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/

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.

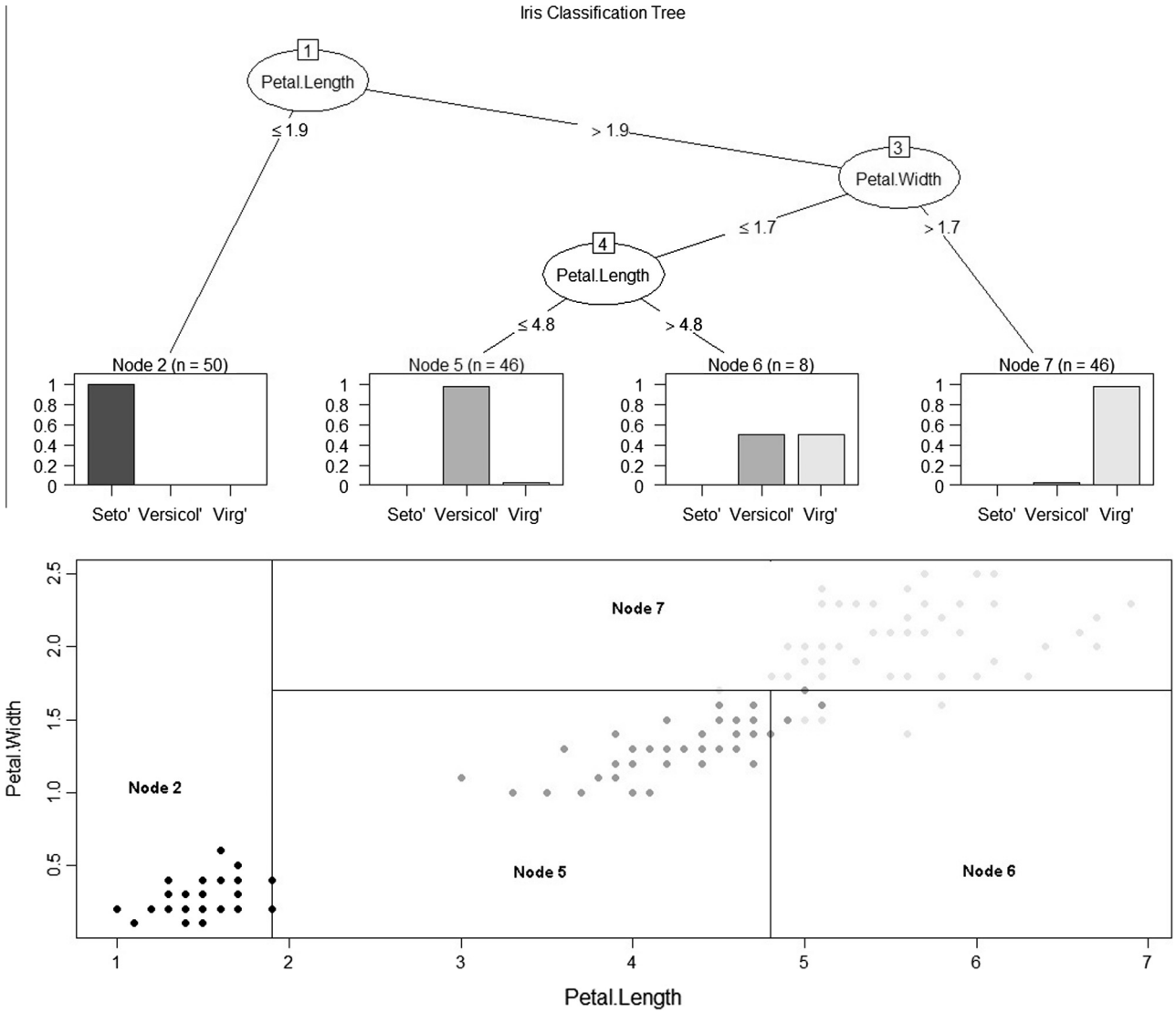

Fig. 1. A classification tree for Iris Dataset.

significantly better decision trees. Another approach to improve greedy decision-tree induction is to generate decision trees through genetic algorithms (GA) or genetic programming (GP). Barros et al. [14] describe approaches for automating the design of decision tree induction algorithms. In particular, they show how a grammar-based GP algorithm can be used for building and evolving individual trees.

2.2. Advantages of decision trees

Decision trees have become a popular method due to the many benefits they offer, including:

1. A self-explanatory and easy to follow nature when compacted. That is to say, if the decision tree has a reasonable number of leaves, it can be grasped by non-professional users. Furthermore, since decision trees can be easily converted to a set of if-then rules, this sort of representation is considered as comprehensible. In addition it provides an implicit feature selection, i.e. by looking at the tree we can easily determine which feature is important.
2. Decision trees are considered to be a non-parametric method, i.e. decisions trees do not hold any assumptions about the space distribution.
3. They are able to handle a variety of input data: nominal, numeric and textual.
4. Compared to other methods, they scale well to big data.
5. They are able to process datasets that may have errors or missing values.
6. They have a high predictive performance for a relatively small computational effort.
7. Decisions trees are available in many open source data mining packages over a variety of platforms.
8. They are useful for various tasks, such as: classification, regression, clustering and feature selection.
9. Decision trees can be combined with other prediction techniques, for example, certain decision trees feature neural networks models or logistic regression models in the terminal nodes.
10. When sensor cost is high, decision trees may be attractive in that they need only for the values of the sensor used along a single path from the root to a leaf. The sensor values can be acquired on demand as we traverse the tree.

11. While decision tree algorithms might have several controlling parameters, in many cases the default values yield sufficiently good predictive performance. If tuning is still required, this usually takes a short tuning session.

2.3. Disadvantages of decision trees

Decision trees also have several known drawbacks. As we will see later, some of these drawbacks can be mitigated by growing a decision forest. Here are the main drawbacks:

(1) Replication Problem: Decision trees use a "divide and conquer" strategy for training the model. Thus, they tend to perform well if a few highly relevant attributes exist, but less so if many complex interactions among the attributes are present. A simple illustration of this phenomenon is the replication problem of decision trees [15]. Since most decision trees divide the instance space into mutually exclusive regions to represent a concept, in certain cases the tree may contain several duplications of the same subtree in order to represent the classifier. For instance, if a tree tries to learn the following Boolean function: $y = (A_1 \; and \; A_2) \; or \; (A_3 \; and \; A_4)$, then the minimal univariate decision tree that represents this function have two copies of the sub-tree that corresponds to the first term $(A_1 \; and \; A_2)$ or two copies of the sub-tree that correspond to the second term $(A_3 \; and \; A_4)$.

(2) Myopic: The myopic nature of most of the decision tree induction algorithms is reflected by the fact that the learning algorithms consider only one level ahead. Specifically, the splitting criterion ranks possible attributes based on their immediate descendants. Such a strategy prefers tests that score high in isolation and may overlook combinations of attributes. This for example is well illustrated when the tree tries to learn a Boolean function such as $y = (A_1 \; xor \; A_2)$. In such case, knowing the values of both attributes determines the value of y. However, knowing the value of either attributes does not help at all in predicating the value of y. Thus, a myopic algorithm might assume that neither attributes are useful, and therefore disregard them. Using deeper lookahead strategies is considered to be computationally expensive and has not been proven useful in all cases [16,17].

(3) Instability: The greedy characteristic of decision trees leads to another disadvantage that should be pointed out. The over-sensitivity to the training set, to irrelevant attributes and to noise [9] make decision trees especially unstable: a minor change in one split close to the root will change the whole subtree below. Small variations in the training set may result in different splits and hence a different decision tree.

(4) Fragmentation problem: This usually happens if many attributes are tested along the path. Then, the fragmentation problem causes partitioning of the data into smaller fragments. Each fragment consists of a relatively small number of instances, and therefore the prediction confidence of this fragment is limited. Note that replication always implies fragmentation, but fragmentation may happen without any replication.

(5) Curse of Dimensionality: If the data splits approximately equally on every split, then a univariate decision tree cannot test more than $O(logn)$ features (where n is the dataset size). This puts decision trees at a disadvantage for tasks with many relevant features.

2.4. Beyond classification tasks

While decision trees are most commonly used for classification tasks, they have been found to be useful for many other predictive tasks. Similarly to classification tasks, their full potential is brought to bear when they are used to constitute a forest.

1. A regression tree [10] is a decision tree that deals with a continuous target attribute. The basic idea is to combine decision trees and linear regression to forecast a numerical target attribute based on a set of input attributes. These methods perform induction by means of an efficient recursive partitioning algorithm. The choice of the best split at each node of the tree is usually guided by a least squares error criterion.
2. A survival tree [18,19] is a decision tree that aims to predict the time until an event of interest occurs. The event might be a death of a patient or when a machine ceases to function. Instead of assuming that the entire population has the same survival pattern, the survival tree splits the population according to its characteristics. In this case, the leaves of the tree hold the hazard functions. The hazard function is the probability that an individual will experience an event (for example, death) within a small time interval, given that the individual has survived up to the beginning of the interval.
3. Clustering tree [20–22]: The aim of clustering is to group instances into subsets in such a manner that similar instances are grouped together, while different instances belong to different clusters. A clustering tree is a decision tree where each node of the tree corresponds to a cluster, and the tree as a whole thus represents a kind of taxonomy.
4. Recommendation tree [23,24]: Often a recommender system attempts to recommend a set of items to users based on their past reactions to other items in the system and the reaction of other (similar) users. In a recommender tree, the terminal nodes hold a weighted list of recommended items. The internal nodes test properties that are known about the user such as a reaction to a certain item or demographic characteristics, such as age.
5. Ranking tree [25–27]: Ranking aims to predict instances of ordinal scale. Decision trees can be extended to address ranking tasks by replacing the splitting criterion with impurity measures that take the ranking into consideration.
6. Markov model tree [28]: The Markov model is a method for sequence learning which is capable of estimating the probability of sequences by training from data. In the Markov model tree, each node in the tree is a Markov model which uses the same states from the higher level, or only some of them, but with different transition probabilities.

In the following sections, we will mostly focus on classification trees with binary splits, but all methods can be generalized to other decision tree models unless specifically stated otherwise.

3. Ensemble learning and decision forest

Ensemble learning refers to the generation and the combination of multiple models to solve a particular learning task. In fact, the ensemble methodology imitates our second nature to seek several opinions before making a crucial decision such as choosing a medical treatment [29]. The core principle is to weigh several individual opinions and combine them in order to reach a decision that is better than the one obtained by each of them separately.

The idea of generating a predictive model that combines several models has been investigated since the seventies when Tukey [30] introduced an ensemble of two linear regression models. Tukey

suggested fitting the first linear regression model to the original data and the second linear model to the residuals. Two years later, Dasarathy and Sheela [31] suggested decomposing the input space using two or more classifiers.

Research on decision forest has started in early 1990s [32], but it only started gaining attention in mid-1990s when Freund and Schapire [33] formulated the well-known AdaBoost (Adaptive Boosting) algorithm.

It has been repeatedly shown that ensemble learning improves the predictive performance of a single model. Ensemble learning works particularly well when decision trees are used as the base models. Combining the ideas of ensemble learning and decision tree learning formed the notion of decision forests. Decision forests are frequently used in many real-world applications in various domains such as medicine, engineering, and information retrieval. Oza and Tumer [34] survey selected applications of ensemble methods to remote sensing, person recognition, one vs. all recognition, and medicine.

3.1. Why does decision forest work?

Marie Jean Antoine Nicolas de Caritat presented the well-known Condorcet's jury theorem as early as 1785. The theorem refers to a jury of voters who need to make a decision regarding a binary outcome (for example, to convict or not to convict a defendant). If each voter has a probability of $p > 0.5$ of being correct (i.e. better than a random guess), then the probability of a majority of voters being correct approaches 1 as the jury size goes to infinity assuming that the votes are independent.

The same idea can be applied for decision forest. If each individual tree in the forest provides a predilection that is better than a random guess, then combining the outputs of all trees can potentially yield an accurate prediction. We still need to comply with the assumption that the outputs of the decision trees are independent. Of course, this is not always the case nor is it easy to obtain such set of independent trees.

Dietterich [35] and Polikar [29] specify several reasons for why decision forest provides an improved predictive performance, including:

- Lack of adequate data: If the available training set is relatively small compared to the size of the space of all possible trees, the learning algorithm is compelled to arbitrarily choose one model from all equally performing models. This arbitrariness may lead to poor performance on unseen instances. Instead of choosing one of them, decision forest enables us to get around the dilemma by choosing all of them and weigh their classifications.
- Representation: In certain cases the hypothesis space does not include any decision tree that fits the data. Combining several trees usually extends the hypothesis space and enables us to obtain a better fit. Fig. 2(a) presents a training set of a simple binary classification task using only two attributes. The aim is to induce a classifier that can classify new emails to either spam (sad face) or non-spam (smiley face) based on the Email-Length (*X*-axis) and Number of Recipients (*Y*-axis). In this example we are using a decision stump as our base model. Recall that a decision stump is a decision tree that consists of a single split which splits the instance space into two sub-spaces either vertically or horizontally. Decision stump is a weak classifier in the sense that it cannot achieve a perfect classification for all training instances. In Fig. 2(a) we can see that each of the decision stumps still misclassify some of the training instances. Nevertheless, Fig. 2(b) shows that by combining all the three decision stumps, we obtain a perfect fit to the training instances.
- Avoiding local minima – Growing an optimal decision tree is known to be NP-hard. Thus, in order practically grow a decision tree, heuristics (such as the greedy algorithm) are used. However, these heuristics can get stuck in local minima. By combining several trees, we can decrease the risk for obtaining a local minimum.

It is well known that the error of a prediction model can be decomposed into three additive components: the intrinsic error, the bias error and the variance error. The intrinsic error, also known as the irreducible error, is the component that is generated due to noise. This quantity is the lower bound of any prediction model. The bias error of an inducer is the persistent or systematic error that the inducer is expected to make. Variance is a concept closely related to bias. The variance captures random variation in the algorithm from one training set to another. Namely, it measures the sensitivity of the algorithm to the actual training set, or the error that is due to the training set's finite size.

Dietterich and Kong [36] examined the tradeoff between variance and bias errors in decision trees. In a simple decision tree (i.e. with small number of nodes), the bias component of the error tends to be high and the variance component of the error tends to be small. As the complexity of the tree increases, the variance component of the error becomes larger and the bias error component becomes smaller. Empirical and theoretical evidence show that some decision forest techniques act as a variance reduction mechanism, i.e., they reduce the variance component of the error. Other decision forest techniques reduce both the bias and the variance parts of the error. In particular, it seems that the bias error is mostly reduced in the early iterations, while the variance error decreases in later ones.

3.2. Decision Forest for Mitigating Learning Challenges

In this section we go over several learning settings that are known to pose a challenge to machine learning algorithms in general, and to decision trees in particular. As described below decision forests have been used to mitigate these challenges:

1. Class imbalance – Class imbalance usually occurs when, in a classification problem, there are many more examples of a certain class than there are of another class. In such cases, standard machine learning techniques may be "overwhelmed" by the majority class and ignore the minority class. One way to deal with this challenge is to adjust the base induction algorithm of the individual trees. In particular, Cieslak and Chawla [37] suggest replacing the splitting criterion in the decision tree with a skew insensitive measure such as Hellinger distance. In addition to revising the base tree induction algorithm, a decision forest as a whole can be used to mitigate the class imbalance. One simple approach is to create a forest where each individual tree is trained using a balanced sub-sample where all minority classes are included and the majority class is under-sampled randomly [38]. Galar et al. [39] introduce a taxonomy for ensemble-based methods to address the class imbalance. Their detailed experimental comparison of the state-of-the-art methods has shown that excellent results can be obtained by combining random under-sampling techniques with ensemble techniques such as bagging or boosting. The predictive performance can be further enhanced using the EUSBoost method [40] which combines boosting and evolutionary undersampling that is adjusted to promote diversity among the individual trees.
2. Concept Drift – In many online learning scenarios, the underlying distribution of the target attribute tends to evolve over time. This results in predictions that are becoming less accurate over

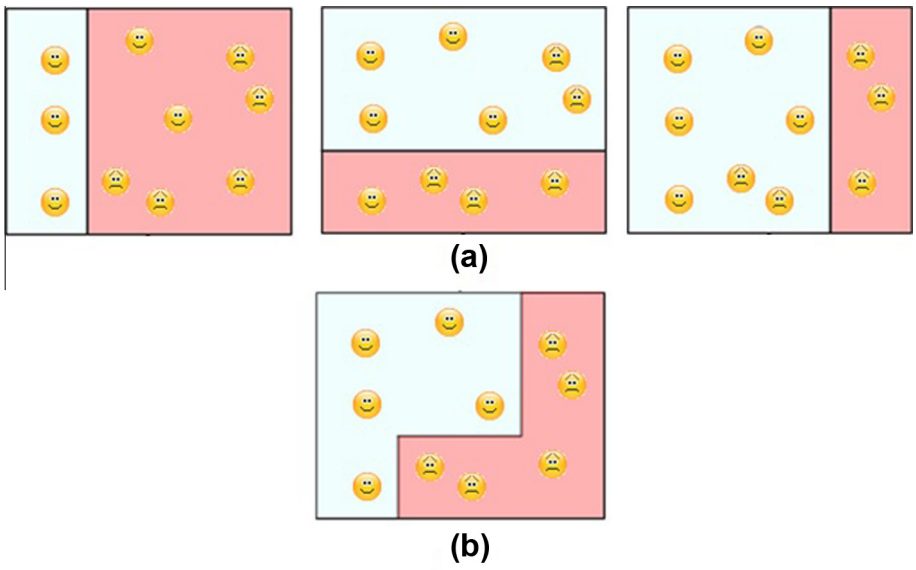

Fig. 2. A forest of decision stumps provides a perfect classification.

time. To prevent this phenomenon, learning algorithms need to adapt timely and accurately to the changes. Gama et al. [41] have extended existing tree induction algorithms to react to concept drift by continuously comparing the class distribution of the last built tree with the class distribution in a time window of the most recent examples. Another possible remedy is to build a forest of trees. Kolter and Maloof [42] develop a Dynamic Weighted Majority (DWM) forest model that dynamically creates and deletes individual trees in response to changes in performance. Minku et al. [43] show that different diversity levels in a forest are required in order to maintain high generalization on both old and new concepts. Moreover, diversity by itself can help to reduce the initial increase in error caused by a drift. Consequently, Minku and Yao [44] suggest an online ensemble learning approach called Diversity for Dealing with Drifts (DDD) that maintains two forests: one with a low diversity level and the second with a high diversity level. The diversity in a forest can be explicitly controlled by using a modified version of online Bagging [43]. Prior to drift detection, only the low diversity forest is used for predictions. Once a concept drift is detected, new low diversity and high diversity forests are trained and classification of new instances are provided by the weighted majority vote of the output of (1) the old high diversity, (2) the new low diversity, and (3) the old low diversity ensemble.

3. Curse of dimensionality – High dimensionality of the input (that is, the number of attributes) increases the size of the search space in an exponential manner and thus increases the chance that the inducer would find spurious classifiers that are not valid in general. It has been estimated that as the number of dimensions increases, the sample size required to train a decision tree should increase exponentially in order to have an effective estimate of multivariate densities. Certain ensemble learning methods can be used to lessen the impact of the "curse of dimensionality." For example, Bryll et al. [45] introduce a method called Attribute Bagging (AB) where each tree is trained on a small randomly selected subset of attributes. AB also searches the appropriate size of the attribute subset by testing prediction performance of variously sized random subsets of attributes. Huang et al. [46] introduce the Improved Decision Forest (IDF) algorithm that nests a feature selection mechanism and therefore is capable of addressing high dimensional datasets, in particular gene expression data that is comprised of ten thousands of genes. Rokach [47] presents a genetic algorithm (GA) based on building a forest for a dataset with many features. Instead of randomly sampling the feature set, the algorithm searches for the best mutually exclusive partitioning of the feature set, such that the individual decision trees are trained on disjoint subsets [48]. This method was found to be particularly useful for datasets with thousands of input features.
4. Decision forests can be used for addressing variants of basic learning tasks. For example, Vens et al. [49] show that decision forests are particularly useful for solving hierarchical multi-label classification tasks, where instances can be simultaneously assigned to several classes at the same time and these classes are organized in a hierarchy. Deng et al. [50] develop specialized decision forest algorithm for classifying time series.

4. Growing a decision forest

Based on the claims presented in Section 3.1, the following principles should be followed in order to generate a useful decision forest:

1. Diversity – Decision forest methods are very accurate, mainly due to the phenomenon that various types of classifiers have different "inductive biases" [51]. In order to make the forest more accurate, the individual trees should be sufficiently different.
2. Predictive performance – The individual trees' predictive performance level should be as high as possible and at least better than a random model.

From first sight, the above two principles seem to be contradictory. However, we are not looking for diversity, per se. In fact, it has been shown that an increase in diversity does not consistently correspond to an improvement in the ensemble accuracy [52]. The two principles can be combined by looking for accurate decision trees that are mistaken in different instances. Moreover, it has been empirically and theoretically shown that the accuracy of the entire forest is positively correlated to the degree of which the errors made by individual decision trees are uncorrelated [53].

There are several approaches in the interest of accomplishing the goal of diverse trees. In the following section we will describe the major approaches.

4.1. Manipulating the training sample

In this case, we vary the input that is used by the algorithm for training. Each tree is trained from a different training set. This method is useful for models in which the variance-error factor is relatively large, i.e. small revisions in the training set may result in a major change in the model.

The simplest approach for manipulating the training set is to use sampling with replacement. Each model is trained using a slightly different sample. The distribution of instances among the different instances could be random or it can be aligned with the class distribution, i.e. the class distribution in each sample is approximately the same as that in the entire dataset [54].

Instead of uniformly selecting the instances, some methods (like AdaBoost) iteratively manipulate the weights that are attached to each instance in the training set. By doing so, the sample of each member is built from a different distribution.

4.2. Manipulating the learning algorithm

In this approach, we manipulate the way in which the tree learning algorithm is used. One way is to alter the way the tree learning algorithm traverses the hypothesis space, thereby leading it to converge to different decision trees [55]. This can be achieved, for example, by injecting randomness' into the algorithm. For example, when growing a decision tree, instead of selecting the best attribute in each stage, it selects randomly (with equal probability) an attribute from the set of the best k attributes.

Another way is to manipulate the learning algorithms' parameters. The tree learning algorithm is usually controlled by a set of parameters, such as minimum training instances and pruning confidence level. Running the algorithm with different parameter settings may lead to different trees [56].

4.3. Partitioning

Partitioning means dividing the original training set into smaller training sets. A different tree is trained on each sub-sample. There are two obvious ways to partition the original dataset: horizontal partitioning and vertical partitioning. In horizontal partitioning the original dataset is partitioned into several datasets that have the same features as the original dataset, each containing a subset of the instances in the original [57]. In vertical partitioning the original dataset is partitioned into several datasets that have the same number of instances as the original dataset, each containing a subset of the original set of features [47].

Ting et al. [58] introduce the Feature-Subspace Aggregating method (Feating), which combines several local trees such that each tree is induced from a distinct local region of the feature-space. In order to obtain the distinct subspaces, the Feating algorithm partitions the feature-space into mutually exclusive local regions that are defined over a fixed number of attributes. The user specifies the number of features used for partitioning the subspace, and by that—controls the level of localization.

4.4. Heterogeneous tree induction algorithms

Gashler et al. [59] introduce diversity in the forest generation by combining various tree induction algorithms. They empirically show that a small heterogeneous forest is better than a large homogeneous forest for a dataset that has many irrelevant attributes.

4.5. Ensemble hybridization

In ensemble hybridization, at least two ensemble strategies are combined in order to generate the decision forest. For example, Zhang and Zhang [60] present the RotBoost algorithm which combines the ideas of Rotation Forest and AdaBoost. RotBoost achieves an even lower prediction error than either one of the two algorithms. In each iteration a new rotation matrix is generated and used to create a dataset. The AdaBoost ensemble is induced from this dataset.

Bernard et al. [61] presented the Dynamic Random Forest algorithm which is based on an adaptive tree induction procedure. The main idea is to guide the tree training such that each tree will complement the existing trees in the forest as much as possible. Here, this is done through a resampling of the training data, inspired by boosting algorithms, and combined with other randomization processes used in the traditional random forest.

Hady et al. [62] proposed the *cotrain-of-trees* scheme which combines the co-training strategy and the tree-structured strategy for solving multi-class problems. First, a tree-structured forest of binary models is trained on each given view. Then, co-training is applied in order to identify the most confident unlabeled examples labeled by one view to add new instances to the other view.

5. Decision fusion

Decision fusion refers to the process of combining the outputs of the individual trees in order to provide a single and unified output. There are three main approaches for fusing the individual outputs. The following subsections briefly describe them.

5.1. Weighting methods

The individual trees' outputs are combined using weights that are assigned to each tree. The weighting methods are best suited for problems where the individual tree performs the same task and has comparable success, or when we would like to avoid problems associated with added learning (such as overfitting or a long training time).

In the simplest setting, which is known as *majority voting*, a prediction of an instance is performed according to the output that obtains the highest number of votes (the most frequent vote). The distribution summation method is a slightly gentler voting schema than simple majority voting. The idea of the distribution summation method is to sum up the conditional probability vector obtained from each classification tree [63]. The selected class is chosen according to the highest value in the total vector.

In other weighting settings, a tree's output has strength proportional to its assigned weight. In particular, the weight of each tree can be set proportional to its accuracy performance on a validation set [64]. In the Bayesian combination method, the weight associated with each tree is the posterior probability of the tree given the training set [65].

Shapire and Singer [66] show that the blending of the decision can be improved if each prediction model can provide the confidences to each of their predictions. Specifically, they give a specific method for assigning confidences to the predictions of decision trees based on the number of instances that reach the corresponding leaf and their class distribution.

Several methods try to set the weight based on the co-existence among the trees. For example, Variance Optimized Bagging, or the *vogging* approach, aims to optimize a linear combination of the trees so as to aggressively reduce variance while attempting to preserve a prescribed accuracy [67].

5.2. Meta-learning methods

Meta-learning is a process of learning from learners. The training of a meta-model is composed of two or more stages, rather than one stage, as with standard learners. In order to induce a meta-model, first the individual trees are trained (stage one), and then the meta-model (second stage). In the combination phase, individual trees provide their outputs, and then the meta-model generates the final output as a function of the individual outputs. Meta-learning methods are best suited for cases in which certain trees have different performances on different subspaces. For example, in the case of classification tasks, the trees may consistently correctly classify, or consistently misclassify, certain instances.

Stacking is probably the most-popular meta-learning technique. By using a meta-learner, this method tries to induce which trees are reliable and which are not. The idea is to create a meta-dataset containing a tuple for each tuple in the original dataset. However, instead of using the original input attributes, it uses the predicted classifications by the individual trees as the input attributes. The target attribute remains as in the original training set. A test instance is first classified by each of the individual trees. These classifications are fed into a meta-level training set from which a meta-classifier is produced. This classifier combines the different predictions into a final one. It is recommended that the original dataset should be partitioned into two subsets. The first subset is reserved to form the meta-dataset and the second subset is used to build the individual trees. Stacking performance can be improved by using output probabilities for every class label from the individual classification trees. In such cases, the number of input attributes in the meta-dataset is multiplied by the number of classes.

In order to avoid over-fitting, it is recommended to partition the training set into two subsets. The first subset is reserved to form the meta-dataset and the second subset is used to build the individual trees. It has been shown that with stacking the ensemble performs (best) when compared to selecting the best classifier from the ensemble by cross validation.

There are many variants of the basic Stacking algorithm. In particular, StackingC [68] and Troika [69] aim to improve the predictive performance in multi-class datasets. In addition to Stacking there are many other meta-learning algorithms designed for combining the individual trees' output, including: Arbiter Trees, Combiner Trees and Grading [70].

One distinct variant of meta-learning is "mixture-of-experts." According to this approach we use a single decision tree (i.e. expert) that is most suitable to the instance that is currently being examined. The premise in the single best output is that there is a competent authority that nominates the best tree for a given instance. The output of the selected tree is referred to as the output of the ensemble as a whole. Very often, the input space is decomposed into several sub-spaces which can have any shape or size. Sometimes the subspaces have soft "boundaries," namely, subspaces are allowed to overlap. Then, for each sub-space we nominate one decision tree to be responsible for providing the prediction. In order to choose the best tree, we usually use another learning algorithm such as neural network or logistics regression.

Omari and Figueiras-Vidal [71] introduce the post-aggregation procedure for fusing a large decision forest. In principle, this procedure is a hybrid fusion method that combines the merit of meta-learning methods with the merit of weighting methods. The post-aggregation procedure consists of two main steps. In the first step, a weighting method (such as distribution summation) is used to fuse the individual trees' outputs and create an initial aggregation. In the second step, a post-aggregation is obtained by applying a machine learning algorithm on the original training instances together with their initial aggregation.

6. Popular decision forest methods

A decision forest can be created by two main means: (1) using a general ensemble method (such as AdaBoost) that can virtually be used with any base learning method, including decision trees, and (2) ensemble methods that were designed specifically for creating a decision forest (such as Random Forest).

Fig. 3 presents a timeline graph which includes the most important decision forest methods published between 1995 and 2006 in a chronological order. The bar is labelled with the publication years with non-iterative methods are displayed to the right and iterative methods are displayed to the left. The method's rectangle has an area proportional to the number of citations it has received in Google Scholar as for May 2015.

In the following sub-sections we briefly review the most popular methods that are shown to be useful in improving the predictive performance of decision trees:

6.1. Bootstrap aggregating

Bagging (bootstrap aggregating) is a simple yet effective method for generating an ensemble of models in general and decision forests in particular. Each decision tree in the forest is trained on a sample of instances taken as a replacement from the original training set. All trees are trained using the same learning algorithm. The final prediction of an unseen instance is determined by performing a majority voting of the individual trees' predications. Since sampling with replacement is used, some of the original instances may appear more than once while other instances may not be included at all. To ensure that there is a sufficient number of training instances in every sample, it is common to set the size of each sample to the size of the original training set. One of the main advantages of bagging is that it can be easily implemented in a parallel mode by training the various ensemble classifiers on different processors.

Often, bagging produces a combined model that outperforms the model that is built using a single instance of the original data. Breiman [72] notes that this is true especially for unstable inducers such as decision trees since bagging can eliminate their instability. In this context, an inducer is considered unstable if perturbations in the learning set can produce significant changes in the constructed classifier.

Breiman [73] refers to bagging decision trees as a particular instance of the *random forest* technique which is described in the next sub-section.

6.2. Random forest and random subspace methods

Random forest is probably the most popular decision forest that was developed. The idea was introduced independently and almost simultaneously by Ho [74] and Amit and Geman [75] in the mid-1990s. It was later perfected and popularized by Breiman [73]. The paper [73] is cited more than 16,500 times according to Google Scholar (as of May, 2015) and its popularity increases every year mainly due to its simplicity, in one respect, and its predictive performance in another.

A random forest uses a large number of unpruned random decision trees that their outputs are combined using an un-weighted majority of class votes. In order to grow a random tree which is still sufficiently accurate, we inject randomness into the decision tree induction algorithm using two randomization processes:

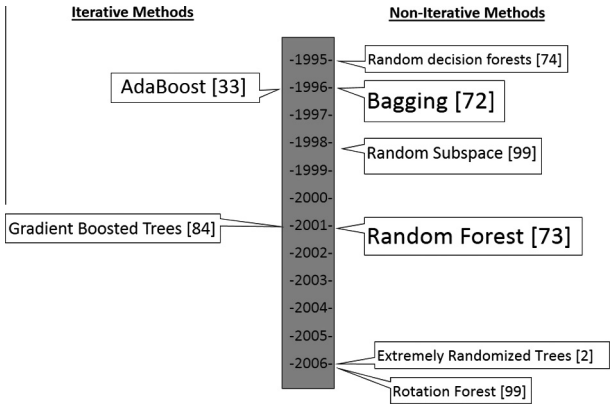

Fig. 3. A timeline graph of the most important decision forest methods published between 1995 and 2006.

1. Bootstrap instances from the training data. Specifically, each tree is trained on a different sample of instances taken as a replacement from the original training set.
2. Rather than choosing the best split among all attributes, the inducer randomly samples a subset of the attributes and chooses the best split among them. The subset size (n) is recommended to be approximately $n = log_2 N + 1$ where N is the number of attributes.

The second randomization process can be implemented differently. Instead of selecting the best attribute at each node (using, for instance, an information gain measure), the attribute is selected randomly such that its probability of being selected is proportional to its measured value. A similar idea has been implemented for a randomized C4.5 decision tree [76]. Instead of selecting the best attribute in each stage, it selects randomly (with equal probability) an attribute from the set of the best n features.

Saffari et al. [77] developed an on-line version of the random forest algorithm that is capable of addressing training data that arrives sequentially or the underlying distribution is continuously changing. Specifically they add a temporal weighting scheme for adaptively discarding some trees based on their out-of-bag-errors in given time intervals and the consequent growing of new trees.

Originally, the random forest algorithm applied only to decision trees, and is not applicable to other types of classifiers, because it involves picking a different subset of the attributes in each node of the tree. Principally the main step of the random forest algorithm can be easily replaced with the broader "Random Subspace Method" [78], which can be applied to many other inducers, such as Support Vector Machines. Notwithstanding, a recent comparison of Random Forest and Random Subspace of decision trees has shown that the former outperforms the later in terms of accuracy [1].

There are other ways to inject randomness to the decision tree induction algorithm, in particular when numeric features are involved. For example, instead of using all the instances to determine the best split point for each numeric feature, a sub-sample of the instances is used [79]. This sub-sample varies with the feature. The feature and split value that optimize the splitting criterion are chosen as the decision at that node. Since the split made at a node is likely to vary with the sample selected, this technique results in different trees which can be combined in ensembles. Another method for randomization of the decision tree uses histograms [80]. The use of histograms has long been suggested as a way of making the features discrete, while reducing the time to handle very large datasets. Typically, a histogram is created for each feature, and the bin boundaries are used as potential split points. The randomization in this process is expressed by selecting the split point randomly in an interval around the best bin boundary.

Désir et al. [81] proposed an extension to random forest called *One Class Random Forests* (OCRF) for solving one class classification tasks. One class classification is a binary classification task for which only one class of samples is available for learning. In OCRF, a procedure for generating artificial outliers is embedded into the original random forest algorithm. This procedure utilizes the two above-mentioned randomization processes that are already offered by random forest to create a relatively small subset of artificial outliers. Referring to the original instances as the positive class and the artificial instances as the negative class, we are now facing a standard binary classification problem that can be solved by a random forest. Kulkarni and Sinha [6] review additional variations of random forest. They present a specific taxonomy of random forest and compare the existing random forest method on the basis of relevant parameters.

Due to its popularity, the implementation of random forest can be found in various software platforms, such as: Weka, R,[3] Matlab,[4] scikit-learn[5] and C#.[6]

6.3. Extremely randomized trees

Similarly to random forest, extremely randomized trees also aim to generate a decision forest while injecting randomness [2]. Recall that random forest selects the best splitting attribute and its best corresponding cut-point among a random subset of features. In contrast, extremely randomized trees randomize both the splitting feature and its corresponding cut-point. For selecting the splitting feature, the algorithm randomly chooses a certain number of features among which the best one is determined. In addition to numerical features, the algorithm draws a random cut-point uniformly in the feature value domain, i.e. the cut-points are selected fully at random independently of the target attribute. In the extreme case, the algorithm randomly selects a single attribute and cut-point at each node, and hence a totally randomized tree grows. Moreover, contrary to random forest, extremely randomized trees do not use the bootstrap instances procedure to construct a set of the training samples for each tree and the same input training set is used to induce all decision trees.

Geurts et al. [2] show that the individual extremely randomized tree tends to have high bias and variance components. However, the variance component can be canceled out by fusing a sufficiently large forest of trees.

6.4. Rotation forest

Rotation forest [82] is another popular method for generating a decision forest. In rotation forest the diversity among the base trees is achieved by training each tree on the whole data set in a rotated feature space. Performing a rotation of the axes just before running the tree induction algorithm may result with a very different classification tree. In addition to ensuring diversity among the trees, rotated trees also relax the constraint of univariate trees which are capable of splitting the input space only into hyperplanes that are parallel to the original feature axes.

More specifically, the main idea is to use feature extraction methods to build a full feature set for each tree in the forest. To this end, we first randomly split the feature set into K mutually exclusive partitions. Then we use a principal component analysis (PCA)

[3] http://cran.r-project.org/web/packages/randomForest/.
[4] https://code.google.com/p/randomforest-matlab/.
[5] http://scikit-learn.org/stable/modules/ensemble.html#forest.
[6] http://www.semanticquery.com/archive/semanticsearchart/researchRF.html.

separately on each feature partition. The idea of PCA is to orthogonally transform any possibly correlated features into a set of linearly uncorrelated features (called principal components). Each component is a linear combination of the original. The transformation guarantees that the first principal component has the largest possible variance. Every subsequent component has the highest variance possible under the constraint that it be orthogonal to the previous components.

The principal components are used to construct the new set of features. The original data set is transformed linearly into the new feature space to construct the new training set. This new training set is fed into the tree induction algorithm which trains a classification tree. Note that different feature partitions will lead to a different set of transformed features, thus different classification trees are generated. The rotation forest algorithm is implemented as part of the Weka suite and as a Matlab code.[7]

The experimental study conducted by the inventors of rotation forest, show that rotation forest outperforms random forest in terms of accuracy. However rotation forest has two drawbacks. First, it is usually more computationally intensive than random forest due to the need of calculating the PCA. Another drawback is that the nodes in the obtained trees are the transformed features and not the original features. This can make it harder for the user to understand the tree because instead of examining a single feature in each node of the tree, the user needs to examine a linear combination of the features in each node of the tree.

Schclar and Rokach [83] suggest using random projections for creating the individual trees rather than using the PCA projection. This reduces the computational complexity burden that arises from the calculation of the principle components. Every derived feature that is used for training the tree is a random linear combination of the original features.

6.5. Adaptive boosting

Boosting is a general method for improving the performance of a weak learner such as a decision stump. The method works by iteratively invoking a decision tree induction algorithm, on training data that is taken from various distributions. The decision trees are then combined into a single strong composite forest. Similarly to bootstrapping methods, boosting methods utilize a resampling mechanism, however instead of randomly sampling the instances in each iteration, boosting revises the sample in order to provide the most useful sample for each consecutive iteration.

AdaBoost (Adaptive Boosting) is probably the most popular boosting algorithm [33]. The main idea is to provide a higher weight to instances that are misclassified in previous iterations. In particular, the weights of all misclassified instances are intensified while the weights of correctly classified instances are decreased. As a consequence, the inducer focuses on the difficult instances. This iterative procedure provides a series of decision trees that complement one another.

6.6. Gradient boosted decision trees

Gradient Boosted Decision Trees is a boosting algorithm designed originally for regression tasks [84]. Similarly to other boosting algorithms, it builds the forest in a stage-wise fashion. Specifically, it computes a sequence of regression trees, where each successive tree aims to predict the pseudo-residuals of the preceding trees given an arbitrary differentiable loss function. Every leaf in the induced tree minimizes the loss function in the corresponding partition. By using a suitable loss function, a gradient boosted tree can also be used for classification tasks. In particular, both Friedman [84] and Li [85] present variation of gradient boosting trees that are adjusted for multi-class classification tasks.

Choosing the right number of trees in the gradient boosted forest (i.e. number of iterations), is very important in order to avoid overfitting. Setting it too high may lead to overfitting while setting it too low may lead to underfitting. One way to select the optimal value is to simply try different forests' sizes and measure the prediction performance on a separate validation data set.

Overfitting can also be avoided by using a stochastic gradient boosting method. The general idea is to consecutively train a tree using an independently drawn random sample taken from the training set. Since every tree in the forest is built using a different sample of instances, the chance for overfitting is reduced and the entire forest generalizes well to new instances.

6.7. IPGA-forest

Martínez-Muñoz and Suárez [86] suggest utilizing the nature of the IPGA decision tree induction algorithm to generate a decision forest. In the base IPGA decision tree induction algorithm the training data is randomly divided into two subsets. In the first iteration the first subset is used to grow the decision tree. The second subset is then used to prune the tree. In the second iteration the roles of the data subsets are interchanged. Starting from the tree that was grown in the first iteration, the tree is grown again using the training instances of the second subset. Then the tree is pruned using the first subset. The roles of the data subsets are interchanged again and the two iterations are repeated until two consecutive pruned trees have the same size. In order to generate a variety of classifiers for the forest, the two subsets are randomly created for each tree in the forest [86].

6.8. Switching classes

Breiman [87] presents a decision forest in which each decision tree in the forest is generated using the original training set but with randomized class labels. The class label of each training instance is switched according to a transition matrix that defines the probability that a class i is replaced with class j. The switching probabilities are chosen to maintain the class distribution of the original training set.

Martínez-Muñoz and Suárez [88] show that the switching classes method is particularly accurate when the forest is large enough and it is created using high class switching rates that does not necessarily maintain the original class distribution. The relaxation of the original class distribution constraint is crucial for using the switching classes method in an imbalanced dataset. In each iteration, a fixed fraction of the original training is randomly selected. The classes of these selected instances are switched at random.

6.9. Comparison of decision forest algorithms

There were several attempts in the literature to compare different decision forest algorithms. Dietterich [76] has compared three methods for constructing a forest of C4.5 classifiers: randomizing, bagging, and boosting. The experiments show that when there is little noise in the data, boosting gives the best results. Bagging and randomizing are usually equivalent.

Another study compared bagging-based and boosting-based decision forests [89]. The study determined that bagging reduced the variance of unstable methods, while boosting methods reduced both the bias and variance of unstable methods but increased the variance for stable methods.

[7] http://www.mathworks.com/matlabcentral/fileexchange/38792-rotation-forest-algorithm.

An additional study that compared bagging-based with boosting-based decision forests indicated that bagging is sometimes significantly less accurate than boosting [90]. The study indicated that the performance of the boosting methods is much more sensitive to the characteristics of the dataset. In particular, boosting may overfit noisy data sets and reduce classification performance.

Villalba Santiago et al. [91] compared seven different boosting algorithms for generating decision forests where the trees are decision stumps, that is, decision trees with only one decision. They conclude that for binary classification tasks—the well-known AdaBoost should be preferred. However for multi-class tasks other boosting methods such as GentleAdaBoost should be considered.

Banfield et al. [92] experimentally evaluated bagging and seven other randomization-based approaches for creating a decision forest. Statistical tests were performed on experimental results from 57 publicly available datasets. When cross-validation comparisons were tested for statistical significance, the best method was statistically more accurate than bagging in only eight out of the 57 datasets. Alternatively, examining the average ranks of the algorithms across the group of datasets, Banfield et al. [92] concluded that Random forest and Boosting with 1,000 trees provided the best performance.

Fernández-Delgado et al. [1] compared the predictive performance of 14 decision tree induction algorithms (including decision stump, REPTree, C4.5, C5.0, etc.), 8 random forest variants, 20 boosting variants (including AdaBoost), 24 variants of bagging and 11 other ensemble methods over 121 datasets concluded that decision forest—and in particular random forest—tends to outperform other learning methods. In particular the best classifier is the parallel random forest variant implemented in *R* using the random Forest and caret packages.

Other than predictive performance, there are additional criteria according to which practitioners can select the most suitable decision forest method for the problem in hand:

- Suitability to the learning setting in hand—Different decision forests methods have different capabilities in addressing specific learning setting, such as imbalance, high dimensionality, multi-label classification and noisy data. The practitioner first needs to characterize the learning task and choose the method accordingly.
- Computational Cost—The complexity cost for building the decision forest and in real time applications also the time required for prediction for new instances. Usually iterative methods such as gradient boosted trees are found to be more computationally effective.
- Scalability—The ability of the decision forest algorithm to scale to big data. Here, random forest and gradient-boosted trees seem to provide the best scalability.
- Software Availability—How many off-the-shelf software packages provide the implementation of the decision forest method. High Availability implies that the practitioner can move from one software to another, without the need to replace the decision forest method.
- Usability—Providing a set of controlling parameters that are comprehensive and can be easily tuned.

6.10. Scaling up decision forests methods

With the recent growth in the amount of data collected by information systems, there is a need for decision tree algorithms that can induce from large datasets. While the availability of large amounts of data is ideal for the needs of any data scientist, it does pose time and memory challenges for many learning algorithms. Big data is a term coined recently to refer to large datasets that are too difficult to process using existing methods.

For small to medium datasets, the computational complexity of decision tree induction algorithms is considered to be relatively low. However, when training a dense forest on Big Data, one can still come across difficulties. Scalability refers to the ability of the method to train the predicative method efficiently given large amounts of data.

The greedy nature of tree induction algorithm does not scale well to a large dataset. For example, to find the best split in the root node, going over all instances in the dataset is required. And when the entire dataset does not fit in the main memory, this full scan becomes a major problem. In the pre-Big Data era, the scalability efforts were focused on relaxing the memory constraint. Several decision tree algorithms were developed that do not require that the entire training data be loaded to the main memory. This category includes the following algorithm: SPRINT [93], SLIQ [94] and FastC4.5 [95]. It should be noted that these algorithms are still limited by the resources of a single machine and by the cost of scanning data from secondary storage.

Thus, in recent years the scalability efforts have focused on parallelization techniques such as MapReduce and MPI that distribute the training effort among many machines. MapReduce is one of the most popular parallel programming frameworks for data mining. In this framework, programmers need to specify a map function which processes key/value pairs to emit a set of intermediate key/-value pairs and a reduce function that aggregates all intermediate values associated with the same intermediate key. The MapReduce framework was pioneered by Google and then popularized by the open-source Apache Hadoop project. While there are other parallel programming frameworks (such as CUDA and MPI), MapReduce has become the industry standard and is implemented on cloud computing services such as Amazon EC2 and various companies such as Cloudera which provide services to ease Hadoop deployment.

PLANET [96] is a decision forest algorithm implemented in the MapReduce framework. The basic idea of PLANET is to iteratively grow the decision tree, one layer at a time, until the data partitions are small enough to fit the main memory and the remaining subtree can be grown locally on a single machine. For the higher levels, the key idea of PLANET is that the splitting criterion at a certain node does not need the entire dataset but a compact data structure of sufficient statistics which in most cases can be fit in-memory. These statistics are calculated on the mappers.

Bootstrapping-based methods for generating a decision forest (such as Random Forest) can be easily emulated by replacing the bootstrap sample with local data, i.e., each machine in the cluster is trained on the data that is locally stored. On the other hand, parallelizing of boosting-based methods such as AdaBoost and Gradient Boosted Decision Trees is not straightforward due to their inherent sequential nature.

Ye et al. [97] present two distributed versions of the Stochastic Gradient Boosted Decision Trees method: the first is implemented in the MapReduce framework and the second in the MPI framework. The methods produce trees identical to those generated by the corresponding serial version. In order to obtain the exact solution, all nodes in the cluster are required to evaluate the potential split points found by all other nodes. Palit and Reddy [98] utilize the MapReduce framework for developing two parallel boosting algorithms: AdaBoost.PL and LogitBoost.PL which are competitive to their corresponding serial versions in terms of predicative performance. Theses algorithms need only one cycle of MapReduce to complete the algorithms. Each Mapper runs a respective AdaBoost algorithm on their own subset of the data to induce the set of weak models. Then, the base models are sorted and are

transmitted along with their weights to the reducers. The reducer averages the weights to derive the weights of the final ensemble.

del Río et al. [99] introduce MapReduce implementation for various common methods that are capable to tackle imbalanced classification tasks using Random Forest. The results show that in most cases, the execution time is reduced when the number of mappers is increased, nevertheless, too many mappers may results in a deteriorated performance.

Various distributed implementations of decision forests are available for the practitioners. In particular, Mahout, which is an Apache project that provides free implementations of scalable machine learning algorithms, includes an implementation of Random Forest[8] on top of Hadoop framework. MLLib, a distributed machine learning framework, provides implementations for Random Forest and Gradient-Boosted Trees[9] on top of Spark framework.

7. Decision forest thinning: making the forest smaller

Some decision forests superfluously contain too many decision trees. This may result in large storage requirements, reduced compressibility, prolonged prediction time [100] and even reduced predictive performance [101].

Hence, forest thinning or pruning is a post-processing step that aims to identify a subset of decision trees that performs at least as good as the original forest and discard any other trees as redundant members. Once thinning is completed, we can keep only the selected trees for the purpose of decision fusion.

In earlier research on ensemble pruning [101], the goal was to use a small size of an ensemble to achieve an equivalent performance of a boosted ensemble. This has been proven to be NP-hard and is even hard to approximate [102]. Additionally, the pruning may sacrifice the generalization ability of the final ensemble. After Zhou et al. [103] proved the "many-could-be-better-than-all" theorem, it became well-known that it is possible to get a small yet accurate decision forest.

In addition, pruning can be performed dynamically to the instance at hand. In this way, we still store the entire decision forest, but for the purpose of providing the classification of a given instance, we can use only some of the decision trees. Hernández-Lobato et al. [104] and Park and Furnkranz [105] present an efficient algorithm that queries only a dynamically determined subset of the decision trees, but still provide the same outcome that would has been provided if all members had been used. The idea is implemented on majority voting for multi-class classification tasks in which the winning class can be determined even without having the votes of all decision trees. Martinez-Muoz et al. [106] investigate several methods for modifying the order of combining the decision trees in the forest. In particular, they show that if an appropriate ordering for the combination process is devised, then the predictive performance reaches a maximum when using only some of the decision trees included in the forest.

Many ensemble pruning methods have been developed recently over the last few years. Roughly speaking, the two most popular approaches for selecting an ensemble subset are ranking-based and search-based methods (see [100] for additional approaches) that are briefly discussed in the following subsections.

7.1. Ranking-based methods

The idea of this approach is to once rank the individual trees according to a certain criterion and choose the top ranked trees according to a threshold. For example, Prodromidis et al. [107] suggest ranking classification trees according to their classification performance on a separate validation set and their ability to correctly classify specific classes. Similarly, Caruana et al. [108] presented a forward stepwise selection procedure in order to select the most relevant classifiers (that maximize the ensemble's performance) among thousands of classifiers. The algorithm FS-PP-EROS generates a selective ensemble of rough subspaces [109]. The algorithm performs an accuracy-guided forward search to select the most relevant members. The experimental results show that FS-PP-EROS outperforms bagging and random subspace methods in terms of accuracy and size of ensemble systems.

In attribute bagging [45], classification accuracy of randomly selected m-attribute subsets is evaluated by using the wrapper approach and only the classifiers constructed on the highest ranking subsets participate in the ensemble voting. Margineantu and Dietterich [101] present an agreement based on ensemble pruning which measures the Kappa statistics between any pair of classifiers. Then, pairs of classifiers are selected in ascending order of their agreement level until the desired ensemble size is reached.

As indicated above, diversity among the trees is an important precondition for the success of a decision forest. Banfield et al. [110] present a diversity-based method for thinning a decision forest. Specifically, the Percentage Correct Diversity Measure (PCDM), evaluates the proportion of classifiers which get each example correct. It focuses on instances in which their classification is ambiguous (i.e. instances for which there is a general consensus that they are not considered useful in the determination of ensemble diversity).

Partalas et al. [111] propose an ensemble pruning ranking measure called Uncertainty Weighted Accuracy (UWA), which considers the uncertainty of the classification of the current ensemble. The UWA measure is strongly influenced by instances for which the members disagree and therefore ensemble's decision is highly uncertain. The ensemble models are evaluated on a pruning set. Withholding a part of the training set for evaluating pruning is done in order to avoid overfitting.

Rokach [112] introduce the collective-agreement-based pruning (CAP) method. Rather than ranking individual decision trees, CAP ranks subsets by considering the individual predictive ability of each tree along with the degree of redundancy among them. Subsets whose members highly agree with the class while having low inter-agreement are preferred.

Zhang and Wang [113] propose a specific method to find a subforest that can achieve the prediction accuracy of a large random forest (in the order of thousands of trees). They consider three measures to determine the importance of a tree in a forest in terms of prediction performance in order to find the minimal size of the forest. The first measure evaluates how removing a certain tree from the forest affects the overall prediction accuracy. The other two measures examine the similarity among the trees and remove trees that are "similar" to other trees in the forest.

7.2. Search-based methods

Instead of separately ranking the members, one can perform a heuristic search in the space of the possible different ensemble subsets while evaluating the collective merit of a candidate subset. The GASEN algorithm was developed for selecting the most appropriate classifiers in a given ensemble [103]. In the initialization phase, GASEN assigns a random weight to each of the classifiers. Consequently, it uses genetic algorithms to evolve those weights so that they can characterize to some extent the fitness of the classifiers in joining the ensemble. Finally, it removes from the ensemble those classifiers whose weight is less than a predefined threshold value. A revised version of the GASEN algorithm called

[8] http://mahout.apache.org/users/basics/algorithms.html.
[9] https://spark.apache.org/docs/latest/mllib-ensembles.html.

GASEN-b has been suggested [114]. In this algorithm, instead of assigning a weight to each classifier, a bit is assigned to each classifier indicating whether it will be used in the final ensemble. In an experimental study, the researchers showed that ensembles generated by a selective ensemble algorithm, which selects some of the trained C4.5 decision trees to make up an ensemble, may not only be smaller in size but also stronger in the generalization than ensembles generated by non-selective algorithms.

Rokach et al. [115] suggest first ranking the classifiers according to their ROC performance. Then, they suggest evaluating the performance of the ensemble subset by using the top ranked members. The subset size is increased gradually until there are several sequential points with no performance improvement.

Prodromidis and Stolfo [116] introduce a backward correlation based pruning. The main idea is to remove the members that are least correlated to a meta-classifier which is trained based on the classifiers' outputs. In each iteration they remove one member and recompute the new reduced meta-classifier (with the remaining members). The meta-classifier in this case is used to evaluate the collective merit of the ensemble.

Windeatt and Ardeshir [117] compared several subset evaluation methods that were applied to Boosting and Bagging. Specifically the following pruning methods have been compared: Minimum Error Pruning (MEP), Error-based Pruning (EBP), Reduced-Error Pruning (REP), Critical Value Pruning (CVP) and Cost-Complexity Pruning (CCP). The results indicate that if a single pruning method needs to be selected then overall the popular EBP makes a good choice.

Zhang el al. [118] formulate the ensemble pruning problem as a quadratic integer programming problem to look for a subset of classifiers that has the optimal accuracy-diversity trade-off. Using a semi-definite programming (SDP) technique, they efficiently approximate the optimal solution, despite the fact that the quadratic problem is NP-hard.

Search-based methods provide a better classification performance than the ranking-based methods [107]. However search-based methods are usually computationally expensive due to their need for searching a large space. Thus one should select a feasible search strategy. Moreover, independent to the chosen search strategy, the computational complexity for evaluating a single candidate subset usually is at least linear in the number of instances in the training set (see [100] for complexity analysis of existing evolution measures).

8. Conclusions

Decision forest aims at improving the generalization ability of a single decision tree by combing the output of several trees. This overview paper has highlighted the major approaches and techniques used in decision forest, specifically the three major aspects involved in using a decision forest: Growing, Thinning and Combining.

The current research trends in decision forest include: developing distributed implementation for addressing the big data challenge; developing decision forest methods for various learning tasks or settings other than the general purpose regular classification and regression; developing static and dynamic methods for decision forest thinning.

Acknowledgments

The author gratefully thanks the editor-in-chief, the Associate Editor and the anonymous reviewers whose constructive comments helped in improving the quality and accuracy of this paper.

References

[1] M. Fernández-Delgado, E. Cernadas, S. Barro, D. Amorim, Do we need hundreds of classifiers to solve real world classification problems?, J Mach. Learn. Res. 15 (1) (2014) 3133–3181.
[2] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, Mach. Learn. 63 (1) (2006) 3–42.
[3] L.I. Kuncheva, Combining Pattern Classifiers Methods and Algorithms, second ed., Wiley, Hoboken, NJ, 2014.
[4] M. Woźniak, M. Graña, E. Corchado, A survey of multiple classifier systems as hybrid systems, Inform. Fusion 16 (2014) 3–17.
[5] A. Verikas, A. Gelzinis, M. Bacauskiene, Mining data with random forests: a survey and results of new tests, Pattern Recogn. 44 (2011) 330–349.
[6] V.Y. Kulkarni, P.K. Sinha, Random forest classifiers: a survey and future research directions, Int. J. Adv. Comput. 36 (1) (2013) 1144–1153.
[7] A. Criminisi, J. Shotton, Decision Forests for Computer Vision and Medical Image Analysis, Springer Science & Business Media, 2013.
[8] R.A. Fisher, The use of multiple measurements in taxonomic problems, Ann. Eugenic. 7 (2) (1936) 179–188, http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x.
[9] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers, 1993.
[10] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and regression trees, Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, CA, 1984, ISBN 978-0-412-04841-8.
[11] I.H. Witten, E. Frank, M.A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, 3rd ed., Morgan Kaufmann, San Francisco, 2011.
[12] T.M. Therneau, E.J. Atkinson, An Introduction to Recursive Partitioning Using the rpartRoutine, Technical Report 61, Section of Biostatistics, Mayo Clinic, Rochester, 1997. <http://www.mayo.edu/hsr/techrpt/61.pdf>.
[13] S.A. Moore, D.M. Daddario, J. Kurinskas, G.M. Weiss, Are decision trees always greener on the open (source) side of the fence?, in: Proceedings of DMIN, 2009, pp. 185–188.
[14] R.C. Barros, André C.P.L.F. de Carvalho, Alex A. Freitas, Automatic Design of Decision-Tree Induction Algorithms, Springer, 2015.
[15] G. Pagallo, D. Huassler, Boolean feature discovery in empirical learning, Mach. Learn. 5 (1) (1990) 71–99.
[16] S. Murthy, S. Salzberg, Lookahead and pathology in decision tree induction, in: IJCAI, 1995, pp. 1025–1033.
[17] S. Esmeir, S. Markovitch, Lookahead-based algorithms for anytime induction of decision trees, in: Proceedings of the Twenty-First International Conference on Machine Learning (ICML '04), ACM, New York, NY, USA, 2004, pp. 33–41.
[18] I. Bou-Hamad, D. Larocque, H. Ben-Ameur, A review of survival trees, Stat. Surv. 5 (2011) 44–71.
[19] A. Eyal, L. Rokach, M. Kalech, O. Amir, R. Chougule, R. Vaidyanathan, K. Pattada, Survival analysis of automobile components using mutually exclusive forests, IEEE Trans. Syst. Man Cybernet.: Syst. 44 (2) (2014) 246–253.
[20] H. Blockeel, L. De Raedt, J. Ramon, Top-down induction of clustering trees, in: Proceedings of the 15th International Conference on Machine Learning, Morgan Kaufmann, 1998, pp. 55–63.
[21] F. Moosmann, E. Nowak, F. Jurie, Randomized clustering forests for image classification, IEEE Trans. Pattern Anal. Mach. Intell. 30 (9) (2008) 1632–1646.
[22] M. Dror, A. Shabtai, L. Rokach, Y. Elovici, OCCT: a one-class clustering tree for implementing one-to-many data linkage, IEEE Trans. Knowl. Data Eng. 26 (3) (2014) 682–697.
[23] A. Gershman, A. Meisels, K.H. Lüke, L. Rokach, A. Schclar, A. Sturm, A Decision Tree Based Recommender System, in: IICS, 2010, pp. 170–179.
[24] L. Rokach, S. Kisilevich, Initial profile generation in recommender systems using pairwise comparison, IEEE Trans. Syst. Man Cybernet. Part C: Appl. Rev. 42 (6) (2012) 1854–1859.
[25] F. Xia, W. Zhang, F. Li, Y. Yang, Ranking with decision tree, Knowl. Inform. Syst. 17 (3) (2008) 381–395.
[26] K.M. Svore, C.J. Burges, Large-scale learning to rank using boosted decision trees, in: Scaling Up Machine Learning, Cambridge U. Press, 2011.
[27] Y. Freund, R. Iyer, R.E. Schapire, Y. Singer, An efficient boosting algorithm for combining preferences, J. Mach. Learn. Res. 4 (2003) 933–969.
[28] L. Antwarg, L. Rokach, B. Shapira, Attribute-driven hidden markov model trees for intention prediction, IEEE Trans. Syst. Man Cybernet. Part C 42 (6) (2012) 1103–1119.
[29] R. Polikar, Ensemble based systems in decision making, IEEE Circ. Syst. Mag. 6 (3) (2006) 21–45.
[30] J.W. Tukey, Exploratory Data Analysis, Addison-Wesley, Reading, Mass, 1977.
[31] B.V. Dasarathy, B.V. Sheela, Composite classifier system design: concepts and methodology, Proc. IEEE 67 (5) (1979) 708–713.
[32] L.A. Clark, D. Pergibon, Tree-based models, in: J.M. Chambers, T.J. Hastie (Eds.), Statistical Models in S, Wadsworth & Brooks, Pacific Grove, CA, 1992.
[33] Y. Freund, R.E. Schapire, Experiments with a new boosting algorithm, in: Machine Learning: Proceedings of the Thirteenth International Conference, 1996, pp. 325–332.
[34] N.C. Oza, K. Tumer, Classifier ensembles: select real-world applications, Inform. Fusion 9 (1) (2008) 4–20.
[35] T.G. Dietterich, Ensemble learning, in: M.A. Arbib (Ed.), The Handbook of Brain Theory and Neural Networks, second ed., The MIT Press, Cambridge, 2002.

[36] T.G. Dietterich, E.B. Kong, Machine Learning Bias, Statistical Bias, and Statistical Variance of Decision Tree Algorithms, Tech. Rep., Oregon State University, 1995.
[37] D.A. Cieslak, N.V. Chawla, Learning decision trees for unbalanced data, in: Machine Learning and Knowledge Discovery in Databases, Springer, Berlin, Heidelberg, 2008, pp. 241–256.
[38] V. Nikulin, G. McLachlan, S. Ng, Ensemble approach for classification of imbalanced data, in: Proceedings of the 22nd Australian Joint Conference on Advances in Artificial Intelligence, Springer-Verlag, 2009.
[39] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, F. Herrera, A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches, IEEE Trans. Syst. Man Cybernet. Part C: Appl. Rev. 42 (4) (2012) 463–484.
[40] M. Galar, A. Fernández, E. Barrenechea, F. Herrera, Eusboost: enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling, Pattern Recogn. 46 (12) (2013) 3460–3471.
[41] J. Gama, R. Fernandes, R. Rocha, Decision trees for mining data streams, Intell. Data Anal. 10 (1) (2006) 23–45.
[42] J.Z. Kolter, M. Maloof, Dynamic weighted majority: a new ensemble method for tracking concept drift, in: Third IEEE International Conference on Data Mining, 2003, ICDM 2003, 2003, pp. 123–130.
[43] L.L. Minku, A.P. White, X. Yao, The impact of diversity on online ensemble learning in the presence of concept drift, IEEE Trans. Knowl. Data Eng. 22 (5) (2010) 730–742.
[44] L.L. Minku, X. Yao, DDD: a new ensemble approach for dealing with concept drift, IEEE Trans. Knowl. Data Eng. 24 (4) (2012) 619–633.
[45] R. Bryll, R. Gutierrez-Osuna, F. Quek, Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets, Pattern Recogn. 36 (6) (2003) 1291–1302.
[46] J. Huang, H. Fang, X. Fan, Decision forest for classification of gene expression data, Comput. Biol. Med. 40 (8) (2010) 698–704.
[47] L. Rokach, Genetic algorithm-based feature set partitioning for classification problems, Pattern Recogn. 41 (5) (2008) 1676–1700.
[48] O. Maimon, L. Rokach, Data mining by attribute decomposition with semiconductor manufacturing case study, in: Data Mining for Design and Manufacturing, Springer, US, 2001, pp. 311–336.
[49] C. Vens, J. Struyf, L. Schietgat, S. Džeroski, H. Blockeel, Decision trees for hierarchical multi-label classification, Mach. Learn. 73 (2) (2008) 185–214.
[50] H. Deng, G. Runger, E. Tuv, M. Vladimir, A time series forest for classification and feature extraction, Inform. Sci. 239 (2013) 142–153.
[51] T. Mitchell, The Need for Biases in Learning Generalizations, Technical Report CBM-TR-117, Rutgers University, Department of Computer Science, New Brunswick, NJ, 1980.
[52] Y. Bi, The impact of diversity on the accuracy of evidential classifier ensembles, Int. J. Approx. Reason. 53 (4) (2012) 584–607.
[53] K.M. Ali, M.J. Pazzani, On the link between error correlation and error reduction in decision tree ensembles, Information and Computer Science, University of California, Irvine, 1995, pp. 95–38.
[54] P.K. Chan, S.J. Stolfo, A comparative evaluation of voting and meta-learning on partitioned data, in: Proceedings of the 12th International Conference on Machine Learning ICML-95, 1995.
[55] G. Brown, J.L. Wyatt, P. Tivno, Managing diversity in regression ensembles, J. Mach. Learn. Res. 6 (2005) 1621–1650.
[56] S.W. Lin, S.C. Chen, Parameter determination and feature selection for C4.5 algorithm using scatter search approach, Soft Comput. 16 (1) (2012) 63–75.
[57] N.V. Chawla, L.O. Hall, K.W. Bowyer, W.P. Kegelmeyer, Learning ensembles from bites: a scalable and accurate approach, J. Mach. Learn. Res. Arch. 5 (2004) 421–451.
[58] K.M. Ting, J.R. Wells, S.C. Tan, S.W. Teng, G.I. Webb, Feature-subspace aggregating: ensembles for stable and unstable learners, Mach. Learn. 82 (3) (2011) 375–397.
[59] M. Gashler, C. Giraud-Carrier, T. Martinez, Decision tree ensemble: small heterogeneous is better than large homogeneous, in: IEEE Seventh International Conference on Machine Learning and Applications, 2008, ICMLA'08, 2008, pp. 900–905.
[60] C.X. Zhang, J.S. Zhang, RotBoost: a technique for combining Rotation Forest and AdaBoost, Pattern Recogn. Lett. 29 (10) (2008) 1524–1536.
[61] S. Bernard, L. Heutte, S. Adam, Dynamic random forests, Pattern Recogn. Lett. 33 (12) (2012) 1580–1586.
[62] M.F.A. Hady, F. Schwenker, G. Palm, Semi-supervised learning for tree-structured ensembles of RBF networks with co-training, Neural Networks 23 (4) (2010) 497–509.
[63] P. Clark, R. Boswell, Rule induction with CN2: some recent improvements, in: Proceedings of the European Working Session on Learning, Pitman, 1991, pp. 151–163.
[64] D.W. Opitz, J.W. Shavlik, Actively searching for an effective neural network ensemble, Connect. Sci. 8 (3–4) (1996) 337–354.
[65] W. Buntine, A Theory of Learning Classification Rules, Doctoral Dissertation, School of Computing Science, University of Technology, Sydney. Australia, 1990.
[66] R.E. Shapire, Y. Singer, Improved boosting algorithms including confidence-rated predictions, Mach. Learn. 37 (1999) 297–336.
[67] P. Derbeko, R. El-Yaniv, R. Meir, Variance optimized bagging, in: European Conference on Machine Learning, 2002.
[68] A.K. Seewald, Towards Understanding Stacking, PhD Thesis, Vienna University of Technology, 2003.
[69] E. Menahem, L. Rokach, Y. Elovici, Troika–an improved stacking schema for classification tasks, Inform. Sci. 179 (24) (2009) 4097–4122.
[70] A.K. Seewald, J. Faurnkranz, Grading Classifiers, Austrian Research Institute for Artificial intelligence, 2001.
[71] A. Omari, A.R. Figueiras-Vidal, Post-aggregation of classifier ensembles, Inform. Fusion 26 (2015) 96–102.
[72] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.
[73] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[74] T.K. Ho, Random decision forests, in: Proceedings of the Third International Conference on Document Analysis and Recognition, vol. 1, 1995, pp. 278–282.
[75] Y. Amit, D. Geman, Randomized Inquiries About Shape: An Application to Handwritten Digit Recognition (No. TR-401). CHICAGO UNIV IL DEPT OF STATISTICS, 1994.
[76] T.G. Dietterich, An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization, Mach. Learn. 40 (2) (2000) 139–157.
[77] A. Saffari, C. Leistner, J. Santner, M. Godec, H. Bischof, On-line random forests, in: 2009 IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops), 2009, pp. 1393–1400.
[78] T.K. Ho, The random subspace method for constructing decision forests, IEEE Trans. Pattern Anal. Mach. Intell. 20 (8) (1998) 832–844.
[79] C. Kamath, E. Cantu-Paz, Creating ensembles of decision trees through sampling, in: Proceedings, 33-rd Symposium on the Interface of Computing Science and Statistics, Costa Mesa, CA, June 2001.
[80] C. Kamath, E. Cantu-Paz, D. Littau, Approximate splitting for ensembles of trees using histograms, in: Second SIAM International Conference on Data Mining (SDM-2002), 2002.
[81] C. Désir, S. Bernard, C. Petitjean, L. Heutte, One class random forests, Pattern Recogn. 46 (12) (2013) 3490–3506.
[82] J.J. Rodriguez, L.I. Kuncheva, C.J. Alonso, Rotation forest: a new classifier ensemble method, IEEE Trans. Pattern Anal. Mach. Intell. 28 (10) (2006) 1619–1630.
[83] A. Schclar, L. Rokach, Random projection ensemble classifiers, in: Enterprise Information Systems, Springer, Berlin, Heidelberg, 2009, pp. 309–316.
[84] J.H. Friedman, Greedy function approximation: a gradient boosting machine, Ann. Stat. (2001) 1189–1232.
[85] P. Li, Abc-boost: adaptive base class boost for multi-class classification, in: ACM Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 625–632.
[86] G. Martínez-Muñoz, A. Suárez, Using all data to generate decision tree ensembles, IEEE Trans. Syst. Man Cyber. C 34 (4) (2004) 393–397.
[87] L. Breiman, Randomizing outputs to increase prediction accuracy, Mach. Learn. 40 (3) (2000) 229–242.
[88] G. Martínez-Muñoz, A. Suárez, Switching class labels to generate classification ensembles, Pattern Recogn. 38 (10) (2005) 1483–1494.
[89] E. Bauer, R. Kohavi, An empirical comparison of voting classification algorithms: bagging, boosting, and variants, Mach. Learn. 35 (1999) 1–38.
[90] D. Opitz, R. Maclin, Popular ensemble methods: an empirical study, J. Artif. Res. 11 (1999) 169–198.
[91] D. Villalba Santiago, J. Rodrıguez Juan, J. Alonso Carlos, An empirical comparison of boosting methods via OAIDTB, an extensible java class library, in: II International Workshop on Practical Applications of Agents and Multiagent Systems – IWPAAMS'2003.
[92] R.E. Banfield, L.O. Hall, K.W. Bowyer, W.P. Kegelmeyer, A comparison of decision tree ensemble creation techniques, IEEE Trans. Pattern Anal. Mach. Intell. 29 (1) (2007) 173–180.
[93] J. Shafer, R. Agrawal, M. Mehta, SPRINT: a scalable parallel classifier for data mining, in: Proceedings of 1996 International Conference Very Large Data Bases, 1996, pp. 544–555.
[94] M. Mehta, R. Agrawal, J. Rissanen, SLIQ: a fast scalable classifier for data mining, in: Advances in Database Technology—EDBT'96, Springer, Berlin, Heidelberg, 1996, pp. 18–32.
[95] P. He, L. Chen, X.H. Xu, Fast C4.5, in: 2007 International Conference on IEEE Machine Learning and Cybernetics, vol. 5, 2007, pp. 2841–2846.
[96] B. Panda, J.S. Herbach, S. Basu, R.J. Bayardo, Planet: massively parallel learning of tree ensembles with mapreduce, Proc. VLDB Endowment 2 (2) (2009) 1426–1437.
[97] J. Ye, J.H. Chow, J. Chen, Z. Zheng, Stochastic gradient boosted distributed decision trees, in: Proceedings of the 18th ACM Conference on Information and Knowledge Management, 2009, pp. 2061–2064.
[98] I. Palit, C.K. Reddy, Scalable and parallel boosting with mapreduce, IEEE Trans. Knowl. Data Eng. 24 (10) (2012) 1904–1916.
[99] S. del Río, V. López, J.M. Benítez, F. Herrera, On the use of MapReduce for imbalanced big data using Random Forest, Inform. Sci. 285 (2014) 112–137.
[100] G. Tsoumakas, I. Partalas, I. Vlahavas, A taxonomy and short review of ensemble selection, in: Workshop on Supervised and Unsupervised Ensemble Methods and Their Applications, 2008.
[101] D. Margineantu, T. Dietterich, Pruning adaptive boosting, in: Proceedings of the Fourteenth International Conference Machine Learning, 1997, pp. 211–218.
[102] C. Tamon, J. Xiang, On the boosting pruning problem, in: Proceedings of the 11th European Conference on Machine Learning, 2000, pp. 404–412.
[103] Z.H. Zhou, J. Wu, W. Tang, Ensembling neural networks: many could be better than all, Artif. Intell. 137 (2002) 239–263.

[104] D. Hernández-Lobato, G. Martinez-Muoz, A. Suárez, Statistical instance-based pruning in ensembles of independent classifiers, IEEE Trans. Pattern Anal. Mach. Intell. 31 (2) (2009) 364–369.
[105] S.H. Park, J. Furnkranz, Efficient prediction algorithms for binary decomposition techniques, Data Min. Knowl. Disc. (2012) 1–38.
[106] G. Martinez-Muoz, D. Hernandez-Lobato, A. Suarez, An analysis of ensemble pruning techniques based on ordered aggregation, IEEE Trans. Pattern Anal. Mach. Intell. 31 (2) (2009) 245–259.
[107] A.L. Prodromidis, S.J. Stolfo, P.K. Chan, Effective and Efficient Pruning of Metaclassifiers in a Distributed Data Mining System, Technical Report CUCS-017-99, Columbia University, 1999.
[108] R. Caruana, A. Niculescu-Mizil, G. Crew, A. Ksikes, Ensemble selection from libraries of models, in: Twenty-First International Conference on Machine Learning, July 04–08, 2004, Banff, Alberta, Canada, 2004.
[109] Q. Hu, D. Yu, Z. Xie, X. Li, EROS: ensemble rough subspaces, Pattern Recogn. 40 (2007) 3728–3739.
[110] R.E. Banfield, L.O. Hall, K.W. Bowyer, W.P. Kegelmeyer, Ensemble diversity measures and their application to thinning, Inform. Fusion 6 (1) (2005) 49–62.
[111] I. Partalas, G. Tsoumakas, I. Vlahavas, An ensemble uncertainty aware measure for directed hill climbing ensemble pruning, Mach. Learn. 81 (3) (2010) 257–282.
[112] L. Rokach, Collective-agreement-based pruning of ensembles, Comput. Stat. Data Anal. 53 (4) (2009) 1015–1026.
[113] H. Zhang, M. Wang, Search for the smallest random forest, Stat. Interface 2 (2009) 381–388.
[114] Z.H. Zhou, W. Tang, Selective ensemble of decision trees, in: Guoyin Wang, Qing Liu, Yiyu Yao, Andrzej Skowron (Eds.), Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing, 9th International Conference, RSFDGrC, Chongqing, China, Proceedings, Lecture Notes in Computer Science, vol. 2639, 2003, pp. 476–483.
[115] L. Rokach, O. Maimon, R. Arbel, Selective voting—getting more for less in sensor fusion, Int. J. Pattern Recogn. Artif. Intell. 20 (03) (2006) 329–350.
[116] A.L. Prodromidis, S.J. Stolfo, Cost complexity-based pruning of ensemble classifiers, Knowl. Inform. Syst. 3 (4) (2001) 449–469.
[117] T. Windeatt, G. Ardeshir, An Empirical Comparison of Pruning Methods for Ensemble Classifiers, IDA2001, LNCS, vol. 2189, 2001, pp. 208–217.
[118] Y. Zhang, S. Burer, W.N. Street, Ensemble pruning via semi-definite programming, J. Mach. Learn. Res. 7 (2006) 1315–1338.