Random Forests: Introduction, Parts, Variants

Tobias Ammann

Literature Study at the Workgroup for Psychological Methods,

Evaluation and Statistics, Department of Psychology.

Supervised by Prof. Dr. Carolin Strobl

June 27, 2013

Author Note

Tobias Ammann

Alte Landstrasse 39

8802 Kilchberg

tag@adnm.ch

Abstract

This report discusses a family of machine learning algorithms called random forests, their structure, their problems, and a number of suggested improvements, in order to give the reader a general introduction, starting with the motivation for machine learning, and the statistical as well as historical background to machine learning and random forests. The aim of this work is to introduce the different parts of random forest algorithms by giving an overview over their limitations and existing suggestions on overcoming these limitations. These components are illustrated with source code snippets and comparisons between various variants of the random forest algorithms are being made. This report also puts forward a few ideas on how to make random forests better and more user-friendly.

Keywords: bagging, boosting, decision trees, ensemble methods, error estimates, fuzzy random forests, introduction, optimality criterion, out-of-bag data, overfitting, PERT, random forests, regression, rotation forests, statistics, variable importance

Random Forests: Introduction, Parts, Variants

Contents

Declaration of Independence

"Hiermit versichere ich, dass ich die vorgelegte Arbeit selbständig und ohne unerlaubte Hilfsmittel verfasst habe. Andere als die angegebenen Hilfsmittel habe ich nicht verwendet."

Tobias Ammann

Date: June 27, 2013

Introduction

Motivation

Psychology has come a long way as a science, with psychological research more and more following the *scientific method*. Research works by the assumption that if we disprove just enough alternative theories, we can eventually tell which theory is probably true, despite positive proof being impossible without complete knowledge. So, the scientific method really is nothing but the use of countless attempts to disprove alternative theories, until only a single such theory remains.

Since there is a large number of theories to disprove, and every researcher likes to see the results of their work as soon as possible, a more speedy method is usually employed, although this comes with a caveat. The speedier method makes the researcher compare his favoured theory against the null hypothesis to show the superiority of his favoured theory. The null hypothesis is either an alternative theory or a chance effect. This is way more efficient than comparing the thousands of theories that researchers have come up with, and continue to come up with. The caveat is commonly called confirmation bias, because researchers focus on immunising their theory instead of disproving those of others. Therefore, the result only has significance in the experimental set-up being tested. On the greater scale of things, i.e. reality, the results might well be completely bogus. Nonetheless, most of the subjects being studied are sufficiently constant or change predictably enough to allow researchers to generalise from the results of an experiment to the world at large while likely remaining correct. This likelihood depends on the size of the effects measured in the experiment, the amount of experimental data collected, and on the properties of the statistical methods involved. In psychological research, where large effects are rare and experiments often only study a handful of psychology students, who notwithstanding research ethics are often forced to participate, it is vital to use powerful statistical methods, because this is the only parameter remaining for researchers to tweak in their favour.

Recently, researchers in psychology began to turn to a new breed of statistical methods, in hope of ever better results. This new breed of statistical methods is called

machine learning.

In this paper, I aim to introduce the reader to *random forests*, which are just one family of machine learning algorithms. I first give the reader some context to random forests in hope that a broader view will help in developing an intuition for random forests.

In the next sections I introduce the reader to machine learning, the differences between the traditional statistical approach and the machine learning approach, and random forests in particular. I then delve into some improvements to random forests that have been suggested in the literature, call for a more user-friendly random forest variant, and finish with a few words on the method of writing this paper, and a discussion.

Machine Learning

In order to understand random forests, it helps to set the stage by briefly discussing machine learning in general. Machine Learning is both a part of predictive statistics and the artificial intelligence branch of computer science.

Predictive statistics is the sub-field of statistics that is concerned with making predictions based on past observations. Its probably most widely known method is *linear regression* that associates two variables y and x in such a way that they describe a straight line: $y = \alpha \cdot x + \beta$, but linear regression is not limited to two variables (Bortz & Schuster, 2010). Linear regression is widely used in psychology because it allows the researcher to tease apart the influence of a number of variables on an outcome variable by making assumptions of the form $Y = \alpha + \beta \cdot X + \epsilon$, where Y is a vector of the outcomes for all participants, α is a constant, β is a vector of weights, X is a matrix with the values of each influence variable for every participant, and ϵ is a vector of random differences. Sometimes this is written as an equation for a single participant: $y_i = \beta_0 + \beta_1 \cdot x_{i1} + \cdots + \beta_n \cdot x_{in} + \epsilon$. This is a standard approach in personality questionnaires.

Artificial intelligence is a field commonly associated with the computer sciences,

where it began with the advent of higher-order programming languages based on mathematical foundations around the *1960s* (Wikipedia, 2013k). Its ultimate aim is to give computers human-like capabilities, so that they can assist us with human intelligence, better rationality and knowledge that surpasses that of every human being. It includes things like *logic programming* (Wikipedia, 2013l; Nilsson, 2009), *expert systems* (Wikipedia, 2013f; Nilsson, 2009), *database management systems* (Wikipedia, 2013d) and *neural networks* (Haykin, 2007; Nilsson, 2009), which all represent some form of storing and querying knowledge. Unfortunately, early computers back then did not have the speed and memory required for artificial intelligence to solve many real problems. This and overly optimistic predictions led to disappointment and the abandonment of AI research, and the field has been largely dormant since, a state widely called *AI winter* (Nilsson, 2009; Wikipedia, 2013j). Only the rather recent coexistence of powerful computers and massive amounts of stored data, sometimes called *big data*, revived artificial intelligence as an important field of research. In fact, the revival and spread of machine learning has made its application commonplace enough that I could listen to a banker telling a friend how he uses random forests for algorithmic Bitcoin trading in his leisure time in Starbucks.

Machine learning is the part of artificial intelligence which is concerned with the computer's learning of facts about the world. These facts can then be stored and queried later on. As such, machine learning is concerned with making statements based on past observations and is therefore close to predictive statistics (Wikipedia, 2013m).

The Machine Learning Life Cycle

This section discusses the difference between machine learning and traditional statistical methods, as well as commonly used terminology. The following section relies on knowledge gained from personal projects a few years back, a university course (Strobl, 2013), as well as Barber (2013) and Wikipedia (2013m).

Authors differ in how they call the different parts of a dataset. The part of a dataset that is being derived or partitioned is commonly called *outcome*, *output*, *output*

variable, *output data*, *class*, or *response*. Some authors also use names that reflect the type of data represented, e.g. *likelihood of smoking* or *cancer risk*. Common names for the parts used to derive output variables are either generic names, such as *variables*, *input variables*, *input*, *input data*, *features*, or problem specific names, such as *median income* or *skin conductance*. This paper will try to be consistent in the names being used. It assumes datasets to be in table form, with columns representing variables and rows representing measurements or participants. The paper uses the word *variable* to refer to columns, and *data* to refer to the whole or parts of the dataset, i.e. rows and columns. In most cases these two terms are interchangeable, but sometimes one is more illustrative or more commonly used in a particular context.

Traditionally, the statistical methods used in psychology take a model (or a family of models) of how the world works, a set of data, and return one of the following things. They either return a probability of how likely an improvement in prediction can be observed at random, or how likely a difference in measurement can be observed at random, or an estimate of what might be observed in different data, or, building on the first two, the model in a family of models that best fits the data. Notice, that these methods do not derive the model on their own.

Regression methods try to derive the values of one variable in the dataset from the other variables in the dataset using a formula the researcher specifies. They then compare the actual values and the prediction of the model under different assumptions, and calculate how probable an improvement in this comparison is to show up due to random variations. These calculations are called *tests*, and can be calculated for the entire model, as well as for individual coefficients. (Wikipedia, 2013q)

Analysis of variance methods partition the dataset according to all but one variable. They then calculate the probability with which the variation in the one variable among the groups could be due to random variations in the dataset. The aim is of course to show that a difference in the values of the outcome variable is unlikely due to chance, and hence dependent on certain input variables (Wikipedia, 2013a).

The probabilities that the methods output are usually called p. Statisticians

usually define a target *significance level*, e.g. 5%, and compare it with p. If p is less than the targeted significance level, then the outcomes predicted by the model and the outcomes predicted by chance differ significantly. For example, a model with $p \leq 5\%$ is better than chance and its predictions will only show up by chance in 5% of all experiments. However, it is important to distinguish between p and $\hat{p}$. The first denotes the probability found in the dataset, while the latter indicates an estimation for p. In most cases the dataset represents a sample of a population and not the population itself, thus the error probability can only be estimated, hence $\hat{p}$.

The most striking difference between traditional statistical methods and machine learning methods is that the researcher cannot specify their model of how the world works, other than through the selection of a machine learning algorithm. Because of this, machine learning algorithms are sometimes described as *black boxes*, meaning that the user can only see what is going into the algorithm, and what is coming out. This is unlike the statistical methods, where the researcher supplies a formula or model. In machine learning, algorithms derive the formula on their own. This is what the "learning" in machine learning means. The second difference is what the algorithms return. Since machine learning algorithms build their models, they cannot just output coefficients and probabilities for a given model, but also the model itself. In short, machine learning provides the researcher with a generic model that adapts to the world. The predictions of such a model can then be calculated for data for which the values of the output variable are known, but which have not been included during the *learning phase*, to calculate the *prediction accuracy*. The problem with this flexibility is, that one cannot really tell what the model looks like, that is, the model is not in a human-readable form. As will be pointed out later, decision trees, the underlying mechanism in random forests, are quite easily understandable, but random forests consist of dozens to thousands of such trees, so that a human can hardly tell what they mean. It is therefore very important to find ways to condense this complexity into something that can be more easily interpreted. The *variable importance measure* of random forests, which will be introduced later, is one such way.

Classification and Regression

The *classification problem* is the problem of classifying new data, i.e. grouping it based on some criteria. It is called a "problem", because in statistics and machine learning the rules to do so are unknown and have to be inferred from an example dataset. The *regression problem* is the related problem, where an *output variable* has to be calculated based on *input variables*. For practical purposes, the difference is just the type of the output variable. Classification usually uses linguistic labels, e.g. "high", "medium" and "low, while regression returns a numeric value of arbitrary precision. One might describe classification as regression with discreet output values, and regression as classification with an infinite number of labels. Output variables are commonly called *classes*, while input variables are called *features*, as has already been discussed above, but other descriptions are also common, sometimes depending on what the dataset represents (Strobl, Malley, & Tutz, 2009).

Randomness

Many machine learning algorithms consume random numbers in different places. While computer generated random numbers are not truly random, they still cause the outcome of the algorithm to change slightly between different executions, due to the fact that random number generators are customarily initialised with the current time at the start of the program. These changes might unnerve a novice, but do not change the outcome of the algorithm systematically, as long as the algorithm is used correctly. Still, for publication purposes it can make sense to set and publish the random number generator's *seed*, i.e. the value the generator is being initialised with. However, doing so during the experiment is a serious mistake. Indeed, it is good practice to run the analysis multiple times to ensure that these random variations do not change the outcome unexpectedly.

Random Forests

Introduction

This part of the paper discusses random forests. "Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest" (Breiman, 2001). Leo Breiman developed random forests with Adele Cutler, building on work by Ho, Amit, Geman, and Dietterich (Wikipedia, 2013p).

Although the name random forests is usually taken to refer to the random forests as defined by Breiman (2001), the large number of variants that have been derived from the original forests, e.g. *Forest-rk* (Bernard, Heutte, & Adam, 2008), *RFW* (Maudes, Rodríguez, García-Osorio, & García-Pedrajas, 2012), *DRF* (Bernard, Adam, & Heutte, 2012), *Fuzzy random forests* (Bonissone, Cadenas, Garrido, & Dıaz-Valladares, 2008), *Rotation forests* (Rodriguez, Kuncheva, & Alonso, 2006), random forests that are more random, e.g. Geurts, Ernst, and Wehenkel (2006), Liu, Ting, and Fan (2005), Cutler and Zhao (2001), and various other improvements by Banfield, Hall, Bowyer, and Kegelmeyer (2007); Robnik-Šikonja (2004); Strobl et al. (2009); G. Zhang and Lu (2012), show that it is better to think of random forests as being a framework instead of being a single method (Wikipedia, 2013p). To understand this framework, it is best to look at the different aspects of random forests, first in a top-down view, and later part by part. The top-down view is based on Breiman (2001), while the part by part discussion will also go into tweaking random forests, which is where the work of other authors will be discussed.

Random forests are *ensemble learning* methods where the ensemble consists of *decision trees*. Every decision tree is constructed on a sample of the input dataset, that has been drawn using bootstrapping with replacement from the original dataset and is equally large. Every node split in the decision tree is an optimal two-way split selected from a random subset of all input variables. The number of randomly selected variables for each split is commonly called `mtry`. If the number of input variables is small, additional input variables can be derived as linear combinations of input variables. The

decision trees are grown maximally without pruning, and new trees are generated until the ensemble of decision trees reaches its target size, usually called `ntree`. Random forests feature error estimates using out-of-bag data. Out-of-bag data are the rows in the dataset that were not included during the bootstrap aggregation, and make up approximately one third of the dataset. Random forests also feature variable importance measures, which are calculated by reclassifying the out-of-bag data, but shuffling the variable under consideration. The variable importance of the shuffled variable is the increase in prediction error.

The reference implementation of random forests is written in Fortran, but a package for the statistical software framework R (R Core Team, 2012), exists under the name `randomForest` (Liaw & Wiener, 2002). An alternative within the R framework, which includes improvements to correct a bias found in variable importance measures is available under the name `party` (Strobl, Boulesteix, Kneib, Augustin, & Zeileis, 2008). A third implementation using Java is available in the WEKA machine learning suite (Hall et al., 2009). For illustration purposes I include a source code example taken from Strobl et al. (2008):

```
load("dat_genes.rda")
library("party")
mycontrols <- cforest_unbiased(ntree=1000, mtry=20, minsplit=5)
mycforest <- cforest(status ~ ., data=dat_genes, controls=mycontrols)
```

Figure 1. R example using the party package.

Random forests are suitable for a wide range of applications. Random forests have been used in applications from psychology and computational biology as is outlined in Strobl et al. (2009), to customer churn prediction (Xie, Li, Ngai, & Ying, 2009), to software testing (Guo, Ma, Cukic, & Singh, 2004), to internet security (J. Zhang & Zulkernine, 2005), and to natural language applications (Xu & Jelinek, 2004; Kobyliński & Przepiórkowski, 2008). The reasons why random forests are such a widely used method, are their prediction accuracy, which is comparable to other state-of-the-art

machine learning algorithms like Adaboost as demonstrated by Breiman (2001), their ability to handle "small n large p" datasets (Strobl et al., 2009), their practical built-in error estimate, and their variable importance measures. The last of which, random forests' variable importance measures, might be their most useful feature. Many domains do not require accurate predictions as much as a model that can be understood by humans. While ensemble methods are unsuitably complex, random forests' variable importance measures can be used to select variables for use in simpler models, e.g. generalised linear models such as logit and probit models, which are more easily interpreted (Strobl et al., 2009).

Ensemble Learning Methods

As mentioned above, random forests build on quite a rich collection of previous work, most of which also concerns ensemble learning methods. In fact, random forests can be seen as a composition of elements from different other ensemble learning methods, e.g. random subspaces (Ho, 1998) and bagging (Breiman, 1996). Although detailed knowledge of these methods is not a requirement to understand random forests, it is important to understand what ensemble learning is.

Ensemble learning is a supervised learning algorithm: its task is to take input and output data, and find a hypothesis that connects the two. This hypothesis can then be used to predict the output data that corresponds to new input data. In theory this problem can be solved by a construct called Bayes optimal classifier, which considers every possible hypothesis, but which unfortunately cannot be implemented except for trivial problems. The principle behind this is, that the combination of hypotheses becomes a stronger hypothesis, because it can represent more functions than every component hypothesis alone could. In short, ensemble methods rely on the principle that the ensemble is more than the sum of its parts. The usual wording of this is, that ensemble learning turns a set of *weak classifiers* into one *strong classifier*. Weak also stands for unstable, meaning that the underlying classifier is susceptible to even small variations in the input data. Ensemble learning methods can be called meta algorithms,

because they rely on other simpler classifier algorithms, and it is theoretically possible to create an ensemble learner for any supervised learning algorithm. (Wikipedia, 2013e; Polikar, 2009)

Bagging. The ensemble learning algorithm that most prominently underlies random forests is bagging. Bagging stands for bootstrap aggregation. Bootstrapping is a term that was used to describe "[...] the process by which lumberjacks hoist themselves up trees [...]" (Wikipedia, 2013b). In statistics it refers to the process of deriving additional samples by resampling the original sample, essentially simulating drawing additional samples from the population. Bootstrap aggregation is an ensemble learning algorithm, which trains each of its underlying weak classifiers on a different set of input data created by bootstrapping. It is another algorithm developed by Breiman (1996).

Bagging typically uses bootstrapping with replacement, which leads to the inclusion of on average two thirds of the original sample in the derived sample. The remaining third is then used as out-of-bag data. This is a common feature of bagging and random forests (Breiman, 2001).

Boosting. Boosting is an ensemble method, where every weak classifiers gets to train on the original sample, but the sample is extended with importance weights. These weights are different for every instance of the weak classifier. They are lower for rows in the dataset that are correctly predicted by the classifiers already in the ensemble, and higher for rows that are classified wrongly. In short, boosting focuses on eliminating one classification error after another, until all data is being classified correctly or a targeted ensemble size is reached. This behavior makes boosting algorithms very fast learners, but susceptible to errors in the dataset (Long & Servedio, 2010). Boosting is not used in random forests as introduced by Breiman (2001), but there are variants that do use boosting, e.g. dynamic random forests (Bernard et al., 2012).

Adaboost (Freund & Schapire, 1995) is probably the most popular boosting algorithm, and the algorithm random forests are most commonly compared to in terms of performance, e.g. Breiman (2001), Banfield et al. (2007) and Rodriguez et al. (2006). The second study compared eight ensemble of decision tree classifiers in a large study

on 57 publicly available datasets and concluded "[...] that boosting, random forests and randomised trees are statistically significantly better than bagging."

Decision Trees

The underlying algorithm of random forests is the decision tree classifier, which can be either a regression tree, or a classification tree. The difference between the two types of trees is the type of output produced. Classifications trees work with discreet output values while regression trees output a continuous numeric value. For classification the most commonly chosen output value among an ensemble of decision trees is assumed to be the right prediction. For regression purposes the output of the ensemble is calculated by averaging the outcomes of all trees. As the name indicates, a decision tree is a tree data structure, i.e. a set of nodes that are connected in a forward manner allowing branching but not cross links and circles. Decision trees usually are binary trees, meaning that each node either has two child nodes, or is a leaf node. Each node represents a decision criterion, each edge a criterion match or a mismatch, and each leaf node represents an outcome of the decision tree. Tree data structures are typically drawn from top to bottom. For example, the following decision tree takes two numeric features `a`, and `b` to predict two values of a discreet output variable called `class`.

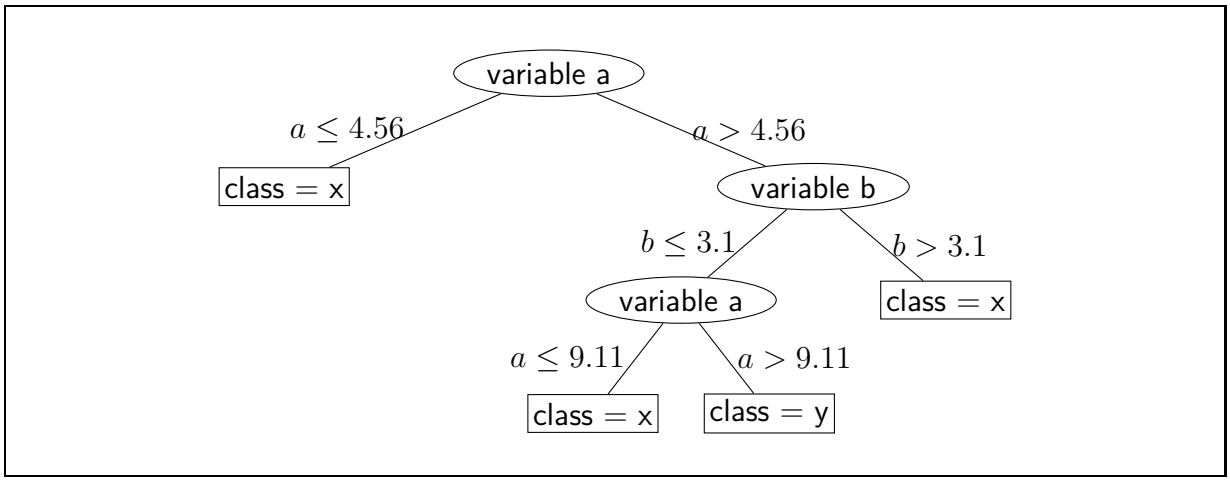

Figure 2. A small decision tree.

The decision trees used in random forests are constructed by randomly selecting a

certain number of input variables as configured by the parameter `mtry` for each node. The algorithm then selects the best split on one of these input variables as the node's criterion, using what is called an optimality criterion. This process is repeated for every child node until a node is reached which satisfies a stop criterion, e.g. only matches rows in the dataset with equal output values. This node becomes a leaf node. This is described as growing a tree maximally. It ensures that each decision tree has a high strength, meaning no decision tree outputs wrong predictions for the subsets of data it matches. (Breiman, 2001)

As a component classifier to an ensemble learning algorithm, every decision tree should be different from the other decision trees in the ensemble, i.e. the decision trees should be diverse enough to cover a wide range of possible interactions between the input data, i.e. hypotheses. This is commonly described as the trees being uncorrelated. Tree strength and inter-tree correlation are what the prediction accuracy of random forests depends on. (Breiman, 2001)

The procedure by which decision trees are grown is modified in some variants of random forests. The PERT (perfect random tree) algorithm (Cutler & Zhao, 2001) for example does not perform a search for the best split on a random selection of input variables, but selects variable and threshold at random. The RFW (random feature weights) algorithm (Maudes et al., 2012) searches all variables but attaches random weights to them, and the DRF (dynamic random forests) algorithm (Bernard et al., 2012) influences the decision tree creation by a procedure inspired by boosting. Other variants, e.g. Van Essen, Macaraeg, Gokhale, and Prenger (2012), change this procedure to grow smaller or more balanced trees to meet performance or hardware requirements.

Random Forests

As has been described in the sections on ensemble learning and decision trees, random forests build ensembles of decision trees. This process depends on two parameters: the ensemble size, commonly called `ntree`, and the randomisation parameter `mtry`. Actually, random forests depend on another parameter, the state of a

random number generator, but being random, it does not and usually should not be specified.

The following is an example implementation of the random forest algorithm roughly following the original version by Breiman (2001). I wrote this implementation to illustrate the simplicity of random forests, as well as its components. The implementation uses entropy instead of the Gini index as the optimality criterion, a naive stop criterion only capable of handling perfectly partition-able data, and lacks variable importance measures. In reference to the higher level programming languages mentioned in the introduction, the following source code is written in a Lisp dialect called Clojure (Wikipedia, 2013c), and uses a functional data structure called *zipper* (Huet, 1997) for efficient tree construction.

For every tree in a random forest ensemble, the algorithm first draws a bootstrap sample from the dataset. The following `bootstrap` function takes a vector of rows as input and returns an equally large bootstrap sample and the corresponding out-of-bag data.

```
(defn bootstrap [dataset]
  (let [n (count dataset)
        s (repeatedly n #(rand-int n))
        d (set (map #(nth dataset %) s))
        o (set (filter (comp not d) dataset))]
    {:dataset d :oob o}))
```

Figure 3. The `bootstrap` function.

For each node in a tree the algorithm first has to select the rows in the dataset that match all criteria up to the given node. The following `select` function takes a vector of rows, and a tuple of a variable name and a split value as input, and returns lists of the matching and the not matching rows.

```
(defn select [[v s] dataset]
  (let [l (reduce (fn [acc t]
                    (if (<= (get t v) s) (conj acc t) acc))
            #{} dataset)
        r (set (filter (comp not l) dataset))]
    [l r]))
```

Figure 4. The `select` function.

For each node the algorithm then progresses to check whether the node should be split, and if so, selects the best split out of a random selection of split variables. The `splits` function takes a dataset and `mtry` as parameters and returns all splits as tuples with a variable name and a split value. The `best-split` function calculates the entropy for every split and applies the best split to two datasets at that node (sample data and out-of-bag data) simultaneously. This is done because both sample and out-of-bag data are stored in each node to simplify the decision tree construction and the error estimate calculation. Since Clojure features immutable data structures this increases memory usage only negligibly.

```
(defn splits [dataset mtry]
  (let [vs (filter (partial not= :class) (keys (first dataset)))
        n (count vs)
        vs (if (<= mtry n) (take mtry (shuffle vs)) vs)]
    (reduce (fn [acc v]
              (reduce (fn [acc t]
                        (conj acc [v (get t v)])) acc dataset))
            #{} vs)))

(defn best-split [dataset oob mtry]
  (let [[_ s l r]
    (first (sort (map (fn [s] (let [[l r] (select s dataset)]
                                [(entropy l) s l r]))
                      (splits dataset mtry))))]
    (let [[oobl oobr] (select s oob)]
      [s (set l) (set r) (set oobl) (set oobr)])))
```

Figure 5. The functions `splits` and `best-split`.

The `extend-node` function uses `leaf?` to check whether the node should be split,

does so if necessary using the `best-split` function, and hands down the appropriate datasets to the child nodes.

```
(defn leaf? [dataset] (= (entropy dataset) 0.0))

(defn extend-node [{:keys [dataset oob] :as node} mtry]
  (if (leaf? dataset)
    (merge node
           {:class (or (first (first (sort-by val >
                                      (frequencies (map :class dataset)))))
                       :unknown)})
    (let [[s l r oobl oobr] (best-split dataset oob mtry)]
      (merge node
             {:criterion s :left  {:dataset l :oob oobl}
                           :right {:dataset r :oob oobr}}))))
```

Figure 6. The stop criterion `leaf?` and the node splitting function `extend-node`.

The following `rf-train` function is the top-level function responsible for building the ensemble of decision trees. It creates `ntree` decision tree root nodes, each with its own bootstrapped sample and out-of-bag datasets, walks through each tree using a zipper, and invokes the `extend-node` for each node in every tree. The different trees in the ensemble are constructed in parallel using the Clojure built-in `pmap`, distributing the work among all available processor cores.

```
(defn rf-train [dataset ntree mtry]
  (set (pmap (fn [sample]
               (loop [loc (zip/zipper
                           (fn [node] true)
                           #(if (:left %)
                              (seq [(:left %) (:right %)]))
                           (fn [node children]
                             (with-meta
                               (merge node
                                      {:left  (first children)
                                       :right (second children)})
                               (meta node)))
                           sample)]
                 (if (zip/end? loc) (zip/root loc)
                   (recur
                     (zip/next (zip/edit loc extend-node mtry)))))
               (repeatedly ntree #(bootstrap dataset)))))
```

Figure 7. The `rf-train` function.

The `rf-predict` is another top-level function. It takes a random forest and a row of input values, and returns the outcome as predicted by the random forest. Note that part of this function also runs in parallel, distributing the work among all available processor cores. In fact, the ability to be parallelised is one of the advantages of random forests.

```
(defn rf-predict [forest features]
  (let [eval-tree
          (fn eval-tree [tree]
            (if (:class tree) (:class tree)
              (if (let [[v s] (:criterion tree)]
                    (<= (get features v) s))
                (eval-tree (:left tree))
                (eval-tree (:right tree)))))]
    (first (first
      (sort-by
        val > (frequencies
                (filter (partial not= :unknown)
                        (pmap eval-tree forest))))))))
```

Figure 8. The `rf-predict` function.

The following top-level function `rf-error` calculates the random forest's error estimate by invoking `tree-error` for each tree in the ensemble. The `tree-error` function uses the out-of-bag data stored in each leaf node and compares it to the predicted outcome of that leaf. Since this implementation only handles noiseless datasets, the error estimate returned by `rf-error` will only ever be zero.

```
(defn tree-error [tree]
  (let [cls (:class tree)]
    (cond (nil? cls) (+ (tree-error (:left tree))
                        (tree-error (:right tree)))
          (= cls :unknown) 0
          :else (count
                  (filter (partial = cls) (:oob tree))))))

(defn rf-error [forest]
  (/ (reduce + 0
       (pmap #(/ (tree-error %) (count (:oob %))) forest))
     (count forest)))
```

Figure 9. The `rf-error` and `tree-error` functions calculate the error estimate.

For information on how to use this implementation of random forests, see the example and the instructions in the appendix.

Parameters. The two parameters random forests depend on, the size of the forest `ntree` and the number of randomly selected variables at each node `mtry`, are problematic in that users set them according to one recommendation or another, or just use the default settings, which again can differ from implementation to implementation. Furthermore, not every dataset is best learned by random forests using the same settings. This lack of a standard way to determine the parameter values makes it hard to compare the accuracy of different methods. The comparative study by Banfield et al. (2007) gives an illustrative overview over the datasets used in previous publications, among which are Breiman (2001) and Dietterich (2000), that each used ensemble sizes of 50-200, and applied the algorithms being compared to 18-27 datasets. In each case the authors concluded that their method was superior to the other methods in the study. However, the more recent study by Banfield et al. (2007) found "no stat. sig.

improvement over bagging in 38 of 57 data sets" when ensemble sizes of up to 1000 trees where used. The study also concludes that bagging with an ensemble size of 1000 and random forests with the randomisation parameter set to the binary logarithm of the number of input variables were the best methods. The study by Banfield et al. (2007) also suggests a mechanism to determine the best size of the ensemble automatically. Methodologically, this would be a welcome improvement, because it eliminates one way researchers can fiddle with the outcome of their calculations, and because it would ensure better results due to the usually larger ensemble sizes. In defense of older studies, like those by Breiman (2001) and Dietterich (2000), one has to mention that computers back then were not as powerful. Breiman (2001) mentions run times for random forests of 4 minutes and 3 hours for Adaboost when building ensembles of size 100 while I can execute all examples in the paper by Strobl et al. (2009) in well under 10 seconds on one core running at 3.40 GHz, consuming about 180 Mb of memory.

The second parameter of random forests is the randomisation parameter `mtry` which controls how many variables are being selected randomly at each decision tree node to search for an optimal split. If `mtry` is set to the number of variables, random forest becomes bagging as proposed by Breiman (1996). Common choices for `mtry` are 1, 2, and other small values in older studies, e.g. Breiman (2001), and the square root of n, or the binary logarithm of n, with n being the number of input variables, becoming increasingly popular in newer literature, e.g. Strobl et al. (2009). As mentioned above, using the binary logarithm of the number of input variables is indeed the best choice for most datasets (Banfield et al., 2007). However, depending on the dataset and the ensemble size, `mtry` needs to be chosen differently. For example in studies of genetics, where the dataset often includes many irrelevant input variables in addition to the dataset being a "small n large p case", it is necessary to use a larger value for `mtry`, because otherwise some variables might never be used in the ensemble at all (Strobl et al., 2009). It might be interesting to consider choosing `mtry` automatically. One way to do this, is to systematically try different values for `mtry` and choose the best value using cross-validation (Strobl, 2013). According to Bernard et al. (2008) a greedy search or

choosing one of the values discussed above is the usual approach in the literature. The same paper shows, that this does not have to be the case, by demonstrating a "push-button" method that automatically derives a suitable value for `mtry` and "is at least as statistically significant as the original" (Bernard et al., 2008).

Out-of-bag Data. Because random forests are based on bagging and use bootstrapped samples, each tree has a set of approximately one third of the original dataset that has not been used to grow the tree. This set can be used to test the prediction accuracy of each tree, and the prediction accuracy of all trees can then be averaged to give an error estimation for the entire random forest. This out-of-bag error estimation is more precise than the naive error estimate, which uses all of the dataset (Strobl et al., 2009). This said, one should not forget that out-of-bag data is not the same thing as a genuine test dataset. Depending on the size of the training dataset and the size of the ensemble, the downsides of bootstrapping might shine through, and lead to an underestimation of the prediction accuracy.

Variable Importance. As has been mentioned above, random forests have a built in variable importance measure, which is calculated by permutating one input variable in the out-of-bag data of every tree, and calculating a new error estimate. The difference between the out-of-bag error estimates with and without randomly permutated input variable is the variable importance. The more important a variable is, the more drastically the prediction error increases when the variable is being randomised. The variable importance measure is sometimes scaled, i.e. z-standardised, but because it strongly depends on the parameter of random forests, it is not possible to compare these variable importances across studies, hence there is little use in doing so, and the practice only encourages problematic comparisons across studies (Strobl et al., 2009).

The idea behind this way of calculating variable importances is, that one would like to compare a prediction model with and without a particular input variable to measure the variable's impact. Obviously, one cannot ignore all trees that incorporate a variable, because each tree incorporates multiple input variables, and their impact on

the prediction would be modified too. By randomly permutating the values of an input variable, the variable's distribution characteristics do not change, but the connection to the output variable is broken. However, according to Strobl, Boulesteix, Zeileis, and Hothorn (2007) the variable importance might still depend on the type of the variable being measured, because the variable importance measure is biased towards variables with many categories and variables with many missing values. Numeric variables usually have as many different values as there are rows in the dataset, meaning that their importance measure is greatly biased due to the large number of "categories". Fortunately, this can be fixed, but "Only when subsamples drawn without replacement, instead of bootstrap samples, in combination with unbiased split selection criteria, are used in constructing the forest, can the resulting permutation importance be interpreted reliably" (Strobl et al., 2007), and correlated input variables still are problematic. The R package `party` includes two functions, `ctree` and `cforest`, that use a different method for split selection and are not affected by this bias. Thus, `cforest` actually calculates another variable importance measure termed "conditional variable importance" (Strobl et al., 2008). Correlated variables can be problematic, because trees that include a pair of correlated input variables are less affected by the random permutation of one of them. Conditional variable importance considers these correlations. However, Grömping (2009) argues, that this problem cannot be avoided as long as `mtry` is smaller than the number of input variables, and that considering all input variables, i.e. bagging, "might already go a long way" towards remedying the problem, although due to the "large p small n case", "unbiased estimation of all coefficients is impossible" in any case.

Another problem that affects the variable importance measure is that some variables might not be well represented in a random forest. This can be due to a small setting for `mtry` and or `ntree` in the presence of many irrelevant input variables, as often is the case in genetics datasets, or if variables show perfect higher order effects, i.e. interaction effects, but no main effects. The latter is called the *XOR problem*. It is important to note, that random forests with a different split selection algorithm do not

have to be affected by this, e.g. PERT (Cutler & Zhao, 2001).

Last but no least, both Strobl et al. (2009) and Grömping (2009) see one of the advantages of random forests in their variable importance measure. (Grömping, 2009) compares linear variable importance measures with the one built into random forests, and finds that the latter are heavily dependent on the mtry parameter. The larger `mtry` is, the better the importance estimates become. Variable importance measures can be used to select input variables for a simpler model, e.g. a generalised liner model, which is more interpretable than a forest of decision trees (Strobl et al., 2009). Variable importance measures are probably the only simple way to "shed some light into the black box of random forests" (Grömping, 2009). An alternative, but rather naive way to estimate variable importance is to count the occurrence of a variable over all trees (Strobl et al., 2009). Other imaginable ways to estimate variable importance would be to use algorithms that analyse the structure of each decision tree. Another interesting point to make is, that random perturbation of an input variable could in theory be used as part of many other models, including but not limited to generalised linear models, because although this is not very useful in the case of generalised linear models, many other complex models are similarly difficult to tease apart, e.g. neural networks.

The way variable selection based on variable importance measures is being done, is to find variables whose randomisation led to an improvement in prediction accuracy. These improvements are just random effects, and function as an indicator for which variable importances are within the band of random fluctuations, and which are true indicators for important variables.

Regression. Random forest can not only be used for classification, but also for regression. To produce the numeric output values necessary for regression, the vote on the most popular output class in the forest is replaced by the average over all tree outputs. One problem with regression using random forests is that more decision trees end up covering the middle range of the output variable, and few trees encode extreme output values. This means that the prediction accuracy is good in the middle of the range of values, but that the predictions for extremer values become more and more

inaccurate (G. Zhang & Lu, 2012). The predictions for large values tend to be too low, while the predictions for small values tend to be too high. G. Zhang and Lu (2012) also suggests five ways to estimate and reduce this problem.

Overfitting. Overfitting (Wikipedia, 2013n) is the situation where an algorithm learns the example dataset well enough to not only reproduce the underlying rules, but to also reproduce the random errors in the dataset. This is also called *generalisation error*, since the algorithm generalises errors in the data by deriving rules for them. Random forests grow optimal decision trees, meaning they try to represent each bootstrapped sample perfectly. Except for the case where different rows in the sample have the same values in their input variables, but different values in their output variable, this leads to decision trees which represent the sample including all of its errors perfectly. The advantage of ensemble learning is that by combining decision trees grown for different samples and constructed using different random split selections, these errors cancel out. However, this also means that there is an upper bound on how accurate random forests can become depending on the noise present in the dataset. Of course, this only applies if the trees in an ensemble are reasonably diverse, i.e. uncorrelated (Breiman, 2001). An analysis by Liu et al. (2005) showed that the generalisation error is at its lowest when the tree ensemble is maximally diverse, and that bootstrapping tends to limit tree diversity.

Different variants of random forests try to grow random forests on data that is less noisy, thus limiting the risk of generalisation errors. FRF stands for *Fuzzy random forests*, and is a method described by Bonissone et al. (2008). FRF use what are called *fuzzy sets* to represent the data in the dataset, and to build their decision trees on top. The advantages of this approach are that the resulting FRF are more immune to noise, and Cadenas, Garrido, Martínez, and Bonissone (2012) extend this framework to handle missing data. Another attempt at constructing random forests from smoother data are rotation forests by (Rodriguez et al., 2006), which use *PCA* (principal component analysis) to find derived input variables that represent the input dimensions better. Although linear combinations of input dimensions are not the same thing as noise, since

datasets are of a limited size and decision trees might well fail to tease out the interactions between all pairs of input variables, they will look like noisy input data if the decision trees do not catch on. Because better input dimensions lead to better splits, and to more uncorrelated random variable selections, rotation forests are often more accurate than random forests (Rodriguez et al., 2006).

Optimality Criterion. Random forests construct the underlying decision trees by selecting the best split at each node from a number of randomly selected variables. To compare the different possible splits, an optimality criterion gets calculated for each split. The most commonly used criterion is the Gini index. The Gini index is "a measure of statistical dispersion developed by the Italian statistician and sociologist Corrado Gini" (Wikipedia, 2013i), but it can also be used as a measure of entropy. The latter is how it is used in random forests. As has been mentioned in the section on variable importance, the fact that the Gini index is biased towards variables with many different values is problematic. Another common optimality criterion is entropy, although it suffers from the same bias towards variables with many different values. The point of using an optimality criterion in the first place is, to grow trees using an impurity reduction algorithm, i.e. to recursively select the split which returns the least dispersed subsets at each node.

A huge disadvantage of using an optimality criterion is the computational cost one has to pay. Trees which use random splits are much faster to grow, e.g. PERT ensembles are claimed to be two orders of magnitude faster than classical random forests (Cutler & Zhao, 2001). Cutler and Zhao (2001) indicate that despite its simplicity, "[...] PERT falls within the range of the other three methods.", meaning Adaboost and two random forest variants (different `mtry`), in terms of its prediction accuracy.

In Search of the Super-Forest

The following section will look at some of the random forest variants that have been mentioned above. It concludes by calling for a more user-friendly random forest variant, which combines many of the features of the proposed variants, and is tailored

to the less statistically adept users in the social sciences.

Alternative: PERT. PERT, or perfect random tree (ensemble), was developed by Cutler and Zhao (2001). The nice feature of PERT is that its strong reliance on randomness gives it a simple structure that is easier to describe and implement than classical random forests by Breiman (2001). In fact, PERT is simple enough that its algorithm is completely described in the following quote from Cutler and Zhao (2001):

> The perfect random tree ensemble, PERT, is so named because the PERT base learner is a random tree classifier that fits the training data perfectly. The PERT base learner is constructed by first placing all data points in the root node. At each step of the tree construction, each non-terminal node is split by randomly choosing two data points from the node until these two belong to different classes. Let $x = (x_1, ..., x_p)$ and $z = (z_1, ..., z_p)$. If this is not possible, all the data points in the node must be from the same class and the node is terminal. Now, randomly choose a feature on which to split, say feature j, and split at $\alpha x_j + (1 - \alpha)z_j$, where α is generated from a uniform(0,1) distribution.
>
> Ties are taken care of by repeating the procedure until a definitive split is obtained. If no such split is found after 10 tries, the node is declared to be terminal (so in this case, the tree would not perfectly fit the data). To form an ensemble, PERT base learners can be combined by simply fitting many PERT base learners to the entire training set and voting these classifiers. Alternatively, PERT base learners can be combined by bagging (Breiman, 1996). In this case, the PERT base learner is fit to many bootstrap samples from the training data and these classifiers are combined by voting.

On the one hand, bagging is not required, because PERT does not depend on randomness introduced through bootstrapping, which means that PERT can avoid the problems associated with bootstrapping. On the other hand, bagging can be used if one requires out-of-bag error estimates or variable importance measures, and as a way to

deal with ties in the dataset. As Liu et al. (2005) have indicated, bootstrapping might lead to less diverse trees, thus it should not be used unless one requires these features.

The main advantage of PERT over random forests is speed. Cutler and Zhao (2001) compare the different run times of random forest construction, as described in Breiman (2001), and PERT to find the latter to be faster by two orders of magnitude, while providing similar accuracy in prediction. Of course, there are other advantages too: The absence of a variable selection criterion means that PERT does not suffer from the bias introduced by it, and, that variable importance estimates are less biased. An instance where the use of a selection criterion is especially problematic is the XOR problem mentioned above in the section on variable importance. Since PERT forests do not depend on an optimality criterion to select their splits, they are able to deal with perfect interaction effects. A disadvantage is that PERT forests are even more difficult to interpret than classical random forests, but since this is rarely tried anyway, it does not matter much in practice. It is important to mention however, that PERT can only be used for classification, since its base learner fits the training data perfectly, applying PERT to regression would “drastically overfit the data” (Cutler & Zhao, 2001).

PERT is not the only random forest variant that emphasises randomness when growing trees. Geurts et al. (2006) suggested what they call *extremely randomised trees*. Despite their name, extremely randomised trees are actually closer to the trees of random forests by Breiman (2001) than to PERT trees, in that they do select the best split from a random subset variables of size `mtry`, although the actual split value for each variable is generated randomly. Furthermore, they do not fit the data perfectly, but allow the user to choose a "minimum sample size for splitting a node" (Geurts et al., 2006). This parameter, called *smoothing strength* by Geurts et al. (2006), prevents overfitting in noisy datasets, and allows the user to configure the behaviour of the algorithm in order to adapt it to dataset at hand.

Alternative: Fuzzy random forests. Fuzzy random forests by Bonissone et al. (2008) use input variables encoded as fuzzy sets. Fuzzy sets (Wikipedia, 2013h) model membership to a category by a value between one and zero, while fuzzy logic

(Wikipedia, 2013g) allows to reason using fuzzy data: For example, a day with a temperature of 45 degrees Celsius is to 100% a hot day, and to 0% a cold day, while a day with 15 degrees Celsius might belong to the label cold day by 40% and to hot day by 60%. The advantage of using fuzzy sets is the added flexibility of representing uncertainty. Cadenas et al. (2012) extended fuzzy random forests to handle missing data, thus making fuzzy random forests even better at representing real world data.

Alternative: Rotation forests. Rotation forests use principle component analysis to transform the dataset before creating the ensemble of decision trees. "Principal component analysis (PCA) is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components." (Wikipedia, 2013o) Rotation forests have been suggested by Rodriguez et al. (2006). Rotation forests are created by applying PCA to the bootstrap sample of each tree prior to building the tree. To apply PCA, the input variables are split into subsets and rows matching randomly selected values of the output variable are removed. The PCA coefficients for each subset are then arranged into the rotation matrix used to transform the sample before building the tree. This means that another parameter, the number of input variable subsets, is required. Rotation forests significantly outperform random forests, bagging, and boosting on many datasets, but due to the transformation of the dataset, rotation forests do not provide variable importance measures (Rodriguez et al., 2006).

The transformation of the input data leads to less correlation between trees, and to improved prediction accuracy, which Rodriguez et al. (2006) demonstrate with the help of diversity-error diagrams.

Future Random Forests. The different algorithms in the random forest framework have been developed iteratively, with each improvement or variation picking up one problem to improve on. The criterion for success has been beating the base algorithm. Similar to how the scientific method has become a competition between researcher and null hypothesis, the field of research on random forests seems to be

plagued by comparisons between the old and the new forests. It might be worth the effort to step back from the specifics for a moment and look at the principle behind random forests.

Every variant in the random forests framework is an ensemble of trees. Beyond that there seems to be little that may not be changed. Two questions that naturally follow from this are:

1. Can a random forest incorporate all of the developed features?
2. How does one prioritise the different features and problems?

Most of the suggested improvements and variants have taken the original random forest algorithm and changed some part of it. Since the changes of the different variants discussed in this report mostly do not overlap, combining them should certainly be possible. The more interesting question is the second. This report has mentioned many problems of random forests, for example the problem of parameter selection. To increase the reach of random forests in general, and particularly in the social sciences, the parameter situation needs to be resolved, because not every researcher can include a simulation study just to show that his methods are not flawed. An accepted solution to this problem is to tune the parameters using cross-validation. While this is a good solution for most applications, it does lead to an underestimation of the error estimate. Obviously this can be fixed by setting aside a dedicated test dataset, but compiling an even larger dataset is often unpractical, since real world datasets are not cheap to compile. By including the suggestions of Banfield et al. (2007) regarding `ntree` and Bernard et al. (2008) regarding `mtry` a new random forest variant could be completely parameterless. Alternatively, a new random variant could use PERT to get rid of the parameter `mtry` and some of the bias in the variable importance measure. Both approaches do not require additional data.

Obviously, there is no single best solution, but this does not make unifying random forest variants a futile task. The simpler and safer a method is to use, the wider it will spread, and by spreading a good default method, all of scientific research will benefit.

Method

Selection of Papers

I started my research on random forests by reading the introductory paper suggested by my supervisor titled *An introduction to recursive partitioning: Rationale, application and characteristics of classification and regression trees, bagging and random forests* (Strobl et al., 2009). I then searched the databases *PsychARTICLES* and *PsychINFO* querying for *random forest* and limiting my results to the last two years. I only considered papers that focus on random forests and dismissed almost all papers where random forests are merely used as a method of analysis. I also used *Google Scholar* to look for the more technical publications outside the field of psychology. I searched for combinations of *random forest*, *comparison*, *analysis*, and *ensemble* without time limitation, and included keywords seen in interesting titles, like *fuzzy*, *perfect*, *full*, *balanced*, *extremely*, and *rotation*. I then looked for some of the referenced publications I had read about in the papers found as mentioned above, and included Strobl et al. (2008), Ho (1998), Breiman (1996), Freund and Schapire (1995), and Long and Servedio (2010).

The final criteria for inclusion were the online availability of a freely downloadable PDF-file, which thanks to *Google Scholar* often turned out to be no problem at all, and my decision on the topic of this report. It turned out, that while Google Scholar is convenient to find freely downloadable PDF-files, its citations are not up to par.

A lot of the information in the fields of computer science, artificial intelligence, machine learning, databases, and especially programming I acquired through different means in the last ten years. As I cannot remember the original sources, I include the pages on Wikipedia and a few books, e.g. Bortz and Schuster (2010); Barber (2013); Haykin (2007); Nilsson (2009), which I used to refresh my memories.

Choice of Topic

Possible options for the topic of this paper were the presentation of exemplary uses of random forests, the discussion of strengths and weaknesses, and any of the more

specialised variants. During my research I had the impression, that there were serious differences in the understanding of random forests among researchers, and even among the designers of improved variants. A study comparing different decision tree ensemble techniques also confirmed this expression by saying that the commonly used methods of comparing random forests were not robust enough (Banfield et al., 2007).

Because of this fuzziness, I decided on a different focus, i.e. to write a very broad - big picture - introduction to random forests.

Discussion

Random forests

Random forests are diverse. There is no "the random forest" method, but rather a wide spectrum of methods that all use similar principles. Random forests are accurate and very powerful, because they can be used on almost any dataset. The way a random forest is grown lends itself to be implemented to run on multiple processor cores at once, as the small example implementation demonstrates. This is an important criterion for anyone who wants to analyse large datasets. On top of that, they are easy to implement and do not require a sophisticated mathematical understanding. In fact, a lot of the random forest variants have been developed by combining known concepts and later demonstrating their viability using a comparative study with simulated and real-world datasets.

However, it has to be said that random forests can be problematic, because they appear to be easy to use, but expose a lot of their inner workings to the user. The user should not have to worry about parameters, and how they interact with a dataset introducing bias in error estimates and variable importance measures. In studies where only small ensembles are used, one has to wonder if the researcher did not just get lucky with his random number generator's seed value. In some ways Banfield et al. (2007) can be seen to ask the same question, since the claimed significant improvement in prediction accuracy of many methods disappear when the comparisons are conducted more rigorously.

The very ease with which random forests can be implemented and changed, should lead to a more thorough research of their qualities, but this seems not to be the case. Most studies just compare a specific variant against the random forests of Breiman (2001), bagging (Breiman, 1996) and Adaboost (Freund & Schapire, 1995). One of the factors for this I believe to be the lack of a standardised and automated testing infrastructure, which would simplify the development and testing of new algorithms.

Methods research

If the datasets and protocols for measuring and comparing algorithms were developed first and standardised, new algorithms could be measured by the push of a button. Authors would not have to use half of their publication to describe the datasets used in a comparison. Furthermore, by combining the knowledge of the particularities of a dataset, especially real-world datasets, authors would have access to immediate indications on why their method might be particularly good or bad when applied to a certain type of data. Such a standardised test repository would also lighten the load of all researchers just using random forests as a tool. They would not have to explain why they choose a particular variant based on their incomplete knowledge of machine learning, but could just refer to a standard. This would certainly be beneficial for research in general, but one does not have to stop there. By standardising research methods, their development and comparison in general, it also becomes easier to replicate a study's results using different methods on the same dataset, and the datasets of a study can be included in the corpus of test-sets for future testing of methods. One of the more well-known repositories of datasets is the *UCI Machine Learning Repository* (Bache & M., 2013), but unfortunately many researchers hesitate to donate their datasets to such a repository.

The "Methods" section of a study and the research methods mentioned in it, should not be separate, because relying on a particular version of a particular tool to decide over the significance of findings is not very scientific. Unfortunately, this happens: Researchers are very careful when selecting theories from their own field, but sometimes flagrantly refer to "default settings" with no further detail, when they use a particular implementation of a method, just because software or random forests are not their field. Default values in implementations do change, and these changes sometimes happen for a reason. It should be possible to check if a study is affected by this change, and reevaluate it if necessary. In keeping with the artificial intelligence origins of machine learning, one can envision a display of a huge dependencies graph with red and green lights representing the state of research fields. Computers would do the

automated rechecking of past studies as soon as new datasets or new methods become available, and notify researchers when the premises of their work or field of study were refuted. Computers could also publish requests for dataset and new methods, when they detect methodical weaknesses and other "anomalies".

Obviously, this is quite a different topic than random forests, but the field of research on random forests nicely illustrates how some of the problems play out. Just because Breiman (2001) uses certain parameters for the comparison, this does not mean that the parameters are a good choice when demonstrating the powers of a new random forest variant. Since random forests all roughly take the same format of input data, it should be trivial to cross-validate all findings with all datasets and all variants, but as it stands this is still difficult to do in practice. One does not know if another study like Banfield et al. (2007) overturns random forests in the near future by finding another method to be superior. For most researchers in other fields it is a safer bet to stick to traditional methods. Unfortunately, this often means to ignore problems with many input variables or not daring to invent models with complex interactions, which are difficult to analyse using traditional methods.

Appendix

The following Clojure source code is the complete version of the example implementation of a random forest discussed earlier in this paper. To run the examples, install the `Leiningen` build system (Hagelberg, 2013), save the following source code to the indicated files, and start up the interactive *REPL* (read eval print loop).

```
$ cd rf/
$ lein deps
$ lein repl
```

Figure 10. Shell commands to start the Clojure REPL.

Once this is done, provided no unexpected errors have occurred, the random forest implementation can be used shown below. Please refer to the source code below for information on the format of the dataset.

```
user=> (require '[rf.core :as r])
nil
user=> (def forest (r/rf-train r/sample 150 3))
#'user/forest
user=> (every? #(= (r/rf-predict forest (dissoc % :class)) (:class %))
  #_=>         r/sample)
true
user=> (r/rf-error rf)
0
user=> (exit)
Bye for now!
```

Figure 11. An example interaction in the REPL illustrating how to use the random forest implementation.

```
; file: rf/project.clj

(defproject rf "1.0.0-SNAPSHOT"
  :description "An example random forest implementation."
  :dependencies [[org.clojure/clojure "1.5.1"]])
```

```
; file: rf/src/rf/core.clj

(ns rf.core
  (:require [clojure.zip :as zip]))

(def sample [
  {:a 0 :b 0 :c 0 :d 0 :class :0} {:a 1 :b 0 :c 0 :d 0 :class :8}
  {:a 0 :b 0 :c 0 :d 1 :class :1} {:a 1 :b 0 :c 0 :d 1 :class :9}
  {:a 0 :b 0 :c 1 :d 0 :class :2} {:a 1 :b 0 :c 1 :d 0 :class :a}
  {:a 0 :b 0 :c 1 :d 1 :class :3} {:a 1 :b 0 :c 1 :d 1 :class :b}
  {:a 0 :b 1 :c 0 :d 0 :class :4} {:a 1 :b 1 :c 0 :d 0 :class :c}
  {:a 0 :b 1 :c 0 :d 1 :class :5} {:a 1 :b 1 :c 0 :d 1 :class :d}
  {:a 0 :b 1 :c 1 :d 0 :class :6} {:a 1 :b 1 :c 1 :d 0 :class :e}
  {:a 0 :b 1 :c 1 :d 1 :class :7} {:a 1 :b 1 :c 1 :d 1 :class :f}])

(defn bootstrap [dataset]
  (let [n (count dataset)
        s (repeatedly n #(rand-int n))
        d (set (map #(nth dataset %) s))
        o (set (filter (comp not d) dataset))]
    {:dataset d :oob o}))

(defn select [[v s] dataset]
  (let [l (reduce (fn [acc t]
                    (if (<= (get t v) s) (conj acc t) acc))
            #{} dataset)
        r (set (filter (comp not l) dataset))]
    [l r]))

(defn entropy [dataset]
  (let [n (count dataset)
        p (map #(/ % n) (vals (frequencies (map :class dataset))))
        l #(/ (Math/log %) (Math/log 2))]
    (* -1 (reduce #(+ %1 (* %2 (l %2))) 0.0 p))))

(defn splits [dataset mtry]
  (let [vs (filter (partial not= :class) (keys (first dataset)))
        n (count vs)
        vs (if (<= mtry n) (take mtry (shuffle vs)) vs)]
    (reduce (fn [acc v]
              (reduce (fn [acc t]
                        (conj acc [v (get t v)])) acc dataset))
            #{} vs)))

(defn best-split [dataset oob mtry]
  (let [[_ s l r]
    (first (sort (map (fn [s] (let [[l r] (select s dataset)]
                                [(entropy l) s l r]))
```

```
                      (splits dataset mtry))))]
    (let [[oobl oobr] (select s oob)]
      [s (set l) (set r) (set oobl) (set oobr)])))

(defn leaf? [dataset] (= (entropy dataset) 0.0))

(defn extend-node [{:keys [dataset oob] :as node} mtry]
  (if (leaf? dataset)
    (merge node
           {:class (or (first (first (sort-by val >
                       (frequencies (map :class dataset)))))
                       :unknown)})
    (let [[s l r oobl oobr] (best-split dataset oob mtry)]
      (merge node
             {:criterion s :left  {:dataset l :oob oobl}
                           :right {:dataset r :oob oobr}}))))

(defn rf-train [dataset ntree mtry]
  (set (pmap (fn [sample]
        (loop [loc (zip/zipper
                      (fn [node] true)
                      #(if (:left %)
                        (seq [(:left %) (:right %)]))
                      (fn [node children]
                        (with-meta
                          (merge node
                                 {:left  (first children)
                                  :right (second children)})
                          (meta node)))
                      sample)]
          (if (zip/end? loc) (zip/root loc)
            (recur
              (zip/next (zip/edit loc extend-node mtry)))))
        (repeatedly ntree #(bootstrap dataset))))))

(defn rf-predict [forest features]
  (let [eval-tree
          (fn eval-tree [tree]
            (if (:class tree) (:class tree)
              (if (let [[v s] (:criterion tree)]
                    (<= (get features v) s))
                (eval-tree (:left tree))
                (eval-tree (:right tree)))))]
    (first (first
      (sort-by
        val > (frequencies
                (filter (partial not= :unknown)
                        (pmap eval-tree forest)))))))))
```

```
(defn tree-error [tree]
  (let [cls (:class tree)]
    (cond (nil? cls) (+ (tree-error (:left tree))
                        (tree-error (:right tree)))
          (= cls :unknown) 0
          :else (count
                  (filter (partial = cls) (:oob tree))))))

(defn rf-error [forest]
  (/ (reduce + 0
       (pmap #(/ (tree-error %) (count (:oob %))) forest))
     (count forest)))
```

References

Bache, K., & M., L. (2013). *UCI machine learning repository.* Retrieved from http://archive.ics.uci.edu/ml

Banfield, R. E., Hall, L. O., Bowyer, K. W., & Kegelmeyer, W. P. (2007). A comparison of decision tree ensemble creation techniques. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *29*(1), 173–180. doi: 10.1109/TPAMI.2007.250609

Barber, D. (2013). Bayesian Reasoning and Machine Learning. In (pp. 291–304). Cambridge University Press. Retrieved from http://web4.cs.ucl.ac.uk/staff/D.Barber/textbook/180613.pdf (DRAFT)

Bernard, S., Adam, S., & Heutte, L. (2012). Dynamic random forests. *Pattern Recognition Letters*, *33*(12), 1580–1586.

Bernard, S., Heutte, L., & Adam, S. (2008). Forest-rk: A new random forest induction method. In *Proceedings of the 4th international conference on intelligent computing: Advanced intelligent computing theories and applications - with aspects of artificial intelligence* (pp. 430–437). Springer Berlin Heidelberg. doi: 10.1007/978-3-540-85984-0_52

Bonissone, P., Cadenas, J., Garrido, M., & Dıaz-Valladares, R. (2008). A fuzzy random forest: Fundamental for design and construction. In *Proceedings of the 12th international conference on information processing and management of uncertainty in knowledge-based systems (ipmu'08)* (pp. 1231–1238). Retrieved from http://www.gimac.uma.es/IPMU08/proceedings/papers/163-BonissoneEtAl.pdf

Bortz, J., & Schuster, C. (2010). Statistik für human- und sozialwissenschaftler. In (7th ed., pp. 183–202). Springer-Verlag.

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140. doi: 10.1023/A:1018054314350

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

10.1023/A:1010933404324

Cadenas, J. M., Garrido, M. C., Martínez, R., & Bonissone, P. P. (2012). Extending information processing in a fuzzy random forest ensemble. *Soft Computing*, *16*(5), 845–861. doi: 10.1007/s00500-011-0777-1

Cutler, A., & Zhao, G. (2001). Pert-perfect random tree ensembles. *Computing Science and Statistics*, *33*, 490–497. Retrieved from `http://www.interfacesymposia.org/I01/I2001Proceedings/ACutler/ACutler.pdf`

Dietterich, T. (2000). Ensemble methods in machine learning. In *Multiple classifier systems* (Vol. 1857, p. 1-15). Springer Berlin Heidelberg. doi: 10.1007/3-540-45014-9_1

Freund, Y., & Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In P. Vitányi (Ed.), *Computational learning theory* (Vol. 904, pp. 23–37). Springer Berlin Heidelberg. doi: 10.1007/3-540-59119-2_166

Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, *63*(1), 3–42. doi: 10.1007/s10994-006-6226-1

Grömping, U. (2009). Variable importance assessment in regression: linear regression versus random forest. *The American Statistician*, *63*(4), 308–319. doi: 10.1198/tast.2009.08199

Guo, L., Ma, Y., Cukic, B., & Singh, H. (2004). Robust prediction of fault-proneness by random forests. In *Software reliability engineering, 2004. issre 2004. 15th international symposium on* (pp. 417–428). IEEE. doi: 10.1109/ISSRE.2004.35

Hagelberg, P. (2013). *Leiningen.* Retrieved from `https://github.com/technomancy/leiningen/tree/2.2.0` (commit: 5eaad5c48d)

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, *11*(1), 10–18. doi: 10.1145/1656274.1656278

Haykin, S. (2007). Neural networks: A comprehensive foundation (3rd edition). In (pp.

23–71). Prentice-Hall, Inc.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *20*(8), 832–844. doi: 10.1109/34.709601

Huet, G. (1997). The zipper. *Journal of Functional Programming*, *7*(5), 549–554. doi: 10.1017/S0956796897002864

Kobyliński, Ł., & Przepiórkowski, A. (2008). Definition extraction with balanced random forests. In B. Nordström & A. Ranta (Eds.), *Advances in natural language processing* (Vol. 5221, pp. 237–247). Springer Berlin Heidelberg. doi: 10.1007/978-3-540-85287-2_23

Liaw, A., & Wiener, M. (2002). Classification and regression by randomforest. *R news*, *2*(3), 18–22. Retrieved from `http://CRAN.R-project.org/doc/Rnews/`

Liu, F. T., Ting, K. M., & Fan, W. (2005). Maximizing tree diversity by building complete-random decision trees. In *Proceedings of the 9th pacific-asia conference on advances in knowledge discovery and data mining* (pp. 605–610). Springer Berlin Heidelberg. doi: 10.1007/11430919_70

Long, P. M., & Servedio, R. A. (2010). Random classification noise defeats all convex potential boosters. *Machine Learning*, *78*(3), 287–304. doi: 10.1007/s10994-009-5165-z

Maudes, J., Rodríguez, J. J., García-Osorio, C., & García-Pedrajas, N. (2012). Random feature weights for decision tree ensemble construction. *Information Fusion*, *13*(1), 20–30. doi: 10.1016/j.inffus.2010.11.004

Nilsson, N. J. (2009). *The quest for artificial intelligence*. Cambridge University Press.

Polikar, R. (2009). Ensemble learning. *Scholarpedia*, *4*(1), 2776. Retrieved from `http://www.scholarpedia.org/w/index.php?title=Ensemble_learning&oldid=91224` doi: doi:10.4249/scholarpedia.2776

R Core Team. (2012). R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria. Retrieved from

http://www.R-project.org/

Robnik-Šikonja, M. (2004). Improving random forests. In J.-F. Boulicaut, F. Esposito, F. Giannotti, & D. Pedreschi (Eds.), *Ecml* (Vol. 3201, p. 359-370). Springer. doi: 10.1007/978-3-540-30115-8_34

Rodriguez, J. J., Kuncheva, L. I., & Alonso, C. J. (2006). Rotation forest: A new classifier ensemble method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *28*(10), 1619–1630. doi: 10.1109/TPAMI.2006.211

Strobl, C. (2013). *Psychologische methoden: Datenerhebung, analyse und darstellung.* University Lecture.

Strobl, C., Boulesteix, A.-L., Kneib, T., Augustin, T., & Zeileis, A. (2008). Conditional variable importance for random forests. *BMC Bioinformatics*, *9*(1), 307. doi: 10.1186/1471-2105-9-307

Strobl, C., Boulesteix, A.-L., Zeileis, A., & Hothorn, T. (2007). Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, *8*(1), 1–21. doi: 10.1186/1471-2105-8-25

Strobl, C., Malley, J., & Tutz, G. (2009). An introduction to recursive partitioning: Rationale, application and characteristics of classification and regression trees, bagging and random forests. *Psychological Methods*, *14*(4), 323.

Van Essen, B., Macaraeg, C., Gokhale, M., & Prenger, R. (2012). Accelerating a random forest classifier: multi-core, gp-gpu, or fpga? In *Field-programmable custom computing machines (fccm), 2012 ieee 20th annual international symposium on* (pp. 232–239). IEEE. doi: 10.1109/FCCM.2012.47

Wikipedia. (2013a). *Analysis of variance.* Retrieved from https://en.wikipedia.org/w/index.php?title=Analysis_of_variance&oldid=552674312

Wikipedia. (2013b). *Bootstrapping.* Retrieved from https://en.wikipedia.org/w/index.php?title=Bootstrapping&oldid=553800549

Wikipedia. (2013c). *Clojure.* Retrieved from https://en.wikipedia.org/w/index.php?title=Clojure&oldid=552903619

Wikipedia. (2013d). *Database.* Retrieved from

https://en.wikipedia.org/w/index.php?title=Database&oldid=552856988

Wikipedia. (2013e). *Ensemble learning.* Retrieved from https://en.wikipedia.org/w/index.php?title=Ensemble_learning&oldid=552635358

Wikipedia. (2013f). *Expert system.* Retrieved from https://en.wikipedia.org/w/index.php?title=Expert_system&oldid=553130112

Wikipedia. (2013g). *Fuzzy logic.* Retrieved from https://en.wikipedia.org/w/index.php?title=Fuzzy_logic&oldid=554084074

Wikipedia. (2013h). *Fuzzy set.* Retrieved from https://en.wikipedia.org/w/index.php?title=Fuzzy_set&oldid=552201293

Wikipedia. (2013i). *Gini coefficient.* Retrieved from https://en.wikipedia.org/w/index.php?title=Gini_coefficient&oldid=555394466

Wikipedia. (2013j). *History of artificial intelligence.* Retrieved from https://en.wikipedia.org/w/index.php?title=History_of_artificial_intelligence&oldid=560258744

Wikipedia. (2013k). *History of programming languages.* Retrieved from https://en.wikipedia.org/w/index.php?title=History_of_programming_languages&oldid=551008740

Wikipedia. (2013l). *Logic programming.* Retrieved from https://en.wikipedia.org/w/index.php?title=Logic_programming&oldid=551620750

Wikipedia. (2013m). *Machine learning.* Retrieved from https://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=552683867

Wikipedia. (2013n). *Overfitting.* Retrieved from https://en.wikipedia.org/w/index.php?title=Overfitting&oldid=543681560

Wikipedia. (2013o). *Principal component analysis.* Retrieved from https://en.wikipedia.org/w/index.php?title=Principal_component_analysis&oldid=554710521

Wikipedia. (2013p). *Random forest.* Retrieved from https://en.wikipedia.org/w/index.php?title=Random_forest&oldid=554887443

Wikipedia. (2013q). *Regression analysis.* Retrieved from `https://en.wikipedia.org/w/index.php?title=Regression_analysis&oldid=553084781`

Xie, Y., Li, X., Ngai, E. N., & Ying, W. (2009). Customer churn prediction using improved balanced random forests. *Expert Systems with Applications*, *36*(3), 5445–5449. doi: 10.1016/j.eswa.2008.06.121

Xu, P., & Jelinek, F. (2004). Random forests in language modeling. In *Proceedings of the 2004 conference on empirical methods in natural language processing.* Retrieved from `http://acl.ldc.upenn.edu/acl2004/emnlp/pdf/Xu.pdf`

Zhang, G., & Lu, Y. (2012). Bias-corrected random forests in regression. *Journal of Applied Statistics*, *39*(1), 151–160. doi: 10.1080/02664763.2011.578621

Zhang, J., & Zulkernine, M. (2005). Network intrusion detection using random forests. In *Pst.*