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

DRAFT Version: June 24, 2013

Author Note

Tobias Ammann

Alte Landstrasse 39

8802 Kilchberg

tag@adnm.ch

Abstract

This report looks at a machine learning algorithms called random forests, their structure, problems, and a number of suggested improvements, to give the reader a more big-picture introduction. The topics discussed are the origins of machine learning, machine learning vs traditional statistics, ensemble learning, bagging, boosting, decision trees, random forests, parameters, out-of-bag data, error estimates, variable importance, regression, overfitting, optimality criterion, the random forest variants PERT, fuzzy random forests, and rotation forests. This report puts forward a few ideas on how to make random forests better and more user-friendly.

Keywords: ensemble methods, introduction, random forests

Random Forests: Introduction, Parts, Variants

Contents

Abstract 2

Random Forests: Introduction, Parts, Variants 3

Introduction **5**

Motivation . . . 5

Machine Learning . . . 6

The Machine Learning Life Cycle . . . 7

Classification and Regression . . . 9

Randomness . . . 9

Random Forests **10**

Introduction . . . 10

Ensemble Learning Methods . . . 12

Bagging . . . 13

Boosting . . . 13

Decision Trees . . . 14

Random Forests . . . 15

Parameters . . . 18

Out-of-bag Data . . . 20

Variable Importance . . . 20

Regression . . . 22

Overfitting . . . 23

Optimality Criterion . . . 24

In Search of the Super-Forest . . . 24

Alternative: PERT . . . 24

Alternative: Fuzzy random forests . . . 26

Alternative: Rotation forests . . . 26

Alternative: Unknown 27

Method **29**

Selection of Papers 29

Aim and Structure of the Paper 29

Discussion **31**

Random forests 31

Methods research 32

Appendix **34**

References **37**

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 only 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 an unweildly number of theories to disprove, and every researcher likes to see the result of his work during his lifetime, a more speedy method is usually employed, although this comes with a caveat. The speedier method has the researcher pit his favored theory against the null hypothesis, a fancy word for chance. 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, commonly called confirmation bias, because researchers focus on immunizing their theory instead of disproving those of others. Therefore, the result only has significance in the experimental set-up being tested. In 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 generalize from the results of an experiment to the world at large, and likely remain 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 freakishly 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 good statistical methods, because that 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 present to the reader some context to random forests, in hope that a more big-picture view will be beneficial.

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 new 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 might be useful 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. It's probably most widely known method is *linear regression* (Wikipedia, 2013l) that associates two variables y and x in such a way that they describe a straight line: $y = \alpha * x + \beta$. Predictive statistics is widely used in psychology because it allows the researcher to look at the unobservable by making assumptions of the form $reaction = mind * stimulus + measurement\ error$. This is the standard approach in personality questionaires.

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 capabilites, 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, 2013m), *expert systems* (Wikipedia, 2013g), *databases* (Wikipedia, 2013e) and *neural networks* (Wikipedia, 2013b), that all represent some form of storing and querying knowledge. Unfortunately, early computers back then didn't have the speed and memory required to push the

envelope far enough, and the field has been largely dormant since. Only the rather recent coexistance 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 their 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 that part of artificial intelligence that is concerned with the computer's learning of facts about the world. These facts can then be stored and subsequently queried later on. As such, machine learning is concerned with making statements based on past observations, and is therefore close to predictive statistics (Wikipedia, 2013n).

The Machine Learning Life Cycle

This section discusses the difference between machine learning and traditional statistical methods. Terminology. The following section heavily relies on information found in Wikipedia, as well as what I learned in statistics lectures in the past years. A good introductory article is (Wikipedia, 2013n).

Traditionally, the statistical methods used in psychology take a model of how the world works, and a set of data, and return one of two 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.

Regression methods try to derive the values of one variable 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 with different inputs, and calculate how probable an improvement in this comparison is to show up due to random variations (Wikipedia, 2013r).

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 variation is more likely to show up in certain groups (Wikipedia, 2013a).

The probabilities the methods output are what statisticians call the *significance*. Statisticians usually define a target *significance level*, e.g. 5%, and compare the output of their statistical calculations to it. If the calculated probability is less than the targeted significance level, the measured effects are said to be significant at the significance level. For example, a result that is significant at 5%, we is less likely to show up at random than in 5% of all experiments.

The most striking difference between traditional statistical methods and machine learning methods is that the researcher can't specify his 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's going into the algorithm, and what is coming out. This is unlike the statistical methods, where the researcher supplies a formula, because in machine learning, algorithms derive the formula on their own. This is what "learning" in machine learning means. The second difference is what the algorithms return. Since machine learning algorithms represent the model, what they output is not a percentage, i.e. significance, but the model itself. In short, machine learning provides the researcher with a generic model that adapts to the world. The prediction of such a model can then be calculated for data for which the values of the output variable are known, but that haven't 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*. Other descriptions are also common, but usually depend 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 initialized with the current time at the start of the program. These changes might unnerve a novice, but don't change the outcome of the algorithm significantly, 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 initialized 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 don't 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, 2013q).

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 forest (Rodriguez, Kuncheva, & Alonso, 2006), random forests that are more random (Geurts, Ernst, & Wehenkel, 2006), (Liu, Ting, & Fan, 2005), (Cutler & Zhao, 2001), and various other improvements, e.g. by (Banfield, Hall, Bowyer, & Kegelmeyer, 2007), (Robnik-Šikonja, 2004), (Strobl et al., 2009), (G. Zhang & Lu, 2012), make it so that it is better to think of random forests as being a framework instead of being a single method (Wikipedia, 2013q). 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 strictly based on (Breiman, 2001), while the part by part discussion will also go into tweaking random forests.

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 selected 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 records in the dataset that were not selected 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 randomizing the variable under consideration. The variable importance of the randomized variable is the increase of misclassifications.

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_smoking.rda")
> library("party")
> myctree <- ctree(intention_to_smoke ~ ., data = dat_smoking)
> class(dat_smoking$intention_to_smoke)
> plot(myctree)
```

Figure 1. R example using the party package.

The use case for random forests is quite wide. 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) and internet security (J. Zhang & Zulkernine, 2005), to natural language applications (Xu & Jelinek, 2004) and (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 in (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 its most useful feature. Many domains don't 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. generalized linear models, 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. Again, this paper will first give an abstract definition and then look at examples in detail.

Ensemble learning is a supervised learning algorithm: its task is to take an example of 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 can't be implemented except for trivial problems. However, the principle that the combination of possible hypotheses becomes a stronger hypothesis, because the ensemble can represent more functions than every component hypothesis could holds. 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, 2013f) (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, 2013c). 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 Leo Breiman (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. Random forests use the remaining third as out-of-bag data (Breiman, 2001).

Boosting. Boosting is an ensemble method, where every weak classifiers gets to train on the original sample, but the sample is improved with importance weights. These weights are different for every instance of the weak classifier. They are lower for records in the dataset that are correctly predicted by the classifiers already in the ensemble, and higher for records 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 isn't used in random forests as introduced by (Breiman, 2001), but there are variants who do, 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 latter compared eight ensemble of decision tree classifiers in a large study on 57 publicly available datasets and concluded "[...] that boosting, random forests and

randomized 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 the single 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 mismatch, and each leaf node an outcome of the represented 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`.

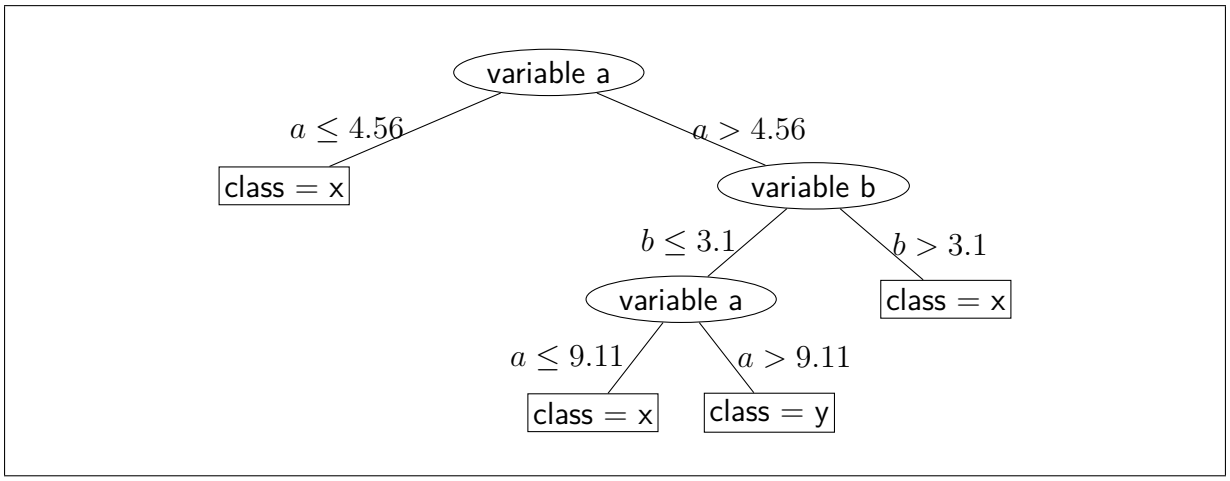

Figure 2. A small decision tree.

The decision trees used by 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 threshold among 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 only matches records in the dataset with the same output value. 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, 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 algorithm (Cutler & Zhao, 2001) for example doesn't perform a search for the best split on a random selection of input variables, but selects variable and threshold at random. RFW (Maudes et al., 2012) searches all variables but attaches random weights to them, and DRF (Bernard et al., 2012) influences the decision tree creation by a procedure inspired by boosting. Other variants, e.g. (Van Essen, Macaraeg, Gokhale, & 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 randomization parameter `mtry`. Actually, random forests depend on another parameter, the state of a random number generator, but being random, it doesn't and usually shouldn't be specified. The following is an exemplary implementation of random forest following the original version by (Breiman, 2001), but with limited features and with entropy instead

of the gini index as optimality criterion. In reference to the higher level programming languages mentioned in the introduction, the following code uses a Lisp dialect called Clojure (Wikipedia, 2013d).

```
(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)))

(def rf (rf-train sample 1 3))

(every? #(= (rf-predict rf (dissoc % :class)) (:class %))
        sample)

(rf-error rf)
```

Parameters. The two parameters random forests depends 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 randomization 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 takes away 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 weren't 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 (Strobl et al., 2009) in well under 30 seconds.

The second parameter of random forests is the randomization 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 1, random forests acts like a linear combination of the input variables. 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`, else some variables might never be used in the ensemble at all (Strobl et al., 2009). It might be interesting to consider choosing `mtry` automatically, and two obvious suggestions for this would be to choose a random `mtry` value for every decision tree being grown, or to

define it via a function of the number of input variables satisfying some statistical distribution criterion. 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 doesn't 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".

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 of the entire random forest. This out-of-bag error estimation is more precise than the standard error estimate, which is using all of the dataset (Strobl et al., 2009). This said, one should not forget that out-of-bag data isn't the same thing as a genuine test dataset. Depending on the size of the 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 randomized. The variable importance measure is sometimes scaled, i.e. z-standardized, but because it strongly depends on the parameter of random forests, it's 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 can't 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 characteristics don't change, but the connection to the output variable is broken. However, according to (Strobl, Boulesteix, Zeileis, & Hothorn, 2007) the variable importance might still depend on which variable is being measured, because the variable importance measure is biased towards variables with many categories and variables with many missing values. Numeric variable usually have as many different values as there are records 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 are not affected by this bias, due to their use yet another variable importance 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 can't 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 the variable importance measure can be affected by, 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 don't

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 are used to select input variables for a simpler model, e.g. a generalized liner model, which is more interpretable than a forest of decision trees (Strobl et al., 2009). Variable importance measures also are the probably only simple way to "shed some light into the black box of random forests" (Grömping, 2009). A alternative, but rather naive way to estimate variable importance is to count the occurance of a variable over all trees (Strobl et al., 2009). Other imaginable ways to estimate variable importance would be to use algorithms that analyze 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 generalized linear models, because although this isn't very useful in the case of generalized 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 by finding variables whose randomization 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 calculation of an average over all tree outputs. One problem with regression using random forests is that more decision trees end up covering the middle range of values of output variable. 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 same paper also suggests five ways to estimate and reduce this bias.

Overfitting. Overfitting (Wikipedia, 2013o) 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 *generalization error*, since the algorithm generalizes 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 records 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 perfectly, including all errors. The advantage of ensemble learning is that by combining decision trees grown for different samples, and constructed using different random split selections, these errors are canceled 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 generalization 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 prone to noise. 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, & Bonissone, 2012) extend this framework to handle missing data too. Another attempt at constructing random forests on higher quality data are rotation forests by (Rodriguez et al., 2006), which use principal component analysis, or PCA, to find derived input variables that represent the input dimensions better. Although linear combinations of input dimensions aren't 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 don't 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, 2013j). 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 select the split which returns the least dispersed subsets is selected recursively 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 the classical random forests (Cutler & Zhao, 2001).

In Search of the Super-Forest

The following section will look at some of the random forest variants that have already 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 technically adept users in the social sciences.

Alternative: PERT. PERT, a variant in the random forests framework was developed by (Cutler & Zhao, 2001). The nice quality of PERT, is that its strong reliance on randomness gives it a simple structure that is easier to describe and implement than the classical random forests by (Breiman, 2001). In fact, PERT is

simple enough that its algorithm is completely described in the following quote from (Cutler & 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 nonterminal 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 doesn't depend on randomness introduced through bootstrapping. On the other hand, bagging is necessary if one requires out-of-bag error estimates or variable importance measures. 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 advantage of PERT over random forests is speed. (Cutler & Zhao, 2001) compare the different runtimes of classical random forests 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 doesn't suffer from the bias introduced by it, thus, variable importance estimates are less affected. 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 don't depend on an optimality criterion to select their splits, PERT has the advantage of being able to deal with perfect interaction effects. A disadvantage is that PERT forests are even more difficult to interpret than the classical random forests, but since this is rarely tried anyway, it probably doesn't 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 isn't the only random forest variant that emphasizes randomness in its algorithm for growing trees. (Geurts et al., 2006) suggested what they call extremely randomized trees. Despite their name, extremely randomized trees are actually closer to the trees of random forests by (Breiman, 2001) than PERT. Extremely randomized trees allow the user to configure the randomness of the constructed decision trees through an additional parameter in order to adapt the random forest algorithm to the problem domain of the 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, 2013i) model membership to a category by a value between one and zero, while fuzzy logic (Wikipedia, 2013h) 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, 2013p) Rotation forests have been suggested by (Rodriguez et al., 2006). Rotation forests are created by calculating a principal component analysis on a bootstrapped sample of the data for every tree. The input variables are split into subsets, and random values of the output variable are removed for the principal component analysis, which means that another parameter for the size of the input variable subsets is required. Rotation forests significantly outperform random forests, bagging, and boosting on many datasets (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.

Alternative: Unknown. 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 the how the scientific method has become a competition between researcher and randomness, 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 nothing that may not be changed. Three questions that naturally follow from this are:

1. Can a random forest incorporate all of the developed features?
2. How does one prioritize the different features and problems?
3. Is there a different and stronger representation for random forests?

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 don't overlap, combining them is certainly possible. The third

question is the hardest to answer, and it's only possible to say that time will tell. The most interesting question is the second. This report has mentioned many problems of random forests, for example the problem of the parameters. 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. 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 both, the parameter `mtry` and some of the bias in the variable importance measure.

Obviously, there is no single best solution, but this doesn't 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 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 research 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 research method. 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*. I also included keywords seen in interesting titles, like *fuzzy*, *perfect*, *full*, *balanced*, *extremely*, and *rotation*. I then looked for some of the referenced publications I read about in the papers found as mentioned above, and included (Strobl et al., 2008), (Ho, 1998), (Breiman, 1996), (Freund & Schapire, 1995), and (Long & 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.

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 can't remember the original sources, I include the pages on Wikipedia, which I used to refresh my memories.

Aim and Structure of the Paper

Possible options for this topic were the presentation of exemplary uses of random forests, the discussion of strengths and weaknesses, and any of the more specialized variants. During my research I had the impression, that there were serious differences in the understanding of random forests among researchers, and even among designers of improved variants. A study comparing different decision tree ensemble techniques also

confirmed this expression by saying that many of the commonly used methods of comparison weren't robust enough for use with random forests (Banfield et al., 2007).

Because of this fuzziness, I decided on a different focus, 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 parallelization, as the small example implementation demonstrates, which is an important criterion for everyone who wants to analyze large datasets. On top of that, they are easy to implement and don't require a sophisticated mathematical understanding. In fact, a lot of the random forest variants have been developed by combining known concepts and later proving the viability using a comparison of methods study on real-world and simulated datasets.

However, it has to be said that random forests are problematic, because they appear as easy to use methods, but expose a lot of their inner workings to the user. The user shouldn't have to worry about parameters, and how they interact with a dataset to introduce bias in error estimates and variable importance measures. In studies where only small ensembles are used, one has to wonder if the researcher didn't just get lucky with his random number generator's seed value. In some ways (Banfield et al., 2007) also ask this question, because many of the significantly better results just disappeared 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 study of its qualities, and the qualities of its variants, but this seems not to be the case. Most comparisons still pitch a newly introduced variant against random forests (Breiman, 2001), bagging (Breiman, 1996) and Adaboost (Freund & Schapire, 1995). One of the factors for this I believe to be the lack of a standardized and automatic 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 standardized, new algorithms could be measured by the push of a button, and authors wouldn't 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 feedback on why their method might be particularly good or bad when applied to a certain type of data. Such a standardized test repository would also lighten the load of all the researchers just using random forests as a tool. They wouldn't 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 certainly is beneficial for research in general, but one doesn't have to stop there. By standardizing research methods, their development and comparison in general, it also becomes easier to replicate a study's results using different methods on the same input data, and the datasets of a study can be included in the corpus of testsets for future testing of methods.

The "Methods" section of a study and the research methods mentioned in it, shouldn't 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 is what 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 an implementation of a method, just because software or random forests are not their field. Default values in implementations do change, and this change sometimes happens 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 an 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, doesn’t 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 and parameters, it should be trivial to cross-validate all findings with all datasets and all variants, but as it stands this won’t be done anytime soon, and since one doesn’t know if another study like (Banfield et al., 2007) overturns random forests in the near future, for most researchers in other fields it’s a safer bet to stick to traditional methods, to ignore problems with many input variables, or to not dare to invent models with complex interactions that are difficult to represent using traditional methods.

Appendix

The following Clojure source code is the complete example implementation of random forests. Parts of it are being explained earlier in this paper.

```
(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)))

(def rf (rf-train sample 1 3))

(every? #(= (rf-predict rf (dissoc % :class)) (:class %))
        sample)

(rf-error rf)
```

References

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

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`

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

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

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

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

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., 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). *Artificial neural network.* Retrieved from https://en.wikipedia.org/w/index.php?title=Artificial_neural_network&oldid=551483619

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

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

Wikipedia. (2013e). *Database.* Retrieved from https://en.wikipedia.org/w/index.php?title=Database&oldid=552856988

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

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

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

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

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

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

Wikipedia. (2013l). *Linear regression.* Retrieved from https://en.wikipedia.org/w/index.php?title=Linear_regression&oldid=553168797

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

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

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

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

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

Wikipedia. (2013r). *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.*