Visual Diagnosis of Tree Boosting Methods

Shixia Liu, Jiannan Xiao, Junlin Liu, Xiting Wang, Jing Wu, Jun Zhu

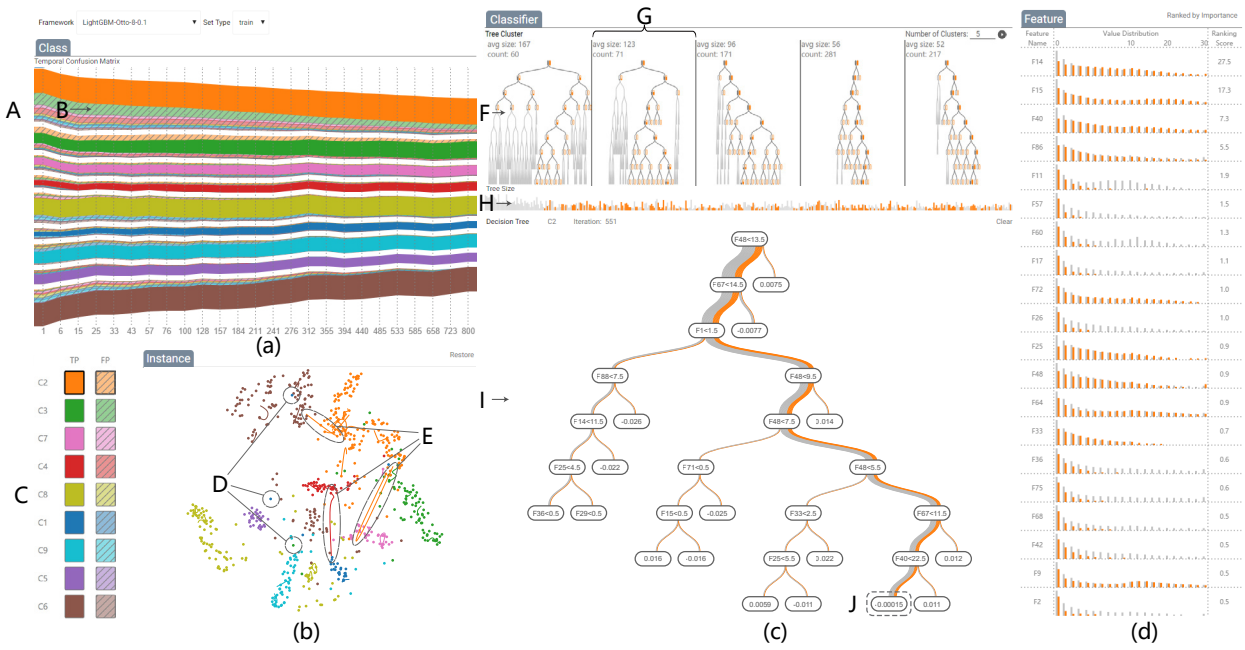


Fig. 1. BOOSTVis: (a) the temporal confusion matrix shows the evolution of model performance at the class-level; (b) the instance view reveals the relationships between instances using the t-SNE projection; (c) the classifier view provides an overview of all the decision trees and displays the selected one; (d) the feature view displays the feature distributions on the selected subsets of instances.

Abstract— Tree boosting, which combines weak learners (typically decision trees) to generate a strong learner, is a highly effective and widely used machine learning method. However, the development of a high performance tree boosting model is a time-consuming process that requires numerous trial-and-error experiments. To tackle this issue, we have developed a visual diagnosis tool, BOOSTVis, to help experts quickly analyze and diagnose the training process of tree boosting. In particular, we have designed a temporal confusion matrix visualization, and combined it with a t-SNE projection and a tree visualization. These visualization components work together to provide a comprehensive overview of a tree boosting model, and enable an effective diagnosis of an unsatisfactory training process. Two case studies that were conducted on the Otto Group Product Classification Challenge dataset demonstrate that BOOSTVis can provide informative feedback and guidance to improve understanding and diagnosis of tree boosting algorithms.

Index Terms—tree boosting, model analysis, temporal confusion matrix, tree visualization.

1 Introduction

Tree boosting, a combination of a set of moderately accurate weak learners (e.g., decision trees), has been demonstrated to be powerful and effective in many applications, such as classification and ranking [59]. Due to its practical effectiveness, tree boosting is one of the most popular methods in data science competitions. For example, the gradient boosting decision tree (GBDT) was used in the winning solution [43] at KDD Cup 2016. In Kaggle competitions, tree boosting has been used in 9 of the 14 first place winning solutions published since 2016 [62]. In comparison, deep neural networks was used in 2. Indeed, tree boosting has comparable popularity to deep learning due to its reputation of being accurate, flexible, and robust, while requiring less computational resources. In addition to the great success in competitions, tree boosting models are also widely used in commercial products, such as web search engines Yahoo [11] and Yandex [68], IBM Watson [61], IBM SPSS predictive analytics software [60], and Microsoft Azure Machine Learning Studio [66].

Despite the popularity and success of tree boosting methods, the development of a high performance tree boosting model still involves an inefficient trial-and-error process. The difficulties arise from three aspects. The first stems from the fact that the development of a good model requires comprehensive performance analysis. Insufficient or misleading information on performance can lead to premature quiting of training or overfitting. The second is caused by the complexity of a tree boosting model. Tree boosting works by sequentially growing new trees. The model's size increases over time (iteration), and its performance depends on the structures of individual trees, and changes with addition of more trees. The intertwining of the temporal and spatial changes makes it difficult to interpret the model's behavior and identify the underlying reasons of an unsatisfactory training process. The third

- *S. Liu, J. Xiao, J. Liu, and J. Zhu are with Tsinghua University and National Engineering Lab for Big Data Software. Email: shixia@tsinghua.edu.cn, xjn16@mails.tsinghua.edu.cn, liujl12@mails.tsinghua.edu.cn, dcszj@tsinghua.edu.cn.*
- *X. Wang is with Microsoft Research. Email: xitwan@microsoft.com.*
- *J. Wu is with Cardiff University. Email: wuj11@cardiff.ac.uk.*

Manuscript received 31 Mar. 2017; accepted 1 Aug. 2017.
Date of publication 28 Aug. 2017; date of current version 1 Oct. 2017.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TVCG.2017.2744378

difficulty lies in feature engineering. "Applied machine learning is basically feature engineering" [36]. The performance of a tree boosting model greatly depends on the features used at the splitting nodes. However, coming up with suitable features is still a time-consuming iterative manual process that requires expert knowledge.

Previous studies [30, 39] have pointed out that interactive debugging is critical to overcoming the difficulties in the development of statistical machine learning models. Following this suggestion, in this paper, we develop BOOSTVis, an interactive visual analytics tool that aims to help machine learning experts better understand the training process of tree boosting, diagnose the underlying reasons for good or bad performances, and make informed improvements. A demo of the prototype is available at `http://shixialiu.com/boostvis/demo/`. A multi-view visualization, as shown in Fig. 1, is at the core of BOOSTVis, which aims to show a variety of performance information at (a) the class-level; (b) the instance-level; (c) the classifier-level; and (d) the feature-level. The views can be put into two categories according to their functionalities. The first two views provide an overview of the model's performance at the class- and instance-levels. In particular, we have developed a temporal confusion matrix to reveal the complex changes of confusing classes along with the iterations in training. We also use a t-SNE projection [32] to visualize instances and their relationships. Following a comprehensive performance analysis, an expert can select the instances and times of interest. The last two views then allow performance diagnosis on the selected subsets. A node-link tree visualization is used to visualize the structure of a selected tree and the distribution of instances on it. A feature view further reveals the important features for the selected subsets. These two views work together to facilitate the examination of feature distributions on trees, and inspire the addition of new features and the use of feature subsampling.

To support our research, we used a previous Kaggle competition, Otto Group Product Classification Challenge [67], as an example, and conducted two case studies with experts. The case studies have shown that BOOSTVis helps better explore the training process and gain novel insights into the influence of tree structures. Moreover, experts can diagnose underlying reasons for unsatisfactory results more efficiently and precisely, which facilitates faster and more effective improvements of models. One of our case studies has demonstrated that by leveraging BOOSTVis, the experts built a model whose performance is better than that of the best single boosting model in the Kaggle competition.

The key technical contributions of this work are:

- **A visual diagnosis tool** that provides a comprehensive performance analysis of tree boosting models from multiple perspectives, and thus helps experts efficiently understand the whole picture of the training process and effectively diagnose an unsatisfactory performance.
- **A temporal confusion matrix visualization** that visually illustrates the complex performance changes of the model throughout the training process.
- **A combination of a tree visualization with the feature view** that helps identify the key features and examine the feature distribution on decision trees, so experts can combine the key features to create new ones, or apply a feature subsampling method.

2 Related Work

Machine learning experts are familiar with using performance metrics, such as accuracy, precision, recall, and F-scores, to evaluate the performance of a learned model [14]. Diagrams such as P-R curve and ROC curve are also widely used to visually compare the performance of several models [13]. Another popular tool is confusion matrix, which reveals the class-level distribution of data and predictions by using a contingency table to contrast actual and predicted classes [47]. Such metrics help users quickly get an overview of the model performance. However, they do not provide detailed information at the instance-, classifier-, and feature-levels, which is useful for thoroughly understanding and diagnosing the model and its training process.

To address this inadequacy, many visualization tools have been developed [1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 17, 19, 21, 22, 23, 24, 26, 27, 28, 33, 35, 37, 38, 40, 41, 42, 45, 48, 49, 50, 51, 52, 53, 54, 57, 58]. Pertinent to BOOSTVis are: 1) visualization of decision trees [4, 26, 50, 54] and ensemble models [21, 26, 48, 49, 53, 58]; 2) visualization tools that facilitates model diagnosis [1, 2, 3, 10, 15, 19, 23, 37, 38, 41]; and 3) visualization of dynamic training processes [5, 17, 21, 40, 57].

Visualization of decision trees and ensemble models. To visualize a decision tree, PaintingClass [50] uses a variation of icicle plots to reveal the local tree structure centered at the node of interest. BaobabView [54] uses a node-link tree visualization aided by confusion matrices and interactively displayed features to support interactive construction and analysis of decision trees. The node-link tree visualization is also used in BOOSTVis because of its intuitiveness. In addition to the visualization of a single tree in [50, 54], BOOSTVis takes into consideration the visualization of the ensemble of trees.

For visualization of ensemble models, EnsembleMatrix [49] presents an interactive visualization of confusion matrices to allow users to compare different classifiers at the class-level. The cascaded scatterplot [21] visualizes the class distribution of data in each stage of Adaboosting. The Transparent Boosting Tree (TBT) [58] is probably the most pertinent to BOOSTVis. It lists the feature groups and the names of all individual trees, visualizes the structures of the selected trees, and provides prediction results local to a selected path. BOOSTVis extends TBT to provide performance overview at both class- and instance-levels, enrich the information provided at the feature-level, and reveal the model evolution more explicitly.

Visualization for model diagnosis. Besides the visualization of specific machine learning models, many visualization tools have been developed to assist the analysis of prediction results and help experts find potential directions to improve a model. Many such visualization tools allow users to drill down into the instance- and feature-levels. A typical example is the visual analysis tool developed by Alsallakh et al. [2]. This tool provides a confusion wheel to display the score distribution for each class in a radial layout. In addition, it allows users to connect performance metrics at the class-level to data at the instance- and feature-levels. Such a connection helps infer the underlying reasons for a poor performance, and is also established in BOOSTVis via the use of a linked multi-view visualization. Upon gaining insights into the potential reasons for a poor performance, users often want to improve the model based on the knowledge gained. To this end, Paiva et al. [37] propose an approach to interactively select training samples, modify their labels, and incrementally improve the classification model towards users' expectation. Although the aforementioned visual diagnosis tools can achieve some success in improving a model, they do not provide a comprehensive analysis of the training process.

Visualization of dynamic training process. In the area of deep learning visualization, in addition to analyzing a snapshot from the training process [12, 27, 28, 33, 51], there have been some efforts to investigate the dynamic training process based on multiple snapshots [17, 40, 57]. Matrix cube, a stacked matrix representation of networks, has been developed to visualize dynamic networks [5]. In the visualization of boosting models, the cascaded scatterplot [21] shows the model evolution in terms of data distribution using horizontally aligned scatterplots. BOOSTVis also uses horizontally aligned segments to represent time steps, but reveals evolution not only at the instance-level, but also at the class- and classifier-levels.

3 Background

Boosting methods belong to the family of ensemble methods that train multiple base learners and combine their predictions to obtain better performance. In tree boosting, specifically gradient boosting decision tree (GBDT), the base learners are decision trees, and they are trained sequentially with the later trees focusing more on the mistakes made by the earlier ones. This makes the decision trees complementary to each other, and hence their ensemble greatly reduces the bias in each individual tree. A cousin of GBDT is random forest (RF) whose base learners are also decision trees, but they are trained in parallel on different data subsets and using random features. This randomness makes the individual trees less dependent. As a result, their ensemble reduces the variance when applying to different data, but cannot reduce the bias. Large individual trees are usually generated in RF to alleviate

the bias. The dependence of the individual trees in GBDT makes it generally perform better than RF, but also makes it harder to understand the relationships between the trees and properly tune the performance. Thus, an analysis of the trees and their evolution would be of great help in building high performance tree boosting models. Parallel computing is non-trivial for tree boosting, but has been achieved in two popular tree boosting systems, i.e., XGBoost [9] and LightGBM [65], which have been chosen to be the underlying boosting systems in BOOSTVis.

XGBoost has achieved great success in data science competitions. LightGBM has been available for only a few months, but initial experiments have demonstrated impressive speed boosts. In recent 4 Kaggle competitions, at least 3 first place winning teams have employed LightGBM in their solutions. This indicates its potential widespread use in real-world applications. As a newly available system, LightGBM is still undergoing the exploration stage. It is therefore of great interest to investigate LightGBM first, then compare it with XGBoost.

4 BOOSTVIS

4.1 Requirement Analysis

The design of BOOSTVis is grounded by interviewing three groups of machine learning experts and practitioners. The first group consists of four major developers (E_1, E_2, E_3, E_4) of LightGBM [65]. One of the developers, Guolin Ke (E_1), contributes to 60% commits of LightGBM on GitHub. The second group contains the initial starter and main contributor of XGBoost [9], Tianqi Chen (E_5), at the University of Washington. The last group consists of three researchers (E_6, E_7, E_8) at Microsoft Research Asia, who often use AdaBoost [59] and RealBoost [44] for face detection and alignment. The interviews were semi-structured with a focus on the processes of the participants' model analysis and diagnosis. In addition, we also examined 129 blogs related to the Kaggle competition, including the winner's blogs as well as competition discussion blogs. Based on the interviews with the experts, experiences shared in the blogs, and our previous research experience, we have identified the following requirements for BOOSTVis.

R1. Examining the model performance for each class and its evolution through the iteration process. A model's performance is the ultimate concern of experts, and is examined at the beginning of any model analysis. The experts said that they usually examine the performance for each class to identify the confusing classes to debug into. This was echoed in some blogs pertaining to Kaggle competitions, where competitors shared their experiences of shaping the reason behind the model construction by using confusion matrices [63, 64]. Because boosting is a sequential process, the experts also wanted to know how the performance evolved through the process. For example, expert E_5 mentioned the need to observe the model performance through the iterative process in order to apply early stopping to avoid overfitting.

R2. Conveying relationships between instances and several important types of instances. Previous research [2, 8, 29, 37, 57] has demonstrated that displaying relationships between instances is useful in confirming known facts and revealing unseen patterns of a model. Expert E_1 also considered such information useful for the identification of important instances such as outliers (e.g., instances whose class labels change among neighboring weak learners in the late training phase) and representative instances, which could enhance the understanding of the model and provide insights for improvements. This consideration has been supported by a recent research study [40], which showed how instance-level information helped experts explore training effects, understand mis-classifications, and improve the model.

R3. Exploring the structures of classifiers (decision trees) and instance distribution on tree structures. A tree boosting model often contains hundreds and thousands of decision trees. To better understand the model, the experts expressed the need to analyze and summarize the tree structures. From the discussions, we collected what they wanted to explore, including the types of tree structures, the most important trees, the evolution of tree size, and the features used in each tree. Expert E_5 gave an example where an end-user would like to explore the tree structures to find important patterns in identifying fraud in his domain. In addition to the tree structures, how instances distributed on the trees is also of concern to the experts. "Unbalanced distribution can cause problems," said expert E_6, which indicate the use of this distribution in model diagnosis.

R4. Examining and comparing important features for the class of interest. All the experts mentioned that they examined feature importance when trying to improve a trained model. "Feature importance gives insights into where we should endeavor to refine a model," said expert E_5. Expert E_1 pointed out that examining feature importance could provide useful feedback for effective feature engineering. After identifying confusing classes in **R1**, the experts emphasized the need of examination at the feature-level to find the underlying reasons. A typical technique they adopted is to rank the features according to their importance in separating the classes. Expert E_1 further mentioned that the distributions of features helped in finding appropriate split values to generate the decision trees.

R5. Connecting model performance with the class, the instance, the classifier, and the feature. All four aspects are important to a model's performance. Expert E_1, E_2, E_5, E_7, and E_8 described the process to improve a boosting model as an iterative rectification of data, features, and classifiers. However, as pointed out in [39], a lot of current tools encapsulate only a portion of the process, which can mislead developers to become over focused on one aspect. In our discussions, the experts clearly expressed the need for an interactive diagnosis tool

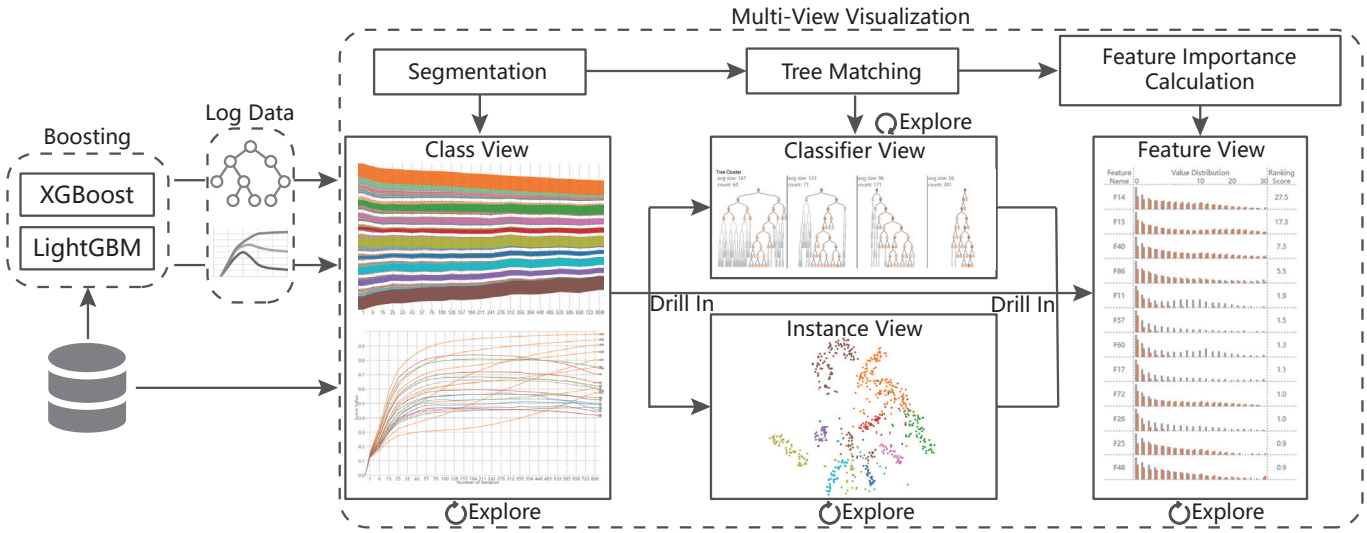


Fig. 2. System pipeline. The training data, the decision tree, and prediction scores at each iteration of training are used to generate the multi-view visualization. The visualization provides four levels of information to allow in-depth examination of the model performance and the training process.

that can support their exploration of all aspects in model diagnosis, including a variety of information at the class-, instance-, classifier-, and feature-levels.

4.2 System Overview

Motivated by the identified requirements, we have developed BOOSTVis, a visual analytics tool that combines the following features:

- A class view that discloses confusions between classes, shows the performance evolution of each class, and displays prediction scores for instances within a specific class (**R1**).
- An instance view that displays the relationships between instances as well as instance clusters based on the prediction scores (**R2**).
- A classifier view that unfolds the structures of the boosted trees. It provides summary information about the model structure, depicts the node-link structure of a specific tree and the instances' distribution on it (**R3**).
- A feature view that lists the important features for discriminating instances in different instance subsets (**R4**).
- Rich interactions that allow experts to smoothly switch between the four levels of information (e.g., class-level and feature-level) and help users more effectively understand and diagnose the tree boosting model (**R5**).

The system pipeline is shown in Fig. 2. BOOSTVis takes the classification training data as the input. The training data is fed into XGBoost or LightGBM to learn a tree boosting model. At each iteration of training, the tree structures and the prediction scores are obtained and recorded in the log data. Both the input and the log data are used to generate the multi-view visualization. The visualization is supported by several mining techniques, such as time series segmentation and tree matching, and provides four levels of information and rich interactions that enable experts to effectively understand and diagnose the model and the training process.

5 Visualization

Since using a familiar visual metaphor enables experts to focus directly on the task itself [34], the basic design principle is to exploit or augment familiar visual metaphors to guide analysis. Based on this principle and the design requirements, we develop four visualizations that convey various diagnosis-related information at the class-, instance-, classifier-, and feature-levels. In particular, a temporal confusion matrix is designed to visually illustrate the performance changes of the model within a training process. A tree visualization is developed to facilitate the understanding of the structures of the classifiers The t-SNE projection and grouped bar charts are employed to display the information at instance- and feature-levels. Furthermore, we also illustrate how these four views work together in the understanding and diagnosis tasks.

5.1 Class-Level as Temporal Confusion Matrix

The training process produces a set of time series at the class-level, which is very useful for performance analysis. The time series data can be classified into two categories: 1) the temporal confusion statistics that measure the rate of the class being confused; and 2) the temporal prediction scores of instances within a selected class. In this section, we describe the visualization design for conveying the confusion statistics over time as well as the non-linear segmentation method for quickly identifying the time ranges of interest.

Design of temporal confusion matrix. In the field of machine learning, a confusion matrix is commonly used to present the accuracy of a learning algorithm. As shown in Fig. 3(a), each column of the confusion matrix represents the instances in a predicted class while each row represents the instances in an actual class. A confusion matrix provides a very convenient way to compare the predicted labels with the actual ones, which can be easily understood by machine learning researchers and practitioners. As a result, we use the confusion matrix as the base of our class-level visualization.

Although the confusion matrix works very well for performance analysis of a single classifier, it fails to convey the temporal perfor-

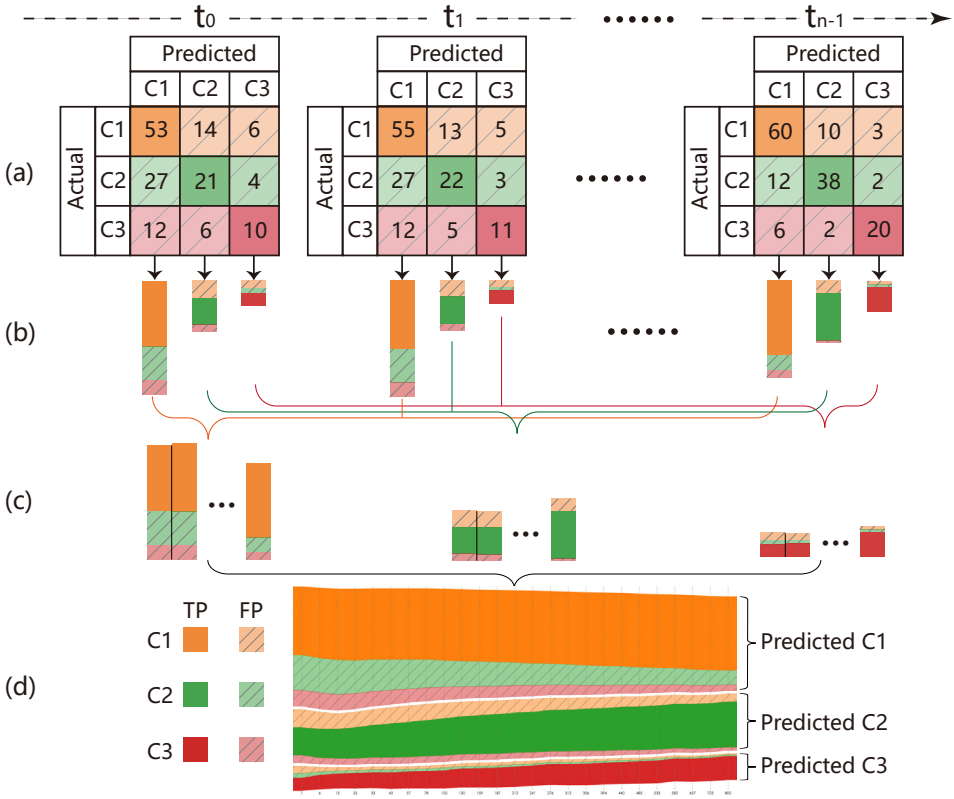


Fig. 3. Design of the temporal confusion matrix. (a) Confusion matrices are organized along the training iterations. (b) Each column is represented by a stacked bar with the solid class color for true positives (TPs, the diagonal cell), transparent and the shaded class colors for false positives (FPs, the non-diagonal cells), and the bar height representing the instance number in the cell. (c) Stacked bars at different iterations are connected to form a stripe. (d) All the stripes are stacked to form the temporal confusion matrix that reveals the changes of TP and FP for each class over iterations.

mance changes of the weak learners in the training process of a boosting model. To tackle this issue, we represent each column/row as a stacked bar. Without loss of generality, we take representation of the column as an example to illustrate the basic idea of the visualization design. First, we create a stacked bar where each bar item corresponds to a cell in the column (the predicted class) of the confusion matrix (Fig. 3(b)). The height of each bar item encodes the instance number of the corresponding cell. Next, for each predicted class, we connect its stacked bars at different time points to form a class stripe (Fig. 3(c)). Within each predicted class stripe (e.g., class C1), the true positives (TPs) are displayed with their actual class color; while the false positives (FPs) are displayed with their actual class color with increased transparency, shaded by oblique lines. In Fig. 3, taking class C1 as an example, a true positive is a C1 instance that is correctly classified as C1; a false positive is an instance in C2 or C3 that is incorrectly classified as C1.

Finally, as shown in Fig. 3(d), all the class stripes are stacked together to form the temporal confusion matrix. The class stripes are arranged in ascending order of class-wise performance with the most problematic class on top. When an expert selects a specific predicted class, the temporal prediction scores of its instances is represented by a line chart (Fig. 10(a)). To reduce the visual clutter, we use K-means, a commonly used clustering method, to cluster the instances with the same actual label according to the adjacencies of their score lines. Fig. 10(a) shows the temporal prediction scores of the instances in the predicted class C1. Here, the line color encodes the actual label of the cluster of instances. It is easy to see that there is an instance cluster with a higher prediction score during the iteration (Fig. 10A). However, its actual label is C2. This is an anomaly cluster in this predicted class. As a result, the temporal prediction scores help identify anomalies quickly.

Segmentation. A tree boosting model often contains hundreds of weak learners. Accordingly, the temporal confusion matrix consists of a set of time series with hundreds of time points. To help experts quickly identify the time points of interest, we need to segment the time series produced in the training process. The segmentation is primarily based on the temporal confusion matrix, because of its importance for understanding. To simplify the calculation, we use a vector to represent the confusion matrix at each time point. Specifically, for each time point, we transform its confusion matrix into a vector by taking its columns one by one. With this transformation, the temporal confusion matrix is transformed into a time series with vector $\boldsymbol{x}(i)$ representing the sample at time i.

We assume a time series s consists of n vectors $\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots, \boldsymbol{x}(n)$ along the time dimension. We denote $s(i,j)$ to be a segment of s: $\boldsymbol{x}(i)$, $\boldsymbol{x}(i+1), \ldots, \boldsymbol{x}(j)$. A k-segmentation of s is a sequence $s_1, s_2, \ldots, s_k$, which divides s into k consecutive segments and each s_i is non-empty. The major objective of time series segmentation is to find k segments that minimize the intra-segment variances. To this end, we formulate time series segmentation as a dynamic-programming problem [20]. Specifically, the cost function to measure the internal variance of each segment is defined as:

$$cost(a,b) = \sum_{l=a}^{b} \| \mathbf{x}(l) - \mu_{ab} \|^2 \tag{1}$$

where μ_{ab} is the average of the vectors between indexes a and b.

Let $f(i,j)$ be the best j-segmentation for the first i samples. Then we have:

$$f(i,j) = \min_{l}\{f(l,j-1) + cost(l+1,i)\} \quad l = j-1, j, \ldots, i-1. \tag{2}$$

The above recursive function can be efficiently solved using dynamic programming. $f(n,k)$ is the best k-segmentation for time series s.

5.2 Classifier-Level as Tree Visualization

As discussed in Sec. 4.1, experts are interested in: 1) the types of tree structures as well as the most important tree in each type; 2) tree size changes; 3) instance distribution and features used on the tree. To support the analysis of the aforementioned information, we have designed a tree visualization that consists of three coordinated components corresponding to the experts' interests. A core challenge here is to deal with hundreds and thousands of trees generated through the iterations. To solve this problem, we have proposed a tree matching algorithm that efficiently and effectively detect the major tree clusters (types) from many decision trees.

Design of the tree visualization. The tree visualization consists of three components, each of which corresponds to one type of information the experts are interested in. The **tree cluster component** (Fig. 1F) provides an overview of the tree structures by displaying the structure clusters. For each cluster (e.g., Fig. 1G), we display its most representative tree (clustering center) as a node-link diagram. The node-link diagram is used here due to experts' familiarity with this representation. To effectively display trees with hundreds of nodes, we follow the idea of "focus + context" and use a tree cut algorithm [31] to highlight the layers of interest, and gray out the others. The **tree size component** (Fig. 1H) shows how tree size changes during the training process. Here tree size refers to the number of nodes in the tree. In this component, the horizontal axis represents time (iteration) and each bar corresponds to the tree size at one iteration. When a user selects a cluster of interest, the bars that correspond to the trees in the cluster will be highlighted. Users can also interact with the bars to select a decision tree of interest. The **decision tree component** (Fig. 1I) then presents the instance distribution on the tree and the features used to split the set of instances. Specifically, the label on the internal tree node shows the feature name and the corresponding split value. For leaf nodes, we follow the suggestions of the experts to display their gradient values. The distribution of the instances on the tree is encoded by using the edge thickness, which is proportional to the number of instances that flow through this edge. In this component, the TPs and selected instances are represented by using the color of their actual class while others are colored gray.

Tree matching. The goal of the tree matching algorithm is to estimate the distances between the tree structures. Based on the distances, the tree clusters can be easily derived by using k-medoid [25]. In BOOSTVis, we extend the widely used tree edit distance [7] to derive a distance more desirable in our application and more computationally efficient.

Given two trees T_1 and T_2, the tree edit distance computes a matching $M = \{(v,w)\}$, where $v \in T_1$, $w \in T_2$ are tree nodes and (v,w) indicates v is matched to w. Taking the matching costs between tree nodes as input, the tree edit distance finds the M with the smallest total cost. Two problems arise when applying this distance. First, calculating the tree edit distance has been shown to be an NP-complete problem [7]. It is very time consuming to calculate the distances between trees with hundreds of tree nodes. Second, in our application, the experts consider the level of a tree node is important. Thus, it is not desirable to match a node at the top of the tree to a node at the bottom.

To solve the aforementioned problems, we extend the original model by adding a level constraint and propose an efficient algorithm to solve the extended model. Specifically, our model is formulated as

$$d(T_1,T_2) = \min_{M_c}\Big(\sum_{(v,w)\in M_c} \gamma(v \mapsto w) + \sum_{v\in T_1} \gamma(v \mapsto \lambda) + \sum_{w\in T_2} \gamma(\lambda \mapsto w)\Big)$$
$$M_c = \{(v,w)|\ |l(v) - l(w)| \le l_{th}\}. \tag{3}$$

Here M_c represents a match with level constraint, $l(v)$ denotes the level of v on the tree, and l_{th} is the maximum level difference allowed for two matching nodes ($l_{th} = 1$ in BOOSTVis). By adding the level constraint, the solution space is largely reduced and the problem can be solved more efficiently. $\gamma(v \mapsto w)$ is the cost of replacing v by w, $\gamma(v \mapsto \lambda)$ and $\gamma(\lambda \mapsto w)$ measure the costs of matching tree nodes to a null node λ. In BoostVis, $\gamma(v \mapsto w) = 0.5(d_{KL}(v,w) + d_{KL}(w,v))$, where $d_{KL}(v,w)$ is the KL-divergence between v and w, each of which is represented by the instance distributions among different classes.

To minimize Eq. (3), we leverage the idea of beam search [16], in which a predefined number (m) of partial solutions are kept as candidates. Specifically, We start by matching the tree nodes at the first level. The top m matches that minimize the cost at this level are stored as candidates. When processing the next level, all matches consistent with the stored matches are calculated and sorted according to their costs. Again, the m-matches with the lowest costs are stored before going to the next level. In our experiment, we found $m = 20$ well balances both efficiency and effectiveness.

5.3 Interactive Understanding and Diagnosis Context

Understanding and diagnosing a boosting model is usually a trial-and-error process. To aid experts with this process, we provide an interactive exploration environment with four coordinated views, i.e., class view, instance view, classifier view, and feature view (Fig. 2). We first introduce the instance view and the feature view; then we illustrate how these views work together through coordinated interactions.

Instance view. The instance view employs the well-known t-SNE projection to demonstrate the relationships between instances of different classes. Each instance is represented by a m-dimensional score vector. Here m is the number of the classes and the i-th component corresponds to the prediction score of the i-th class. As shown in Fig. 1(b), instance points are colored according to their actual classes. The color-coding scheme is shown in Fig. 1C. In addition to normal points, instance view allows users to investigate outliers, such as instances that move between different classes in the late training phase (Fig. 1E). To efficiently handle a large number of instances, we employ an outlier-based random sampling method to improve scalability. Specifically, we set a higher sampling rates for the outliers than that of the normal instances. Although sampling can reduce visual clutter effectively, it may also lose some important instances. To address this issue, we employ an incremental t-SNE projection method. Particularly, for each new instance to be visualized, its initial position is determined by several already placed instances, which are close to this instance. All the instances with their positions are input to t-SNE for their final projection result.

Feature view. As shown in Fig. 1(d), the feature view consists of a set of grouped bar charts to illustrate the feature distribution. Experts are interested in the role that features play in separating the selected subset of instances from others. Accordingly, for each feature (e.g. Fig. 4A), we display its distribution on the selected subset of interest (colored bars) as well as its distribution on other instances (grey bars). To rank the features, we leverage the feature importance metric adopted by LightGBM [65] and XGBoost [9], which measures the importance by the increase of purity after splitting instances by using the feature.

Interactive analysis. The coordinated interactions among the four views facilitate the examination of a variety of information at different granularities, thus forming a convenient process of hypothesis generation and verification. Particularly, the confusion matrix and prediction scores in the class view, as well as t-SNE projection in the instance view, provide an overview of the model's performance at the class- and instance-levels. Based upon the examination of the model's performance at different instance subsets, an expert can easily discover the subset(s) of interest. After selecting an instance subset, their distributions on different decision trees are displayed in the classifier view (Fig. 1(c)). In addition, the feature distribution within these subsets is displayed in the feature view (Fig. 1(d)). With such information, an expert further analyzes and diagnoses the training process. Examples of the interactive analysis will be given in the case studies (section 6).

6 Case Study

In this section, we demonstrate the usefulness of BOOSTVis by conducting two case studies on a real-world dataset used in the Kaggle competition - Otto Group Product Classification Challenge [67]. The dataset contains 206,246 products sold by the Otto Group, one of the world's biggest e-commerce companies. The products belong to nine categories and are described by 93 obfuscated numerical features. In addition, the instance distribution over the nine categories is imbalanced. As a result, this dataset is very difficult to tackle. The competitors were asked to train a model for the classification of the products. The model's performance was measured using the multi-class logarithmic loss function (denoted as LogLoss for simplicity's sake):

$$L = -\frac{1}{n}\sum_{i}^{n}\sum_{j}^{m} y_{ij}\log(p_{ij}). \tag{4}$$

Here n is the number of products in the test set, $m = 9$ represents the number of classes, y_{ij} is 1 if and only if product i belongs to class j, and p_{ij} denotes the predictive probability that product i belongs to class j. Smaller L indicates better model performance.

6.1 Understanding the Training Process

This case study was collaborated on with the major developers of LightGBM (E_1, E_2, E_3, E_4). The evaluations were mainly based on LightGBM, and focused on evaluating the effectiveness of BOOSTVis in terms of delivering useful and innovative information about a model and its training process. The experts emphasized exploring all aspects of a training process to enhance understanding, which would in turn facilitate subsequent model diagnosis. A comparison between the training processes using LightGBM and using XGBoost was also conducted by the experts.

Performance for each class (R1). The experts began the analysis by observing the performance for each class. After looking at the temporal confusion matrix (Fig. 1(a)), they immediately realized that the most problematic class was the orange one. As shown in Fig. 1A, throughout the training process, most of the misclassified instances were incorrectly assigned to the orange class. Among all the misclassified instances, those belonging to the green class were the most confused with the orange class (Fig. 1B).

Relating performance to features (R4). The performance for each class differed significantly from each other. To find possible explanations, the experts turned their attention to the feature view.

Expert E_1 first selected the violet class, which had good classification results, and examined its important features (Fig. 4(a)). He immediately noticed feature *F34*, which had a disproportionally large importance value. Looking further into the histograms, he noticed that *F34* apparently had different distributions on instances of the violet class (violet) and on instances of other classes (gray). Thus, E_1 believed *F34* had an important role in discriminating the violet class from the others. For verification, he checked the corresponding tree in the first iteration of training (Fig. 5). He found that *F34* was used in the top layers of the tree, and the first layer already separated most instances of the violet class from the others, which explained the good classification results for the violet class.

E_1 also examined the features for the orange and green classes in the feature view (Figs. 4(b)(c)). No features with dominant importance were observed for the two classes. The distributions of important features on instances of the orange or green class did not differentiate well from their distributions on instances of other classes. "This means, unlike the violet class, there is no single feature that can separate

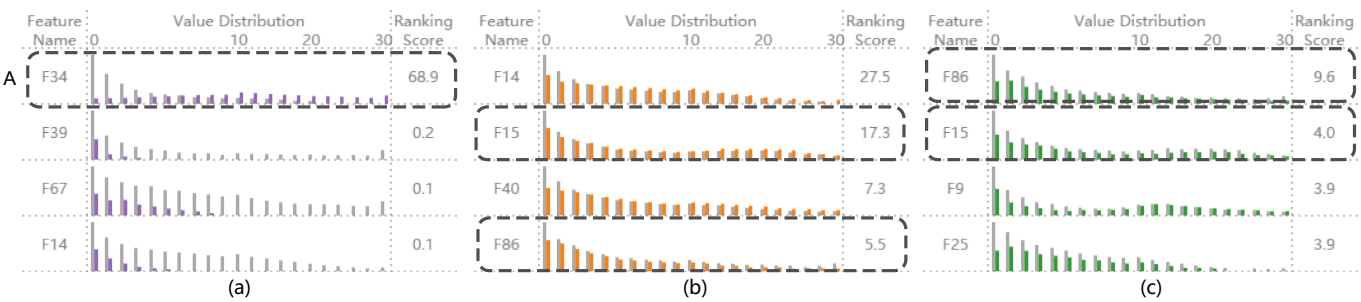


Fig. 4. Comparing feature distributions for (a) the violet class, (b) the orange class, (c) the green class with other classes (gray). Dominant feature (*F34*) is found in (a) because of the apparently different distributions. Overlapping features (*F15* and *F86*) are found in (b) and (c).

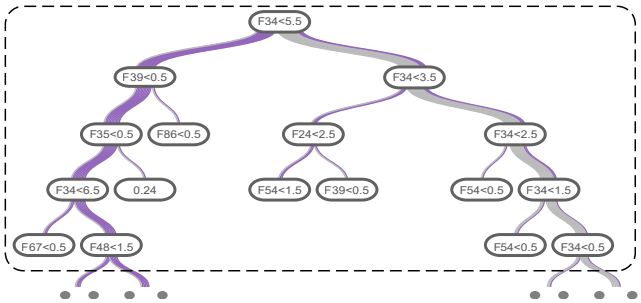


Fig. 5. Decision tree for the violet class in iteration 1. Feature *F34* at the top layer can separate most instances of the violet class from the others.

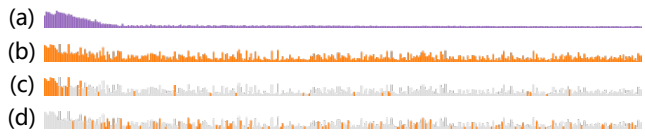


Fig. 6. Changes of tree size throughout the training process for (a) the violet class, (b) the orange class. (a) is more "top-heavy" than (b). (c) and (d) highlight the sizes of trees in the 1^{st} and 4^{th} clusters (Fig. 1F) for the orange class respectively. Trees in the 1^{st} cluster are more centered at the beginning of training than trees in the 4^{th} cluster.

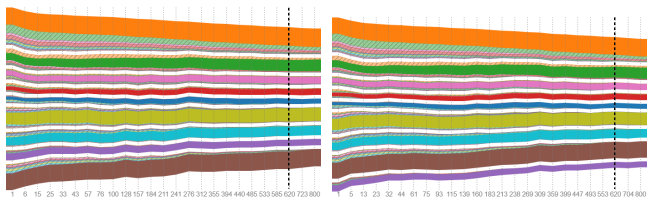
the orange or green instances from the others," said E_1. He further discovered important features that overlapped between the two classes, such as features *F15* and *F86*, and reasoned it to be the cause of the confusion between the two classes, "because their important features are not distinct enough."

Exploring iterative classifier updates (R3). In addition to examining the features, the experts also wanted to explore the series of classifiers to better understand the training process.

E_1 examined the classifiers for the violet class. Fig. 6(a) shows a bar chart showing how the tree size changed throughout the training process. It can be seen from the bar chart that larger trees occurred at the beginning of the training process, while the later iterations generated trees that were smaller. Tree size is an indicator of fitting ability; larger trees with more node splittings are more important to training. This "top-heavy" bar chart made it clear to the expert that a successful training process tended to have important classifiers centered at the beginning. E_2 observed a similar "top-heavy" bar chart for the orange class (Fig. 6(b)). However, unlike the violet class, large trees still occurred during the middle and later iterations of training, which was considered by E_2 as an indicator of struggles to separate some orange instances from the instances of other classes.

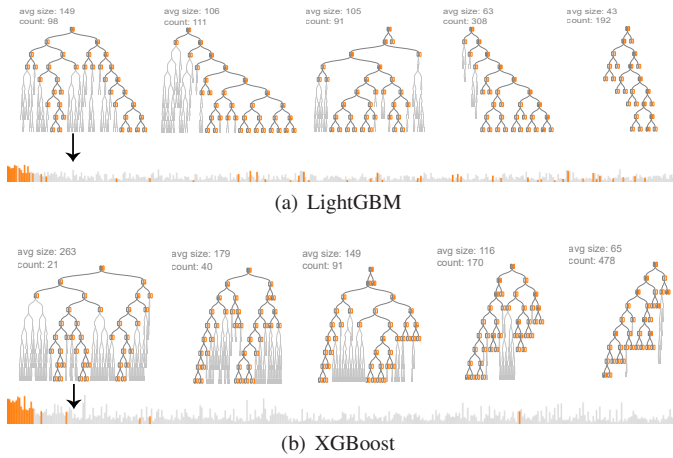
The experts then looked at the clusters of trees for the orange class (Fig. 1F). The first cluster had a more balanced distribution of instances on the leaf nodes. Highlighting this cluster of trees in the bar chart (Fig. 6(c)), the experts found that these trees were larger and mostly occurred at the beginning of the training process. The fourth cluster, on the other hand, consisted of trees with one prominent leaf node containing the majority of the instances. The bar chart (Fig. 6(d)) showed that these trees were smaller and occurred in later stages of training. This exploration revealed that trees with more balanced instance distributions tended to be generated at the beginning of a training process.

The two explorations indicated some correlation between a tree's "importance" and "balance". Expert E_4 selected the representative decision tree for the 4^{th} cluster, namely the 551^{th} tree, to investigate further into this matter. Fig. 1I shows the tree for the orange class at the 551^{th} iteration. The expert checked the leaf node where the most instances were centered (Fig. 1J), and found that the value added to these instances was only -0.00015, insignificant compared to the values (magnitudes larger than 0.01) added at other leaf nodes. This finding verified there was little contribution from this classifier to the majority of the instances passed through it. Checking the main path that the instances followed, expert E_4 noticed the very high threshold value at each node on the path. "It is a very useful finding", claimed



(a) LightGBM (b) XGBoost

Fig. 7. Comparing performances of LightGBM and XGBoost models. The black lines mark the 620^{th} iteration in (a) and (b), where (b) has better performance and faster convergence than (a).



(a) LightGBM

(b) XGBoost

Fig. 8. Comparing classifiers of LightGBM and XGBoost models. (b) has bigger average tree size than (a). Larger trees are more centered at the beginning of training in (b) than in (a).

E_4, "...highest value is favored in LightGBM when there is not a good threshold...an issue I haven't noticed before."

Comparing LightGBM and XGBoost (R5). The above evaluations were based on the models in LightGBM. At the request of the experts another model trained on the same data in XGBoost with comparable settings was also passed into BOOSTVis to facilitate a comparison between the two. For simplicity's sake, we will refer to the two models as the LightGBM model and the XGBoost model.

Fig. 7 shows the temporal confusion matrices conveying the performances of the two models. After close observation, the experts agreed that the training in XGBoost converged faster than the training in LightGBM. For example, at the marked 620^{th} and the final 800^{th} iterations, the shaded green or red stripes adjoining the orange one were noticeably narrower in XGBoost than in LightGBM. The narrower shaded stripes means less misclassified instances. The experts also noticed the smaller time segments in the training of the XGBoost model. The marked 620^{th} iteration was in the 20^{th} segment in training the LightGBM model, but was in the 21^{st} in training the XGBoost model. Based on the segmentation objective (Eq. 1), smaller segments result from bigger variance over time, which indicated the training of the XGBoost model had faster performance change, i.e., faster convergence.

To investigate the underlying reasons for this, expert E_4 examined the two models' tree clusters (the glyphs). Fig. 8 shows the tree clusters for the orange class. Expert E_4 noticed that the average tree sizes of the XGBoost model were bigger than those of the LightGBM model. Highlighting the cluster with the biggest average size in the bar chart, he further observed that large trees were more centered at the beginning of training the XGBoost model. The same observations were made for the other classes. As larger trees usually indicate stronger fitting ability, this observation gave an explanation for the faster convergence of the training in XGBoost. But on the other hand, XGBoost's stronger fitting ability may have a more adverse effect on its generalization ability. This was confirmed from the LogLoss on the validation set, which was

0.45330 for the XGBoost model, slightly worse than the 0.45269 for the LightGBM model.

6.2 Diagnosing an Unsatisfactory Training Process

This case study shows how BOOSTVis helps experts efficiently improve an unsatisfactory base tree boosting model and make it perform better than the best single boosting model in the Kaggle competition. The base boosting model was trained by using grid search, which is a traditional parameter optimization method. Specifically, we trained 35 tree boosting models by using different tree depths ($d \in \{2,3,...,8\}$) and learning rates ($\eta \in \{0.01,0.05,0.1,0.5,1\}$). The model with the lowest (best) LogLoss on the validation set ($L = 0.45467$) was chosen as the base model. We then asked experts E_1 and E_7 to analyze the base model by using BOOSTVis. Their findings were collected via emails and phone meetings. We iteratively retrained the model by following their suggestions. After the following first three steps, the LogLoss of the model was decreased to 0.43777, which is better than the best single boosting model in the competition ($L = 0.43902$).

Tuning parameters by connecting model performance with class-level and classifier-level information (R3, R5). From the temporal confusion matrix (Fig. 1(a)), the experts noticed the obvious confusion between the orange and green classes. Curious about the underlying reasons for this confusion, the experts selected instances that belonged to the orange ($\boldsymbol{I}_o$) and green ($\boldsymbol{I}_g$) classes respectively and observed how these instances were distributed in the decision trees (classifiers).

Fig. 9(a) shows the decision tree at the 50^{th} iteration. After observing this tree, expert E_1 realized that although some tree nodes (e.g., Fig. 9A) worked well when differentiating $\boldsymbol{I}_o$ and $\boldsymbol{I}_g$ from instances that belonged to other classes (gray), no tree nodes could easily differentiate $\boldsymbol{I}_o$ from $\boldsymbol{I}_g$. As shown in Fig. 9B, $\boldsymbol{I}_g$ is still mixed with $\boldsymbol{I}_o$ even on the leaf node. This phenomenon was also observed on other trees. E_1 speculated that to improve the purity of the corresponding clusters, more features are needed. Accordingly, he suggested that we change the control parameter of the tree size. To avoid overfitting, LightGBM allows users to control the tree size by limiting the maximum depth of the decision trees. Since the maximum tree depth was predefined, $\boldsymbol{I}_o$ and $\boldsymbol{I}_g$ could only be differentiated by using a small number of features (no larger than the maximum tree depth). To increase the number of features used while still avoiding overfitting, E_1 suggested that we limit the number of leaf nodes on the tree rather than limit the tree depth.

Fig. 9(b) shows an example decision tree of the new model built by following E_1's suggestion. The tree grew much deeper when its depth was not limited and some tree nodes at the bottom (e.g., Fig. 9C) could easily differentiate between $\boldsymbol{I}_g$ and $\boldsymbol{I}_o$. After this step, the LogLoss was decreased from 0.45467 to 0.44568.

Adding new features by examining and comparing feature distributions (R4). Expert E_7 then tried to tune the model by adding features. It has been observed that adding interaction features (the combinations of existing features) might be helpful in improving model performance [56]. A limitation of tree boosting is that only one feature can be used each time the instances are splitted. Adding interaction features addresses this limitation and thus is useful in improving the model accuracy. To find good interaction features, E_7 tried three different strategies, all of which were supported by BOOSTVis: 1) adding features to shorten the distance between high-confidence TPs and low-confidence TPs; 2) adding features to enlarge the distance between TPs and FPs; and 3) adding features to shorten the distances between TPs and FNs.

Fig. 10 shows how the interaction feature *F64 + F9* was discovered. Since the orange class was the most problematic, expert E_7 first examined the prediction scores for the instances classified to the orange class (Fig. 10(a)). He then selected two instance clusters: $\boldsymbol{I}_{o1}$ whose prediction scores were high during the iteration and $\boldsymbol{I}_{o2}$ whose prediction scores were low. Although both $\boldsymbol{I}_{o1}$ and $\boldsymbol{I}_{o2}$ were TPs, instances in $\boldsymbol{I}_{o1}$ were classified into the correct class with a high confidence while instances in $\boldsymbol{I}_{o2}$ were classified into the class with a low confidence. To shorten the distance between $\boldsymbol{I}_{o1}$ and $\boldsymbol{I}_{o2}$ (strategy 1), E_7 examined the important features of $\boldsymbol{I}_{o1}$ and $\boldsymbol{I}_{o2}$. As shown in Fig. 10(b), both feature *F64* and feature *F9* have a similar distribution on $\boldsymbol{I}_{o1}$ and $\boldsymbol{I}_{o2}$, especially for the part with larger feature values. Thus, he suggested that we try

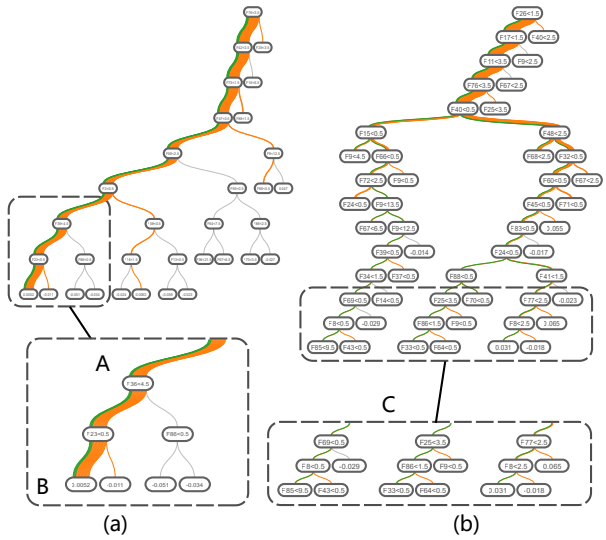


Fig. 9. Comparing tree structures constructed by (a) limiting the tree depth, and (b) limiting the number of leaf nodes. The tree in (b) grows much deeper and includes nodes with good differentiating ability, e.g. C.

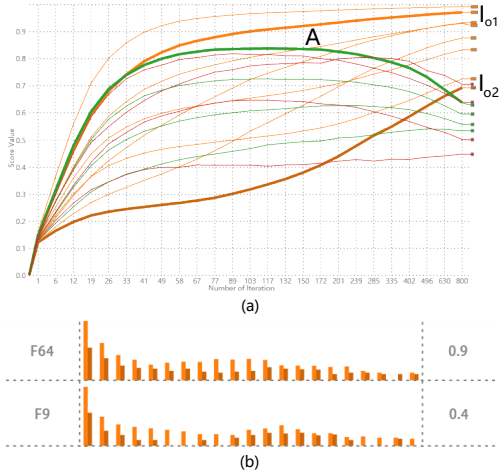


Fig. 10. Finding interaction features: (a) identifying high-confidence TPs ($\boldsymbol{I}_{o1}$) and low-confidence TPs ($\boldsymbol{I}_{o2}$); (b) comparing distributions of important features of $\boldsymbol{I}_{o1}$ and $\boldsymbol{I}_{o2}$. Both the features have similar distributions on $\boldsymbol{I}_{o1}$ and $\boldsymbol{I}_{o2}$, suggesting a combination of them to shorten the distance between the two orange lines in (a).

combining the two features to generate a new interaction feature. After trying four combinations (i.e., *F64 + F9*, *F64 - F9*, *F64 ÷ F9* and *F64 × F9*), we found *F64 + F9* worked the best (LogLoss was decreased to 0.44430), and thus added this feature to improve the model.

After trying the aforementioned three strategies, expert E_7 finally added four interaction features, and the LogLoss was decreased from 0.44568 to 0.44379 after this step. Although this improvement was not very significant, the experts considered it reasonably good, considering the meaning of all the features were obfuscated. Expert E_1 commented, "It is hard to add features when the meaning of the features was unknown. What we usually do is analyze the meaning of the features and add one that can increase the diversity of the feature set."

Recognizing the need for subsampling by observing an evolution pattern of decision trees (R3). To further improve the model, expert E_1 closely examined the evolution pattern of the tree structures. After analyzing the tree structures generated at the first three iterations, he realized that the features used on the top of the tree remained too stable. Fig. 11(a) shows the top three levels of the three trees. The features that did not appear at the top three levels of the previous tree were highlighted. Only one new feature appeared in the second tree and two new features appeared in the third tree. Since too much stability

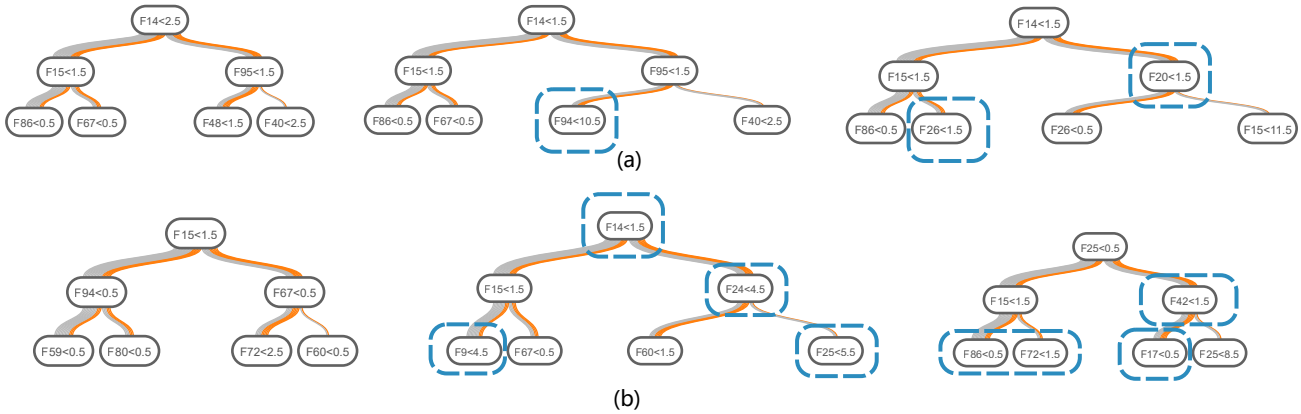


Fig. 11. Tree structures (top three levels) generated at the first three iterations before (a) and after (b) applying subsampling. New features added to the top three levels at each iteration are highlighted. Far more new features are introduced by applying subsampling.

could result in poor generalization ability, to solve this problem, E_1 suggested us to use feature subsampling. By using feature subsampling, each decision tree is built with only a randomly chosen percentage of the features. The trees built after applying subsampling are shown in Fig. 11(b). As shown in the figure, the tree structures evolved more rapidly, with four new features appearing in the second and third tree. As a result, the generalization ability of the model is improved. After applying subsampling, LogLoss decreased from 0.44379 to 0.43777.

Detecting outliers by analyzing relationships between instances (R2). When observing the relationships between the instances, expert E_7 detected some outliers. There were two types of outliers. The first type corresponds to instances whose neighbors belonged to different classes. Examples of such outliers found by E_7 was shown in Fig. 1D. It has been observed that this kind of instances may easily be misclassified [40] and may even be hard for humans to classify. The second type includes instances that moved between two clusters during the iteration process. This kind of outliers can be detected by observing the trails of the instances. Examples of such outliers is given in Fig. 1E. After exploring for a few minutes, E_7 detected nine outliers, six of them are of the first type and three of them are of the second type. After we removed these outliers and retrained the model, the model accuracy was not improved. Expert E_1 commented that this is reasonable since tree boosting is robust to outliers in the input space [18].

7 Discussion

The case studies demonstrate the usefulness and effectiveness of BOOSTVis in helping experts understand the inner workings of tree boosting and diagnose an unsatisfactory training process. Intrigued by BOOSTVis, the LightGBM team plans to integrate it with LightGBM. Nevertheless, there is certainly room for improvements.

General applicability. Although BOOSTVis is designed for tree boosting methods, it can be easily applied to other boosting methods that does not employ tree learning methods as weak learners. The only change needed is to design a new visualization for the new type of classifiers. In addition, it is not difficult to apply the BOOSTVis visualizations to other types of ensemble learning methods such as bagging. The major change that must be made is: in the class view, the time points along the time dimension are treated as classifier indexes in the bagging model.

Color scalability. In the class and instance visualizations, color-coding is utilized to encode different classes. The experts said that they could easily differentiate around ten classes with different colors. This is also consistent with the previous findings that only a small number of colors can be used effectively as category labels [55]. To compensate for this, we only display at most ten different classes in these two visualizations. A number of experiments have demonstrated that the estimated limitations of human comparison capacity usually vary from three to seven objects [46]. As a result, showing ten classes on the screen works for most real-world analysis tasks.

Learning curve. The BOOSTVis visualizations are based on familiar visual metaphors, such as the confusion matrix, line chart, t-SNE projection, tree visualization, and bar chart. As a result, the experts quickly became familiar with these visualizations and the adopted encodings. Because of the four coordinated views in the interactive exploration environment, several types of interactions are provided in BOOSTVis. We observed that the experts usually took from several minutes to a dozen minutes to become thoroughly familiar with the interactions between the views. The transition between a specific predicted class in the temporal confusion matrix and the temporal prediction scores of its instances (line chart) confused some of the experts. One expert said that he did not realize the connection between these two components until an unintentional clicking on one of the class stripes. The experts suggested providing a quick tour function, which would illustrate how to use this tool step by step. This will enable a wide adoption of this tool among machine learning experts and practitioners.

8 Conclusion

In this paper, we have presented a visual analytics tool, BOOSTVis, to help machine learning experts better understand the boosting process, diagnose underlying problems, and make informed improvements. A multi-view visualization, supported by time series segmentation, t-SNE, tree matching, etc. allows experts to explore a model's performance from different aspects and track the training process. Two case studies were conducted to demonstrate the usefulness of BOOSTVis for promoting comprehensive understanding and facilitating experts' diagnosis and refinement of a tree boosting model.

There are several directions to follow in our future work to improve BOOSTVis. First, we plan to conduct a formal user study and further improve our system based on the collected feedback. Second, we are interested in extending its application to a wider range of methods. Currently, BOOSTVis is used for analysis of tree boosting models. An immediate extension is the application to tree models trained using other boosting methods, such as Adaboost. Looking further ahead, as discussed in section 7, we aim to generalize BOOSTVis to analyze other ensemble learning methods such as bagging. Third, we would like to investigate how to support online analysis of the training process. In the current pipeline, the model is pre-trained, and BOOSTVis reproduces the training process from the log data. Experts expressed the desire to monitor the real-time running results and stop the training process if necessary. This requires the development of a set of visualization and data mining algorithms that can effectively convey streaming data, and automatically detect anomalies. Interactivity can be further improved allowing experts to modify parameters and change settings for additional tests.

Acknowledgments

S. Liu, J. Xiao, J. Liu, X. Wang are supported by National NSF of China (No. 61672308). J. Zhu is supported by the National NSF of China (61620106010 and 61621136008) and the Tsinghua Tiangong Intelligent Technology Institute. The authors would like to thank Dr. Xin Tong, Guolin Ke, Tianqi Chen, Changjian Chen, Xi Ye, and all the experts in our workshops for insightful discussions.

REFERENCES

[1] D. T. A. Kapoor, B. Lee and E. Horvitz. Interactive optimization for steering machine classification. In *CHI*, pages 1343–1352, 2010.

[2] B. Alsallakh, A. Hanbury, H. Hauser, S. Miksch, and A. Rauber. Visual methods for analyzing probabilistic classification data. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1703–1712, 2014.

[3] S. Amershi, M. Chickering, S. M. Drucker, B. Lee, P. Simard, and J. Suh. ModelTracker: Redesigning performance analysis tools for machine learning. In *CHI*, pages 337–346, 2015.

[4] M. Ankerst, C. Elsen, M. Ester, and H.-P. Kriegel. Visual classification: An interactive approach to decision tree construction. In *KDD*, pages 392–396, 1999.

[5] B. Bach, E. Pietriga, and J.-D. Fekete. Visualizing dynamic networks with matrix cubes. In *CHI*, pages 877–886, 2014.

[6] B. Becker, R. Kohavi, and D. Sommerfield. Visualizing the simple baysian classifier. In U. Fayyad, G. G. Grinstein, and A. Wierse, editors, *Information Visualization in Data Mining and Knowledge Discovery*, pages 237–249. Morgan Kaufmann, 2002.

[7] P. Bille. A survey on tree edit distance and related problems. *Theoretical computer science*, 337(1):217–239, 2005.

[8] D. Caragea, D. Cook, H. Wickham, and V. Honavar. Visual methods for examining svm classifiers. In S. J. Simoff, M. H. Böhlen, and A. Mazeika, editors, *Visual Data Mining*, pages 136–153. Springer, 2008.

[9] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *KDD*, pages 785–794, 2016.

[10] J. Choo, H. Lee, J. Kihm, and H. Park. iVisClassifier: An interactive visual analytics system for classification based on supervised dimension reduction. In *VAST*, pages 27–34, 2010.

[11] D. Cossok and T. Zhang. Statistical analysis of Bayes optimal subset ranking. *IEEE Transactions on Information Theory*, 54(11):5140–5154, 2008.

[12] A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. In *CVPR*, pages 4829–4837, 2016.

[13] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[14] C. Ferri, J. Hernández-Orallo, and R. Modroiu. An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, 30(1):27–38, 2009.

[15] E. Frank and M. Hall. Visualizing class probability estimators. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 168–179, 2003.

[16] D. Furcy and S. Koenig. Limited discrepancy beam search. In *IJCAI*, pages 125–131, 2005.

[17] Google. Tensorboard: Visualizing Learning. `https://www.tensorflow.org`, 2017. Accessed: 31-Mar-2017.

[18] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 3 edition, February 2009.

[19] F. Heimerl, S. Koch, H. Bosch, and T. Ertl. Visual classifier training for text document retrieval. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2839–2848, December 2012.

[20] J. Himberg, K. Korpiaho, H. Mannila, J. Tikanmaki, and H. T. Toivonen. Time series segmentation for context recognition in mobile devices. In *ICDM*, pages 203–210, 2001.

[21] B. Höferlin, R. Netzel, M. Höferlin, D. Weiskopf, and G. Heidemann. Inter-active learning of ad-hoc classifiers for video visual analytics. In *VAST*, pages 23–32, 2012.

[22] A. Jakulin, M. Možina, J. Demšar, I. Bratko, and B. Zupan. Nomograms for visualizing support vector machines. In *KDD*, pages 108–117, 2005.

[23] D. H. Jeong, C. Ziemkiewicz, B. Fisher, W. Ribarsky, and R. Chang. iPCA: An interactive system for PCA-based visual analytics. *Computer Graphics Forum*, 28(3):767–774, 2009.

[24] D. C. M. Jr., E. A. de Oliveira, U. M. Braga-Neto, R. F. Hashimoto, and R. M. C. Jr. Signal propagation in Bayesian networks and its relationship with intrinsically multivariate predictive variables. *Information Sciences*, 225:18–34, 2013.

[25] L. Kaufman and P. Rousseeuw. *Clustering by means of medoids*. North-Holland, 1987.

[26] J. Krause, A. Perer, and K. Ng. Interacting with predictions: Visual inspection of black-box machine learning models. In *CHI*, pages 5686–5697, 2016.

[27] T.-Y. Lin and S. Maji. Visualizing and understanding deep texture representations. In *CVPR*, pages 2791–2799, 2016.

[28] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):91–100, January 2017.

[29] S. Liu, W. Cui, Y. Wu, and M. Liu. A survey on information visualization: recent advances and challenges. *The Visual Computer*, 30(12):1373–1393, 2014.

[30] S. Liu, X. Wang, M. Liu, and J. Zhu. Towards better analysis of machine learning models: A visual analytics perspective. *Visual Informatics*, 1(1):48–56, 2017.

[31] S. Liu, J. Yin, X. Wang, W. Cui, K. Cao, and J. Pei. Online visual analytics of text streams. *IEEE Transactions on Visualization and Computer Graphics*, 22(11):1–15, 2016.

[32] L. v. d. Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[33] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, pages 5188–5196, 2015.

[34] P. McLachlan, T. Munzner, E. Koutsofios, and S. C. North. LiveRAC: Interactive visual exploration of system management time-series data. In *CHI*, pages 1483–1492, 2008.

[35] M. Možina, J. Demšar, M. Kattan, and B. Zupan. Nomograms for visualization of naive Bayesian classifier. In *European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 337–348, 2004.

[36] A. Ng. Machine Learning and AI via brain simulations. `http://ai.stanford.edu/~ang/slides/DeepLearning-Mar2013.pptx`, 2013. Accessed: 31-Mar-2017.

[37] J. G. S. Paiva, W. R. Schwartz, H. Pedrini, and R. Minghim. An approach to supporting incremental visual data classification. *IEEE Transactions on Visualization and Computer Graphics*, 21(1):4–17, 2015.

[38] K. Patel, N. Bancroft, S. Drucker, J. Fogarty, A. J. Ko, and J. A. Landay. Gestalt: Integrated support for implementation and analysis in machine learning. In *UIST*, pages 37–46, 2010.

[39] K. Patel, J. Fogarty, J. A. Landay, and B. Harrisonl. Examining difficulties software developers encounter in the adoption of statistical machine learning. In *AAAI*, pages 1563–1566, 2008.

[40] P. E. Rauber, S. G. Fadel, A. X. Falcao, and A. C. Telea. Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):101–110, January 2017.

[41] D. Ren, S. Amershi, B. Lee, J. Suh, and J. D. Williams. Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):61–70, 2017.

[42] P. Rheingans and M. DesJardins. Visualizing high-dimensional predictive model quality. In *Visualization*, pages 493–496, 2000.

[43] V. Sandulescu and M. Chiru. Predicting the future relevance of research institutions - the winning solution of the KDD Cup 2016. *arXiv e-prints:1609.02728*, 2016.

[44] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[45] C. Seifert and E. Lex. A novel visualization approach for data-mining-related classification. In *International Conference Information Visualisation*, pages 490–495, 2009.

[46] T. A. Sørensen and S. Kyllingsbæk. Short-term storage capacity for visual objects depends on expertise. *Acta psychologica*, 140(2):158–163, 2012.

[47] S. V. Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment*, 62(1):77–89, 1997.

[48] G. Stiglic, M. Mertik, V. Podgorelec, and P. Kokol. Using visual interpretation of small ensembles in microarray analysis. In *IEEE International Symposium on Computer-Based Medical Systems*, pages 691–695, 2006.

[49] J. Talbot, B. Lee, A. Kapoor, and D. S. Tan. Ensemblematrix: Interactive visualization to support machine learning with multiple classifiers. In *CHI*, pages 1283–1292, 2009.

[50] S. T. Teoh and K. l. Ma. Paintingclass: interactive construction, visualization and exploration of decision trees. In *ACM SIGKDD*, pages 667–672, 2003.

[51] F.-Y. Tzeng, E. B. Lum, and K.-L. Ma. An intelligent system approach to higher-dimensional classification of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):273–284, 2005.

[52] F.-Y. Tzeng and K. L. Ma. Opening the black box - data driven visualization of neural networks. In *IEEE Visualization*, pages 383–390, 2005.

[53] S. Urbanek. Exploring statistical forests. In *Proceedings of the 2002 Joint Statistical Meeting*, 2002.

[54] S. van den Elzen and J. J. van Wijk. BaobabView: Interactive construction and analysis of decision trees. In *VAST*, pages 151–160, 2011.

[55] C. Ware. *Information visualization: perception for design*. Elsevier, 2012.

[56] H.-f. Yu, H.-y. Lo, H.-p. Hsieh, J.-k. Lou, T. G. Mckenzie, J.-w. Chou, P.-h. Chung, C.-h. Ho, C.-f. Chang, J.-y. Weng, E.-s. Yan, C.-w. Chang, T.-t. Kuo, P. T. Chang, C. Po, C.-y. Wang, Y.-h. Huang, Y.-x. Ruan, Y.-s. Lin, S.-d. Lin, H.-t. Lin, and C.-j. Lin. Feature engineering and classifier ensemble for KDD cup 2010. In *KDD Cup*, 2010.

[57] T. Zahavy, N. B. Zrihem, and S. Mannor. Graying the black box: Understanding DQNs. In *ICML*, pages 1899–1908, 2016.

[58] T. Zhou, T. Chen, and L. He. Transparent boosting tree: An interactive machine learning framework. Technical report, University of Washington, 2014. Final Project of CSE 512 Data Visualization.

[59] Z.-H. Zhou. *Ensemble Methods: Foundations and Algorithms*. CRC press, 2012.

[60] IBM SPSS Software. `http://www.ibm.com/analytics/us/en/technology/spss/`. Accessed: 31-Mar-2016.

[61] IBM Watson. `https://www.ibm.com/watson/`. Accessed: 31-Mar-2016.

[62] No free hunch–the official blog of kaggle.com. `http://blog.kaggle.com/category/winners-interviews/`. Accessed: 31-Mar-2017.

[63] Titanic - guided by a confusion matrix. `https://www.kaggle.com/vinceallenvince/titanic/titanic-guided-by-a-confusion-matrix`. Accessed: 31-Mar-2017.

[64] Confusion matrix with probabilities. `https://www.kaggle.com/mark42/otto-group-product-classification-challenge/confusion-matrix-with-probabilities/code`. Accessed: 31-Mar-2017.

[65] Microsoft LightGBM. `https://github.com/Microsoft/LightGBM`. Accessed: 31-Mar-2017.

[66] Microsoft Azure Machine Learning Studio. `https://studio.azureml.net/`. Accessed: 31-Mar-2017.

[67] Otto Group Product Classification Challenge. `https://www.kaggle.com/c/otto-group-product-classification-challenge`, 2015. Accessed: 31-Mar-2017.

[68] Yandex corporate blog entry. `https://yandex.ru/blog/webmaster/5707`. Accessed: 31-Mar-2017.