Performance Analysis of Decision Tree, Random Forest, XG Boost Tree and SVM classifier

Subhadeep Dash and V. Sai Rathan

***Abstract*—This article discusses in detail about three popular tree based classifying algorithms namely Decision Trees, Random Forests and XG Boost Trees. Support Vector Machines have also been considered for comparison with the tree based algorithms. The main aim of this article is to showcase the functionality of each and depict which classifier is comparatively better in terms of performance.**

I. INTRODUCTION

Classification is one of the common problems in data mining. It refers to classifying the data into various classes i.e. each record in a data-set is associated with a well-defined class. Grouping data into various classes makes the work easier for data scientists who prefer to research in a specific field they are interested in. Several classifying algorithms have been introduced from the time data enthusiasts have realized the importance of data and the wonders they can perform with the stored information.

Tree based algorithms have been quite popular in classification [1]. Decision Tree based algorithm is more generalized one in which the building of association rules is quite easier and transparent. Decision making is quite evident in decision trees unlike many other classifying methods such as neural networks, clustering techniques [2] where a large amount of computation is done and is quite complex to understand. It allows partitioning of data in a much deeper level which is not easily achieved by some decision making classifiers such as logistic regression and support vector machines.

Random Forests [3] were introduced in 2001. They consist of many Decision Trees and are one of the most accurate classifiers available [4]. A Random Forest is typically best-suited for categorical data. For each data sample, it is assigned the label of the terminal node in which it ends up. This procedure is repeated for each decision tree in the forest and finally, the average vote over all the trees is considered as the label for the data sample provided. Random Forests are well known for their over-fitting nature on the training data.

XG Boost [5] is a relatively newer algorithm than the above mentioned classifying techniques. XG Boost was introduced in 2014 and it stands for Xtreme Gradient Boosting. This tree implements Gradient Boosting algorithm, which was developed specifically for its high predictive capability. The Gradient Boosting Trees build one tree at a time unlike the Random Forests where each tree is built independently. Every time it builds a new tree, it tries to rectify the errors that were encountered in the previously built tree. It generally takes longer time than the random forests because the trees have to be built in a sequential manner. However, results show that gradient boosting has proved to have obtained better results than the Random Forest.

Support Vector Machines are another type of classifiers which don't classify data by construction of a tree and hence this is the reason why it has been considered in this paper to compare other mentioned tree-based algorithms. They are considered extremely good when no prior information on the data has been provided.

II. DECISION TREES

A. Overview

Decision Trees are considered as the foundation for many of the tree based classifying algorithms. They are implemented in almost every advanced tree based classifying algorithms. In Decision Trees, it is very easy to visualize the rules based on which a data sample is classified. It has been observed that Decision Trees are quite accurate in predicting the class of a data sample.

It is quite easy to construct a Decision Tree as it doesn't require normalization of the training data, creation of dummy variables or removal of blank values. The cost of using the tree is approximately equal to the logarithm of the total number of training data samples. It can handle both categorical and numerical data with ease.

B. Construction of a Decision Tree

The following mentioned steps are to be followed to build a Decision Tree:

- The attribute which seems to be the best to categorize the training samples is to be placed at the root of the decision tree.
- After setting up a node in the tree, we classify the training set into different subsets based on the values of the attribute, hence leading to branching of the tree. In each of these subsets, all elements should have the same attribute value as others i.e. the training sample data is categorized into various subsets based on the attribute chosen while the tree is being constructed level by level. As a result, as the depth of the tree increases, we obtain subsets of the training data having similar values for each attribute.
- The above mentioned steps are performed recursively until the leaf nodes of the tree are obtained in all branches of the tree.

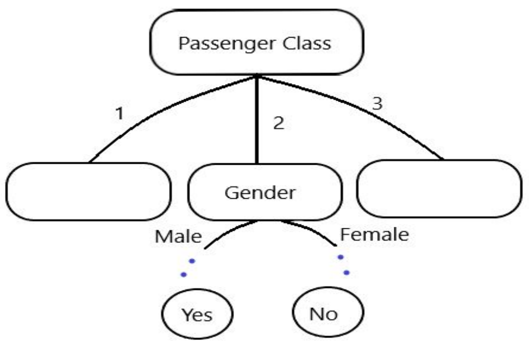

Fig. 1. Decision Tree

The above figure is a sample representation of the Decision Tree Construction for the Titanic data-set on which we have worked on. The training data samples with class 1 proceed towards the leftmost branch. The ones with class 3 traverse towards the rightmost branch and the data samples of those of class 2 move towards the middle branch. Each of the samples in the dataset end up with a leaf node Yes or No survival leading to classification of the people present on the ship as dead or alive.

Some of the points are to be noted before the construction of the Decision Tree:

- At the initialization of the Decision Tree, whole training set is assumed to be present at the root level after which they are divided into separate subsets as the branching of the tree occurs.
- The data samples are distributed recursively based on their respective attribute values until they reach a leaf node beyond which no branching occurs.
- The order of setting attributes to nodes is calculated based on some statistical approach on the training data.
- Decision Trees are best suited for categorical data but if the data is continuous, they are converted to discrete based on some preprocessing technique prior to construction of the tree.

Selection of Attributes: If a dataset consists of k number of attributes, it is necessary to decide in which order will be the attributes placed level by level for classification. There are two types of criteria for solving this issue. They are namely *Information Gain* and *Gini Index*.

1) Information Gain: Entropy is used to calculate the randomness or uncertainty of a random variable X.

$$Entropy(X) = -\sum_{x \in X} p(x) logp(x) \tag{1}$$

Information Gain(IG) of a random variable X is obtained by subtracting the entropy of the attribute value from the entropy of the target attribute(Target).

$$IG(X) = Entropy(Target) - Entropy(X) \tag{2}$$

The attributes are sorted in decreasing order of their Information Gain values and the attribute with the highest Information Gain value is taken as the root. From then, level by level an attribute is chosen for classification based on this order obtained.

2) Gini Index: Gini Index is a measure of how often a randomly chosen data sample can be incorrectly classified. An attribute with lower Gini Index should be preferred. Gini Index of a random variable X can be defined as follows:

$$GiniIndex = 1 - \sum_j p_j^2 \tag{3}$$

For example let us consider the Gini Index of a attribute Gender in the Titanic Dataset where the Survival attribute is the output attribute i.e. the classifying attribute. Assume there are 20 data samples as shown below. We need to find out the Gini Index of Gender attribute.

PassengerId	Survived	Sex
1	0	male
2	1	female
3	1	female
4	1	female
5	0	male
6	0	male
7	0	male
8	0	male
9	1	female
10	1	female
11	1	female
12	1	female
13	0	male
14	0	male
15	0	female
16	1	female
17	0	male
18	1	male
19	0	female
20	1	female

TABLE I
TITANIC DATASET

In the above table, we observe that there are 11 females and 9 males. For males and alive, probability is $1/9$ and for death it is $8/9$. For females and alive probability is $9/11$ and for death it is $2/11$.

$$Gini(1, 8) = 1 - ((1/9)^2 + (8/9)^2) = 0.1975308642 \tag{4}$$

$$Gini(9, 2) = 1 - ((9/11)^2 + (2/11)^2) = 0.2975206612 \tag{5}$$

Gini Index with respect to the target is obtained by

$$\begin{aligned} &Gini(target, gender) \\ &\quad = (9/20) * 0.1975308642 \\ &\quad + (11/20) * 0.2975206612 \\ &\quad = 0.2525252526 \end{aligned} \tag{6}$$

The attribute having least Gini(target, attribute) value is given highest precedence in the order i.e. it is placed in the root and all the other attributes too are sorted in increasing order and placed accordingly. CART(Classification And Regression Trees) and ID3(Iterative Dichrotomiser 3) algorithms are used to build Decision Trees. CART uses Gini Index as metric where as ID3 uses Information Gain and Entropy function as metrics for building these structures.

Overfitting is quite common in Decision Trees. So techniques like Pre-pruning and Post-pruning are employed to curb overfitting. In Pre-pruning, it stops the tree construction early by deciding not to split a node if the goodness measure is below some threshold value. In Post-pruning, a complete tree is built first. Later if overfitting arises, cross validation is performed and the node is only allowed to expand if its expansion doesn't cause overfitting else it is pruned off i.e. the current node should be converted into a leaf node.

C. Advantages and Disadvantages

1) Advantages:

- The logic of even a complex Decision Tree can be visualized in such a way that any person could easily understand it since these follow the same approach human beings usually follow to make decisions.
- They don't require normalization of data, removal of blank values or even creation of dummy variables.
- The cost of building a decision tree is quite less.

2) Disadvantages:

- There is a high probability of over-fitting in Decision Trees which can be corrected by Pre-Pruning or Post-Pruning.

D. Scikit-learn implementation

Scikit-learn module is a free software machine learning library for Python which includes in-built classification, regression and clustering algorithms. This has an in-built *DecisionTreeClassifier()* class. Using this class in Python, we can test the accuracy by varying the parameters of the Decision Tree classifier.

This class enables the user to pass various parameters. The prototype of the class is mentioned below:

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')

Using this class, we can limit the maximum number of leaf nodes, maximum depth, minimum samples to be split, maximum number of features to be chosen and the criterion to choose the attributes too. This helps us in controlling overfitting to a large extent. This library is very much useful when it comes to tweaking values of parameters of already built classifiers and obtaining better accuracy.

III. Random Forests

A. Overview

As the name suggests, Random Forests are Forests consisting of a large number of Decision Trees. The more the number of Trees, the higher the accuracy. All the Trees constructed are independent of each other are simultaneously constructed. For each data sample, the Random Forest takes into account the label of each Decision Tree in it and considers the mode of the outputs obtained as the label for the given data sample.

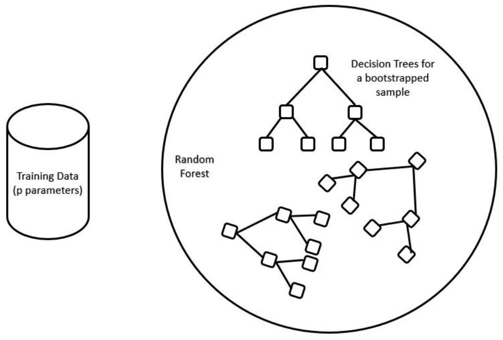

Fig. 2. Random Forest

B. Construction of a Random Forest

The following procedure is to be followed to build a Random Forest:

- A Bootstrap sample of size N is drawn from the training data-set. A Bootstrap sample is a smaller sample which is bootstrapped from a larger sample of data.
- Randomly select 'm' number of features from a total of 'p' features in the data-set (where $m << p$) and build a Decision Tree for these m no. of features. The number m remains constant throughout the construction of the forest.
- Split the node into daughter nodes by finding the best attribute as discussed earlier in case of a Decision Tree.
- Recursively repeat the above steps until a threshold value is achieved for the size of a node.
- Therefore, a Decision Tree in the Forest is successfully created. Similarly, B no. of Decision Trees can be constructed in this manner for the Random Forest.

Algorithm 1 Construction of a Random Forest:

```
1: procedure RandomForestClassifier
2:     for $i \leftarrow 1$ to $B$ do
3:         Draw a bootstrapped sample of size N
4:         TreeConstruct:
5:         Select m variables randomly from p variables.
6:         Pick the best variable among m and split.
7:         if $node_{size} > node_{min}$ then
8:             goto TreeConstruct
9:         $T_i \leftarrow$ ConstructedTree
10:    return $\{T_i\}_1^B$
```

Estimation of Error: Out-of-bag error, also known as the out-of-bag estimate, is used to estimate test error in Random Forests. It is similar to leave-one-out cross-validation but almost without any computational burden. It is the mean prediction error on each training sample x_i, using only the trees which didn't have x_i in their bootstrap sample.

C. Advantages and Disadvantages

1) Advantages:

- Random Trees can be used for both Classification and Regression.

- Since Random Trees contain many number of Decision Trees, they are lesser prone to over-fitting.
- No need of feature normalization, removal of blank values or creation of dummy variables.

2) Disadvantages:

- Over-fitting is still considered a problem, though they are better than naive Decision Trees in this scenario.
- They cannot be easily interpreted or visualized like Decision Trees.
- Computational Cost is higher than Decision Trees as many Trees need to be constructed and hence, it is relatively slower to build.
- The hyperparameters namely the number of Decision Trees in a Random Forest and the no. of variables considered should be selected prior to construction of the Forest.

D. Scikit-learn implementation

RandomForestClassifier() class is present in scikit-learn library of Python which can be used to experiment on any data-set and classify them. The prototype for it is given below:

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)

IV. XG Boost Trees

A. Overview

More than half of the winning solutions in Machine Learning challenges hosted at Kaggle adopt XGBoost. In terms of training, Gradient Boost tries to add new trees that compliments the already built ones. This normally helps to attain better accuracy with less number of trees. Unlike Random Forests, this algorithm builds trees in the forest sequentially. It learns from the errors caused by the previously constructed tree and tries to rectify it using a particular learning rate in the next tree.

B. Elements in a XGBoost Tree

1) Models and Parameters: In a linear model, the prediction y_i from input x_i can be denoted as

$$\hat{y}_i = \sum_j \theta_j x_{ij} \tag{7}$$

where θ denotes the undefined parameters which needs to be learnt from the data.

2) Objective Function: The objective function consists of two parts: training loss and regularization term. It can written as follows:

$$Obj(\theta) = L(\theta) + \Omega(\theta) \tag{8}$$

where $L(\theta)$ is the training loss and $\Omega(\theta)$ is the regularization term.
Generally $L(\theta)$ is the mean squared error given by

$$L(\theta) = \sum_i (y_i - \hat{y}_i)^2 \tag{9}$$

Another commonly used loss function is the logistic loss used for logistic regression.

$$L(\theta) = \sum_i [y_i \ln(1 + e^{-\hat{y_i}}) + (1 - y_i)\ln(1 + e^{\hat{y_i}})] \tag{10}$$

The regularization term controls the complexity of the model resulting in avoidance of over-fitting.

3) Models and Parameters in a Decision Tree(CART): We classify the sample data records in various leaf nodes and assign a score to each. The prediction scores of each individual tree are summed up to get the final score. [6]

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F \tag{11}$$

where K is the number of trees, f is a function in the functional space F and F is the set of all CARTs. The objective function is optimized as follows:

$$obj = \sum_i^n l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{12}$$

4) Tree Boosting: Considering tree boosting at step t in case of gradient boosting, we can take the objective function as:

$$obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t)}) + \sum_{i=1}^{t} \Omega(f_k) \tag{13}$$

where

$$\hat{y_i}^{(t)} = \hat{y_i}^{(t-1)} + f_t(x_i) \tag{14}$$

$$l(y_i, \hat{y_i}^{(t)}) = (y_i - \hat{y_i}^{(t)})^2 \tag{15}$$

$$\sum_{i=1}^{t} \Omega(f_k) = \Omega(f_t) + constant \tag{16}$$

Further the objective function can be simplified as follows:

$$\begin{aligned}
obj^{(t)} &= \sum_{i=1}^{n} (y_i - \hat{y_i}^{(t)})^2 + \Omega(f_t) + constant \\
&= \sum_{i=1}^{n} (y_i - (\hat{y_i}^{(t-1)} + f_t(x_i)))^2 + \Omega(f_t) + constant \\
&= \sum_{i=1}^{n} ((y_i - \hat{y_i}^{(t-1)})^2 + 2(\hat{y_i}^{(t-1)} - y_i)f_t(x_i) + f_t(x_i)^2) \\
&\quad + \Omega(f_t) + constant \\
&= \sum_{i=1}^{n} (l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2) \\
&\quad + \Omega(f_t) + constant
\end{aligned}$$

$$\text{where } g_i = \delta_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y_i}^{(t-1)}),$$

$$h_i = \delta^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \tag{17}$$

Finally the objective function can be finalized as:

$$obj^{(t)} \approx \sum_{i=1}^{n}(l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2}h_i f_t(x_i)^2) + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \quad [\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2] \tag{18}$$

From the above mentioned equations, the best w_j for a given structure $q(x)$ and the best objective reduction we can get is $w_j^* = -\frac{G_j}{H_j+\lambda}$ and $obj^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$ where $G_j = \sum_{i\in I_j} g_i$ and $H_j = \sum_{i\in I_j} h_i$.

C. Advantages and Disadvantages

1) Advantages:

- XG Boost builds one tree at a time, which rectifies the errors made by the previously built tree.
- Boosting focuses step by step on difficult samples which help handling unbalanced data-sets.
- Since these trees are built by optimizing an objective function, it can be used to solve almost all objective function that we can write gradient out like Poisson regression and ranking, which is harder to achieve using Random Forests

2) Disadvantages:

- It is harder to tune XG Boost when compared to Random Forests.
- More sensitive to over-fitting if the data is noisy.
- Training takes more time since trees are built sequentially.
- Logics or rules cannot be easily interpreted or visualized.

D. Implementation

xgboost library is available for Python using which *XGBClassifier()* can be implemented.

```
XGBClassifier(base_score=0.5, booster='gbtree',
colsample_bylevel=1, colsample_bytree=1, gamma=0,
learning_rate=0.1, max_delta_step=0, max_depth=3,
min_child_weight=1, missing=None, n_estimators=100,
n_jobs=1, nthread=None, objective='binary:logistic',
random_state=0, reg_alpha=0, reg_lambda=1,
scale_pos_weight=1, seed=None, silent=True, subsample=1)
```

V. Support Vector Machines

A. Overview

Support Vector Machines(SVMs) are quite popular and unlike the other three algorithms described earlier, it is not a tree based algorithm. The main strength of SVMs lies in their kernel.Generally there are two kinds of classifiers namely Linear and Non-linear SVM classifiers.

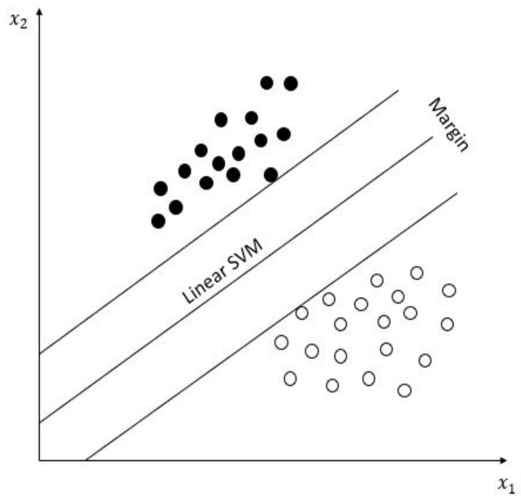

Fig. 3. Linear Support Vector Machine

B. Construction of a Support Vector Machine

1) Selecting Hyperplanes: For linearly separable data, two parallel hyperplanes are selected that can separate the two classes of data in such a way that distance between both hyperplanes is maximum. The maximum margin hyperplane is the one that lies in the middle of them. The equation for the maximum margin hyperplane can be written as

$$\vec{w}x_i - b = 0 \tag{19}$$

The equations for both the hyperplanes can be written as below:

$$\vec{w}x_i - b = 1 \tag{20}$$

$$\vec{w}x_i - b = -1 \tag{21}$$

where $||\ \vec{x}\ ||$ is normal vector to the hyperplane, x_i denotes features and b is constant. If for an element, $\vec{w}x_i - b \geq 1$, then it belongs to a particular class, else if $\vec{w}x_i - b \leq -1$, then it belongs to the other class. The maximum distance between two hyperplanes, i.e. the maximum width of the margin is $\frac{2}{||\vec{w}||}$.

2) Selecting Kernel: Non-linear hyperplanes can be drawn using kernel trick. Every kernel holds a non-linear kernel function. Some of the standard kernels are:

- Polynomial(homogeneous) Kernel : $K(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j})^d$
- Polynomial(non-homogeneous) Kernel : $K(x, y) = (x^T y + c)^d$
- Radial Basis Function(rbf) Kernel : $K(x, x') = \exp(-\frac{||x-x'||^2}{2\sigma^2})$ where x, y, x_i, x_j, x' are vectors in the feature space, c and σ are free parameters and d is the degree of polynomial function.

C. Advantages and Disadvantages

1) Advantages:

- Support Vector Machines are effective even the number of features is quite large i.e. it works even when the number of features is greater than the total number of training samples.

- They are defined by convex optimization function(no local minima).
- They are proved to be better when there is no prior information about the data.
- The risk of over-fitting is less in Support Vector Machines.
- Any complex problem can be solved by choosing an appropriate kernel.
- It is a robust model for classification since it maximizes the margin.

2) Disadvantages:

- Choosing an appropriate kernel is the biggest limitation since wrong selection of kernel can lead to large errors leading to increase in error percentage.
- The final model is difficult to visualize or interpret.
- They have good generalization performance but the testing phase could be extremely slow due to its high algorithmic complexity.

D. Scikit-learn implementation

Support Vector Classifier is available in scikit-learn for implementation and it's class constructor is as shown below:

SVC(C=1.0, kernel=rbf, degree=3, gamma=auto_deprecated, coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape=ovr, random_state=None)

VI. RESULTS ON REAL DATA

After analysing all the four classifiers, they have been tested in Python environment for the Titanic data-set to determine whether a person is alive or dead based on the parameters like passenger class, age, gender, etc. The following are the accuracies obtained using each classifier:

Classifiers	Training Accuracy(%)	Testing Accuracy(%)
Decision Tree	84.43	79.10
Random Forest	96.63	84.70
XGBoost Trees	88.44	85.07
SVM Classifier	78.97	77.99

TABLE II
PERFORMANCE COMPARISON OF CLASSIFIERS

It is observed that Random Forest over-fits with the training data and obtains an accuracy of 84.70% while testing which is quite better than the Decision Tree and SVM Classifier which obtain 79.10% and 77.99% accuracy respectively. We observe that the over-fitting is less in case of XGBoost Trees when compared to Random Forest. XGBoost Trees obtain an accuracy of 88.44% on training set where as for Random Forest it reaches 96.63%. XGBoost Trees give the maximum accuracy while testing when compared to any other tree based algorithm. It is observed that SVM classifiers are way less accurate than tree based algorithms while both training and testing. Even the Decision Tree surpasses SVM in terms of accuracy. Therefore, tree based classifying algorithms have an edge over SVM in terms of accuracy whereas SVM is known for its robustness. Hence, the performance analysis for all the four classifying algorithms have been conducted and statistics have been noted.

REFERENCES

1. B. Gupta, A. Rawat, A. Jain, A. Arora, and N. Dhami, "Analysis of various decision tree algorithms for classification in data mining," 2017.
2. B. N. Patel, S. G. Prajapati, and K. I. Lakhtaria, "Efficient classification of data using decision tree," *Bonfring International Journal of Data Mining*, vol. 2, no. 1, pp. 06–12, 2012.
3. V. Y. Kulkarni and P. K. Sinha, "Random forest classifiers: a survey and future research directions," *Int J Adv Comput*, vol. 36, no. 1, pp. 1144–53, 2013.
4. E. Scornet, G. Biau, J.-P. Vert, *et al.*, "Consistency of random forests," *The Annals of Statistics*, vol. 43, no. 4, pp. 1716–1741, 2015.
5. T. Chen and C. Guestrin, "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.
6. T. Chen and C. Guestrin, "Xgboost: Reliable large-scale tree boosting system. arxiv 2016; 1–6," *DOI: http://dx. doi. org/10.1145/2939672.2939785*, 2016.