Generalized weighting for bagged ensembles

by

Hieu Trung Pham

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Industrial Engineering

Program of Study Committee:
Sigurður Ólafsson, Major Professor
Daniel Nordman
Gül Okudan-Kremer
Justin Peters
Lizhi Wang

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation/thesis. The Graduate College will ensure this dissertation/thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2018

DEDICATION

To family and friends.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

Thank you Siggi, Dan, Gül, Justin, Lizhi, Heike, Steve, and Lawrence.

ABSTRACT

Ensemble learning is a popular classification method where many individual simple learners contribute to a final prediction. Constructing an ensemble of learners has been shown to consistently improve prediction accuracy over a single learner. The most common types of ensembles include: bootstrap aggregated (bagged), boosted, and stacked. Each are different, yet has the same foundation of combining multiple learners.

In this dissertation, we focus our attention to bagged ensembles; namely we propose a generalization by way of model weighting. The new method is motivated by the potential instability of averaging predictions of trees that may be of highly variable quality. To alleviate this, we replace the usual arithmetic average with a Cesáro average for weighted trees in the random forest. We provide both a theoretical analysis that gives exact conditions under which we would expect this weighted ensemble approach to do well, and numerical analysis that shows the new approach is competitive to other bagged ensembles when training a classification model on numerous realistic data sets.

Going a step further we generalize our weights such that we allow simultaneous control over bias and variance. In particular, we introduce a regularization term that controls the variance reduction for bagged ensembles. Therefore, a new tunable weighted bagged ensemble framework is proposed, resulting in a very flexible method for classification. Using this methodology, we explore the impact tunable weighting has on the votes of each learner in an ensemble.

To aid in the applicability of this body of work, the author discusses an R package that allows users to implement our proposed weighting scheme to arbitrary bagged ensembles. The

package provides tools for constructing tunable bagged ensembles in the form of weights and is titled **wbensembleR**.

CHAPTER 1. INTRODUCTION

1.1 Background

Data Science, as currently termed, is not just statistics, mathematics, or computer science, yet integrates all three. Today data science can be defined as an interdisciplinary field that employs techniques and theories of the aforementioned subjects to extract knowledge or insight from data (Blei and Smyth, 2017). The rapid growth and popularity of this new field is apparent; various universities as well as a plethora of online platforms around the world are offering degrees specializing in data science.

The foundational aspects of data science are well set. In *50 Years of Data Science*, Donoho (2017) describes data science as *six divisions*:

1. Data Exploration and Preparation
2. Data Representation and Transformation
3. Computing with Data
4. Data Modeling
5. Data Visualization
6. Science about Data Science

With regard to the first division Hadley Wickham, a well-known contributor in the world of R and data science created **tidyverse**, a collection of R packages, for the specific purpose of systematically formatting *messy* data (Wickham, 2016). Wickham's *Tidy Data* (Wickham,

2014) estimates that 80% of data science is spent on preprocessing data, that is, cleaning and preparation. Certainly Wickham's point is well justified as an intuitive understanding of data is necessary before meaningful conclusions can be inferred.

The third and fourth division can be seen in parallel with *machine learning*. Max Kuhn's book, *Applied Predictive Modeling*, defines machine learning as the use of tools that utilizes information to find relevant patterns (Kuhn and Johnson, 2013). Acknowledging Kuhn's definition, machine learning is separable into two parts: supervised and unsupervised learning. The former is a way to describe problems where the outcome is known, that is, predictive modeling. Whereas the latter allows a computer to find relationships between data without a known goal, in other words, clustering. For an application of (hierarchical) clustering to food hub systems refer to (Mittal and Krejci, 2017) and (Mittal, 2016).

Both subsets of machine learning play a vital role in data science and each possesses their own unique challenges. For the scope of this dissertation, we limit ourselves to supervised learning; specifically, we only consider classification models as opposed to regression models.

Given a standard supervised learning problem, a machine learning model is given training data of the form $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$ where $\mathbf{x}_i$ denotes a vector of predictor variables or *features*, and y_i denotes a response. If $Y = \bigcup_{i=1}^{n} y_i$ is a finite set of values such as: {Yes, No}, {Low, Medium, High}, {0,1}, etc, we consider this to be a *classification* problem. If Y is expected to be a continuous response we take this as a *regression* problem. For the purposes of our research, we focus on two-class classification problems.

1.2 Motivation

A few common machine learning models for classification are: decision trees (Breiman et al., 1984), k-nearest neighbors (KNN) (Cover and Hart, 1967) and naïve Bayes (Rish, 2001). Each utilizes data in different ways and possess unique scenarios in which they

perform well. For example, a decision tree's structure exploits hierarchy in the importance of features. Naïve Bayes expects predictor variables to be independent, and KNN assumes feature variables can be mapped meaningfully into a metric space.

Since these can be considered *simple learners*, they are not without fault. Namely, each model has low bias but high variance (Kuhn and Johnson, 2013). Every predictive model contains error from bias and variance with the amount of each being determined by the data and model choice. *Bias* is defined as a model's understanding of underlying relationship between features and target outputs; whereas, *variance* is the sensitivity to perturbations in training data. Naturally, a model with low bias and low variance is desired but not always producible. One common approach to reduce variance among learners is to create a bootstrapped aggregated ensemble of models where an ensemble can be defined as a collection of individual objects; in our case, the objects are machine learning models.

An ensemble of learners has been shown to consistently improve prediction accuracy over a single learner (Zhou, 2012). Bagging and boosting are the most common ensemble methods, each with distinct advantages (Quinlan, 2006). While boosting methods are typically very tunable with numerous parameters, to date, the type of flexibility this allows has been missing for bagging methods such as the random forest (Breiman, 2001). Indeed boosted models, namely boosted trees, seem to be winning a variety of data science competitions leaving the random forest seemingly forgotten. In **xgboost**, a popular boosting package in R and Python, there exists a variety of parameters that users can tune to better fit their data sets (Chen et al., 2018). However, as of now there is not a package nor any foundation theory that allows tunability for bagged ensembles.

Bagging is an ensemble method for reducing variance of simpler learners with low bias but high variance (Breiman, 1996a), but there is also evidence that simply reducing variance is not always the best approach. In fact, it is reported that bagging works well with decision stumps (Martínez-Muñoz et al., 2007), which have higher bias but lower variance than un-

pruned trees. And while the best known bagging approach, namely the random forest, uses unpruned trees, its feature sampling has a similar effect. Namely, restricting the number of features available to a tree will on the average increase the bias. This is analogous to methods that focus primarily on reducing bias. Boosted trees are a popular ensemble method where the purpose of boosting is reduced bias, but implementations of boosted trees include one or more regularization parameters that are used to control the variance (Schapire, 2003). Similarly, deep neural networks are successful because adding layers to the network reduces bias, but at the same time numerous regularization parameters are used to reduce variance (Schmidhuber, 2015). However there is a cost. Namely the training times for these models are long and there can be upwards of 30+ parameters to tune in an implementation. Moreover, the tunability used to reduce bias and control for variance is not an easy task to determine optimal parameters to have minimal bias and variance.

The success of such highly tunable methods motivates our new ensemble approach. While the main idea of bagging is to reduce variance, we add tunable weights to the ensemble that allows for simultaneously controlling the bias. In this dissertation, we acknowledge these short comings and seek a way to address these missing components. The authors develop a framework which provides parameters for added flexibility to bagged ensembles. Specifically we propose a new tunable weighted bagged ensemble methodology, resulting in a flexible method for classification. We present this in incremental improvements. Namely,

1. Consider a new weighted scheme for the random forest that we call the Cesáro Random Forest

2. Generalize our weighted scheme to arbitrary bagged ensembles that allows for the simultaneous control of bias and variance using just two parameters

Using this methodology, we explore the impact tunable weights have on each learner in an ensemble and compare the results with the best known bagged ensemble method, namely

the random forest. Each methodology is expanded upon in separate chapters.

1.3 Ensemble Learning

Here we discuss three popular ensemble methods, each with their unique benefits. In practice, ensembles are a popular method for obtaining very high accuracy by combining less accurate models (Dietterich, 2000). Hence it is understandable why this remains a popular choice for predictive modeling.

1.3.1 Bootstrap Aggregating (Bagging)

In statistics, bootstrapping is a well known resampling with replacement technique with a firm theoretical foundation. However, in terms of data mining, we only consider bootstrapping in the context of building predictive models. For a concrete example, consider a data set $\Gamma = \{(X_i, Y_i), i = 1, ..., N\}$ where X_i are the predictor variables or features, Y_i is the class and N is the total number of observations. Let $\Gamma_1^* \subset \Gamma$ denote the bootstrap sample of observations. From Γ_1^*, model one is trained and tested against $\Gamma \backslash \Gamma_1^*$, the complement of Γ_1^* in Γ, where the out of bag (OOB) error rate for model one is found by calculating the number of wrong predictions. These steps are then repeated up to Γ_k^*, where k is a user parameter for the number of models desired (Azizi et al., 2016). In a classification problem each model provides a vote for its prediction of the class of each observation. The class with the greatest number of votes, amongst all models, is the final casted prediction. Considering only two-class data sets we can relabel classes as $\{0, 1\}$. This formulation allows calculations of an arithmetic mean since the votes form a sequence whose terms are zero or one. Namely, if the average of this sequence is greater than .5 then class 1 is predicted, and 0 if the average is less than .5. In the event of a tie the implementer decides the class label, but typically a class is chosen at random. One can think of this type of ensemble as a majority voting

scheme where the class with the greatest number of votes wins.

Breiman (1996a) provides a theoretical and numerical treatment of bagging learners such as: decision trees, KNN and more. His numerical results show that bagging can consistently outperform a single learner.

Arguably, the most popular bagged ensemble learner is the random forest (Breiman, 2001). The random forest is an ensemble of bagged decision trees with the notable exception being at each node only $\sqrt{p}$ attributes, where p is the total number of features, are considered for splitting and no pruning of the tree is performed. Breiman (2001) first suggested bagging; then the creation of the random forest as a specific bagging method followed. However, aside from the number of trees generated there are no tunable parameters for the random forest or any bagged ensemble.

With that said, an important question when creating an ensemble of learners is deciding its size, namely, the number of models in the ensemble. Oshiro et al. (2012) suggested that the random forest should have between 64 - 128 trees; by doing this there is a balance between computational time and accuracy. However, in general, this is not an easy question to answer. Namely, as the number of classes in your data set increases so should the number of models in the ensemble (van Dop and Steyn, 1990). Dietterich (2000) gives a detailed explanation of bagged ensembles from a numerical perspective and compares them to other ensemble methods such as boosting.

Still today, ensembles are widely used across various disciplines. In the area of health care, Bashir et al. (2016) applied a weighted ensemble to predict heart disease in patients. Whereas Valentini et al. (2004) constructed a bagged ensemble of support vector machines for cancer recognition. Both were able to accomplish their goal in a more accurate manner compared to other existing methods.

Moreover, bagged ensembles is still an active research area. In the area of deep learning, Mosca and Magoulas (2018) considered a bagged ensemble as a distillation method for reg-

ularization. From their findings, they were able to conclude that ensembling smaller neural networks, one can approximate the structure of a deep neural network. From the perspective of class imbalance, Collell et al. (2018) combined bagged ensembles with threshold-moving to adapt the performance measure using a natural class distribution. Their work provides insight and a new method to combat class imbalance with bagged ensembles. From this summary, it is clear that bagged ensembles continually play an active role in theoretical and practical research. It goes without saying that our work of tunable parameters will be a welcomed and popular addition for this field.

1.3.2 Boosting

Boosting, an idea similar to bagging, primarily relies on many weak learners; that is, models which perform slightly better than random. However, instead of a large ensemble containing many models, the result of boosting is a single strong learner; namely a classifier that is well correlated to the underlying structure of the data (Setak et al., 2017). Unlike bagging, which attempts to reduce variance, boosting aims to reduce bias. It is proven that as long as the performance of each individual learner is better than random guessing, the final model will converge to a strong learner.

Historically, two boosting methods were discovered by Schapire (1990) and Freund (1995) independently. However, neither were able to fully utilize the structure of the weak learner.

To combat these issues, Freund and Schapire (1996) worked together to produce a new boosting algorithm called adaptive boosting or *adaboost*. This algorithm is very popular and perhaps has the most significance and impact in the machine learning world. For their work, they were awarded the Gödel Prize in 2003 (Freund and Schapire, 1997).

The procedure for adaboost is as follows, a sequence of weak learners is generated where at each step in the learning process the algorithm finds the best classifier with respect to observation weights. The instances which are incorrectly classified in the ith step are given

more weight in the $i + 1$st step, whereas samples that were correctly classified receive less weight. From this, observations which are difficult to classify are given increasing larger weight until the algorithm constructs a model which can correctly label these observations. At each stage, the new step weight is computed based on the error rate of that iteration.

Similar to bagging, boosting can be applied to a variety of classification models; however, some learners should be considered over others. For example, decision trees and stumps are popular choices for boosting since these can satisfy the weak learner assumption due to the ability to restrict depth (Breiman, 1998).

With the theory and applicability of adaboost established, many generalizations followed. Some examples include, the logitboost, gradient boosting, extreme gradient boosting and stochastic gradient boosting (Schapire, 2003). The collection of these models is jointly termed gradient boosting machines. Each differs in a unique way; however, all being boosting methods, they have the goal of converting weak learners into a single strong leaner.

Boosting, particularly extreme gradient boosting (XGBoost) of decision trees, has become the most reliable method for data scientists to achieve state-of-the-art results. In 2015 Kaggle, a well-known data science website, held 29 competitions. Of the 29 winning solutions, 17 utilized XGBoosting in some form (Chen and Guestrin, 2016).

Aside from Kaggle competitions, the significance of boosted ensembles is well represented. Recently, Gupta et al. (2015) quantified the influence music has on the moods of listeners by constructing a new framework called *Boosted Ensemble of Single feature Filters Models*. This model improved the signal to noise ratio by a factor of 1.92 over the best baseline models. In some theoretical work, Jing et al. (2008) altered the structure of a Bayesian network to incorporate a boosted structure with numerical results supporting the benefit of their findings. From these examples, one can see the impact Fruend and Schapire had on the data science world.

However, boosting is not without faults. On some data sets, boosting methods will

severely overfit; that is, not generalize to a new data set well. Indeed, Buhlmann and Hothorn (2007) and Jiang (2000) both explain the consequences of boosting models and overfitting. This overfitting can be associated to the fact that boosting intentionally tries to learn each observation exactly. However, most implementations of boosting algorithms have a plethora of tunable parameters to combat this; such as restricting the number of learners or altering a learning rate (Fattahi et al., 2015). Vezhnevets and Barinova (2007) and Elith et al. (2008) give further insights about how modifying the traditional boosting algorithm can reduce overfitting. This versatility for boosted ensemble is a big advantage that general bagged ensembles do not have.

1.3.3 Stacking

Although less common than bagging and boosting another ensemble learning approach is stacking (Zenko, 2004). This involves the construction of a wrapper algorithm that combines the predictions of numerous other learning algorithms. Unlike the aforementioned ensembles, which only consider a single model, stacking involves the collection of different learning algorithms.

In detail, stacking can be visualized in two levels. To begin, one must first separate the training data into two parts, say X_1, and X_2. The first step is building many distinct learners on X_1 with complete parameters. Once these are built, one proceeds to evaluate their performance on X_2. Using the results of X_2, we create a new data frame which has the same number of rows as our original training data, but the difference being that the column labels are now each of the models with results from X_2. On the second level, a new wrapper learning model is used to perform the final prediction on the actual testing data. In practice a logistic regression is commonly used for binary classification.

This concept of stacking was first developed by Wolpert (1992). However his paper contained no theoretical foundation but simply an ad-hoc analysis. These problems were

somewhat solved by van der Laan et al. (2007) where the authors established that stacked ensembles represent an asymptotically optimal learning system. Although, the authors attempted to put stacked ensembles on top of a pedestal, they were not completely decisive in constructing the theoretical foundation. Today, stacked ensembles is seen as a black box algorithm. In *Stacked Ensemble Models for Improved Prediction Accuracy* (Gunes et al., 2017), a group of researchers from the SAS Institute, attempts to give an intuitive explanation to stacked ensembles, but lacked sufficient details for a complete theoretical framework.

Although bagging and boosting are given the most attention, it does not imply that stacked ensembles are not of value. In fact the top two performers in the Netflix competition utilized a form of stacking called blending (Sill et al., 2009). Moreover, Ozay and Vural (2012) showed that stacking can be successfully applied to classification and even outperform adaboost and the random forest.

In specific scenarios, constructing a stacked ensemble has seen considerable interest in the area of health care. Both Bhasuran et al. (2016) and Ekbal and Saha (2013) considered a stacked ensemble approach to draw out additional information in their data. Bhasuran et al. (2016) developed a classifier combined with fuzzy matching to recognize rare diseases. Whereas Ekbal and Saha (2013) applied his ensemble to understand and design insights into their data. Although one can consider this ensemble type to be a black box, it is still of use and widely applicable to various fields.

1.4 Weighted Averages

In this section, we explain different types of weights for consideration in a weighted average. The traditional mean is given by the arithmetic average; that is, $\frac{1}{|I|}\sum_{i\in I} i$ where I is some finite set. However, given any finite set, we can perform a weighted average: $\sum_{i\in I} w_i i / \sum_{i\in I} w_i$ where w_i is a vector of weights. The strength of the weight is usually determined

by the application area or the implementer. Below we review different weighting methods to determine ways to obtain w_i.

1.4.1 Out of Bag Error

Given that our focus is restricted to bagged ensembles, we initially consider weights utilizing the out of bag error. As previously mentioned, recall that when bagging, random observations are sampled with replacement from a data set, Δ. Hence, the observations that were never sampled are used as a testing set and an error value is obtained, this is defined as the out of bag error (OOB). Breiman (1996b) gives a thorough study of this in his paper, *Out-of-Bag estimation*. With this established, one can develop weights, w_i, that are proportional to the error of that model. Naturally given a set of OOB errors $\{o_1, ..., o_n\}$, we can define our weights to be $w_i = 1/o_i$. In practice we have $o_i \in [0, 1]$; thus, $o_i = 0$ is certainly possible. In this case, we can define some arbitrary weight value as long as the value is the largest amongst all w_i. From this, it is apparent that as $o_i \to 1$, the strength of the weight decreases. Using the OOB for a weighted error measure is the most natural method in the context of bagged ensembles.

Specifically, El Habib Daho et al. (2014) proposed a weighted stacking method using the out-of-bag error as a way to measure tree importance. Their research provided some fruitful results of improved prediction accuracy. Moreover, Li et al. (2010), Ronao and Cho (2015), and Winham et al. (2013) all suggested weighted trees again using the OOB as a measure of tree importance. Ronao and Cho (2015) and Winham et al. (2013) were able to increase the prediction accuracy of their desired classification problem; however, they did so in an ad-hoc nature. Namely, they were only to attempting to answer a specific question. On the other hand, El Habib Daho et al. (2014), and Li et al. (2010) developed the theory of their weighted schemes, but were unable to determine specific factors where their weighted scheme worked best.

As we can see, using the OOB as a measure corresponding to a weighted average is indeed popular. As previously stated, this the most common and natural measure.

1.4.2 Cesáro Averages

Given a real-valued sequence $\{a_n\}_{n=1}^{N}$, the *Cesáro averages* are the terms of $\{c_n\}_{n=1}^{N}$, where $c_n = \frac{1}{n}\sum_{k=1}^{n} a_k$. For simplicity, we refer to $\{c_n\}$ as the *Cesáro sequence*

For a bagged ensemble with n models, let $\{v_i\}_{i=1}^{n}$ denote the sequence of votes up to model i. By applying the Cesáro averages to $\{v_i\}_{i=1}^{n}$ we have the weight associated to vote v_i to be $w_i = \sum_{j=i}^{n} \frac{1}{j}$.

We further discuss the Cesáro averages in Chapter 2.

1.4.3 Bayesian Model Averaging

Considering a probabilistic approach, Hoeting et al. (1999) provided a method called Bayesian Model averaging (BMA) where weights are computed from a posterior distribution on the data set Δ. Graefe et al. (2015) applied this idea to forecasting in social sciences. Essentially BMA attempts to average over all possible combinations of feature variables as it attempts to best model the data set under uncertainty. Indeed BMA assigns higher weight to models that differ significantly from other models in the ensemble. However, some concerns are that lower weight can be assigned to two different models but each model contains unique information. Moreover, it is possible that one model may be given all the weight due to BMA's convergence properties; in which case, this becomes an optimal model selection problem. That is, BMA becomes a feature selection method. In fact, Wu et al. (2015) has shown that using BMA for feature selection can improve the accuracy of the typical naïve Bayes classifier.

1.4.4 Borda Count

A different type of weighted voted that has been applied by Ponti (2011) is the Borda count (Saari, 1985) which has its roots from decision theory (Adressi et al., 2016). In this method, each vote is given a point ranking usually in reference to the number of classes. In the end, the class with the largest point total wins. As an overview of the Borda count, Emerson (2013) gives a thorough review of the original Borda count method from a numerical and theoretical perspective. However, this sort of measurement system is typically applied in cases where there are more than two classes such as with elections (Klamler, 2004) or sports hall of fame voting (Fraenkel and Grofman, 2014) . However, since our scope is limited to two-class data sets, a weighted vote system such as the Borda count cannot be effectively applied.

Related to the Borda count is also the system where better models are given more than one vote. That is, in an ensemble if one model is much better than others, that model should be given at least two plus votes while the others simply cast a single vote. However, the problem of this is how to decide how many extra votes the model should receive as well as determining the better models.

1.5 Organization of Dissertation

Chapter 1 contains the introduction, motivation and literature review necessary for a comprehensive understanding for the remaining parts of this dissertation. Next, Chapter 2 presents the Cesáro Random Forest (CRF) supported theoretically and numerically. Chapter 3 allows for a generalization of the ideas in Chapter 2. That is, we allow for arbitrary learners and a generalized weighting scheme. An R package is presented in Chapter 4 which allows for users to implement the ideas from Chapters 2 and 3. Finally, Chapter 5 contains the conclusion along with future work ideas.

CHAPTER 2. CESÁRO RANDOM FOREST

2.1 Introduction

The random forest (RF) methodology is a widely used machine learning technique for solving classification and regression problems (Breiman, 2001). It has been applied in various applications spanning many disciplines. To give a couple of recent examples, Naghibi et al. (2017) applied the random forest model in the domain of geography for potential water mappings. Whereas Subasi et al. (2017) successfully diagnosed kidney disease using the random forest.

The heart of the random forest lies with the construction of an ensemble of weak learners, namely, decision trees employing a restricted subset of features or predictor variables. These learners are flexible, easy to visualize, and have fast training times (Friedman and Hall, 2007). The usual implementation of the RF involves the creation of many individual decision trees through a bootstrap resampling process. That is, each tree is generated by a random subset of possibly repeated data points. Since some data points are not included in each resampled subset, for each tree, we have an associated out of bag (OOB) error. This error estimates how accurate each tree is when applied to data that was not used to generate the tree. Additionally, at all nodes, each candidate split is made from a random subset of features and evaluated by the Gini impurity index (Strobl et al., 2007). Typically, for a classification problem with n features, $\sqrt{n}$ features are considered at each split. Unlike in the case of a single decision tree no pruning is done.

While the RF is a highly effective method, a potential downside is that predictions are sometimes highly volatile due to the purposeful randomness, that is, the resampling of the

training data and the subspace sampling of the predictor variables. Because of this, if we run the RF algorithm repeatedly it is possible to get a different prediction each time. Asymptotically the random forest should be stable; in other words, using an arbitrarily large number of trees one should get repeatable predictions. However, if this convergence is slow it may cause difficulties.

Ideally, given a machine learning model we would hope that the same (correct) prediction is made each time. However, this is not always true for any algorithm, including the random forest. To illustrate this phenomena take the classical *Titanic* data set (Hendricks, 2015) where we attempt to predict the gender of a passenger. Running the classical random forest 10 times to predict the 341^{st} observation, we obtain the following predictions using a 100 tree ensemble:

{Male, Female, Female, Male, Female, Male, Male, Female, Male, Female}.

For this small ensemble the RF thus predicts the two class values equally; however, this improves as we increase the number of trees. From a 1000 tree ensemble we have a 6/4 divide in favor of males. Using 20000 and 50000 trees we obtain 8/2 and 10/0 split, respectively. Indeed, for a large number of trees we obtain stable predictions for male, which is the correct class; however, requiring a large ensemble for a small data set seems impractical. Although this is certainly an atypical example, this serves to illustrate our point.

In this chapter our goal is twofold. For one, we propose a numerically weighted tree scheme to add stability; that is, create a modified RF so that repeated runs will produce the same or near-same predictions. Secondly, with this added stability we want to study the impact on the accuracy for two-class classification problems. To test the behavior of this proposed method, we apply this to classical data sets commonly used in the data mining research community.

2.2 Methodology

2.2.1 Cesáro Averages

To increase the stability of the RF we propose using Cesáro averages (Stein and Shakarchi, 2003).

Definition 2.2.1. *Given a real-valued sequence* $\{a_n\}_{n=1}^{N}$*, the Cesáro averages are the terms of* $\{c_n\}$*, where* $c_n = \frac{1}{n}\sum_{k=1}^{n} a_k$*. For simplicity, we refer to* $\{c_n\}$ *as the Cesáro sequence*

The historical roots of the Cesáro averages comes from the field of harmonic analysis. Typically, this is used to give quasi-convergence to a divergent sequence. For example, consider the infinite sequence $\{b_n\} = \{1, 0, 1, 0, ...\}$. Clearly, this does not converge to any real number due to its oscillating behavior. Intuitively, one might be inclined to say that the *limit* of the sequence is 1/2. This intuition would serve well as one can easily verify that the Cesáro sequence does indeed converge to 1/2. Hence for a divergent series, we can add stability for possible convergence and thus motivates using such averages in our context. A well-known result shows that if a sequence converges to a number c, then the Cesáro sequence converges to c as well. However, the converse is not necessarily true. For our purposes, we only consider a sequence of finite length. Namely, the number of trees generated dictate the length of our sequence.

For the duration of this chapter we let $\{v_i\}$ denote the sequence of votes up to tree i for a given instance, and $\{\tilde{v}_i\}_{i=1}^{n}$ denote the cumulative average of $\{v_i\}_{i=1}^{n}$ up to term i, that is, $\tilde{v}_n = \frac{1}{n}\sum_{k=1}^{n} v_k$. We define the n^{th} term of our Cesáro sequence to be $c_n = \frac{1}{n}\sum_{i=1}^{n} \tilde{v}_i$. From this, it is determined that the vote casted, for a single instance, by the entire random forest is based off the value of the last term in the Cesáro sequence, c_n. However, our construction of the Cesáro sequence does not make the numerical weights applied to each tree clear. To address this we refer to the following lemma.

Lemma 2.2.1. *Given a random forest with n trees, let* $\{v_i\}_{i=1}^n$ *denote the sequence of votes up to tree* i*. By applying the Cesáro averages to* $\{v_i\}_{i=1}^n$ *we have the weight associated to vote* v_i *to be* $w_i = \sum_{j=i}^{n} \frac{1}{j}$

Proof. Observe,

$$\begin{aligned} c_n &= \frac{1}{n}\sum_{i=1}^{n} \tilde{v}_i \\ &= \frac{1}{n}\left[\frac{v_1}{1} + \frac{v_1+v_2}{2} + ... + \frac{v_1+v_2+...+v_n}{n}\right] \\ &= \frac{1}{n}\left[\left(\sum_{i=1}^{n}\frac{1}{i}\right)v_1 + \left(\sum_{i=2}^{n}\frac{1}{i}\right)v_2 + ... + \left(\frac{1}{n}\right)v_n\right] \end{aligned}$$

Hence we have our weighted sequence $\{w_i\}_{i=1}^n$ where $w_i = \sum_{j=i}^{n} \frac{1}{j}$, with our normalizer being $\sum_{i=1}^{n}\sum_{j=i}^{n} \frac{1}{j}$. □

We first note that each weight can be represented as the partial sums of the harmonic series. From this equivalent representation of the Cesáro averages in terms of numerical weights, we have a clear schema for weighted votes. Additionally, from these weights there are some conclusions to infer. To begin, the first few votes are given heavy importance due to the construction of the weights. This can be readily seen by a simple observation of the summation. Indeed, the harmonic series grows slowly, so that terms in the beginning are much greater than the end. In fact, approximately 2.5×10^8 terms of the harmonic series are needed to sum to 20 (Weisstein, 2004). Due to such importance being placed at the beginning, it makes sense to order the trees in the *best* way possible with the best performing trees being placed at the beginning. We therefore need to address how to identify the *best* trees.

2.2.2 Cesáro Random Forest and Tree Sequencing

For our implementation of the RF we split our data set into two parts. Specifically, the training data set consists of 3/4 of the original sample with the remaining 1/4 being used for testing. Here we propose two different ways to sequence the trees, that is, giving the *best* trees the most weight. They are:

- **Order 1**: Out of Bag (OOB) Error Rates
- **Order 2**: Accuracy on a Second Training Set

As stated earlier, for each tree there is an associated out of bag error. Based on this, trees with a lower OOB error rate are better performers and should be placed first, that is, given more weight. Using this as a way to sequence our trees is the most natural method.

For Order 2, with our training set of size 3/4 the original we build a second training data set. This second set is size 1/4 of the first training set and essentially acts as an independent testing set. To be explicit, the trees generated from our first training set are tested against the second training set. From this we determine the best tree by evaluating which trees had the highest number of correct predictions.

Given the different types of sequencing and the weights based off the Cesáro averages, we have our algorithm for what we will call the Cesáro Random Forest (CRF) in Algorithm 1.

As previously stated the user determines what occurs if the weighted average equals 1/2. For our implementation we decided to label the class zero as opposed to deciding at random.

Algorithm 1 Cesáro Random Forest

Input n - Size of ensemble, Δ - Training data set, Λ - Testing data set
Output Vector of predictions

for i from 1 to n **do**
    Generate i^{th} decision tree, T_i, based upon classical random forest procedure
    Store out-of-bag error for T_i
    Test T_i on Λ to receive votes for each observation
end for
Sequence $\{T_i\}_n$ by out-of-bag error
Compute vector of weights using Lemma 2.2.1
for each observation j in Δ **do**
    Compute $c_j = \left[\frac{\sum_{i=1}^{n} w_i v_i}{\sum_{i=1}^{n} w_i} \right]$
    Store c_j
end for
return Vector $\{c_j\}$

2.3 Results

2.3.1 Theory

As noted earlier, due to the nature of the harmonic series, most of the mass of the partial sums is near the beginning. An easy way to visualize this is with the function $f(x) = 1/x,\ x > 0$, a close approximation to $\sum_n \frac{1}{n}$, which decreases rapidly to 0 and has the $\lim_{x \to 0^+} 1/x = +\infty$. From this reasoning we establish a few results.

Lemma 2.3.1. *Let $\{a_i\}_{i=1}^n$ denote a sequence where each term is either a or b. Suppose the first k terms are the same. If m of the remaining n-k terms are the opposite of the first k then the Cesáro weighted average of $\{a, ..., a, b, ..., b, a, ..., a\}$ will be closer to b than any other ordering where our weights are $w_i = \sum_{j=i}^{n} \frac{1}{j}$.*

Proof. Without loss of generality, define $\Phi : \{a, b\} \to \{0, 1\}$ as a bijection. Suppose $\{b_n\} = \{0, ..., 0, 1, ..., 1, 0, 1, 0, ..., 0\}$ is a worse ordering. Observe

$$\frac{1}{n}\left[\frac{1}{k+1}+...+\frac{3k-1}{4k-1}+\frac{3k-1}{4k}+\frac{3k}{4k+1}+...+\frac{3k}{n}\right]$$

$$<\frac{1}{n}\left[\frac{1}{k+1}+...+\frac{3k-1}{4k-1}+\frac{3k}{4k}+\frac{3k}{4k+1}+...+\frac{3k}{n}\right]$$

This is a contradiction to assuming $\{b_n\} = \{0, ..., 0, 1, ..., 1, 0, 1, 0, ..., 0\}$ was a worse ordering.

□

Essentially, Lemma 2.3.1 explains that the given sequence of trees will provide the worst scenario in terms of being numerically close to b. This simple result will become useful in application to other proofs.

Given that most of the mass of the harmonic series is given near the beginning, one might wonder if there exists a natural number, N, such that if the first N trees vote the same class the entire Cesáro RF will predict that class. That is, since the harmonic series is so dominant at the beginning, is there a point that the remaining trees have no significance if the beginning trees vote a certain way? This leads to our first theorem.

Theorem 2.3.1. *Let {0,1} be the two classes for a given data set. Let $n \geq 34$, where n is the number of trees. Given any instance, if the first $k = .2n \in \mathbb{N}$, trees predict the same class, then the Cesáro random forest will predict that class.*

The interpretation of Theorem 2.3.1 is that if the first $.2n$ trees cast the same vote then regardless of what happens with the remaining $.8n$ trees, the Cesáro RF will vote in-line with the first $.2n$ trees. From this we can see how important this implies the sequencing of trees to be. If we choose the best way to order the trees, then we are able to completely determine the outcome of any instance. However, such assumptions are too extreme in the sense that it is unlikely that the first 20% of trees will vote the same class or predict the correct class. This leads to our next theorem that provides a bit more flexibility.

Theorem 2.3.2. *Let {a,b} be the two classes of a data set. Let $n \geq 59$ be the number of trees. Given any instance, suppose the first $0 < k \leq .2n$ trees vote class {a}, $k, n \in \mathbb{N}$. If at most $3k$ of the remaining $n - k$ trees votes class {b} then the Cesáro random forest will predict {a} as the majority class.*

Before providing proof of these statements we must first reference a lemma that gives an asymptotic representation on the partial sums of the harmonic series.

Lemma 2.3.2.

$$\sum_{n=1}^{x} \frac{1}{n} = \ln(x) + \gamma + \mathcal{O}\left(\frac{1}{x}\right)$$

$$> \ln(x+1)$$

where γ is the Euler-Mascheroni constant.

Proof. Apostol (1976) page 55 proves the equality. For the inequality observe that since $f(x) = 1/x$ is a strictly decreasing function we have:

$$1 + \frac{1}{2} + ... + \frac{1}{n} > \int_{1}^{x+1} \frac{dt}{t} = \ln(x+1).$$

□

As further explanation of Lemma 2.3.2 we define the *Big O* notation ($\mathcal{O}(\cdot)$) in Definition 2.3.1.

Definition 2.3.1. *If $g(x) > 0$ for all $x \geq a$, we write $f(x) = \mathcal{O}(g(x))$ to mean there exists a constant $M > 0$ such that*

$$|f(x)| \leq Mg(x) \text{ for all } x \geq a$$

In Figure 2.1, we approximate M to be approximately $1/2$.

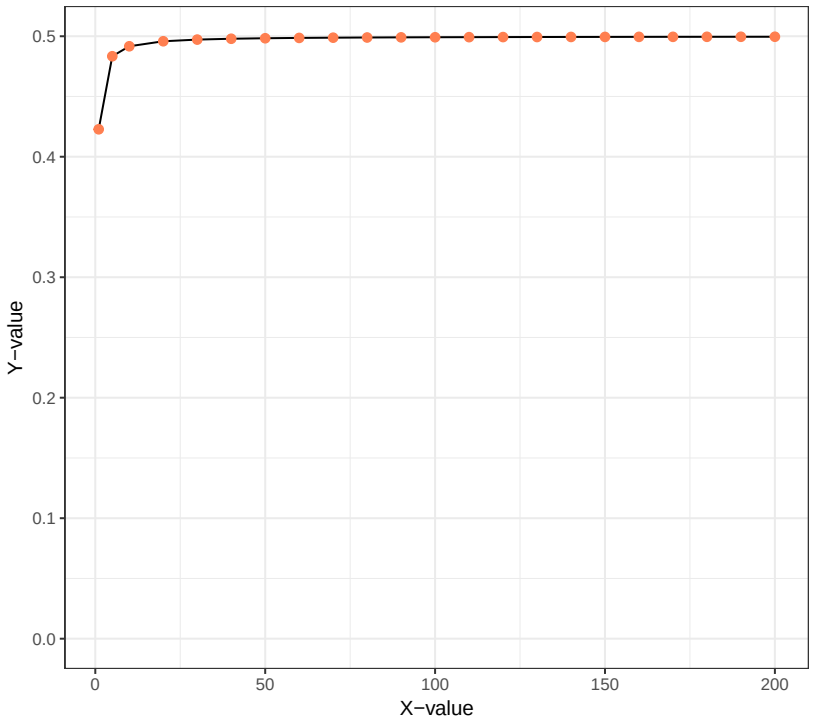

Figure 2.1: Asymptotic bound for Lemma 2.3.2

With our preliminaries set, we move forward to proofs of Theorems 2.3.1 and 2.3.2, respectively.

Proof of Theorem 2.3.1. Without loss of generality, suppose the first $k = .2n \in \mathbb{N}$ predictions for n trees are all of class 1. Set $\alpha = \frac{1}{1} + \frac{2}{2} + ... + \frac{k}{k} + \frac{k}{k+1} + ... + \frac{k}{n}$. It suffices to show that $c_n = \frac{1}{n}\alpha \geq .5$. Observe:

$$\begin{aligned} c_n &= \frac{1}{n}\alpha \\ &= \frac{1}{n}\left[\sum_{i=1}^{k} 1 + \sum_{i=k+1}^{n} \frac{k}{i}\right] \\ &= \frac{1}{n}\left[\sum_{i=1}^{k} 1 + k\left(\sum_{i=1}^{n} \frac{1}{i} - \sum_{i=1}^{k} \frac{1}{i}\right)\right] \\ &= \frac{k}{n} + \frac{k}{n}\left(\sum_{i=1}^{n} \frac{1}{i} - \sum_{i=1}^{k} \frac{1}{i}\right) \\ &= .2 + .2\left(\sum_{i=1}^{n} \frac{1}{i} - \sum_{i=1}^{k} \frac{1}{i}\right) \end{aligned}$$

$$
\begin{aligned}
&= .2 + .2\left(\sum_{i=1}^{n}\frac{1}{i} - \sum_{i=1}^{k}\frac{1}{i}\right) \\
&> .2 + .2\left(\ln(n+1) - \ln(k+1)\right) \qquad \text{by Lemma 2.3.2} \\
&= .2 + .2\left[\ln\left(\frac{n+1}{k+1}\right)\right]
\end{aligned}
$$

Our result follows for $n \in \mathbb{N}$ satisfying $.02 + +.2\left[\ln\left(\frac{n+1}{k+1}\right)\right] \geq .5$. Using a numerical solver, we obtain $n \approx 33.5869$. Hence for $k, n \in \mathbb{N}$ satisfying $k = .2n$, $n \geq 34$ we have our result.

□

Proof of Theorem 2.3.2. Define $\Phi : \{a, b\} \to \{0, 1\}$ as a bijection. Showing the result holds for worst case is sufficient, that is, without loss of generality, let $\{v_i\}_{i=1}^{n} = \{0, ..., 0, 1, ..., 1, 0, ..., 0\}$ be the sequence of votes. From Lemma 2.2.1, we have our weighted sequence $\{w_i\}_{i=1}^{n}$ where $w_i = \sum_{j=i}^{n}\frac{1}{j}$. It is enough to show that:

$$
\frac{\sum_{i=1}^{n} w_i v_i}{\sum_{i=1}^{n} w_i} = \frac{\sum_{i=k+1}^{4k} w_i}{\sum_{i=1}^{n} w_i} \leq \frac{1}{2} \text{ or equivalently, } \sum_{i=k+1}^{4k}\sum_{j=i}^{n}\frac{1}{j} \leq \frac{1}{2}\sum_{i=1}^{n}\sum_{j=i}^{n}\frac{1}{j} = \frac{n}{2}
$$

Observe,

$$
\begin{aligned}
\sum_{i=k+1}^{4k}\sum_{j=i}^{n}\frac{1}{j} &= \sum_{j=k+1}^{4k}\frac{j-k}{j} + \sum_{j=4k+1}^{n}\frac{3k}{j} \\
&= \sum_{j=k+1}^{4k} 1 - \sum_{j=k+1}^{4k}\frac{k}{j} + \sum_{j=4k+1}^{n}\frac{3k}{j} \\
&= 3k - k\sum_{j=k+1}^{4k}\frac{1}{j} + 3k\sum_{j=4k+1}^{n}\frac{1}{j} \\
&= 3k - k\left(\sum_{j=1}^{4k}\frac{1}{j} - \sum_{j=1}^{k}\frac{1}{j}\right) + 3k\left(\sum_{j=1}^{n}\frac{1}{j} - \sum_{j=1}^{4k}\frac{1}{j}\right) \\
&= 3k - k\sum_{j=1}^{4k}\frac{1}{j} + k\sum_{j=1}^{k}\frac{1}{j} + 3k\sum_{j=1}^{n}\frac{1}{j} - 3k\sum_{j=1}^{4k}\frac{1}{j}
\end{aligned}
$$

$$= 3k - 4k\sum_{j=1}^{4k}\frac{1}{j} + k\sum_{j=1}^{k}\frac{1}{j} + 3k\sum_{j=1}^{n}\frac{1}{j}$$

$$\leq 3k - 4k(\ln(4k)+\gamma) + k(\ln(k)+\gamma+\frac{1}{k}) + 3k(\ln(n)+\gamma+\frac{1}{n}) \qquad \text{by Lemma 2.3.2}$$

$$= 3k - 4k\ln(4k) + k\ln(k) + 3k\ln(n) + 1 + \frac{3k}{n}$$

$$= 3k - 4k\ln(4) - 4k\ln(k) + k\ln(k) + 3k\ln(n) + 1 + \frac{3k}{n}$$

$$= 3k - 4k\ln(4) - 3k\ln(k) + 3k\ln(n) + 1 + \frac{3k}{n}$$

Let $k = \alpha n$, this implies $0 < \alpha \leq .2$. We now have:

$$\sum_{i=k+1}^{4k}\sum_{j=i}^{n}\frac{1}{j} \leq 3\alpha n - 4\alpha n\ln(4) - 3\alpha n\ln(\alpha n) + 3\alpha n\ln(n) + \frac{3\alpha n}{n} + 1$$

$$= 3\alpha n - 4\alpha n\ln(4) - 3\alpha n\ln(\alpha) - 3\alpha n\ln(n) + 3\alpha n\ln(n) + 3\alpha + 1$$

$$= n(3\alpha - 4\alpha\ln(4) - 3\alpha\ln(\alpha)) + 3\alpha + 1$$

For $0 < \alpha \leq .2$, $g(\alpha) = 3\alpha - 4\alpha\ln(4) - 3\alpha\ln(\alpha)$ obtains a maximum at $\alpha = \frac{1}{4\sqrt[3]{4}}$, and $h(\alpha) = 3\alpha$ obtains a maximum at $\alpha = .2$.

Hence we obtain a new bound:

$$\sum_{i=k+1}^{4k}\sum_{j=i}^{n}\frac{1}{j} \leq n\left(\frac{3}{4\sqrt[3]{4}} - \frac{1}{\sqrt[3]{4}}\ln(4) - \frac{3}{4\sqrt[3]{4}}(-\frac{4}{3})\ln(4)\right) + \frac{3}{5} + 1$$

$$= \frac{3n}{4\sqrt[3]{4}} + \frac{8}{5}$$

To ensure $\sum_{i=k+1}^{4k}\sum_{j=i}^{n}\frac{1}{j} \leq \frac{n}{2}$, we want n large enough to satisfy $\frac{3n}{4\sqrt[3]{4}} + \frac{8}{5} \leq \frac{n}{2}$. This holds if

$$n \geq \frac{\frac{8}{5}}{\frac{1}{2} - \frac{3}{4\sqrt[3]{4}}} \approx 58.12$$

Thus, for $k, n \in \mathbb{N}$ satisfying $k \leq .2n$ and $n \geq 59$, we have our result.

□

In Theorem 2.3.2 we have a condition that is more attainable. Namely, we no longer need all $.2n$ trees to vote the same class to ensure the correct class as in Theorem 2.3.1. Now, if the first $k \leq .2n$ trees vote a particular class, then we just have to ensure that there are only so many wrong votes. From this, we do not imply from these two theorems that this is the only way to predict the correct class. We are simply giving a statement to guarantee the correct prediction if the assumptions of either theorem are satisfied. With this reasoning, we state some corollaries.

Corollary 2.3.1. *If the conditions of either Theorem 2.3.1 or 2.3.2 are satisfied and the first k votes are of the correct class, then the prediction accuracy of the Cesáro random forest will always be greater than or equal to the classical random forest.*

Proof. It is clear that if the classical random forest is 100% accurate then the Cesáro RF will also be 100% accurate. Now suppose the Cesáro RF and classical random forest differ on at least one vote. By assumption, we are guaranteed that the Cesáro RF will predict the correct class. Hence, in the instance where the classical RF predicted a different class, the Cesáro random forest will be correct. Thus we will either have greater or equal prediction accuracy. □

Although in practice one may not know the correct class, Corollary 2.3.1 can be seen as useful in another way. In an implementation of the Cesáro Random Forest algorithm if newly constructed trees do not contribute to the top 20% of *best* trees, then one can terminate the algorithm. That is, we have motivated a stopping condition for the number of trees needed in our ensemble. We formalize this in another corollary.

Corollary 2.3.2. *Given a Cesáro random forest with n trees, if any additional trees are constructed but not placed in the top 20% of all trees then the vote of the Cesáro random*

forest will be unchanged for a given observation.

Proof. Immediate consequence of Theorem 2.3.1. □

In practice we can select some finite number of m trees, and if none of those are in the top 20% of all trees it is reasonable to terminate. Specifically, if a user is unsatisfied with the performance of n trees, then constructing $n + m$ trees will have no effect on the prediction accuracy unless the m additional trees are sequenced in the top 20% of all trees. Hence, a cutoff to the number of trees can be determined for asymptotic stability and, from a practical standpoint, can reduce computational time.

2.3.2 Numerical Results and Interpretation

In this section we provide numerical results to compare the classification accuracy of the classical and Cesáro random forest, as well as compare different tree sequencing methods. For each data set we used 3/4 for training and reserved 1/4 for testing. All experiments were run using R, a statistical computing software. Recall that we only consider two-class data sets.

2.3.2.1 Results on Traditional Data Sets

To evaluate the effectiveness of our Cesáro RF, ten traditional data sets were tested. The data sets were acquired from the UCI (University of California-Irvine) repository (Bache and Lichman), CRAN (Hendricks 2015), or collected by the authors. Table 2.1 summarizes the data sets used.

To begin, we applied the Cesáro RF to each data set with different sequencing methods, and varying number of trees. For comparison, we also ran the classical RF implementation. In Table 2.2, we give the mean of each of the ten trials from 500 trees. In addition we also display the standard deviation of the classical random forest and the *best ordering*, where

Table 2.1: Data set description for CRF experiments

Dataset	Instances	Features	Source
Titanic	714	7	CRAN
Sonar	208	60	UCI
NBA Playoffs	683	92	Authors
Teaching Evaluations	101	5	UCI
Bupa Liver	345	7	UCI
Hill-Valley	606	100	UCI
Musk	476	167	UCI
Gisette	5999	5000	UCI
Medallion	2000	500	UCI
Arcene	200	10000	UCI

the *best ordering* is the sequencing method, aside from the classical RF, that provided the highest accuracy on average.

Table 2.2: Average accuracy % (500 trees, 10 trials) comparing CRF and RF

	Cesáro RF				***Standard Deviation***	
Dataset	**1**	**2**	**None**	**Classical RF**	**Best Order**	**Classical RF**
Titanic	**81.7**	80.4	80.6	80.3	**2.7**	2.9
Sonar	**84.0**	81.9	81.4	82.0	**7.1**	7.7
NBA Playoffs	94.0	94.9	94.0	**95.1**	1.3	1.3
Teaching Evaluations	72.3	71.2	69.4	**76.2**	**7.3**	9.2
Bupa Liver	**73.5**	**73.5**	70.5	72.5	3.1	**3.0**
Hill-Valley	**56.8**	55.1	53.1	56.3	**1.7**	2.0
Musk	**89.5**	87.4	86.1	89.2	**1.6**	1.8
Gisette	97.2	96.9	96.4	97.2	0.4	0.4
Medallion	**72.8**	69.5	66.5	69.2	**1.1**	1.2
Arcene	**82.9**	80.3	79.5	82.0	**5.8**	6.0

From Table 2.2, we can draw some conclusions about the Cesáro random forest. To begin, our method performs well relative to the classical random forest in 7 out of 10 data sets, and ties in another one. Additionally, this displays the usefulness of our ordering scheme.

However, note that there exists a clear distinction in the two different orderings. Specifically, *Order 2* consistently outperformed *Order 1*, sorting by OOB error. The ineffectiveness of *Order 2* is possibly due to the small size of training data for these examples. Nonetheless, we are confident in saying that ordering is better than not-ordering. Moreover, from observing the standard deviations, it can be determined that the Cesáro random forest does indeed provide more consistent results; thus providing the expected stability. Lastly, the Cesáro random forest seems to work well on problems with a large number of features. This is supported by large differences in the *Medallion*, *Arcene*, and *Musk* data sets.

2.3.2.2 Comparison of Methods

In Tables 2.3 and 2.4, we display results that compare the CRF and the RF with 100 trials with varying number of trees. We initially notice that:

1. For the RF there are minimal differences beyond the construction of 500 trees. That is, percent changes between the number of trees are at most $\pm.2\%$.

2. The CRF, in some cases, still sees improvements of at most $\pm.4\%$.

From this one may conclude that, in most cases, the RF stabilizes after 500 trees, which is the default parameter in most libraries. Additionally, the CRF still performs well relative to the RF. This can be seen as a fact that as one constructs more trees, there is still a chance that *good* trees will be constructed and placed near the beginning of the sequencing, and thus further increasing accuracy.

Additionally, we notice that using 500 and 1000 trees appear to work better than 100 or 2000 trees. In particular, the usage of 500 and 1000 trees seemly works best with the possibility that a number in between may be optimal. Although 100 and 2000 trees perform well relative to the classical RF, in some cases they are not as reliable. This can be contributed to the sensitivity of the Cesáro random forest. This sensitivity can be contributed

Table 2.3: Average accuracy % of CRF with ordering 1 and various number of trees (100 trials)

Dataset	100 Trees	500 Trees	1000 Trees	2000 Trees
Titanic	79.4	79.8	**79.9**	**79.9**
Sonar	81.7	**82.6**	82.1	82.3
NBA Playoffs	94.1	94.3	**94.4**	94.2
Teaching Evaluations	76.2	76.6	**76.9**	**76.9**
Bupa Liver	71.9	72.1	**72.2**	72.1
Hill-Valley	56.0	56.2	56.2	56.2
Musk	89.1	89.5	89.5	89.5
Gisette	97.0	97.2	97.2	97.2
Medallion	69.6	72.1	**72.3**	72.2
Arcene	80.5	81.6	81.5	**81.5**

to the small number of trees, and the strength of the weight being applied. By constructing just 100 trees and applying the Cesáro RF we are going to have a diverse range of trees being built, and hence by our weighting scheme a large emphasis is placed on possibly worse trees. However, this sensitivity can also be positive. This is in the sense that if we construct excellent trees, then more weight can be placed on them and the bad trees at the middle and end will have little to no impact on the voting outcome. This cannot be said for the classical random forest since the good and bad trees are averaged together.

2.3.2.3 Number of Trees Comparison

Further studying the ineffectiveness of 100 and 2000 trees, we observe Table 2.5. From this, we notice that ordering the trees may do more harm than good if too many or too few trees are constructed. In this case, not-ordering is competitive with *Order 1*. Yet again this can be attributed to the sensitivity of the Cesáro random forest. From this analysis, we observe that the construction of trees between 500-1000 performs competitively compared to that of the classical random forest.

In regards to different sequencing schemes, we notice that *Order 1*, sorting by OOB error,

Table 2.4: Average accuracy % of RF and various number of trees (100 trials)

Dataset	100 Trees	500 Trees	1000 Trees	2000 Trees
Titanic	79.6	79.6	**79.8**	**79.8**
Sonar	81.6	81.9	**82.0**	**82.0**
NBA Playoffs	94.9	95.0	**95.1**	**95.1**
Teaching Evaluations	76.0	76.5	**76.9**	76.8
Bupa Liver	72.5	**72.7**	72.5	72.6
Hill-Valley	56.0	**56.2**	56.0	**56.2**
Musk	88.9	**89.1**	89.0	**89.1**
Gisette	97.0	97.2	97.2	97.2
Medallion	67.4	68.9	69.2	**69.3**
Arcene	80.4	**80.9**	80.8	80.8

consistently provides the best accuracy. The ineffectiveness of *Order 2* can be associated with the lack of instances to train on. Indeed, for *Order 2* by further subsetting the training data set, we are constructing each tree off a smaller number of instances. This problem can most likely be remedied by obtaining a data set containing a very large number of instances.

2.3.2.4 Application of Ensemble Size Stopping Criteria

Here we provide an application of the stopping conditions on the ensemble size suggested by Theorem 2.3.1 and Corollary 2.3.2. Table 2.6 considers such a scenario where we construct a random forest on 100 trees with the *Sonar* data set.

An additional $m \in \{10, 15, ..., 35\}$ trees are added to the RF to be considered for a stopping rule. We successively append m trees until none of those m trees are placed in the top 20%. Each trial is run 100 times with the aggregated results displayed. The *tree range* is the minimum and maximum number of trees obtained by the 100 trials; whereas, the third column is the median ensemble size of the 100 trials. It should be noted that the size of the CRF and RF are the same.

From this, we immediately notice that the prediction accuracy of the CRF is greater

Table 2.5: Average accuracy % of CRF with ordering 1 with 100 and 2000 trees (10 trials)

	100 Trees		*2000 Trees*	
Dataset	**Order 1**	**None**	**Order 1**	**None**
Titanic	78.5	**78.9**	**79.9**	79.5
Sonar	80.6	**81.5**	**83.3**	82.9
NBA Playoffs	94.2	**95.5**	**95.3**	95.1
Teaching Evaluations	**72.7**	71.5	70.0	**72.7**
Bupa Liver	69.8	**72.8**	**72.3**	71.4
Hill-Valley	**57.1**	56.2	55.1	**55.6**
Musk	**88.1**	86.0	87.7	**88.2**
Gisette	96.9	**97.1**	**97.1**	96.6
Medallion	**70.3**	61.9	**73.0**	67.6
Arcene	**83.0**	78.5	**81.3**	80.1

than the RF and the standard deviation is at most its counterpart. Additionally, using a stopping rule of 20 trees we obtain the prediction accuracy of 83.1%. Although not the highest, using this criteria to construct 720 trees can be seen as a trade-off between size and accuracy versus building a random forest with 19070 trees.

Table 2.6: Sonar data set with a stopping criterion (100 Trials)

Additional		**Median**	**Cesáro RF**		**Random Forest**	
Trees	**Tree Range**	**of Trees**	**Avg. Acc.**	**SD**	**Avg. Acc.**	**SD**
10	[100 , 640]	140	82.3	5.2	82.0	5.3
15	[100 , 2050]	345	82.1	5.4	81.7	5.7
20	[130 , 4370]	720	83.1	5.1	82.6	5.1
25	[130 , 10000]	2745	82.3	4.9	82.0	5.2
30	[130 , 20000]	6205	83.6	6.0	83.4	6.2
35	[540 , 40000]	19070	83.7	6.1	83.3	6.1

2.4 Conclusion

From our numerical results we recognize that our weighted methodology performs well relative to that of the traditional random forest when applied to existing data sets. In addition, as hypothesized, the sequencing of trees plays a vital role. This is easily seen by observing the results of the traditional data sets. Additionally, from the traditional data sets, the number of trees constructed is an important parameter.

Although the Cesáro random forest appears to be competitive to the classical RF it is not without limitations. There are two obvious limitations that stem from our motivation. The first being how to determine the sequencing of trees. If we do not accurately place the best trees first we are not able to guarantee the effectiveness of the Cesáro RF; in fact, it is probable that we will be significantly worse. The reason for such importance being placed on sequencing is due to our next limitations, that is, the strength of the weights we are using. As mentioned early most of the mass of the harmonic series is near the beginning, which explains why the first few trees are so important, as is explained by Theorem 2.3.1.

Another limitation of using the CRF is that users lose the probability estimates of class membership which can be useful for quantifying uncertainty. Hence, one may obtain a higher prediction accuracy but will not be able to see probability estimates. In some cases, the trade-off between prediction accuracy and information gained may be worthwhile.

In conclusion, we studied the effect of applying tree level weights by way of the Cesáro averages to incorporate a sense of quasi-stability. The weights were in the form of the harmonic series. From this we were able to prove theoretical results about this methodology. Numerically we were able to provide evidence that this new weighting method compares favorably to the traditional random forest algorithm as well as demonstrate the stability of our results through reduced variance.

CHAPTER 3. GENERALIZED WEIGHTING SCHEME FOR BAGGED ENSEMBLES

3.1 Introduction

Bagging is an ensemble method for reducing variance of simpler learners with low bias but high variance (Breiman, 1996a), but there is also evidence that simply reducing variance is not always the best approach. In fact, it is reported that bagging works well with decision stumps (Martínez-Muñoz et al., 2007), which have higher bias but lower variance than unpruned trees. And while the best known bagging approach, namely the random forest, uses unpruned trees, its feature sampling has a similar effect. Namely, restricting the number of features available to a tree will on the average increase the bias. This is analogous to methods that focus primarily on reducing bias. Boosted trees are a popular ensemble method where the purpose of boosting is reduced bias, but implementations of boosted trees include one or more regularization parameters that are used to control the variance (Schapire, 2003). Similarly, deep neural networks are successful because adding layers to the network reduces bias, but at the same time numerous regularization parameters are used to reduce variance (Schmidhuber, 2015). Thus, in both approaches several tuning parameters are used to control variance while the main idea of the method is to reduce bias. The success of such highly tunable methods motivates the proposed new ensemble approach. While the main idea of bagging is to reduce variance, we add tunable weights to the ensemble that allows for simultaneously controlling the bias.

In the literature, there are various ways to increase prediction accuracy of bagged ensembles, including feature sampling and model stacking (Bryll et al., 2003). Others have

considered giving weights to different votes in specific concepts (Lacasse et al., 2010). However, a general weighting scheme that can be tuned to fit data *does not exist*. Namely, there is not a parameter, p, such that we can alter values of p to change the strength of a weighted vote. As mentioned above, having such tunable parameters has been found to be important for other ensemble methods such as boosted trees (Freund and Schapire, 1996), which motivates the potential usefulness of the new approach.

In this chapter, we will generalize bagged ensemble learning to a weighted vote by considering different ways of averaging. For the duration for this chapter, we refer to the notation provided in Table 3.1.

Table 3.1: Summary of notation

Notation	**Definition**
Δ	Data set with binary classes
δ_i	i^{th} bootstrapped data set from Δ
v_i	Vote for the class of a particular observation
n	Number of models in ensemble
Γ	Ensemble of models
γ_i	i^{th} predictive learner
w_i	i^{th} weight
c_i	class of observation i
$p \in [0, \infty)$	tunable parameter 1
$\alpha \in [0, 1]$	tunable parameter 2

3.2 Methodology

3.2.1 Weighting Scheme

For an ensemble of n models, we denote this as $\Gamma = \bigcup_{i=1}^{n} \gamma_i$ where each γ_i is some predictive learner. Given a learning problem, once each model provides a vote, v_i, we apply weights of the form:

$$w_i = \begin{cases} \sum_{j=1}^{n} \frac{1}{j^p} & i = 1 \\ \sum_{j=i}^{n} \frac{1}{j^p} + \sum_{j=1}^{i-1} \frac{\alpha}{j^p} & i \geq 2 \end{cases} \tag{3.1}$$

where n is the ensemble size and $\alpha \in [0, 1]$ and $p \in [0, \infty)$ are tuning parameters to be discussed further below. It follows that for a given observation k, we have the predicted class to be:

$$c_k = \left[\frac{\sum_{i=1}^{n} w_i v_i}{\sum_{i=1}^{n} w_i} \right] \tag{3.2}$$

where $[\cdot]$ is the nearest integer function.

Thus when $\alpha = 1$ we have that all weights are equal and we have an unweighted ensemble. Fixing $\alpha = 1$, we obtain

$$w_{i \geq 2} = \sum_{j=i}^{n} \frac{1}{j^p} + \sum_{j=1}^{i-1} \frac{1}{j^p} = \sum_{j=1}^{n} \frac{1}{j^p} = w_1. \tag{3.3}$$

Hence any pure bagged ensemble can be seen as a special case of our weighted ensemble method. If we would further restrict the ensemble to use decision trees with feature sampling at each node, we obtain the random forest.

Bagged ensembles reduce the variance of the base classifiers, which is what the proposed weighted ensemble will do if $\alpha = 1$. For $\alpha \in [0, 1)$, the effect is more complex, but in the extreme case, if (almost) all the weight is given to the single best model then bias would be reduced (and variance increased). This is therefore a more flexible ensemble method. Specifically, the p parameter allows us to tune the rate of decrease for our weights. This is readily seen by observing the function $f : \mathbb{R} \to \mathbb{R}$ where $f(x) = 1/x^p$ which can be considered an approximation to $\sum_{j=i}^{n} \frac{1}{j^p}$. In Figure 3.1, we provide such a visual using varying values of

p. Indeed, we notice that small changes in p results in various decreases in the function, and hence our weights. Analogously, as $p \to \infty$ we allow more weight to be placed towards the beginning and less towards the end. By doing this, we advertently give higher importance to the first few learners.

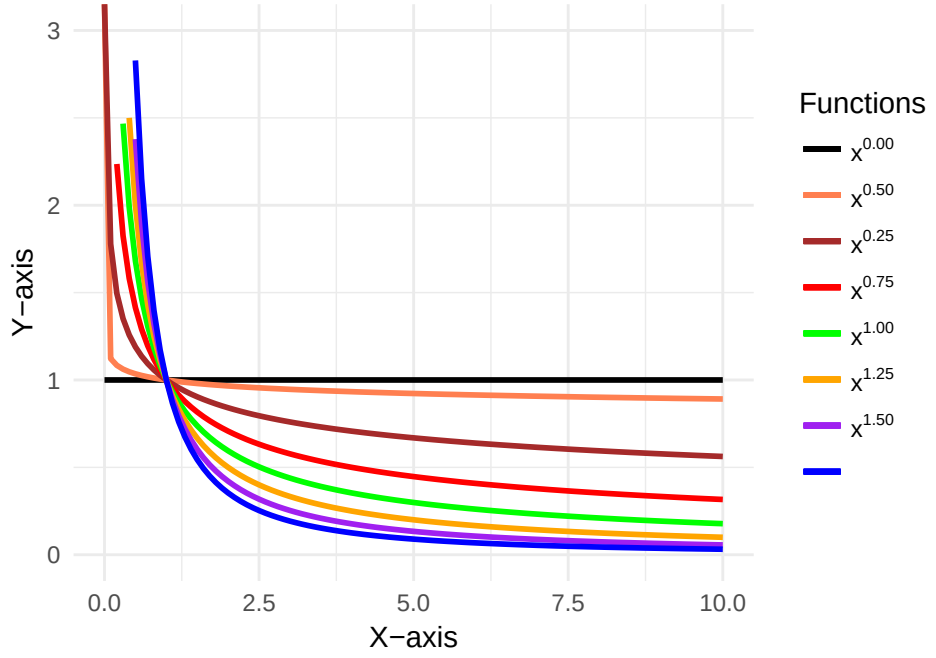

Figure 3.1: Comparison of p-values for $f(x) = \dfrac{1}{x^p}$

Moreover, we may consider $\sum_{j=1}^{i-1} \dfrac{\alpha}{j^p}$ to be a regularization term with a tunable constant α, where maximum regularization is achieved with $\alpha = 1$. A direct benefit of α is that it allows each weight to be equal and hence obtaining a pure bagged ensemble. We take $\alpha \in [0, 1]$ since for $\alpha \notin [0, 1]$ we no longer have a decreasing function. With this, we establish our tunable weight scheme with two parameters that can be applied in any bagged ensemble setting.

3.2.2 Weighted Bagged Ensemble Learning

Like any weighting scheme, determining where to apply the weight is of utmost importance. With an ensemble of γ_i models, how do we sequence each model to allow the best models to be given the most weight? Our solution to this problem is to sequence our models by their out-of-bag error; this is the most natural way of determining the quality of a learner. (Other methods may be to use cross-validation or a second training set.) With this, we have our complete learning algorithm. Algorithm 2 is the pseudo-code for our learning procedure.

Algorithm 2 Weighted Bagged Ensemble Learning

Input n - Size of ensemble, Δ - Training data set, Λ - Testing data set, predictive learner
Output Vector of predictions

for i from 1 to n **do**
    Generate bootstrapped data set, δ_i, from Δ
    Construct model γ_i from δ_i
    Store out-of-bag error for γ_i
    Test γ_i on Λ to receive votes for each observation
end for
Sequence $\Gamma = \bigcup_{i=1}^{n} \gamma_i$ by out-of-bag error
Compute vector of weights using equation (2)
for each observation j **do**
    Compute $c_j = \left[\sum_{i=1}^{n} w_i v_i \Big/ \sum_{i=1}^{n} w_i \right]$
    Store c_j
end for
return Vector $\{c_j\}$

For our implementation, in the event of a tie, the class will be labeled zero. In other applications one may consider allowing the class to be chosen at random. The predictive learners, γ_i, considered can be any learner of choice. Though as previously stated, since bagging reduces variance it may be expected that any low bias, high variance model will be the most improved (Fumera et al., 2005). On the other hand, for $\alpha < 1$ this is not a pure bagging approach and in the extreme case of setting the parameters such that most of the

weight is on one model, this would not be the case.

3.3 Numerical Results

In this section, we provide numerical results for our weighted ensemble scheme by testing on traditional and synthetically generated data sets. We consider ensembles containing each of the following:

1. CART (Classification and Regression Tree) (Breiman et al., 1984)
2. Naïve Bayes Classifier (Rish, 2001)
3. K-Nearest Neighbors (KNN) with $k = \sqrt{\#\text{ of observations}}$ (Cover and Hart, 1967)
4. Support Vector Machine (SVM) with a linear kernel (Cortes and Vapnik, 1995)
5. Neural Network with one hidden layer and ten nodes (Rosenblatt, 1958)
6. Logistic Regression (Cox, 1958)

We consider these models due to the fact that they are six different learners and each posses situations in which they perform well. In all, we hope to establish criteria in which one would consider this weighted ensemble methodology.

3.3.1 Traditional Results

We first apply our weighted ensemble method to traditional data sets as listed in Table 3.2. That is, data sets which are commonly used in the data mining community. These were collected from a combination of UCI (University of California Irvine) repository (Bache and Lichman, 2013), CRAN (Hendricks, 2015) or created by the authors.

For all experiments we consider $p \in \{0, .25, .50, .75, 1.0, 1.5, 2.0\}$, $n \in \{100, 500, 1000\}$ and $\alpha = 0$. In later trials we adjust our α value. Furthermore, to avoid any ambiguity, for

Table 3.2: Traditional data sets for general weighted bagged ensemble experiments

Data Set	Observations	Features	Source
Teaching Evaluations	101	5	UCI
LSVT Voice Rehab	126	310	UCI
Sonar	208	60	UCI
Heart (Statlog)	267	44	UCI
Haberman	306	3	UCI
Bupa Liver	345	7	UCI
Ionosphere	351	34	UCI
Musk	476	167	UCI
Breast Cancer	569	30	UCI
Hill-Valley	606	100	UCI
NBA Playoffs	683	92	Authors
Titanic	714	7	CRAN
Diabetes Pima Indians	768	8	UCI
German Credit Card	1000	25	UCI
Medallion	2000	500	UCI
Musk Updated	7074	167	UCI
Polish Bankruptcy	10503	64	UCI
HTRU	17898	8	UCI
Taiwan Credit Card	30000	24	UCI
Online News Popularity	39644	60	UCI

each data set we utilize 3/4 for training and holdout the remaining 1/4 for testing. Given each data set, we run 10 trials. In all, we hope to establish an optimal ensemble, robust default parameters, and preferred types of learning models.

Moreover, we compare our results to two benchmarks: pure bagging and the random forest. This is a sensible comparison since we want to understand if and when using weights outperforms pure bagging, and the random forest is the bagged ensemble of choice for most machine learning applications.

3.3.1.1 Determining optimal ensemble size and sequencing method

For our first experiment our objective is to determine the optimal ensemble size, be that 100, 500, or 1000 along with a preferred sequencing method. Displayed in Table 3.3 are the results from our first experiment.

We display the highest averages obtained across all models as well as the corresponding ensemble size that produced the maximal accuracy. Moreover, in addition to sequencing by out-of-bag error, we report two other methods: *None* and *Pure Bagging*. We take *None* to mean there is no sequencing of the models and that weights are applied to each model in its construction order. Whereas *Pure Bagging* is the usual majority vote; that is, each vote is given equal priority as in the random forest. From Table 3.3, important takeaways are that:

1. The new weighted ensemble (including pure bagging) performs well relative to the random forest in 14 out of 20 data sets.

2. Of those 14 data sets, sequencing by the out-of-bag error to determine model quality was the better ordering method 9 times.

3. An ensemble of size 500 or 1000 provides the highest averages more often than an ensemble of 100 models.

In some cases, we outperform the random forest by over four percentage points as seen in the *Teaching Evaluation*, *LSTV* and *Haberman* data sets. Moreover, one may notice that data sets where we heavily underperform have a large number of features compared to observations. Indeed this holds true for the: *Musk*, *Musk Updated*, *Hill-Valley*, and *NBA* data sets with the *Bupa Liver* data set being the exception. Additionally shown is that an ensemble of size 500 or 1000 is preferred over one of size 100. However in the cases where we are competitive to the random forest an ensemble of size 500 may be preferable.

Table 3.3: Prediction accuracy (%) from 10 trials. Average accuracy ± one standard deviation comparing weighted bagging, pure bagging, and the random forest

	Sequencing Method			***Benchmarks***	
Data Set	**OOB**	**None**	**Size**	**Pure Bagging**	**Random Forest**
Teaching Evaluations	**79.6±6.0**	79.2±6.6	1000	71.5±7.1	77.7±9.7
LSVT Voice Rehab	86.6±4.7	86.6±5.5	1000	**86.9±4.6**	85.2±5.1
Sonar	**84.0±5.4**	83.7±4.3	100	77.1±5.1	82.5±4.9
Heart (Statlog)	82.1±4.2	**82.2±4.4**	500	82.0±4.6	81.0±4.6
Haberman	**77.9±7.0**	77.4±5.0	500	75.9±5.2	73.7±4.5
Bupa Liver	67.8±4.9	67.4±4.6	1000	67.7±4.6	**72.2±4.3**
Ionosphere	**93.6±2.2**	**93.6±2.1**	100	90.6±3.2	93.4±2.2
Musk	89.0±2.5	88.2±2.5	500	85.5±5.6	**90.6±1.9**
Breast Cancer	**97.1±1.4**	96.9±0.82	1000	94.8±2.3	96.0±1.7
Hill-Valley	99.5±0.0	99.5±0.0	1000	**99.8±0.0**	58.1±3.5
NBA Playoffs	**98.0±1.2**	97.8±1.2	500	97.8±1.1	94.7±1.7
Titanic	78.6±1.8	78.6±3.7	500	78.7±1.7	**80.2±2.6**
Diabetes Pima Indians	**76.6±2.1**	**76.6±3.0**	500	76.5±2.6	76.3±2.7
German Credit Card	**81.4±1.6**	**81.4±1.5**	100	**81.4±1.5**	80.1±1.9
Medallion	67.0±2.7	65.7±3.0	1000	65.7±3.1	**69.0±2.3**
Musk Updated	94.8±0.3	94.8±0.3	1000	94.9±0.0	**97.4±0.1**
Polish Bankruptcy	95.8±0.1	95.8±0.0	500	**96.3±0.0**	96.1±0.1
HTRU	**98.1±1.1**	98.0±0.9	500	98.1±1.0	97.9±0.5
Taiwan Credit Card	82.1±0.7	81.1±1.2	100	**82.2±0.4**	81.8±0.6
Online News Popularity	65.2±0.7	65.2±1.2	1000	65.2±1.2	**67.3±0.8**

In comparison with pure bagging, there does not exist a data set in which we statistically differ when using a weighted ensemble. However, there are instances where, on average, we can perform much higher. Statistically we are able to outperform the random forest in the *Hill-Valley* and *NBA Playoffs* data set. Across most data sets, the standard deviation is generally high, and although we are able to beat pure bagging and the random forest on average, there is no statistical difference with our weighted ensemble scheme in 18 out of 20 cases.

In terms of the sequencing, it is of no surprise that ordering our models by OOB error is better than not sequencing. In some cases not ordering and applying the weighted scheme

is better; this can most likely be attributed to high variability in the construction of each model. The same can be said for giving each model equal weight (pure bagging).

In all, it may be concluded that an ensemble containing anywhere from 500-1000 models may be sufficient. Though one should be cautious about the computational affects of increasing the size. From these observations, we do not yet make assertions about preferred p-values or a recommended model.

3.3.1.2 Determining existence of optimal p and α parameters

The next objective is to analyze the effect of an α parameter. To determine the significance of this parameter as well as its relationship with p we will take the *Bupa Liver* and *Sonar* data set and attempt to find an optimal p and α value. In finding the optimal to near-optimal parameters, we hope to resolve two hypotheses:

1. Can choosing specific (optimal) p and α significantly increase prediction accuracy?

2. Will these values be robust to any bootstrapped data?

For this experiment, we will use an ensemble of size 500, and sort by the out-of-bag error. From Table 3.3 we see that in one data set (*Bupa Liver*) we are unable to outperform the random forest, while in the other (*Sonar*) we do. In Table 3.4, we report five values: the accuracy obtained from a simple majority vote (pure bagging), the minimum and maximum prediction accuracy, and the α and p values which corresponds to the maximum value. To determine an optimal p and α value a grid search was performed over 1681 possible combinations. Namely $p \in \{0, .05, .10, .15, ..., 2.0\}$ and $\alpha \in \{0, .025, .050, .075, ..., 1.0\}$.

With these values, we are now able to outperform the random forest in the *Bupa Liver* data set with an SVM. Moreover, in the *Sonar* data set we still were able to provide a higher accuracy than the random forest. In both cases, the parameters appear to have little affect

on the CART. Another noticeable pattern is the gap between the minimum and maximum value for each model. Indeed, from this we can deem that our weighted scheme has an impact on the final prediction accuracy and that finding optimal p and α parameters can yield high dividends.

One key observation is that for most cases, $p = 0 \implies \alpha > 0$, and $\alpha = 0 \implies p > 0$. This can be seen as a further analogy to our bias variance trade-off. Namely when $\alpha = 0$ we have a pure bagging scheme (variance reduction) and in those cases we obtain the highest accuracy. Whereas for $p > 0$ we are simply reducing bias and obtain the highest accuracy when $\alpha = 0$.

Moreover, we see that our p and α values are indeed robust. That is, in each both trials, generally higher p-value is preferred. The exception would be in the *Sonar* data set on trial 2 when using NB and SVM. It should be noted, that maximum values obtained are different due to the randomness of the bootstrapped data each learner was constructed on. However, this is not a concern since our purpose is to establish a connection between p and α values.

3.3.1.3 Determining default p-value

Having established that a small α value is preferred, can we decide on a recommended p-value that is robust to each ensemble size? In the following Figure 3.2 we plot a histogram of the p-values where we outperformed the random forest.

It is clear that each ensemble size prefers a different p-value. The reasoning can be explained as follows. First when creating a large ensemble, there may be substantial more *bad* learners hence by using a smaller p-value each model is given near same weight. On the other hand, if the ensemble contains many good learners then a very high p-value is desired as to allow the best models to have the most weight. Namely for an ensemble of size 100 a high p-value is preferred and for size 500 a smaller one is better. The biggest difference is that a p-value of 2.0 is clearly optimal for a small ensemble and a p-value of 0 is optimal for

Table 3.4: Test optimal values (p and α)

Data Set	Trial	Model	Minimum	Maximum	Optimal p	Optimal α	Pure Bagging
Bupa Liver	1	CART	70.1	**71.2**	0	0.025	**71.2**
		NB	65.5	70.1	1.85	0	66.7
		KNN	59.8	67.8	1.75	0	64.4
		Log. Reg.	67.8	70.1	0	0.175	69.0
		SVM	67.8	69.0	0	1.0	69.0
		N. Network	64.4	67.8	1.55	0	65.5
	2	CART	67.8	70.1	0	0.625	70.1
		NB	65.5	74.7	1.95	0	66.7
		KNN	59.8	66.7	1.9	0	64.4
		Log. Reg.	63.2	66.7	1.80	0	63.2
		SVM	74.7	**78.2**	1.75	0	77.0
		N. Network	70.1	72.4	1.85	0	70.1
Sonar	1	CART	61.5	65.4	0	0.025	65.4
		NB	71.2	75	1.75	0	73.1
		KNN	80.8	**84.6**	1.75	0	80.8
		Log. Reg.	73.1	76.9	0	1.0	76.9
		SVM	78.8	78.9	2.0	0	80.8
		N. Network	82.7	84.6	1.95	0	82.7
	2	CART	65.4	69.2	1.95	0	67.3
		NB	67.3	71.2	0	0	67.3
		KNN	69.2	**78.8**	1.7	0	69.2
		Log. Reg.	59.6	65.4	1.2	0.075	63.5
		SVM	69.2	76.9	0.95	0	76.9
		N. Network	82.7	86.5	1.90	0	84.6

a size 500 ensemble.

In total from both these plots, one may loosely say that a default p-value should be near .75, and that by proper tuning via say, grid-search, one may find an optimal p.

3.3.1.4 Determining model choice

A natural question to ask is, does the new ensemble approach work better with some of these methods? In this section, we attempt to provide an answer.

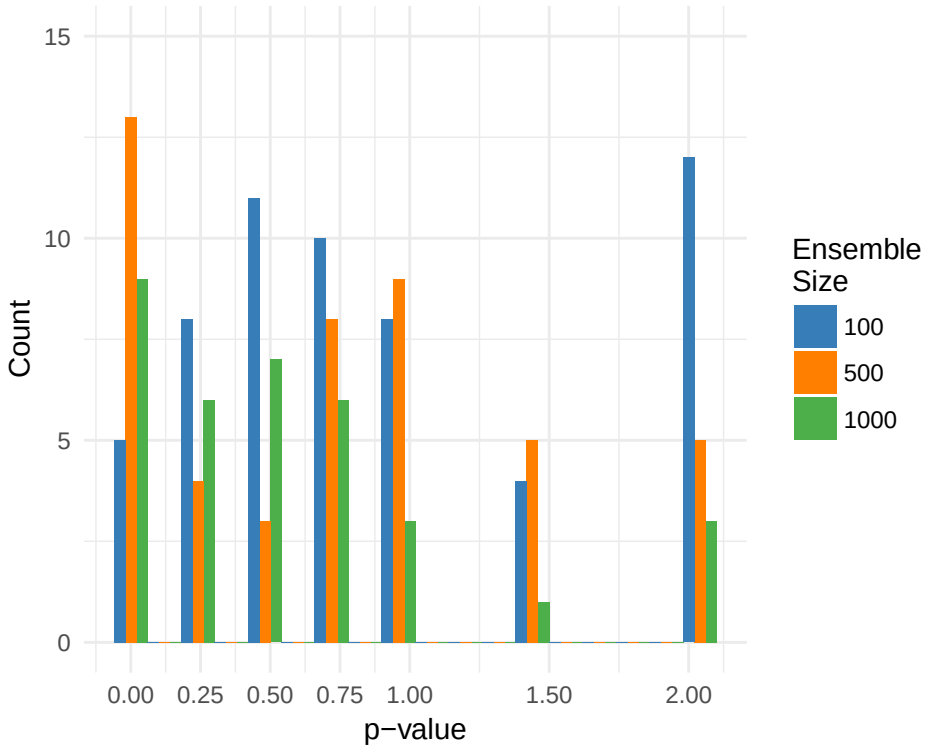

Figure 3.2: Comparison of p-values that outperform the random forest

In Table 3.5 we provide a binary indicator where one means the model outperformed the random forest and zero where we did not. From this we see that the number of cases where CART is greater than the others with KNN and SVM following closely.

However, for the 20 data sets, we notice that there are cases in which each model will perform well within the weighted ensemble.

3.3.1.5 Computation time

For insights into computational efficiency we provide results for a subset of our experiments in Figure 3.3. For this visualization, we only display: *Teaching Evaluation*, *Medallion*, and *Online News Popularity*. These are considered since they are the smallest and largest data sets with *Medallion* being a middle ground for a relative comparison. The CPU being

Table 3.5: Learners that outperform the random forest

Data Set	CART	Naïve Bayes	KNN	Log. Reg.	SVM	N. Network
Teaching Evaluations	0	1	0	0	1	0
LSVT Voice Rehab	1	1	1	0	1	0
Sonar	0	0	1	0	0	1
Heart (Statlog)	1	1	1	0	1	1
Haberman	1	1	1	1	0	0
Bupa Liver	0	0	0	0	0	0
Ionosphere	1	0	0	0	0	0
Musk	0	0	1	0	0	0
Breast Cancer	0	0	1	0	1	0
Hill-Valley	0	0	0	1	1	1
NBA Playoffs	0	0	0	0	1	0
Titanic	0	0	0	0	0	0
Diabetes Pima Indians	1	1	1	1	1	0
German Credit Card	1	1	1	1	1	0
Medallion	1	0	0	0	0	0
Musk Updated	0	0	0	0	0	0
Polish Bankruptcy	1	0	0	0	0	0
HTRU	0	0	0	0	0	1
Taiwan Credit Card	1	0	0	0	0	0
Online News Popularity	0	0	0	0	0	0
Sum Total	**9**	**6**	**8**	**4**	**8**	**4**

used is two 2.6 GHz 8-Core Intel E5-2640 v3 processors.

For our figure we plot our seven models and report a transformed value of time (log(Time)) in seconds for constructing 1000 models. For our smallest data set, we were able to construct 1000 models in a matter of seconds. Even for the *Medallion* data set computational times were modest across all models. However, for our largest data set, *Online News Popularity*, KNN and SVM cost the most computationally. Namely constructing 1000 models took approximately $e^{12} \approx 162,755$ seconds apiece. Hence in practice for such a large data set, other models may be considered.

3.3.2 Synthetic Results

By testing synthetic data sets, we hope to provide insights into when our weighted bagged ensemble is competitive to the random forest. Namely we consider data sets with the fol-

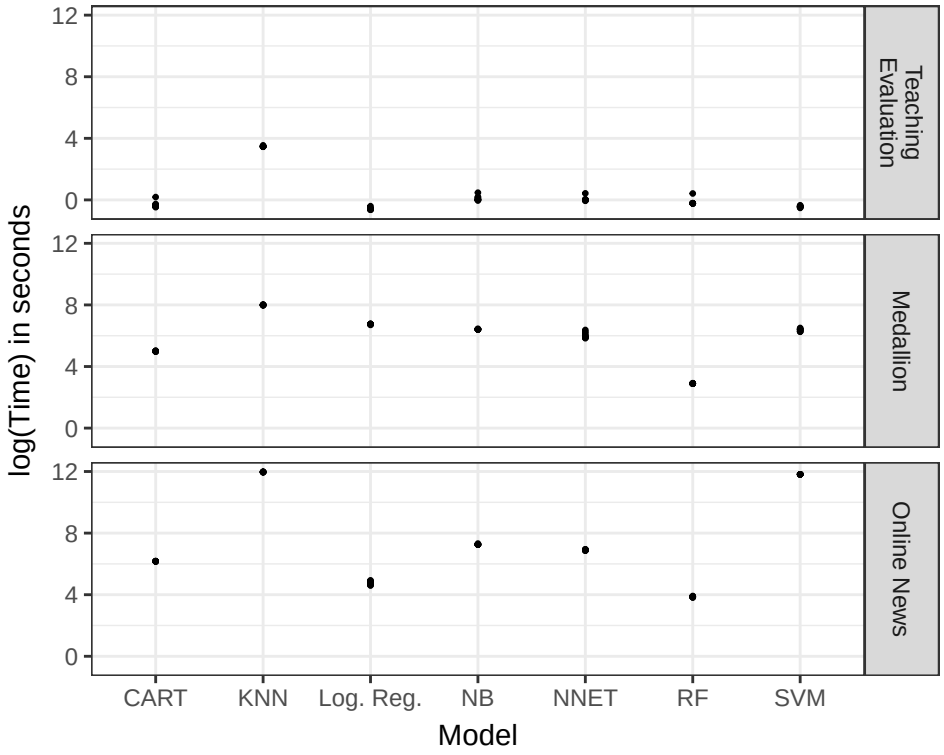

Figure 3.3: Comparison of run times for 3 data sets

lowing properties: sparsity, spuriosity, noise, and ratio of observations to features. To study this we alter four previously tested data sets each for a specific purpose, that is,

1. *Breast Cancer*: Sparsity

2. *Heart (Statlog)*: Outliers

3. *Haberman*: Noise

4. *German Credit Card*: Ratio of observations to features

By modifying these four data sets, we are able to test each condition while having a baseline as comparison. For the purposes of this section, we restrict our ensemble size to 500 learners and only sequence by the out-of-bag error as per the results of Table 3.3. However, the six different learning models are still utilized.

3.3.2.1 Effect of sparsity

Given the *Breast Cancer* data set with 569 observations and 30 features, we randomly replace entries with zero. We do this for five different levels; that is, we randomly replace $\{10, 20, 40, 60, 80\}\%$ of the data set with zeros. For each sparsity level, we run 10 trials where in each trial we replace a different random sample with the given percentage of zeros.

In Table 3.6 we report our findings. One will notice there is a 0% column; this represents our baseline, that is, the results obtained from the original data set without any altercations.

Table 3.6: *Breast* data set with various levels of sparsity (average ± one standard deviation %)

	Sparsity Level					
Model	0%	10%	20%	40%	60%	80%
CART	94.3±2.7	93.6±1.8	92.7±1.1	89.4±2.5	81.9±3.6	75.2±2.4
NB	94.8±1.8	94.5±1.7	94.2±1.1	92.3±2.4	89.6±2.9	84.8±1.8
KNN	**96.7±3.5**	90.8±1.5	86.4±1.2	77.4±2.2	71.6±0.0	72.2±3.6
Log. Reg.	95.6±1.2	94.1±2.1	90.8±2.0	86.9±2.3	80.8±2.2	73.6±4.0
SVM	97.2±1.4	93.4±2.2	90.6±2.0	86.6±2.9	82.4±2.1	76.6±2.8
N. Network	91.6±2.1	89.1±4.9	83.2±4.4	75.1±3.9	67.0±3.4	66.1±3.6
RF	96.0±0.9	**95.4±0.1**	**94.5±0.8**	**93.2±0.6**	**92.0±0.6**	**88.0±1.3**

From this, we are able to see that the random forest consistently outperforms our weighted ensemble scheme. In fact, on average, our prediction accuracy is much worse. This can likely be attributed to the learners chosen in our ensemble, and the randomness associated to the random forest to learn different parts of the data. Moreover the rate of decrease in prediction accuracy of the random forest is much slower than any of the three models.

3.3.2.2 Effect of spurious data points

Next we analyze the *Heart* data set which contains 267 observations and 45 features. Here, we add synthetic observations with random values generated between zero and one along with a class value. Specifically to the *Heart* data set, {25, 75, 125}% more rows are

added. The results are summarized in Table 3.7 where column 0% is considered our baseline comparison.

Table 3.7: *Heart* data set with various levels of spurious data points (average ± one standard deviation %)

	Spuriosity Level			
Model	0%	25%	75%	125%
CART	**82.1 ± 4.4**	74.0 ± 5.4	68.0 ± 4.9	64.7 ± 4.5
NB	79.7 ± 4.3	**74.3 ± 3.1**	**71.5 ± 3.0**	**65.2 ± 3.0**
KNN	79.9 ± 4.3	71.1 ± 2.7	66.5 ± 4.3	62.2 ± 4.0
Log. Reg.	75.8 ± 3.8	67.1 ± 3.4	64.7 ± 2.6	59.5 ± 2.3
SVM	80.0 ± 5.2	68.0 ± 4.5	64.4 ± 3.5	60.1 ± 2.1
N. Network	80.3 ± 6.1	71.2 ± 4.7	64.7 ± 3.2	62.1 ± 3.7
RF	80.9 ± 4.7	73.5 ± 1.4	67.1 ± 0.9	64.5 ± 0.9
NB-Not Bagged	75.1 ± 4.3	72.5 ± 4.5	66.8 ± 3.3	63.8 ± 2.1

The results above support the claim that under the presence of spurious data points we would expect this weighted ensemble scheme to hold form. Indeed, the naïve Bayes classifier consistently provides a higher average than that of the random forest. Hence a fair statement is that a weighted ensemble of the correct classifier may outperform the random forest in the presence of spurious observations. Moreover, when comparing the bagged ensemble of NB to that of a non-bagged naïve Bayes, we see that there is a significant increase in prediction accuracy. This further supports the benefit a bagged ensemble as opposed to a single model approach.

3.3.2.3 Effect of noise

The *Haberman* data set containing 306 observations and 3 features is used to provide insights into how we perform with the addition of noisy features. To understand this, we adjoin columns with random values generated between zero and one; that is, {50, 100}% more columns are added. Table 3.8 contains our summary.

Table 3.8: *Haberman* data set with various levels of noise (average ± one standard deviation %)

	Noise Level		
Model	0%	50%	100%
CART	76.8 ± 5.0	76.8 ± 4.5	75.3 ± 4.1
NB	76.4 ± 3.7	75.8 ± 3.7	74.9 ± 4.7
KNN	**77.9 ± 4.0**	**76.5 ± 3.3**	**75.7 ± 3.6**
Log. Reg.	75.6 ± 4.6	73.9 ± 4.8	71.0 ± 5.6
SVM	72.1 ± 4.5	73.6 ± 2.8	74.0 ± 3.7
N. Network	74.2 ± 3.1	73.9 ± 4.8	71.0 ± 5.6
RF	73.7 ± 3.5	72.3 ± 1.6	72.6 ± 2.0
KNN-Not Bagged	74.9 ± 2.0	73.8 ± 1.2	73.6 ± 2.1

With the addition of noisy features, we still perform competitively to the random forest. From our table, it is clear that an ensemble of the k-nearest neighbors will allow us to maximize our potential. However, the change in accuracy in the random forest from 0% to 100% is 1.3 which is less than the KNN. Hence the random forest could be considered to be more invariant to noise than the KNN ensemble. Additionally, when comparing with a non-bagged KNN, we notice how much of improvement is obtainable with our weighted ensemble. Thus, we further defend the usefulness of a bagged ensemble approach, and in particular, our weighting scheme.

3.3.2.4 Effect of ratio of observations to features

Lastly, we consider the *German Credit Card* data set that has 1000 observations and 25 features. To test the impact of the ratio of observations to features, we randomly subset rows until our desired fraction is obtained. We test ratios of the form {40:1 (original), 20:1, 10:1, 5:1}.

From Table 3.9 we are still able to attain an higher prediction accuracy than the random forest with CART and KNN as candidate models for the ensemble. From the results, there

Table 3.9: *German Credit Card* data set with varying ratios (average ± one standard deviation %)

	Ratio (Obs. to Feats.)			
Model	40:1	20:1	10:1	5:1
CART	81.0 ± 1.9	**81.0 ± 2.9**	**80.3 ± 6.8**	82.8 ± 9.1
NB	77.5 ± 2.1	78.4 ± 2.4	78.3 ± 5.2	76.9 ± 13.8
KNN	**80.9 ± 1.9**	80.7 ± 3.3	80.0 ± 5.5	**83.1 ± 8.6**
Log. Reg.	79.5 ± 1.3	76.6 ± 2.8	75.0 ± 3.4	60.1 ± 2.1
SVM	78.5 ± 2.5	78.6 ± 4.3	75.7 ± 3.4	75.7 ± 5.9
N. Network	78.0 ± 1.5	77.2 ± 4.1	77.9 ± 4.9	76.6 ± 2.7
RF	80.1 ± 1.9	79.0 ± 2.2	77.0 ± 3.6	78.6 ± 8.2

does not appear to be an obvious pattern between the ratios and prediction accuracy. This is readily seen since we obtain the two maximal values in the 5:1 scenario. This may possibly associated to the fact that there were very few good models due to the lack of training observations and a higher p-value was chosen.

3.4 Conclusion

In this work we present a tunable weighting method applied to bootstrapped aggregated ensembles. This weighted scheme has the potential to outperform a pure bagged ensemble as well as the random forest. We support these claims by experimental results on traditional data sets and synthetically generated data sets.

Although, we usually can outperform pure bagging, and in some cases, we can out perform the random forest, our weighted scheme is not without faults. Namely, there are time considerations in effectively choosing α and p parameters aside from a grid search. Moreover, by introducing weights, there is the possibility of increase overfitting on for the model, that is, unlike for pure bagging it is now possible to decrease bias to the extent that the benefits are outweighed by the increased variance.

CHAPTER 4. WBENSEMBLER: AN R PACKAGE FOR WEIGHTED BAGGED ENSEMBLE LEARNING

4.1 Introduction

In the context of predictive modeling, ensembles refer to the collection of (potentially) different learning models such as: decision trees, support vector machines, logistic regressions, etc. to form a single strong learner which is (hopefully) more accurate than each individual component. One such collection is a bootstrapped aggregated (bagged) ensemble that is, learning models that are generated from a bootstrapped data set. Ensembles such as these are widely applicable and very popular with the most well known being the random forest (Breiman, 2001).

Nevertheless another ensemble method, boosting, also has wide spread popularity, and seems to be winning a variety of data science competitions leaving the random forest to be seemingly forgotten. One aspect that existing boosting models have that bagged ensembles do not are tunable parameters. In **xgboost**, a popular boosting package, there exists a variety of parameters than users can alter to better fit data sets (Chen et al., 2018). However, as of now there is not a package that allows any tunability for general bagged ensembles.

Due to this gap, the authors have developed a packaged call **wbensembleR** that provides parameters for added flexibility to bagged ensembles; namely, we allow for the simultaneous control of bias and variance. This framework considers added flexibility in the form of weighted votes. In particular, we consider these weights to be of the form:

$$w_i = \begin{cases} \sum_{j=1}^{n} \frac{1}{j^p} & i = 1 \\ \sum_{j=i}^{n} \frac{1}{j^p} + \sum_{j=1}^{i-1} \frac{\alpha}{j^p} & i \geq 2 \end{cases}$$

where n is the ensemble size and $p \in [0, \infty)$ and $\alpha \in [0, 1]$ are tuning parameters to be discussed further below. It follows that for a given observation, we have the predicted class to be which ever class obtains the most weight.

Observe that in the case where $\alpha = 1$ we have that all weights are equivalent that is, $w_i = w_j$ for all i, j. Indeed, fixing $\alpha = 1$, we obtain

$$w_{i \geq 2} = \sum_{j=i}^{n} \frac{1}{j^p} + \sum_{j=1}^{i-1} \frac{1}{j^p} = \sum_{j=1}^{n} \frac{1}{j^p} = w_1$$

Hence any pure bagged ensemble can be seen as a special case of our weighted ensemble method. If we would further restrict the ensemble to use decision trees with feature sampling at each node, we obtain the random forest.

Bagged ensembles reduce the variance of the base classifiers, which is what the proposed weighted ensemble will do if $\alpha = 1$. For $\alpha \in [0, 1)$, the effect is more complex, but in the extreme case, if (almost) all the weight is given to the single best model then bias would be reduced (and variance increased). This is therefore a more flexible ensemble method. Specifically, the p parameter allows us to tune the rate of decrease for our weights. This is readily seen by observing the function $f : \mathbb{R} \to \mathbb{R}$ where $f(x) = 1/x^p$ which can be considered an approximation to $\sum_{j=i}^{n} \frac{1}{j^p}$. Analogously, as $p \to \infty$ we allow more weight to be placed towards the beginning and less towards the end. By doing this, we advertently give higher importance to the first few learners.

Moreover, we may consider $\sum_{j=1}^{i-1} \frac{\alpha}{j^p}$ to be a regularization term with a tunable constant α, where maximum regularization is achieved with $\alpha = 1$. A direct benefit of α is that it allows each weight to be equal and hence obtaining a pure bagged ensemble. We take $\alpha \in [0, 1]$

since for $\alpha \notin [0, 1]$ we no longer have a decreasing function. With this, we establish our tunable weight scheme with two parameters that can be applied in any bagged ensemble setting.

4.2 Overview of wbensembleR

For the duration of this chapter, we will utilize ToothGrowth, a built-in data set with 60 observations, 2 features, and binary classes. Although not an large data set, this serves its purpose as a demonstration guide. However, in practice, one would use **wbensembleR** on much larger data sets. Below is a summary of ToothGrowth.

```
> str(ToothGrowth)
'data.frame':        60 obs. of  3 variables:
$ len : num  4.2 11.5 7.3 5.8 6.4 10 11.2 11.2 5.2 7 ...
$ supp: Factor w/ 2 levels "OJ","VC": 2 2 2 2 2 2 2 2 2 2 ...
$ dose: num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...


> summary(ToothGrowth)
len              supp         dose
Min.   : 4.20   OJ:30   Min.   :0.500
1st Qu.:13.07   VC:30   1st Qu.:0.500
Median :19.25           Median :1.000
Mean   :18.81           Mean   :1.167
3rd Qu.:25.27           3rd Qu.:2.000
Max.   :33.90           Max.   :2.000


> head(ToothGrowth)
```

```
len     supp dose
1  4.2   VC  0.5
2 11.5   VC  0.5
3  7.3   VC  0.5
4  5.8   VC  0.5
5  6.4   VC  0.5
6 10.0   VC  0.5
```

One can view **wbensembleR** as a wrapper or companion to Max Kuhn's **caret** in the sense that our weighted scheme works with the outputs of `caret::train()` and `caret::predict.train()` (Kuhn et al., 2017). Namely, we sequence our models by out of bag error produced by `caret::train()` and weigh the votes of `caret::predict.train()`.

Our package consists of the following functions:

1. `bagged.ensembler()`
2. `predict.ensemble()`
3. `weights()`
4. `matrix.weights()`

Each function has various input and output structures. In practice, the most useful are `bagged.ensembler()` and `predict.ensemble()`. The other commands can be seen as helper functions; that is, commands that assist the main functions in doing their computations. Additionally since our package can be viewed as an extension of **caret**, every model, in **caret** is available for constructing a bagged ensemble. Here we give an overview of the helper functions and in the following section, we further describe our main functions.

The two helper functions are embedded in the `predict.ensemble()` operation. `weights()` simply outputs a vector containing the calculated weight for a given length n, p and α value where n should be the size of the bagged ensemble.

```
weights <- function(n = NULL, p = NULL, alpha = NULL)


> weights(n = 5, p = .25, alpha = .75)
[1] 5.0 4.5 4.0 3.5 3.0
```

An extension of `weights()` is `matrix.weights()`. Whereas `weights()` simply gives a vector, `matrix.weights()` constructs a matrix where each column corresponds to a weighted sequence. Namely, this is used when either p or α are vectors of length greater than one. The arguments of both functions are the same, but have a different structural output. The following is an example of the generated output.

```
> matrix.weights(n = 5, p = 0:2, alpha = seq(0,1,.5))


        [,1]      [,2]      [,3] [,4]     [,5]      [,6] [,7]     [,8]
p          0 1.0000000 2.0000000  0.0 1.000000 2.0000000    0 1.000000
alpha      0 0.0000000 0.0000000  0.5 0.500000 0.5000000    1 1.000000
weight1    5 2.2833333 1.4636111  5.0 2.283333 1.4636111    5 2.283333
weight2    4 1.2833333 0.4636111  4.5 1.783333 0.9636111    5 2.283333
weight3    3 0.7833333 0.2136111  4.0 1.533333 0.8386111    5 2.283333
weight4    2 0.4500000 0.1025000  3.5 1.366667 0.7830556    5 2.283333
weight5    1 0.2000000 0.0400000  3.0 1.241667 0.7518056    5 2.283333
        [,9]
p       2.000000
```

```
alpha   1.000000
weight1 1.463611
weight2 1.463611
weight3 1.463611
weight4 1.463611
weight5 1.463611
```

As we can see aside from the first two rows, which states p and α, we obtain the output of `weights()` as columns. This function is necessary to aid in the computations of `predict.ensemble()`. As implied by our weighted scheme, we notice that when $\alpha = 1$, we have that all weights are equivalent and hence we no longer have a weighted average but a typical arithmetic mean.

Although these functions assist the main ones in their computations, they are still useful in their own right. Specifically, if a user wants to observe the weights before they are applied to the casted vote of each model, they can use one of these two commands.

4.3 Implementation of Code

Now that we have established the simpler functions, we can delve into the important aspects of our package, namely, `bagged.ensembler()` and `predict.ensemble()`.

4.3.1 Model Training

`bagged.ensembler()` takes the form:

```
bagged.ensembler <- function(formula = NULL, data = NULL, size = NULL,
                             model = NULL, sample.fraction = 0.632, ...)
```

The inputs of `bagged.ensembler()` include: the prediction formula, a reference to some data frame object, the number of models to construct, the model of choice, the bootstrap fraction, and other additional inputs. Note that "..." is a feature that allows additional optional arguments to be passed through to be embedded in functions that support them. An example of an input structure for `bagged.ensembler()` can be seen below where the additional operation arguments are **preProcess** and **tuneLength**. For a complete list of available models and optional parameters refer to **caret**'s documentation.

```
models <- bagged.ensembler(formula = supp ~., data = ToothGrowth,
                           size = 100, model = "knn",
                           preProcess = c("center","scale"),
                           tuneLength = 20)
```

The output of `models` is a list structure containing sorted out of bag error rates for each model, **caret**'s default output for each model and the provided formula.

```
> summary(models)
   Length   Class   Mode
[1,] 200    -none-  numeric
[2,] 100    -none-  list
[3,]   3    formula call
```

To output each object in the list simply call: `models[[x]]` where x $\in \{1, 2, 3\}$. Below we display the out of bag errors and the formula.

```
> head(models[[1]])
    OOBError    Model
20 0.2758621    20
```

```
50 0.2758621    50
85 0.2903226    85
38 0.3000000    38
43 0.3142857    43
23 0.3333333    23

> models[[3]]
supp ~ .
```

`models[[2]]` will print out every model generated hence resulting in a long output. To view a single model, issue the command `models[[2]][i]` with $i \in \{1, ..., \text{size}\}$. However, the aforementioned command will only issue the summary of the *ith* learner. To view all relevant details of the *ith* model, one must *unlist* the object. Below we provide an example.

```
> models[[2]][1]
[[1]]
k-Nearest Neighbors

37 samples
2 predictor
2 classes: 'OJ', 'VC'

Pre-processing: centered (2), scaled (2)
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 37, 37, 37, 37, 37, 37, ...
Resampling results across tuning parameters:
```

```
k   Accuracy   Kappa
5  0.5169010  0.076477981
7  0.5219981  0.090561341
9  0.5169446  0.079573591
11  0.5081771  0.074939383
13  0.5079328  0.077090662
15  0.4961933  0.066593492
17  0.4978584  0.069072450

Accuracy was used to select the optimal model using  the largest value.
The final value used for the model was k = 7.

> model.details <- unlist(models[[2]][1], recursive = F)
> summary(model.details)
            Length  Class        Mode
method        1     -none-     character
modelInfo    13     -none-     list
modelType     1     -none-     character
results       5     data.frame list
pred          0     -none-     NULL
bestTune      1     data.frame list
call          6     -none-     call
dots          0     -none-     list
metric        1     -none-     character
```

```
control      27     -none-     list
finalModel    8     knn3       list
preProcess   21     preProcess list
trainingData  3     data.frame list
resample      3     data.frame list
resampledCM   6     data.frame list
perfNames     2     -none-     character
maximize      1     -none-     logical
yLimits       0     -none-     NULL
times         3     -none-     list
levels        2     -none-     character
terms         3     terms      call
coefnames     2     -none-     character
xlevels       0     -none-     list
```

We notice that once we unlist our object, we can retrieve the usual **caret** output of each model. Hence no information is lost by `bagged.ensembler()`.

Additionally, although not shown here, built into **caret** is the ability to construct learners in parallel. If a user wishes to implement multi-processing all that is required to register a parallel backend. One simple way to implement a backend is with the library **doParallel** (Microsoft Corporation and Steve Weston, 2017).

```
library(doParallel)
cl <- makeCluster(5)
registerDoParallel(cl)
```

Once the parallel backend is registered, all subsequent models are run in parallel (if supported by the chosen model).

4.3.2 Model Evaluation

Having obtained the output of `bagged.ensembler()`, we can make use of the next main function that is, `predict.ensemble()`. The format is as follows:

```
predict.ensemble <- function(object = NULL, data = NULL,
                             p = NULL, alpha = NULL)
```

Arguments of `predict.ensemble()` include: a `bagged.ensembler()` object, a reference to a data frame, p and α values. Here p and α can a single number, a vector or a combination of both. However, it should be observed that as the length of p and alpha increase so will the computational time. Namely if p and α are vectors of length n and m, respectively, we obtain a grid with $n * m$ unique combinations. Below we provide an example where p is a vector containing integers one through five, and α is a sequence from zero to one in increments of .25.

```
prediction <- predict.ensemble(object = models, data = ToothGrowth,
                               p = 1:5, alpha = seq(0,1,.25))
```

The output consists of two list. The first is a list of each model (sorted by OOB) and their corresponding vote for each observation, the other is a list with accuracy values with a corresponding p and alpha.

```
> head(prediction[[1]],5)
       1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
model4  2 1 2 2 2 1 1 1 2  2  2  2  2  2  1  2  2  2  2  2  1  2  1  1
```

```
model16 2 2 2 2 2 2 2 2 2  2  2  2  2  1  2  2  2  2  2  1  1  1  1
model42 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
model70 2 1 2 2 2 1 1 1 2  2  2  2  2  2  1  2  2  2  2  2  1  1  1  2
model94 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
        25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
model4   1  1  1  2  1  1  1  1  2  1  1  1  1  1  2  1  2  1  1  1  2
model16  1  1  1  1  1  1  2  2  2  2  2  2  2  2  2  2  2  1  1  1  2
model42  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
model70  1  1  1  2  2  1  1  1  1  1  1  1  2  1  1  1  2  1  1  1  2
model94  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
        46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
model4   1  1  2  2  1  1  1  2  1  1  1  1  1  1  1
model16  1  1  1  2  1  1  1  1  1  1  1  1  1  1  1
model42  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
model70  1  1  2  2  1  1  1  2  1  2  1  1  1  1  2
model94  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
```

```
> prediction[[2]]
            1    2      3      4      5      6    7    8    9   10
p         1.0000 2.00 3.0000 4.0000 5.0000 1.0000 2.00 3.00 4.00 5.00
alpha     0.0000 0.00 0.0000 0.0000 0.0000 0.2500 0.25 0.25 0.25 0.25
accuracy 0.6833 0.65 0.6667 0.6667 0.6667 0.7167 0.70 0.70 0.70 0.70
          11  12  13  14  15   16   17   18   19   20  21  22  23  24
p         1.0 2.0 3.0 4.0 5.0 1.00 2.00 3.00 4.00 5.00 1.0 2.0 3.0 4.0
alpha     0.5 0.5 0.5 0.5 0.5 0.75 0.75 0.75 0.75 0.75 1.0 1.0 1.0 1.0
accuracy 0.7 0.7 0.7 0.7 0.7 0.70 0.70 0.70 0.70 0.70 0.7 0.7 0.7 0.7
```

```
          25
p         5.0
alpha     1.0
accuracy 0.7
```

Recall that from the construction of our weighted scheme, when $\alpha = 1$ we have that all weights are equivalent. That is, we no longer have a weighted average but a usual arithmetic mean. As we can see from this output we can determine numerous parameter values and their corresponding accuracy. With this information, users can determine which weighted method and the strength of weights that are appropriate for their data set. As we can see given $p = 1$ and $\alpha = .25$ we obtain the highest predicted value which is larger than the usual arithmetic average when $\alpha = 1$. With this, it is possible to create a finer grid in-which we can (hopefully) produce a larger increase in accuracy.

4.4 Package Limitations and Expansion

There are many future usability extensions that can be made to enhance the **wbensembleR** package. One idea, previously mentioned, would the inclusion of multi-class data sets. This would be of particular interest since a great deal of classification problems are non-binary. Another idea, would allow this to generalize to allowing different models; as of now only one type of model can be considered. However, it may be of use to aggregate several different models.

CHAPTER 5. CONCLUSION

Since the beginning, this body of work was aimed at solving a specific problem in the data science area. As supervised learning is a large part of machine learning and has garnered increased attention over the years, it makes intuitive sense to develop more ideas and applications to support this trend. To that end, for various models, tunable parameters are a way to add a layer of flexibility so that the predictive learner can adhere to the data set more precisely. However, not all models possess such parameters; namely, this is a component missing for bagged ensembles.

This dissertation contains the entire gamut with respect to this issue. That is, through theoretical derivations, numerical computations, and an implementation of a software package, we have designed a framework that can be used by numerous individuals for tuning bagged ensembles.

Incrementally, the author first applied a weighted scheme to trees of the random forest by way of Cesáro averages. In doing this, the results showed that we can be competitive the random forest in certain cases in terms of prediction accuracy and more stable predictions. Moreover, from a theoretical standpoint, we were able to show conditions in which our methodology can be expected to outperform the random forest. Through the theory, we were able to determine a stopping condition which results in a way to potentially obtain higher accuracy and more consistent results while not constructing so many decision trees.

The next step taken was to generalized this to arbitrary models and a generalized weighting scheme. The implications of this framework resulted in a way to simultaneously control bias and variance. Namely a regularization term was added that allowed for convergence to a pure bagged ensemble. However, determining the optimal parameters can be a difficult

task. In our study, we performed a grid-search of numerous values. While this is a way to find the best parameters, another method may be better. Moreover, using this methodology it is unlikely that we are able to reduce bias to the levels of a boosted tree or a deep neural network (DNN); however, our methodology has shorter training times as well as only two parameters to tune unlike boosted trees and DNN which can have upwards of 30+ parameters. This leads to a simplification of use as well as a way to control for bias and variance in a convenient manner. With this, results were obtained that show the weighted methodology can perform competitively well to pure bagging and the random forest.

From a practical standpoint, the R package constructed acts as a wrapper around Max Kuhn's **caret**. Among practitioners, **caret** is a widely used and popular machine learning library since it contains many predictive learners as well as different combinations of structural combinations. Our R package **wbensembleR** works with **caret** so that no information is lost as shown in chapter 4. User's can download the package from `github.com/hieu-phamt/wbensembleR`.

It is the hope of the author that this dissertation comprehensively covers the first known theoretical foundation for tunability of bagged ensembles. In the end, we anticipate this theory and R package to used amongst the data science community.

5.1 Future Extension

Comparable to most open research problems, due to time limitations and other constraints, there are still many unsolved problems left to investigate. Within our weighted ensemble scheme, we only consider a particular weight; however, as a generalization to the theory and R package, one could consider having other sets of weighted averages to allow users more flexibility. To that end, we only consider a single model when bagging. However as stated in the introduction, one could consider a bagged stacked ensemble; although a

black-box, one could apply our idea to help strengthen the prediction accuracy of such ensembles. Lastly, our entire context was limited to two-class classification problems. Whereas binary classification is indeed a large area, there are still numerous data sets with more than two classes. For a generalization of our theory, one could represent and generalize the theoretical foundation to allow for multi-class classification. Moreover, as an extension of Chapter 3, we intend to further explore performance and default parameter settings in the context of the bias and variance of the base classifier, with potentially both a theoretical and empirical analysis.

BIBLIOGRAPHY

Adressi, A., Hassanpour, S. T., and Azizi, V. (2016). Solving group scheduling problem in no-wait flexible flowshop with random machine breakdown. *Decision Science Letters*, 5:157–168.

Apostol, T. (1976). *Introduction to Analytic Number Theory*. Springer New York.

Azizi, V., Jabbari, M., and Kheirkhah, A. S. (2016). M-machine, no-wait flowshop scheduling with sequence dependent setup times and truncated learning function to minimize the makespan. *International Journal of Industrial Engineering Computations*, 7(2):309–322.

Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository.

Bashir, S., Qamar, U., and Khan, F. H. (2016). A Multicriteria Weighted Vote-Based Classifier Ensemble for Heart Disease Prediction. *Computational Intelligence*, 32(4):615–645.

Bhasuran, B., Murugesan, G., Abdulkadhar, S., and Natarajan, J. (2016). Stacked ensemble combined with fuzzy matching for biomedical named entity recognition of diseases. *Journal of Biomedical Informatics*, 64:1–9.

Blei, D. M. and Smyth, P. (2017). Science and data science. *Proceedings of the National Academy of Sciences*.

Breiman, L. (1996a). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (1996b). Out-of-Bag Estimation. *Technical Report; Department of Statistics: UC Berkeley*, pages 1–13.

Breiman, L. (1998). Arcing classifiers. *Annals of Statistics*, 26(3):801–849.

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

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*, volume 19.

Bryll, R., Gutierrez-Osuna, R., and Quek, F. (2003). Attribute bagging: Improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition*, 36(6):1291–1302.

Buhlmann, P. and Hothorn, T. (2007). Boosting algorithms: Regularization, predicitng and model fitting. *Statistical Science*, 22(4):477–505.

Chen, T. and Guestrin, C. (2016). XGBoost : Reliable Large-scale Tree Boosting System. *arXiv*, pages 1–6.

Chen, T., He, T., Benesty, M., Khotilovich, V., and Tang, Y. (2018). *xgboost: Extreme Gradient Boosting*. R package version 0.6.4.1.

Collell, G., Prelec, D., and Patil, K. R. (2018). A simple plug-in bagging ensemble based on threshold-moving for classifying binary and multiclass imbalanced data. *Neurocomputing*, 275:330–340.

Cortes, C. and Vapnik, V. (1995). Support-Vector Networks. *Machine Learning*, 20(3):273–297.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, 20(2):215–242.

Dietterich, T. G. (2000). Ensemble Methods in Machine Learning. *Multiple Classifier Systems*, 1857:1–15.

Donoho, D. (2017). 50 Years of Data Science. *Journal of Computational and Graphical Statistics*, 26(4):745–766.

Ekbal, A. and Saha, S. (2013). Stacked ensemble coupled with feature selection for biomedical entity extraction. *Knowledge-Based Systems*, 46:22–32.

El Habib Daho, M., Settouti, N., Lazouni, M. E. A., and Chikh, M. E. A. (2014). Weighted vote for trees aggregation in Random Forest. In *International Conference on Multimedia Computing and Systems -Proceedings*, volume 0, pages 438–443.

Elith, J., Leathwick, J. R., and Hastie, T. (2008). A working guide to boosted regression trees.

Emerson, P. (2013). The original Borda count and partial voting. *Social Choice and Welfare*, 40(2):353–358.

Fattahi, P., Azizi, V., and Jabbari, M. (2015). Lot streaming in no-wait multi product flowshop considering sequence dependent setup times and position based learning factors. *International Journal of Engineering, Transactions A: Basics*, 28(7):1064–1071.

Fraenkel, J. and Grofman, B. (2014). The Borda Count and its real-world alternatives: Comparing scoring rules in Nauru and Slovenia. *Australian Journal of Political Science*, 49(2):186–205.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285.

Freund, Y. and Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Freund, Y. and Schapire, R. R. E. (1996). Experiments with a New Boosting Algorithm. *International Conference on Machine Learning*, pages 148–156.

Friedman, J. H. and Hall, P. (2007). On bagging and nonlinear estimation. *Journal of Statistical Planning and Inference*, 137(3):669–683.

Fumera, G., Roli, F., and Serrau, A. (2005). Dynamics of variance reduction in bagging and other techniques based on randomisation. Multiple Classifier Systems, pages 316–325. Springer Berlin Heidelberg.

Graefe, A., Küchenhoff, H., Stierle, V., and Riedl, B. (2015). Limitations of Ensemble Bayesian Model Averaging for forecasting social science problems. *International Journal of Forecasting*, 31(3):943–951.

Gunes, F., Wolfinger, R., and Tan, P.-Y. (2017). Stacked Ensemble Models for Improved Prediction Accuracy. *SAS*, pages 1–19.

Gupta, R., Kumar, N., and Narayanan, S. (2015). Affect prediction in music using boosted ensemble of filters. In *2015 23rd European Signal Processing Conference, EUSIPCO 2015*, pages 11–15.

Hendricks, P. (2015). *titanic: Titanic Passenger Survival Data Set*. R package version 0.1.0.

Hoeting, J. A., Madigan, D., Raftery, A. E., and Volinsky, C. T. (1999). Bayesian Model Averaging: A Tutorial. *Statistical Science*, 14(4):382–417.

Jiang, W. (2000). Does boosting overfit: Views from an exact solution. Technical report.

Jing, Y., Pavlović, V., and Rehg, J. M. (2008). Boosted Bayesian network classifiers. *Machine Learning*, 73(2):155–184.

Klamler, C. (2004). The Dodgson ranking and the Borda count: A binary comparison. *Mathematical social sciences*, 48(1):103–108.

Kuhn, M. and Johnson, K. (2013). *Applied predictive modeling*.

Kuhn, M., Wing, J., Weston, S., Williams, A., Keefer, C., Engelhardt, A., Cooper, T., Mayer, Z., Kenkel, B., the R Core Team, Benesty, M., Lescarbeau, R., Ziem, A., Scrucca, L., Tang, Y., Candan, C., and Hunt., T. (2017). *caret: Classification and Regression Training*. R package version 6.0-76.

Lacasse, A., Laviolette, F., Marchand, M., and Turgeon-Boutin, F. (2010). Learning with randomized majority votes. volume 6322 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 162–177.

Li, H. B., Wang, W., Ding, H. W., and Dong, J. (2010). Trees Weighting Random Forest method for classifying high-dimensional noisy data. In *Proceedings - IEEE International Conference on E-Business Engineering, ICEBE 2010*, pages 160–163.

Martínez-Muñoz, G., Hernández-Lobato, D., and Suárez, A. (2007). Selection of Decision Stumps in Bagging Ensembles. International Conference on Artificial Neural Networks, pages 319–328.

Microsoft Corporation and Steve Weston (2017). *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*. R package version 1.0.11.

Mittal, A. (2016). Hybrid Simulation Modeling for Regional Food Systems. Master's thesis, Iowa State University.

Mittal, A. and Krejci, C. C. (2017). A hybrid simulation modeling framework for regional food hubs. *Journal of Simulation*, pages 1–14.

Mosca, A. and Magoulas, G. D. (2018). Distillation of deep learning ensembles as a regularisation method. In *Smart Innovation, Systems and Technologies*, volume 85, pages 97–118.

Naghibi, S. A., Ahmadi, K., and Daneshi, A. (2017). Application of support vector machine, random forest, and genetic algorithm optimized random forest models in groundwater potential mapping. *Water Resources Management: An International Journal, Published for the European Water Resources Association (EWRA)*, 31(9):2761–2775.

Oshiro, T. M., Perez, P. S., and Baranauskas, J. A. (2012). How many trees in a random forest? In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7376 LNAI, pages 154–168.

Ozay, M. and Vural, F. (2012). A New Fuzzy Stacked Generalization Technique and Analysis of its Performance. *CoRR*.

Ponti, M. P. (2011). Combining classifiers: From the creation of ensembles to the decision fusion. In *Proceedings - 24th SIBGRAPI Conference on Graphics, Patterns, and Images Tutorials, SIBGRAPI-T 2011*, pages 1–10.

Quinlan, J. R. (2006). Bagging, boosting, and C4.5. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 5(Quinlan 1993):725–730.

Rish, I. (2001). An empirical study of the naive Bayes classifier. *Empirical methods in artificial intelligence workshop, IJCAI*, 22230(JANUARY 2001):41–46.

Ronao, C. A. and Cho, S. B. (2015). Random forests with weighted voting for anomalous query access detection in relational databases. In *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, volume 9120, pages 36–48.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408.

Saari, D. G. (1985). The Optimal Ranking Method is the Borda Count. Discussion Papers 638, Northwestern University, Center for Mathematical Studies in Economics and Management Science.

Schapire, R. E. (1990). The Strength of Weak Learnability. *Machine Learning*, 5(2):197–227.

Schapire, R. E. (2003). The boosting approach to machine learning: an overview. *Nonlinear Estimation and Classification*, 171:149–171.

Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. *Neural Networks*, 61:85–117.

Setak, M., Azizi, V., Karimi, H., and Jalili, S. (2017). Pickup and delivery supply chain network with semi soft time windows: metaheuristic approach. *International Journal of Management Science and Engineering Management*, 12(2):89–95.

Sill, J., Takács, G., Mackey, L. W., and Lin, D. (2009). Feature-weighted linear stacking. *CoRR*, abs/0911.0460.

Stein, E. M. and Shakarchi, R. (2003). *Fourier Analysis: An Introduction*.

Strobl, C., Boulesteix, A. L., and Augustin, T. (2007). Unbiased split selection for classification trees based on the Gini Index. *Computational Statistics and Data Analysis*, 52(1):483–501.

Subasi, A., Alickovic, E., and Kevric, J. (2017). Diagnosis of Chronic Kidney Disease by Using Random Forest. In *CMBEBIH 2017*, volume 62, pages 589–594.

Valentini, G., Muselli, M., and Ruffino, F. (2004). Cancer recognition with bagged ensembles of support vector machines. *Neurocomputing*, 56(1-4):461–466.

van der Laan, M. J., Polley, E. C., and Hubbard, A. E. (2007). Super Learner. *Statistical Applications in Genetics and Molecular Biology*, 6(1).

van Dop, H. and Steyn, D. G. (1990). *Air Pollution Modeling and Its Application VIII*.

Vezhnevets, A. and Barinova, O. (2007). Avoiding Boosting Overfitting by Removing Confusing Samples. *Work*, pages 430–441.

Weisstein, E. (2004). Harmonic series. Champaign, IL, 2018.

Wickham, H. (2014). Tidy Data. *Journal of Statistical Software*, 59(10).

Wickham, H. (2016). *tidyverse: Easily Install and Load 'Tidyverse' Packages.*

Winham, S. J., Freimuth, R. R., and Biernacka, J. M. (2013). A weighted random forests approach to improve predictive performance. *Statistical Analysis and Data Mining*, 6(6):496–505.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5(2):241–259.

Wu, G., Sanner, S., and Oliveira, R. F. S. C. (2015). Bayesian model averaging naive bayes (bma-nb): Averaging over an exponential number of feature models in linear time. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 3094–3100. AAAI Press.

Zenko, B. (2004). Is Combining Classifiers Better than Selecting the Best One? *Machine Learning*, 54(3):255–273.

Zhou, Z. (2012). *Ensemble Methods: Foundations and Algorithms*. Chapman and Hall.