifiers in terms of excitatory and inhibitory training examples. Their method decomposes the pre-activation prediction of a query instance into a linear combination of training point activations. Unfortunately, this approach is specific to deep learning models, and requires a suitable differentiable loss function to compute the "representer values" (weights) of the training-instance activations.

Tree ensembles are non-differentiable due to the step functions created by feature splits in each tree. To adapt the representer point framework to tree ensembles, we introduce TREX (**T**ree-ensemble **R**epresenter-point **EX**planations). TREX leverages the structure of the trees to build a tree ensemble kernel, which acts as a similarity measure between data points. Then, we train a kernelized model on this new kernel representation by solving the dual problem to obtain weights for the training instances. We are then

able to decompose any prediction as a linear combination of these training instances, resulting in an instance-attribution explanation of positive and negative training points. We show that TREX is not only able to aid in dataset debugging and model understanding, but is also more scalable than previous methods.

Our contributions are as follows:

1. We define a new kernel for tree ensembles, LeafOutput, based on the path through and the numerical output of each tree in the ensemble.

2. We introduce TREX, a method for generating global and local explanations for tree ensembles by using a kernelized surrogate model within the representer point framework.

3. We evaluate TREX on four benchmark datasets, demonstrating that: (1) the surrogate models accurately approximate the original tree ensembles; (2) in a data cleaning setting, TREX identifies noisy instances better than the previous state-of-the-art; (3) model performance decreases fastest when removing the examples TREX identifies as influential; (4) TREX is orders of magnitude faster than other methods; (5) TREX's local explanations can identify and explain errors due to dataset shift.

2. Background

2.1. Instance-attribution Explanations

(Koh & Liang, 2017) derived an approximation of the influence functions framework from classical statistics (Cook & Weisberg, 1980) for deep learning models. This allows one to compute the influence of each training instance on the model's prediction without having to perform leave-one-out retraining, which is intractable in most cases. Their approach is specific to deep learning models, requiring the computation of the Hessian vector product and first-order derivatives of the loss function. (Yeh et al., 2018) introduce the representer point framework for deep neural networks, a way of efficiently decomposing the pre-activation predictions of a neural network into a linear combination of the training samples.

2.2. Tree Ensembles

Since influence functions cannot be directly applied to decision trees, (Sharchilev et al., 2018) introduce LeafInfluence, an extension of influence functions to gradient boosted decision trees. Their approach considers the tree-ensemble structure to be fixed, allowing them to analyze the changes in leaf values with respect to the weights of the training samples.

(Davies & Ghahramani, 2014) introduce the idea of a random forest kernel, a type of random partition kernel which clusters the data by first sampling a level d for a tree in the forest, and then assigning datapoints to clusters (denoted by nodes at level d) based on whose ancestors were in which nodes at that level. They show that this supervised kernel is a viable and scalable alternative to the more popular linear and radial basis function (RBF) kernels. (He et al., 2014) use a random forest kernel with logistic regression to improve performance of predicting ad clicks at Facebook.

(Bloniarz et al., 2016) also build on the idea of a random forest kernel and introduce SILO, a local linear modeling technique using random forests to identify supervised neighbors. Given an instance x^t, SILO generates a local training distribution based on how often a training instance x^i ends at the same terminal node as x^t. This distribution is then used to fit a weighted linear regression model, whose prediction approximates $E(y|x)$.

(Plumb et al., 2018) apply this idea and introduce MAPLE, a model-agnostic supervised local explainer, used to generate feature-based explanations similar to LIME (Ribeiro et al., 2016). MAPLE fits a regression forest to the outputs of a black-box model, then uses a feature importance selector called DSTUMP (Kazemitabar et al., 2017) to select the most important features. When an explanation is desirable, it fits a weighted linear regression model to these features using the local training distribution as training sample weights.

Instead of fitting a separate weighted linear model for each prediction, our approach trains a kernelized model that globally approximates the tree ensemble; we can use this model to get a global perspective of which training samples are influential as well as provide local explanations as to which samples are influential to one or multiple test instances. We achieve this by introducing a new supervised tree ensemble kernel based on the leaf outputs of the trees in the ensemble.

3. Methodology

In this section, we present TREX (**T**ree-ensemble **R**epresenter-point **EX**planations), an extension of the representer point framework (Yeh et al., 2018), initially designed for deep learning models, to non-differentiable tree ensembles such as gradient boosted trees. The main idea is to decompose a prediction from the tree ensemble into a linear combination of the training points; this enables one to identify the training instances that contribute the most towards a given prediction.

In order to apply the representer theorem and obtain this linear representation, we first need to define a kernel that captures the structure of the tree ensemble. Second, we need to use this kernel to train a kernel logistic regression (KLR) or a support vector machine (SVM) model that approximates

the original tree ensemble. This gives us a kernel expansion with learned weights — what (Yeh et al., 2018) describe as "representer values" — for the training examples. Our kernelized model can then represent a prediction as a linear combination of the training examples.

3.1. Preliminaries

We assume an *instance space* $\mathcal{X}$ defined over d features, $\{x_1, x_2, \ldots, x_d\}$. For simplicity, we assume that all attributes are real-valued. In binary classification, our goal is to find a function $f : \mathcal{X} \to \{-1, +1\}$ that maps each instance to either the positive $(+1)$ or negative (-1) class.

A *decision tree* is a tree-structured model where each leaf is associated with a categorical or real-valued prediction, each internal node is associated with an attribute x_i, and its outgoing branches define a partition over the attribute's values. Given an instance $x \in \mathcal{X}$, the prediction of a decision tree can be found by traversing the tree, starting at the root and following the branches consistent with the attribute values in x. The traversal ends in a single leaf node, and the prediction of the tree is equal to the value of the leaf node.

A *tree ensemble* is a set of decision trees, each defined over the instance space $\mathcal{X}$. Given an instance $x \in \mathcal{X}$, the prediction of the tree ensemble is the sum of the predictions of all trees in the set. See Figure 1 for an example of a tree ensemble containing two trees, each defined over three attributes. As depicted, the left branch of each split is associated with the value 0 and the right branch is associated with 1. For the instance $x = (1, 0, 0)$, the first tree evaluates to 5 and the second tree evaluates to 3.8, for a total prediction of 8.8. Since 8.8 is positive, the predicted label for x is positive.

3.2. Tree Ensemble Kernels

The first part of our method is to define a kernel that computes the similarity between pairs of data points based on how they are processed by a specific tree ensemble, T. Intuitively, two data points are processed identically if they are assigned to the same leaf in each tree in the ensemble. The degree of similarity between two data points can thus be defined by comparing the specific leaf or leaf value that each data point is assigned by each tree in the ensemble.

We define our tree ensemble kernels as dot products in an alternate feature representation defined by the feature mapping ϕ: $k(x^i, x^j; T) = \phi(x^i; T) \cdot \phi(x^j; T)$. Note that the kernel is parametrized by T, since the computation necessarily depends on the structure of the tree ensemble. Different choices of ϕ emphasize different aspects of the tree ensemble structure: *LeafPath* is a tree-based kernel (Bloniarz et al., 2016; He et al., 2014; Plumb et al., 2018) where the elements in this new feature vector represent the leaves of all the trees in T; a value of 1 means x^i traversed to that leaf, otherwise the value is 0.

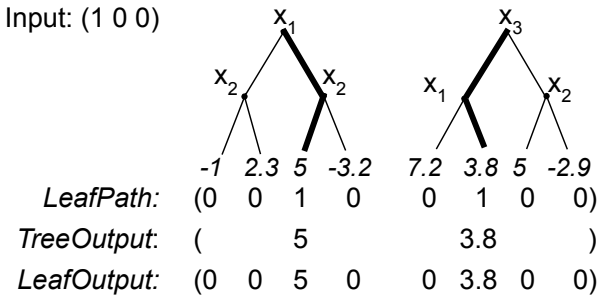

Figure 1. Different transformations of a single data instance from a simple two-tree ensemble. The numbers at the leaves represent leaf values, and the lines in bold represent the paths taken through each tree in the ensemble given the input instance.

TreeOutput is a new tree-based kernel that takes the leaf value from each tree, resulting in a vector whose length is equal to the number of trees in the ensemble. This kernel supports the fact that two instances can take different paths through a tree but still contribute to the same label. Finally, *LeafOutput* is a combination of the previous two kernels, in which we take the LeafPath representation and replace each value of 1 with the actual value of the corresponding leaf. See Figure 1 for a simple example of each kernel's feature representation.

3.3. Representer Point Decomposition

Representer theorems (Schölkopf et al., 2001) state that the optimal solutions of many learning problems can be represented in terms of the training examples. In particular, the nonparametric representer theorem (Theorem 4 from (Schölkopf et al., 2001)) applies to empirical risk minimization within a reproducing kernel Hilbert space (RKHS). This covers a wide range of linear and kernelized machine learning methods.

Representer theorems provide a natural way to explain classifiers and their predictions in terms of the training instances. Given a representation in the following form,

$$f(\cdot) = \sum_{i=1}^{m} \alpha^i k(\cdot, x^i), \tag{1}$$

the value of each weight α^i describes the global contribution of training instance x^i to the overall classifier $f(\cdot)$. For an individual prediction on a query instance x^t, the contribution of x^i to the prediction is simply $\alpha^i k(x^t, x^i)$, the instance weight times the kernel function applied to x^t and x^i.

However, a tree ensemble is not an RKHS. In order to develop a representer point method for tree ensembles, we need to define a kernel and empirical risk minimization problem that approximates the tree ensemble as closely as

possible. To this end, we use the transformation techniques from the previous section to define a tree ensemble kernel, and combine it with the empirical risk minimization methods in the following sub sections.

Kernel Logistic Regression Consider a tree ensemble T trained on a dataset $D^* = \{x^i, y^i\}_1^n$ where $x^i \in \mathbb{R}^d$ and $y^i \in \{-1, +1\}$ are the predicted labels from T. We fit a kernel logistic regression (KLR) model by optimizing the following dual objective (Yu et al., 2011):

$$\min_{\alpha} \frac{1}{2}\alpha^T Q \alpha \sum_{i=1}^{n} \alpha^i \log \alpha^i + (C - \alpha^i)\log(C - \alpha^i), \quad \text{s.t. } 0 \leq \alpha^i \leq C \tag{2}$$

where C is a penalty parameter, $Q^{ij} = y^i y^j k(x^i, x^j; T)$ $\forall i, j$, and $k(\cdot, \cdot)$ is one of the kernels from the previous section. This allows us to solve for α directly, giving weights to all training instances. Yeh et al. (Yeh et al., 2018) coin these terms "representer values"; semantically they represent the resistance of training instance x^i to minimizing the norm of the weight matrix.

After solving for α, we can decompose the prediction of a new test instance x^t as in equation (1). Thus, we can describe the contribution of training instance x^i to the overall prediction as its kernel similarity $k(\cdot, x^i)$ weighted by the its representer value α^i. The resulting value can be positive or negative, leading to excitatory or inhibitory examples that contribute towards or away from the predicted label of x^t. We end up with an explanation defined in terms of all training instances, since most of the values in α are non-zero; for a sparse-solution, we turn to support vector machines.

Support Vector Machine Following the same setup, we use an SVM (Cortes & Vapnik, 1995) as our empirical risk minimizer. Again, we optimize the dual objective (Yu et al., 2011) to find instance weights α:

$$\min_{\alpha} \frac{1}{2}(\alpha^T (Q + D)\alpha) - e^T \alpha, \text{ s.t. } 0 \leq \alpha^i \tag{3}$$

where e is a vector of all ones, D is a diagonal matrix, and $D_{ii} = 1/2C, \forall i$. Since the support vectors of an SVM are the only instances with non-zero weights, the resulting explanation is sparse.

4. Experiments

Datasets and Framework Our evaluation uses the following datasets: `Churn` ($n = 7,043$, $d = 19$) (Kaggle, 2018), which tracks customer retention; `Amazon` ($n = 32,769$, $d = 9$) (Kaggle, 2013), where the task is to predict employee access for certain tasks; `Adult` ($n = 48,842$, $d = 14$) (Dua & Graff, 2019), a dataset containing information about personal incomes; and `Census` ($n = 299,285$, $d = 41$) (Dua & Graff, 2019), a population survey dataset collected by the U.S. Census Bureau. The Churn dataset does not have a predefined train/test split, so we randomly select 20% to use as a test set.

Table 1. Test Accuracy of GBDT vs. Interpretable Models

Model	Churn	Amazon	Adult	Census
GBDT	**0.813**	**0.947**	**0.868**	**0.958**
LR	0.806	0.940	0.824	0.948
SVM (Linear)	0.806	0.940	0.822	0.946
SVM (RBF)	0.759	0.941	0.764	0.938
KNN	0.762	0.939	0.802	0.946

Our experiments use CatBoost (Prokhorenkova et al., 2018), an open source implementation of gradient boosted trees. When training TREX with logistic regression, we use liblinear (Fan et al., 2008) on the tree-ensemble feature representation to solve the L2 regularized dual problem from equation (2). For SVMs, we use the SVC solver from liblinear (Pedregosa et al., 2011) to solve equation (3).

Hyperparameter Tuning In our experiments, we measure predictive performance using accuracy, and we select the hyperparameters of our tree ensemble by performing two-fold cross-validation, where we tune the number of trees and the maximum depth of each tree. To tune the surrogate model that approximates the tree ensemble, we randomly select 10% of the training data and select the surrogate model whose predictions have the highest Pearson correlation to the tree ensemble on this data.

4.1. Tree-Ensemble Performance

First, we verify that tree ensembles are necessary to obtain good predictive performance on these baselines. In particular, if other methods that are simpler and easier to interpret perform just as well, then there is no need to explain a complex tree ensemble. Table 1 shows that gradient boosted decision trees (GBDTs) are consistently more accurate than simpler models, justifying the use of a tree ensemble and the need for methods to explain them.

4.2. Fidelity

To generate explanations, TREX first trains a surrogate model that approximates the predictive behavior of a tree ensemble. We compare the fidelity of TREX-KLR and TREX-SVM to a KNN model built using the tree-ensemble kernel, which we denote TE-KNN. Figure 3 shows that TREX with the LeafOutput kernel is able to approximate the tree ensemble very accurately, better than TE-KNN in all scenarios. The LeafOutput kernel generally had higher fidelity than the LeafPath and TreeOutput kernels; thus, we focus on the LeafOutput kernel for the remaining experiments.

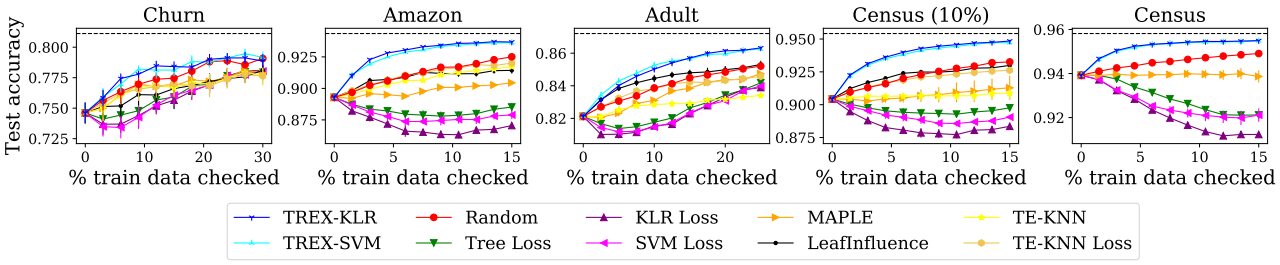

Figure 2. Change in test accuracy as training points are checked and fixed; the dashed line represents test accuracy before label corruption. Each experiment is repeated 5 times to obtain standard error bars. LeafInfluence and *TE-KNN* were too slow in generating predictions for the Census dataset, so we also include results for a 10% subset of the Census dataset.

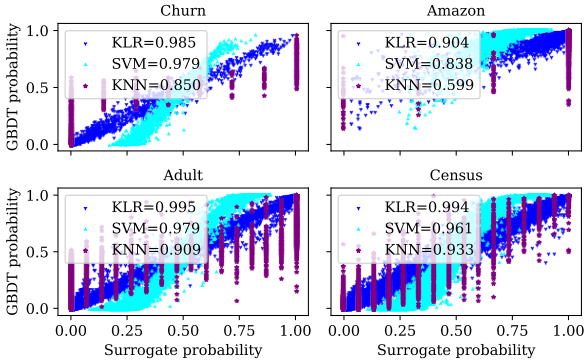

Figure 3. Comparison between a tree ensemble and three surrogate models: two TREX models and one KNN model, all using the LeafOutput kernel. The numbers in each plot represent the Pearson correlation between the GBDT and surrogate predictions.

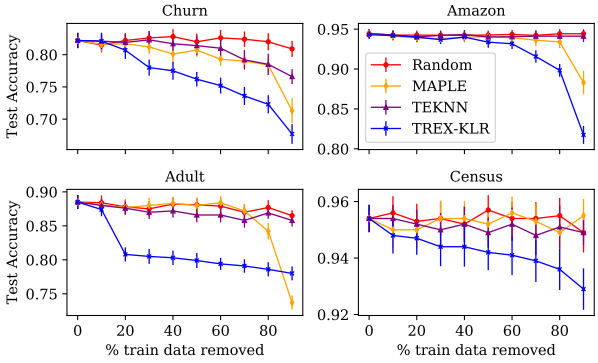

Figure 4. Change in test performance as the training samples with the most positive influence on the test set are removed in 10% increments up to 90%.

4.3. Dataset Cleaning

Datasets often contain missing or noisy labels that can degrade the performance of a classifier. We can use TREX to efficiently identify problematic training instances and relabel them as appropriate. Following a similar experimental setup as (Koh & Liang, 2017), we corrupt a training set by randomly flipping 40% of the training labels and training a tree ensemble model on the resulting noisy dataset. We then use TREX to order the training instances to be manually checked, fixing them if they had been previously flipped; the model is then evaluated on a held out test set. We take the same approach as (Yeh et al., 2018) and sort the training instances by the absolute value of their weights, α.

We compare to the following baseline orderings: *Random*, a completely random ordering; *Tree Loss*, ordered by the loss of the tree-ensemble predictions; *Surrogate Loss*, ordered by the loss of the surrogate model predictions; *TE-KNN*, ordered by neighborhood density; *MAPLE* (Plumb et al., 2018), ordered by similarity density; and *LeafInfluence* (Sharchilev et al., 2018), ordered by the influence of each training sample on itself (Koh & Liang, 2017).

Results are shown in Figure 2. On every dataset, TREX-KLR and TREX-SVM achieve the highest accuracy for each amount of data checked. In other words, TREX identifies the training instances that (after relabeling) have the greatest impact on the model's test performance. In contrast, LeafInfluence, MAPLE, and several other baselines often perform *worse than a random ordering*.

4.4. Remove and Retrain

Inspired by a recent approach that measures explanation quality for feature attribution techniques, we adapt the ROAR (**R**em**O**ve **A**nd **R**etrain) framework (Hooker et al., 2019) from measuring feature importance to measuring the influence of training samples on a set of model predictions. In this experiment, each method generates and aggregates instance-attribution explanations for a randomly selected set of $n = 50$ test instances, and orders the training data from most positively influential to most negatively influential. Then, the training data is removed in 10% increments, where the best explanatory methods cause the sharpest degradation in performance. Each experiment is repeated 20 times.

We observe that each dataset is quite robust to the deletion of

Table 2. Average Time (in seconds) to compute the impact of all training instances on a single test instance. MAPLE did not finish (DNF) fine-tuning on the Census dataset after running for 12 hours, so the computation time for that dataset is not applicable (N/A).

Fine-Tune

Model	Churn	Amazon	Adult	Census
LeafInf	0	0	0	0
MAPLE	700	5,400	11,500	DNF
TE-KNN	30	530	300	14,300
TREX-KLR	25	110	60	650
TREX-SVM	15	190	90	800

Computation

Model	Churn	Amazon	Adult	Census
LeafInf	100	3,500	1,500	29,900
MAPLE	0.031	0.048	0.059	N/A
TE-KNN	0.012	0.438	0.187	1.405
TREX-KLR	0.045	0.556	0.243	2.609
TREX-SVM	0.033	0.457	0.276	2.805

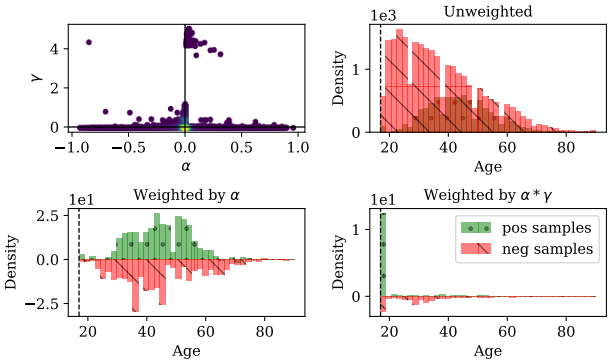

Figure 5. Detecting shift in the `Adult` dataset. Top-left: representer value α and similarity γ of each training point to a chosen test point. Top-right: histogram of positive and negative training examples in each age range. Bottom-left: same, multiplied by α. Bottom-right: same, multiplied by the product of α and γ.

training instances, maintaining relatively high accuracy even as 90% of the training data is randomly removed (Figure 4). However, we find that TREX is generally able to find the training instances that, when deleted, cause the earliest and largest degradation in model performance. LeafInfluence was too inefficient to run (see timing experiments below).

4.5. Runtime Comparison

In this section, we measure the time it takes for each method to generate an instance-attribution explanation for a single random test instance. We compare TREX to LeafInfluence (using its fastest setting); we also compare against MAPLE and TE-KNN. The measurement is broken up into fine-tune and computation costs. Fine-tuning is a one-time cost to setup the explainer (tuning any hyperparameters and training the explainer), while computation is the time it takes to use the explainer to generate an explanation for a new test instance. Each experiment is repeated five times.

TREX incurs a relatively modest one-time fine-tuning cost to train the kernelized model, roughly 1-2 orders of magnitude faster than MAPLE. After this step, TREX is able to quickly generate an explanation for a test instance, roughly 3-4 orders of magnitude faster than LeafInfluence. Our approach is fast enough to generate instance-attribution explanations in real time for individual queries, or even for groups of training and test instances, as shown in Section 4.4.

4.6. Case Study: Detecting Dataset Shift

To evaluate the utility of TREX in explaining individual predictions, we created a domain mismatch within the `Adult` dataset. In the original training set, all 395 people under the age of 18 are labeled as making less than or equal to $50K per year (negative). We reduced that set to 98 people and flipped 83 of the labels, so that 83 out of 98 17-year-olds in the training data are positive. This inevitably caused incorrect predictions in the test set, where 17-year-olds were predicted to have incomes over $50K per year. We then selected one of these incorrect predictions and used TREX-KLR[1] with the LeafOutput kernel to help explain it. In Figure 5, we first plot the representer value α and similarity to the test point γ for each training example (Top-left). We see a clump of high-similarity points, along with a few other outliers. All 97 points with $\gamma > 2.0$ have age=17.

Investigating further, we show the distribution of the age attribute in three histograms. In the overall dataset (Top-right), most examples with low age (< 25) are negative. After weighting by α (Bottom-left), their contribution to the overall model is small but positive. However, for this one test example (Bottom-right), these same points make a very large, positive contribution, explaining the final prediction of a positive label.

5. Conclusion

In this work we have developed TREX[2], a method of explaining tree ensemble predictions via the training data. We extended the representer point framework (Yeh et al., 2018) to work for non-differentiable tree ensembles by exploiting the tree-ensemble structure to create a new tree-based kernel, from which we can train a kernelized model. We demonstrated that this model is capable of closely approximating the predictive behavior of the tree ensemble, and can be used to aid in dataset debugging and help better understand model behavior.

[1] We see similar results with TREX-SVM.

[2] https://github.com/jjbrophy47/trees

References

Bloniarz, A., Talwalkar, A., Yu, B., and Wu, C. Supervised neighborhoods for distributed nonparametric regression. In *Artificial Intelligence and Statistics*, pp. 1450–1459, 2016.

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

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

Cook, R. D. and Weisberg, S. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics*, 22(4):495–508, 1980.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Davies, A. and Ghahramani, Z. The random forest kernel and other kernels for big data from random partitions. *arXiv preprint arXiv:1402.4293*, 2014.

Dua, D. and Graff, C. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2019.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9(Aug): 1871–1874, 2008.

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

He, X., Pan, J., Jin, O., Xu, T., Liu, B., Xu, T., Shi, Y., Atallah, A., Herbrich, R., Bowers, S., et al. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pp. 1–9. ACM, 2014.

Hooker, S., Erhan, D., Kindermans, P.-J., and Kim, B. A benchmark for interpretability methods in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 9734–9745, 2019.

Kaggle. Amazon.com - employee access challenge. https://www.kaggle.com/c/amazon-employee-access-challenge/data, 2013. [Online; accessed 28-April-2020].

Kaggle. Dataset surgical binary classification. https://www.kaggle.com/omnamahshivai/surgical-dataset-binary-classification, 2018. [Online; accessed 16-April-2020].

Kazemitabar, J., Amini, A., Bloniarz, A., and Talwalkar, A. S. Variable importance using decision trees. In *Advances in Neural Information Processing Systems*, pp. 426–435. Curran Associates, Inc., 2017.

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*, pp. 3146–3154. Curran Associates, Inc., 2017.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1885–1894. JMLR, 2017.

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

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

Plumb, G., Molitor, D., and Talwalkar, A. S. Model agnostic supervised local explanations. In *Advances in Neural Information Processing Systems*, pp. 2515–2524. Curran Associates, Inc., 2018.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. Catboost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, pp. 6638–6648. Curran Associates, Inc., 2018.

Ribeiro, M. T., Singh, S., and Guestrin, C. Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144. ACM, 2016.

Schölkopf, B., Herbrich, R., and Smola, A. J. A generalized representer theorem. In *International conference on computational learning theory*, pp. 416–426. Springer, 2001.

Sharchilev, B., Ustinovskiy, Y., Serdyukov, P., and de Rijke, M. Finding influential training samples for gradient boosted decision trees. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 4577–4585, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/sharchilev18a.html.

Yeh, C.-K., Kim, J., Yen, I. E.-H., and Ravikumar, P. K. Representer point selection for explaining deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 9291–9301. Curran Associates, Inc., 2018.

Yu, H.-F., Huang, F.-L., and Lin, C.-J. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1-2):41–75, 2011.