Information Fusion 27 (2016) 111–125

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

# Decision forest: Twenty years of research

Lior Rokach [*]

*Department of Information Systems Engineering, Ben-Gurion University of the Negev, P.O.B. 653, Beer-Sheva 84105, Israel*

ARTICLE INFO

*Article history:*
Received 5 May 2015
Received in revised form 8 June 2015
Accepted 15 June 2015
Available online 20 June 2015

*Keywords:*
Decision tree
Decision forest
Random forest
Classification tree

ABSTRACT

A decision tree is a predictive model that recursively partitions the covariate's space into subspaces such that each subspace constitutes a basis for a different prediction function. Decision trees can be used for various learning tasks including classification, regression and survival analysis. Due to their unique benefits, decision trees have become one of the most powerful and popular approaches in data science. Decision forest aims to improve the predictive performance of a single decision tree by training multiple trees and combining their predictions. This paper provides an introduction to the subject by explaining how a decision forest can be created and when it is most valuable. In addition, we are reviewing some popular methods for generating the forest, fusion the individual trees' outputs and thinning large decision forests.

## 1. Introduction

A decision tree is a predictive model expressed as a recursive partition of the covariates space to subspaces that constitute a basis for prediction. Decision trees have become one of the most powerful and popular approaches in data science—the science and technology of exploring large and complex bodies of data in order to discover useful patterns. Decision trees, originally developed in decision theory and statistics, were enhanced by researchers in other fields, such as: data mining, machine learning, and pattern recognition.

While a decision tree has many advantages, such as comprehensibility, it still suffers from several drawbacks—instability, for instance. One way to realize the full potential of decision trees is to build a decision forest. A decision forest, as its name implies, consists of several decision trees in which their predictions are combined into a final prediction. By building a forest, the mistakes of a single decision tree are compensated for by other decision trees in the forest. A recent study [1] that empirically compared 179 classification algorithms arising from 17 learning families over 121 datasets concluded that decision forests, and in particular random forests, tend to outperform other learning methods. The key to the improved predictive performance is the complementarity of the base decision trees included in the forest. Moreover, while the decision forest requires growing several decision trees, it is still considered very attractive in terms of computational cost because of the low computational cost of the base tree induction algorithm [2].

There are several excellent reviews on ensemble learning that were recently published in the literature [3,4]. However, these current reviews do not specifically focus on the decision forest, but rather discuss ensemble learning in general. Thus, the insights that were gained for decision forest and cannot be generalized to other ensemble learning methods are usually not sufficiently emphasized, to say the least. On the other hand, there are several surveys [5,6] which solely focus on random forest and its variants. While random forest is the most popular decision forest algorithm, there are other useful methods for building decision forests. A recent book edited by Criminisi and Shotton [7] presents a unified model of decision forests for addressing various learning tasks. This excellent and highly recommenced book covers theoretical foundations, practical implementation and application of decision forests. However, it does not discuss important issues such as scaling up decision forests methods for big data and decision forest thinning.

This paper aims to serve as an introductory reading to decision forest and to survey the state-of-the-art methods in the field. The rest of the paper is organized as follows: Section 2 briefly introduces decision trees that serve as the building blocks of decision forest. In particular, we show how decision trees are usually trained, what their advantages and disadvantages are, and how they can be used in various learning tasks other than classification. Section 3 introduces the concept of the decision forest, explains why decision forests work, and in which problem setting they are particularly useful. Section 4 presents various approaches for growing a decision forest. Section 5 explains how the outputs of the individual decision trees are fused. Section 6 reviews some of

* Tel.: +972 8 6479338; fax: +972 8 6477527.
*E-mail address:* liorrk@bgu.ac.il

http://dx.doi.org/10.1016/j.inffus.2015.06.005
1566-2535/© 2015 Elsevier B.V. All rights reserved.

the most popular methods for building decision forests. Section 7 discusses how the size of a decision forest can be reduced by discarding trees that do not contribute to the forest performance as a whole. Finally, Section 8 concludes the paper.

## 2. Individual decision trees – the building blocks of decision forest

A decision tree is a predictive model expressed as a recursive partition of the covariates space to subspaces that constitute a basis for prediction. The decision tree consists of nodes. Starting with the entire dataset, the root node corresponds to the first split which specifies how the data should be divided into disjoint partitions. The succeeding children nodes continue to split the data into smaller partitions until no further partitioning is required. The leaves represent the final partitions.

The "root" node has no incoming edges. The internal nodes (also known as "test" nodes) have exactly one incoming edge and two or more outgoing edges. Every internal node denotes a test to be checked on the instances. Each branch (outgoing edge) represents an outcome of the test. The "leaves" nodes (also known as "decision" nodes) have no outgoing edges and hold the predicted value or the prediction model.[1] A single node decision tree which contains a root node and leaves and no other internal nodes is sometimes called a *decision stump*.

Decision trees are commonly used for classification tasks. The aim of classification is to classify an object or an instance into a predefined set of classes based on their attributes' values (features). For example, in the well-known Iris flower dataset [8], the goal is to classify the flowers into three sub-species of Iris (Iris Setosa, Iris Versicolor and Iris Virginica) based on four classified measurements of the flower: the petals' width, the petals' length, the sepals' width and sepals' length—all in centimeters. When a decision tree is used for classification tasks, it is most commonly referred to as a classification tree. In this case the leaves' nodes hold either the predicted class or class distribution.

Fig. 1 illustrates a classification tree for the Iris dataset. The internal nodes contain the tested feature. The leaves contain the number of instances and the class distribution.[2] Similarly to the classification tree presented in Fig. 1, most decision trees are univariate in the sense that the branching test that determines which instances fall into which partition, uses only one feature at a time. Instances are classified by navigating them from the root of the tree down to a leaf according to the outcome of the tests along the path similar to a flowchart. First, we test the petal's length in the root node (node #1). If the petal's length is less than 1.9 cm, we branch to the left and reach a leaf node (node #2). As indicated in the figure, this leaf node consists of 50 instances ($n = 50$) and the class distribution is clearly biased toward the Iris Setosa class. If the petal's length is greater than 1.9 cm, we branch to the right and reach to another test node (node #3) which tests the petal's width. The tree continues to branch out until we reach a leaf. Note that the prediction is based only on which leaf a given instance falls into.

Decision trees split the covariate space into disjointed partitions. One way to better understand the decision tree is to visualize the covariate space along with the selected partition boundaries. The bottom part of Fig. 1 presents a two-dimension projection scatterplot using the features that were chosen to be tested in the tree. A univariate decision tree divides the space into axis-parallel hyperplans. The scatterplot illustrates the corresponding partition of the feature space by adding the partition boundaries. In this case, there are four hyperplans (boxes) in the plot. Each box corresponds to exactly one leaf in the classification tree. We can visually induce the class distribution of each box based on the instances that are located in it.

### 2.1. Growing a decision tree

The basic decision tree induction algorithm constructs decision trees in a top-down recursive divide-and-conquer manner. In each iteration, the algorithm searches for the best partition of the dataset. Recall that many decision trees are univariate i.e. the dataset is split according to the value of a single attribute. Thus, in such cases, the algorithm needs to find the best splitting attribute. The selection of the most appropriate attribute is made according to certain splitting criteria, such as information gain or the Gini coefficient. All possible attributes are evaluated according to the splitting criterion and the best attribute is selected. After the selection of an appropriate split, each node further subdivides the training set into smaller subsets and the process continues in a recursive manner. Note that for numeric attributes, it is common to create a binary partition which splits the attribute's value range into two parts. Because there are many possible cut points, the induction algorithm searches for the best cut point by evaluating the splitting criterion on each possible cut point.

The growing phase continues until a stopping criterion is triggered. Many stopping criteria can be used to control the growing process. For example the process can be stopped if all instances in the current partition belong to a single class or if the number of instances in the terminal node is less than a predefined minimum. In addition, statistically motivated stopping criteria can be implemented via hypothesis tests. The null hypothesis states that the current tree and the resultant tree obtained—following an additional split—perform equally. If the null hypothesis cannot be rejected than the growing process is stopped.

Employing tight stopping criteria tends to create small and underfitted decision trees. Conversely, using loose stopping criteria tends to generate large bushy decision trees. Complicated decision trees might have limited generalization capabilities. Although they seem to correctly classify all training instances, they fail to do so in new and unseen instances, mainly because they are overfitted to the training set. One way to mitigate this dilemma is to allow the decision tree to overfit the training set. Then the overfitted tree is pruned into a smaller tree by removing sub-branches that are not contributing to the generalization predictive performance. It has been shown in various studies that this pruning approach can improve the generalization predictive performance of a decision tree, especially in noisy domains.

Probably the most popular decision tree induction algorithms are C4.5 [9], and CART [10]. Both algorithms are top-down induction algorithms that use splitting criterion and pruning methods as described above. There are several open source implementations of the above two popular algorithms: For example, J4.8 is a Java implementation of the C4.5 algorithm in the Weka data mining tool [11] and rpart package [12] is an R package implementation of the CART algorithm. While these implementations are expected to perform similarly, a comparative study indicates that this is not always the case [13].

In this section, we focused on greedy top-down induction of decision trees due to their simplicity and popularity. The lookahead strategy aims to improve the greedy strategy by exploring the space of all possible decision trees up to a fixed depth. This more extensive search quickly leads to intolerable time consumption. Moreover, limited a lookahead search does not produce

[1] In classification trees the leaves usually hold either the predicted class (most frequent class) or the class distribution. In regression trees it is usually the mean value of the target variable. But more complex models exist such as survival trees that hold a survival model in each node.

[2] The visualized tree contains only the counts for the tree's leaves. However, it is easy to calculate the counts for the internal nodes by simply summing the values of each node's children recursively, from the leaves to the root node.