Stockholms universitet

Evaluating Forecast Accuracy of Random Forests Versus LS_Boost on Simulated Time Series

Adam Goran

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2021:9
Matematisk statistik
Juni 2021

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Matematiska institutionen

Mathematical Statistics
Stockholm University
Bachelor Thesis **2021:9**
http://www.math.su.se

Evaluating Forecast Accuracy of Random Forests Versus LS_Boost on Simulated Time Series

Adam Goran*

June 2021

Abstract

No machine learning algorithm is perfect, but some are superior at providing accurate predictions and forecasts. In this report, we compare the ability of random forests and LS_Boost to produce one-step forecasts. The study regards mainly the object of finding which algorithm is able to attain the smallest forecast error for unseen data. Different situations are considered to diversify the work and accomplish a fair comparison. Data is obtained from simulations, where three established financial time series models serve as frameworks. In order to understand and apply the algorithms correctly we introduce the theory of regression trees, bagging and boosting. The specific algorithm for random forests and LS_Boost respectively is also presented, along with some general knowledge about the three time series models. We find that LS_Boost achieves lower forecast error in terms of both mean squared error and mean absolute error for all simulations. This is the case regardless of whether the parameters of the algorithms are tuned. Random forests is close in performance though and does actually approach the performance of LS_Boost as the simulated data becomes more sparse.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: adam.goran@hotmail.com. Supervisor: Ola G H Hössjer, Kristofer Lindensjö.

Acknowledgements

This is a bachelor's thesis of 15 ECTS in Mathematical Statistics at the Department of Mathematics at Stockholm University. I am grateful for the guidance from my special supervisors Ola G H Hössjer and Kristofer Lindensjö. Their valuable help has facilitated the work.

Abstract

No machine learning algorithm is perfect, but some are superior at providing accurate predictions and forecasts. In this report, we compare the ability of random forests and LS_Boost to produce one-step forecasts. The study regards mainly the object of finding which algorithm is able to attain the smallest forecast error for unseen data. Different situations are considered to diversify the work and accomplish a fair comparison. Data is obtained from simulations, where three established financial time series models serve as frameworks. In order to understand and apply the algorithms correctly we introduce the theory of regression trees, bagging and boosting. The specific algorithm for random forests and LS_Boost respectively is also presented, along with some general knowledge about the three time series models. We find that LS_Boost achieves lower forecast error in terms of both mean squared error and mean absolute error for all simulations. This is the case regardless of whether the parameters of the algorithms are tuned. Random forests is close in performance though and does actually approach the performance of LS_Boost as the simulated data becomes more sparse.

Contents

1 Introduction

Machine learning is a growing field of mathematical statistics and becomes more relevant within several professions for each year. In turn, supervised learning is the largest field of machine learning for justified reasons. One reason is its applicability to real life problems. New algorithms or modifications of existing ones appear frequently, as there exists an endeavour to beat established algorithms and reach the lowest possible errors.

Time series models possess some properties that make them different from other regression models and they hence become interesting to understand and predict. It is a difficult task, especially if the underlying model is unknown. For this reason, we want to apply rather complex machine learning methods along with linear methods for estimation on different time series to evaluate their performance. Simulations will provide data from three common time series models.

Among the most popular machine learning algorithms are boosting and random forest. They make an excellent subject for comparison, as they both exploit the idea to combine many simple, often bad in their self, learners to achieve better results. The learner is in this case regression trees, which are inspired by the decision making process in real life and are easy to interpret. These algorithms have proven to be highly competitive and not very hard to implement, hence their popularity.

Random forests was developed from its precursor bagging and does not have many well known variants. Boosting on the other hand, comes with various implementations and variants, each with a unique name. One of them was proposed in the same year as random forests, namely the LS_Boost algorithm. This algorithm has a natural connection with regression problems with a relatively simple implementation. Understanding LS_Boost and gradient boosting in general is a good idea before getting into the more recent variants of boosting, such as LightGBM, XGBoost and CatBoost.

1.1 Outline

This paper starts off with a theory section, where the different subsections follow a natural order so that it is preferable to read from beginning to end. The purpose of this section is to give the reader proper information about the algorithms. An experienced programmer should be able to implement the algorithms without any prior knowledge of them. Following is the simulation section, where the aim is to find out which algorithm performs best. Readers with good background knowledge can skip to this section.

2 Theory

2.1 General concepts of machine learning

This study falls into the category of supervised learning where the object is to predict an output y from inputs x. We say that x is an observation from the random variable X with unknown distribution. There are in general p input variables such that $X = (X_1, X_2, \ldots, X_p)$ is multivariate. Furthermore, y is an observation from Y that is assumed to be connected to X via

$$Y = f(X) + \epsilon.$$

Hence, Y is another random variable. The term ϵ stands for the noise occurring between Y and $f(X)$. Noise can be explained as a random error that is assumed to be 0 on average and independent of X. The aim is thus to estimate $f(X)$ only. We do this using a training set consisting of N paired observations. Consequently, for observation $i = 1, 2, \ldots, N$ the input-output pair is (x_i, y_i) where $x_i = (x_{i1}, x_{i2}, ..., x_{ip})$. These notations are in line with [1] (introduced in section 2.1, pages 15-16).

In the context of machine learning, bias and variance is vaguely defined and referred to in many different situations. Formally, bias is defined as

$$\text{Bias}(\hat{f}(X)) = E(\hat{f}(X)) - f(X)$$

for an estimator $\hat{f}(X)$ of the fixed but unknown true function $f(X)$. This is known as the systematic deviation of our estimator from the true function. The variance is per usual defined as

$$\text{Var}(\hat{f}) = E((E(\hat{f}) - \hat{f})^2)$$

where the dependence of $\hat{f}$ on X has been made invisible for clarity. This quantity is known as the source of error originating from the estimator's variation around its mean. It is a measure of sensitivity when small changes are made in the training set. If we were to repeat an experiment on new data points from the same distribution, high variance means that the estimates would differ a lot on average. Note that the bias function is defined conditionally on X, so it is actually a function of X, whereas X is random in the definition of the variance. Now, from [1] (page 34, equation 2.7) we get that the average squared prediction error on a previously unobserved data point x_0 can be decomposed as

$$E(y_0 - \hat{f}(x_0))^2 = \text{Var}(\hat{f}(x_0)) + \text{Bias}(\hat{f}(x_0))^2 + \text{Var}(\epsilon), \tag{1}$$

which is known as the expected test mean squared error (MSE). The term $\text{Var}(\epsilon)$ can serve as a threshold value for how low the MSE can go because it is an irreducible source of variation. Equation (1) can be derived from

$$E(y - \hat{y})^2 = E(f + \epsilon - \hat{f})^2 = E(f - \hat{f})^2 + \text{Var}(\epsilon)$$

since $\hat{y} = \hat{f}$. A separate testing set is thus required in order to estimate the MSE.

Bias is in this report and occasionally in the literature referred to as intercept or constant terms. It is also referred to as training error because variance defined as above does not exist on training error due to the data set being fixed. Bias in the setting of training error is closely related to the concepts of overfitting and underfitting. When bias is very low the model has probably picked up some of the noise, leading to an overfit. On the contrary, high bias probably means that the important structures of the data are not modelled entirely, which is known as underfitting. Meanwhile, variance is sometimes referred to test error due to the evaluation on new data. It will be clear from the context what each term stands for and it is usually the definitions above.

2.2 Regression trees

Regression trees have a major role in both of the algorithms compared in this report and is clarified later on. It is therefore critical for the purpose of the analysis to understand them. A regression tree is a decision tree where the response variable is continuous. They appeal to our intuition since many decisions in real life are made using the same logic. All material in this subsection will be taken from [2] (section 9.2) unless stated otherwise. The implementation used here, called CART (classification and regression trees), was introduced in [3].

Tree-based methods in general are powerful since they can capture non-linear complex structures in the data and have low bias when grown deep, see [2] (page 305). They are also simple from a computational perspective and work with single as well as multiple regressors. The process of growing a tree is recursive such that the data is splitted in a binary fashion according to some condition in each step. In other words, the data is first divided into two pieces, or branches, then each branch is divided into two parts and so on. Eventually, a sufficient tree is formed in which the final partition defines regions for the explanatory variables. The interpretability of this method is one of its advantages, since they for instance are easy to display graphically.

A mathematical description will now be provided, in accordance with [2] (page 307), and the notation here is used throughout the report. First off, a decision is needed regarding how to model the final regions that the method produces. This is an issue of how to make predictions of the response variable within each region $R_1, R_2, ..., R_J$, where J is the total number of regions. The common choice is to model the regions with different constants, that is, for region j we model the response with a constant c_j, but other functions could be used as well. Only constants are considered from now on. It is easy to show that the optimal constant according to the sum of squared deviance

criterion is the average of the y_i within the region. So for region j we set

$$\hat{c}_j = \text{mean}(y_i | x_i \in R_j),$$

corresponding to equation 9.10 in [2]. This can be shown by differentiating

$$L = \sum_{x_i \in R_j} (y_i - c_j)^2$$

with respect to c_j and then solve $L' = 0$. One should interpret $\hat{c}_j$ as the optimal constant given a fixed training set.

Next up, we treat how the regions are formed by defining a similar procedure as in [2] (page 307). It is often impossible to find the best partition of data in terms of least squares, which is one of the flaws that regression trees exhibit. Instead, the way to go is to start with all the data in a greedy approach. The task is to find an optimal variable k to split and an optimal point s that splits the data into two regions, namely

$$R_1(k, s) = \{X | X_k \leq s\} \text{ and } R_2(k, s) = \{X | X_k > s\},$$

where the upper-case letters now denote unobserved quantities. To be precise, the split points are always chosen to be one of the values in the data, for example x_{72}, the seventh observation of the second explanatory variable. Now, given the data points of pairs (x_i, y_i), we want to solve

$$\min_{k,\, s} \left[\min_{c_1} \sum_{x_i \in R_1(k,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(k,s)} (y_i - c_2)^2 \right].$$

This equation is gathered from [2] (equation 9.13). With insight from before the inner sums are minimized by choosing the constants as the mean of the response variables within the region, given k and s. It thus turns down to scanning through the different explanatory variables and its different inputs, which is very feasible for a computer (as long as the size of the data set is not humongous). When the optimal values are found the algorithm can be applied to each region in the new partition of the data. With a final tree, one can predict a response from new input belonging to region R_j with the value $\hat{c}_j$.

In Figure 1, there is an example of a regression tree. At first, the split is made at $x_1 = 0$ and divides the data into two parts where observations larger or equal to 0 go down the right side of the tree. Then, more splits are made, eventually resulting in five regions where $c_1, \ldots, c_5$ are shown at the terminal nodes. This tree includes two variables, but the dataset could nevertheless consist of more, seemingly irrelevant, input variables.

The question arises when to finish the process, which is discussed in [2] (page 308). This is an important question since overfitting will become an

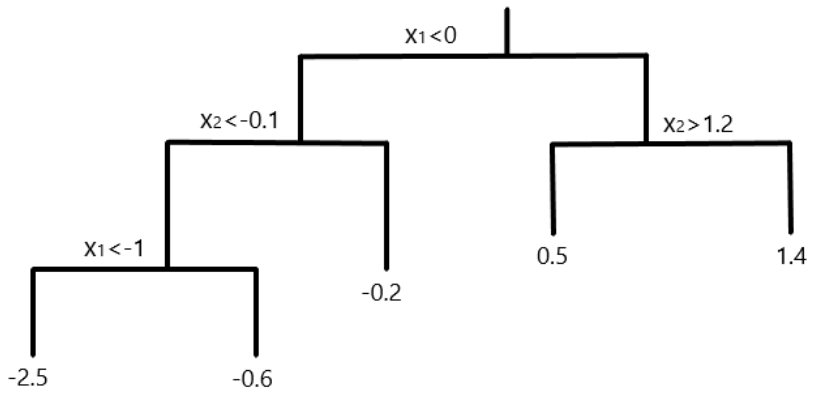

Figure 1: A simple regression tree including two variables

issue with a very large tree while a too small tree probably will underfit. However, this is of less practical importance since the programs available often deal with this automatically in an effective way. Nevertheless, a preferred method to address this, called *cost-complexity pruning*, will be described in the following.

The term pruning comes from the idea of growing a large tree T_0 as a starting point and then trying to make it smaller without increasing the sum-of-squares too much. The regions defined by T_0 should only contain some minimal number of data points, say 5, which governs how large T_0 can be. To begin with, let $T \subset T_0$ be any subtree. That is, we can obtain T from T_0 with pruning, which means that regions are collapsed. Further, let $|T|$ be the number of resulting regions from T and denote by N_j the number of observations within region R_j. We now adopt the cost complexity criterion (cf. [2], equation 9.16)

$$C_\alpha(T) = \sum_{j=1}^{|T|} \sum_{x_i \in R_j} (y_i - \hat{c}_j)^2 + \alpha|T| \tag{2}$$

that should be minimized for a given $\alpha \geq 0$. In fact, α is a tuning parameter that controls the tradeoff between tree size and parsimonity. When $\alpha = 0$ the solution is the full tree T_0. One can show that for each α there exists T_α, the smallest unique subtree that minimizes $C_\alpha(T)$. To find this tree, one should create a sequence of subtrees where two or more regions are collapsed successively until there are no more branches or no decrease in $C_\alpha(T)$. The regions to collapse in each step should cause the double sum of squares in (2) to increase the least and they should be determined by testing to cut all nodes of the tree except the terminal ones. It is possible to show that

the optimal tree is included in this sequence. Using *cross-validation* one can find the optimal value of α, resulting in the tree $T_{\hat{\alpha}}$. For a description of cross-validation, see [1] (pages 175-181). Pruning will not be used for neither LS_Boost nor random forests, but will come in handy when we want to fit a single tree to data. One can argue that decision trees in general are not optimal for regression problems, since they produce piecewise constant approximations of a smooth function.

2.3 Random forests

2.3.1 Bagging

Bagging is the precursor of random forests. By describing bagging it is easy to implement random forests since they only differ in a minor fashion. The name is an abbreviation of bootstrap aggregating, a name that describes the idea behind it. All information in this section about bagging is gathered from [4], which is the original source of this algorithm. Following is a brief introduction to the method in general, after which more details will be presented. We will only consider the version of bagging used for regression problems.

In [4] (section 1) the concept is introduced, but the notation differs from the description here. Starting with a data set D with a response variable and one or multiple explanatory variables, a predictor $\hat{f}(x, D)$ is formed that depends on the training data D and the vector of explanatory variables x to use in the prediction. Random forests and often bagging as well use a regression tree as prediction function. The underlying idea of this method is that many predictors outperform a single predictor. Hence, the assumption is that there exists a sequence of data (learning) sets $\{D_k\}$, where $k = 1, \ldots, B$, that originate from the same distribution as D and consists of N independent observations. Only the sequence of predictors $\{\hat{f}(x, D_k)\}$ will be used in the final prediction, not $\hat{f}(x, D)$. To obtain a prediction, one simply takes the average of $\hat{f}(x, D_k)$ over all k.

It is not realistic to believe that the sets $\{D_k\}$ already are available in real life situations. The solution is to replicate the original data set D by drawing bootstrap samples with replacement, see [4] (section 1). This means that any observation (y_i, x_i) could appear multiple times in the bootstrapped sample D_k or not at all. We draw B such samples in which each observation from D is randomly selected with probability $1/N$. The size of the bootstrapped sample D_k is usually chosen to be N, the same size as D, which thus on average contains about $1 - e^{-1} \approx 63\%$ distinct observations from D. The latter statement can be verified from the binomial distribution with parameters N and $1/N$. Regression trees work well as prediction functions $\hat{f}$ in the bagging estimate since they generally are good but rather unstable in regression problems. They are unstable in the sense that small changes

in the data set can result in large changes in its predictions.

Let us now justify the bagging algorithm and show why it will work, as presented in [4] (section 4.1). Assume that each observation in D is independently drawn from a probability distribution P with $\hat{f}(x, D)$ as predictor. The aggregated predictor over D can be defined as

$$\hat{f}_A(x) = E_D \hat{f}(x, D). \tag{3}$$

Remember that the aggregated predictor takes the average of the different predictions, which explains its connection to the expected value. The lowered D in (3) clarify that the expectation is taken over the data set and not the explanatory variables, and the lowered A denotes aggregation. Now, let y be a response and x be taken as fixed input and write

$$E_D(y - \hat{f}(x, D))^2 = y^2 - 2yE_D\hat{f}(x, D) + E_D\hat{f}^2(x, D), \tag{4}$$

corresponding to equation 4.1 in [4]. If the inequality $EX^2 \geq (EX)^2$ is applied to the third term in the right hand side of (4) and the substitution in (3) is used afterwards one obtains

$$E_D(y - \hat{f}(x, D))^2 \geq (y - \hat{f}_A(x))^2. \tag{5}$$

Both sides of (5) can be integrated over the joint distribution of x and y. This corresponds to taking the sum over all observations if D is estimated, as discussed below. Then, the conclusion can be drawn that the mean-squared error of $\hat{f}_A(x)$ is lower than that of $\hat{f}(x, D)$ averaged over D. The size of the difference of the two sides in the inequality is governed by how much bigger the left side is than the right side in

$$E_D\hat{f}^2(x, D) \geq (E_D\hat{f}(x, D))^2.$$

We can interpret the past arguments as the original predictor having a variance around its own mean, which is decreased by aggregating, since the mean is the theoretical bagging estimate. It is also easy to see why unstable predictors benefit from bagging. If a replicate of D is used in the prediction, causing it to change a lot, then the two sides in (5) will be far from equal.

So far we have assumed that the aggregated predictor depends on x and implicitly on the distribution P, but a bagged estimate in practice depends on another distribution P_D, which is the bootstrap approximation of P. See [4] (section 4.3). The distribution P_D puts equal weight $1/N$ at each point $(y_i, x_i), i = 1, 2, \ldots, N$ of D. Since B is a finite and often small number, our estimate is an approximation of the true bagged estimate. In fact, it is a Monte Carlo estimate of the estimate, converging to the true bagging estimate as $B \to \infty$.

More insights can be given that further highlight the limitation of bagging, see [4] (end of section 4.1). A stable predictor will produce an estimate

from the data set similar or equal to the estimate of the corresponding aggregated estimate that is based on the underlying distribution of the data. This means that there are no gains in trying to estimate the distribution through a bootstrap. Another drawback with bagging is that it will impair predictions that are close to the lowest attainable limit in terms of standard error.

The value of B does not need to be large, around 20 is often enough as discussed in [4] (section 6), although it might be reasonable to increase this number if the running time is low. Larger samples than N make it more likely for more distinct observations to be included in each sample, but are reported not to improve accuracy. Overall, bagging is a simple method that is easy to implement. When used on regression trees it almost certainly increases accuracy, but the straightforward interpretation of one tree is ruined. This comes from the fact that a bagged tree is no longer a tree.

2.3.2 Random forests algorithm

Random forests exploit the idea of bagging and are in many respects the preferable choice. In this section, the general concept and the algorithm will first be described, after which theoretical properties will be provided. Random forests are popular and improve bagging on trees by trying to reduce the correlation between the individual trees. Unless otherwise stated, all the material referred to is taken from [2]. The original source is [5], but it is not used here because the content of that article is presented mostly in the setting of classification. The general procedure for random forest is described in Algorithm 1, in similarity to [2] (algorithm 15.1, page 588).

A closer look at Algorithm 1 exposes a few tuning parameters: the number of trees B, the sample size N, the minimum region size n_{min} and the number of variables m to split at random. We can see that the only difference from bagging is that the candidates of variables to split should be chosen at random, instead of all variables being taken into account. Thus, if we only have $p = 1$ input variable or lag in a time series model, the result will be identical to bagging. One could also use the regularisation described in the regression tree section to obtain pruned and more optimal trees according to the cost-complexity criterion, but it is not common and will be disregarded here since random forests seldom overfit the training data. The output, or estimator, corresponding to line 2 of Algorithm 1 can be expressed as

$$\hat{f}_{\text{rf}}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x).$$

The reason for the random variable split is to reduce the correlation between trees and thus reduce variance of the estimate, see [2] (page 588). Suppose that we have K i.i.d. random variables with variance σ^2. An average

Algorithm 1 Random forests for regression

1. For $b = 1$ to B :
   (a) Draw D^*, a bootstrap sample of size N from the data devoted to training.
   (b) Build a random-forest tree T_b, which is somewhat different from the regular regression tree, to the bootstrapped data. The following steps should be recursively repeated on each terminal node of the tree, starting with the whole data set D^*, until this terminal node contains at most the minimum number n_{min} of observations.
      i. There are p input variables from which m should be chosen at random.
      ii. Among the m variables, pick the best variable to split and the best splitting point of the data, according to the theory of regression trees.
      iii. Split the data into two new regions.
2. The output is now the ensemble of trees $\{T_b\}_1^B$. To predict a response from a new input, simply apply the input to all trees and take the average.

of these variables has variance $\frac{1}{K}\sigma^2$. But if the variables are not independent and have a pairwise positive correlation of ρ the variance of the average instead becomes

$$\rho\sigma^2 + \frac{1-\rho}{K}\sigma^2. \tag{6}$$

The second term of (6) is neglectable when K is large, but the first term remains. Random forests may increase the variance σ^2 to some extent, but will decrease ρ and therefore it is a better estimate, at least for large K. This is the only way to improve the accuracy of an estimate with the bagging technique because bias cannot be reduced by averaging. The past statement follows from the fact that the expectation of one tree is the same as the expectation over the average of many trees, implicating that the bias is the same too.

In Algorithm 1, the parameter m is recommended to be $p/3$ rounded down to the nearest integer (cf. [2], page 592), although in practice it is good to try different values, especially the case $m = 1$. Also, the authors of [2] recommend that the minimum size n_{min} of each region should be 5. The number of bootstrap samples B need not to be very large, but increasing the number of samples seldom affects the time it takes for computers to make calculations. We will always set N to be the same number as the number of training observations, which is the commonly accepted choice.

Random forests only problem with overfitting occurs when there are many variables, but few relevant ones. Choosing a higher value of m might fix this if the data set is large. Also, if fully grown trees are used, with $n_{min} = 1$, overfitting may occur as $B \to \infty$. The reader is referred to [2] (page 596) for a thorough discussion.

Some more aspects of random forests will now be considered, referring to [2] (pages 597-599). First, denote by Θ_b the characteristics of the bth tree such that $T(x; \Theta_b)$ is completely defined in terms of what variables and values are splitted and the values of the final regions. Now, letting $B \to \infty$, the estimator of the random forest becomes

$$\hat{f}_{\text{rf}}(x) = \text{E}_{\Theta}(T(x; \Theta)), \tag{7}$$

where it is understood that Θ depends on the data used for training and x denotes a single new data point. From (6) we get that the variance of this estimator is

$$\text{Var}(\hat{f}_{\text{rf}}(x)) = \rho(x)\sigma^2(x).$$

Here, ρ and σ depend on x since they are defined as

$$\rho(x) = \text{corr}[T(x; \Theta_1(D_1^*), T(x; \Theta_2(D_2^*)]$$

and

$$\sigma^2(x) = \text{Var}(T(x; \Theta(D^*)).$$

That is, $\rho(x)$ is a sampling correlation of two randomly drawn trees from the forest built on randomly sampled data sets D_1^* and D_2^*. Whereas $\sigma^2(x)$ is interpreted theoretically as the, of any randomly drawn tree, sampling variance. It is important to recall that this is only true as B converges to infinity, whilst in practice the true estimate is an approximation of (7) of varying accuracy.

There is an important feature with random forests that makes it more practical than many other non-linear estimators, see [2] (page 593). It is the use of so called out-of-bag (OOB) samples. If for each observation $z_i = (x_i, y_i)$ we construct a random forest predictor by averaging the trees from bootstrap samples in which z_i did not appear, we can estimate the OOB error. This estimate is obtained by taking the mean of the difference of the predictions and the actual values. Performing N-fold cross validation is close to an identical estimate, which thus shows the importance of this feature. The process of fitting a random forest model can be done in one sequence, terminating when the OOB error stabilises. OOB error is irrelevant in this report, but the OOB samples prove to be useful anyway.

2.3.3 Variable importance

Variable importance needs to be discussed for random forests, see [2] (page 593). Unlike boosting, explanatory variables will not be entirely ignored in

the process of growing the trees because of the random variable selection (unless for rare situations). This may be one of the weaknesses of random forests as it tends to overrate the importance of some variables. Still, variable importance tests can be performed to evaluate if some variables are neglectable. One option is to follow the same procedure as for gradient boosted models. Another option is to use the OOB samples in the following fashion. For the bth tree, record the accuracy of the predictions made on the OOB sample, after which the values for the jth variable should randomly be permuted in the sample and the accuracy should be computed again. This generally decreases the accuracy, where the important variables tend to decrease the accuracy more as they are more common as split variables. The process is done for each variable and the average of the decrease in accuracy of the prediction error is taken over all the trees. If the accuracy on the shuffled sample instead is increased, it basically means that the variable has very small predictive power and should advantageously be excluded from further analysis.

2.4 LS_Boost

Boosting is a popular method because it has proven to be truly competitive, see [2] (page 337). New algorithms keep getting developed to further improve the performance. LS_Boost is an application of regular gradient boosting where the loss function equals the least squares, hence the name LS. The general idea of boosting is to build trees sequentially, so that the next tree is built conditionally on information from the previous tree. Modifications to the original training set are made based on this information. Because the next tree uses information from the past tree, this means that any tree in the sequence depends on all the previous trees.

The idea that motivates the boosting procedure for regression trees is that a single tree must be grown large to capture the patterns of the data, making it sensitive to overfitting, cf. [1] (page 322). Boosting instead uses small trees to slowly improve $\hat{f}$ where it did not perform well. Other statistical learning methods tend to perform well if they learn slowly. The reason why small trees are used, other than making the predictor learn slowly, is due to their low variance and bias, since boosting in theory only decreases bias. Another factor is that overfitting seldom becomes a problem with small trees.

2.4.1 LS_Boost algorithm

In this subsection, most of the material is drawn from [7], where the definition of this algorithm for regression problems is established. However, it is referred to as LS_TreeBoost in the article even though it is not defined explicitly (see specifically algorithm 2 and 3). This is because LS_Boost, as defined

in [7], does not exclude the possibility to be used in conjunction with other base learners than trees. The regression tree function is called a base learner in this context, which basically means the function being boosted. Other examples of base learners could be neural networks or k-nearest neighbour algorithms. The algorithm used in this report is summarised in Algorithm 2.

Algorithm 2 LS_Boost for regression trees

1. $f_0(x) = \bar{y}$
2. for $b = 1$ to B do:
   (a) $\tilde{y}_i = y_i - f_{b-1}(x_i)$, for all $i = 1, 2, \ldots, N$
   (b) fit $\{R_{jb}\}_1^J$, a tree with J regions on $\{\tilde{y}_i, x_i\}_1^N$
   (c) $\gamma_{jb} = \text{mean}_{x_i \in R_{jb}}\{y_i - f_{b-1}(x_i)\}$, $j = 1, \ldots, J$
   (d) $f_b(x) = f_{b-1}(x) + \upsilon \sum_{j=1}^J \gamma_{jb} 1(x \in R_{jb})$
3. output: $f_B(x)$

Algorithm 2 uses the indicator function $1(\cdot)$ that equals zero if its argument is false and 1 otherwise. Line 3, the output line, can equivalently be expressed as

$$\text{output: mean}(y) + \upsilon \sum_{b=1}^{B} \sum_{j=1}^{J} \gamma_{jb} 1(x \in R_{jb}).$$

We can see that the loss function L is least squares since line 2 (a) of Algorithm 2 uses the gradient

$$\frac{-\partial L(y_i, f(x_i))}{2\partial f(x_i)} = \frac{-\partial (y_i - f(x_i))^2}{2\partial f(x_i)} = y_i - f(x_i).$$

This also explains why the algorithm is called a gradient boosting algorithm.

In Algorithm 2, the first step is to determine a preliminary predictor $\bar{y}$ of the output. Proceeding to the loop, if $\hat{y}$ is the predictor of y after $b-1$ iterations, the residuals $\tilde{y} = \hat{y} - y$ are calculated during iteration b. Then, a regression tree is fitted to these residuals using the explanatory variables. The next step is to calculate the $\{\gamma_{jb}\}_{j=1}^J$ values, which is a problem of finding the optimal constants to piecewise update the output. Piecewise updates refers to the fact that the tree $\{R_{jb}\}_{j=1}^J$ decides boundaries according to which the updates are made. Anyhow, the object of finding optimal constants corresponds to solving

$$\gamma_{jb} = \min_{\gamma} \sum_{x_i \in R_{jb}} L(y_i, f_{b-1}(x_i) + \gamma)$$

for an arbitrary loss function L. The least squares loss function gives the solution of line 2 (c) in Algorithm 2, since

$$\gamma_{jb} = \min_{\gamma} \sum_{x_i \in R_{jb}} (y_i - (f_0(x_i) + \gamma))^2 = \min_{\gamma} \sum_{x_i \in R_{jb}} ((y_i - f_{b-1}(x_i)) - \gamma))^2.$$

This equation clearly has the solution

$$\gamma_{jb} = \text{mean}_{x_i \in R_{jb}}\{y_i - f_{b-1}(x_i)\} = \text{mean}_{x_i \in R_{jb}}\{\tilde{y}_i\}$$

according to previous statements, see for example the section about regression trees. Furthermore, the output function is then adjusted using these constants shrinked by υ. This is afterwards repeated for the updated output function until the process terminates. Note that the boundaries for x are typically different in each iteration, so for example

$$\{x_i \in R_{j=1,b=1}\} \neq \{x_i \in R_{j=1,b=2}\}.$$

The point here is that the output function becomes more complex as b grows. Note also that large residuals are given more weight than smaller ones as they to some degree dictate how the regions are defined. An interesting result of the algorithm is that the training error decreases for every iteration, since the output by definition yields smaller residuals every time it is updated.

There are a few tuning parameters for this algorithm, including B, J and υ. Empirical studies have shown that so-called stumps work well together with large values of B and are sometimes even optimal, cf. [7]. Stumps designate trees with only one split, that is, where $J = 2$ in this context. It could be mentioned though that more input variables (a larger p) often leads to a higher optimal value of J.

In the algorithms of [7] the parameter $0 < \upsilon \leq 1$ is not included, but suggested later in the article (section 5) as an improvement technique. It is referred to as the shrinkage parameter. The process of updating the output in Algorithm 2 can be counterproductive in prediction problems where training data is fitted too closely. One obvious way to solve this is to choose the value B from cross validation to make sure that the model does not overfit the training data. A better option is to regulate υ, since it is found that combining a smaller υ with a higher value of B provides better predictions in general. This is a common technique in boosting methods, which gives more accurate results in general at the cost of increased computational time. However, this should not be a serious problem as large trees are not desirable and unlikely to be necessary. Therefore the number of computations per iteration b is kept so small that the total computation time remains feasible.

2.4.2 Variable importance

In [7] (section 8) it is described how to interpret a boosted model. When boosting is applied to really small trees that only contain one split of the

data, the ensemble fits an additive model, which follows from the fact that each term involves only one variable. The advantage of this model lies in interpretability, because the final predictor is just the arithmetic sum of the effects of the individual predictors. Other than that, boosting is generally difficult to interpret, but one can at least estimate the relative importance of explanatory variables to attain some degree of interpretation. The following measure will be used for a single tree T and it is defined in equation (44) of [7] as

$$F_\ell^2(T) = \sum_{t=1}^{J-1} \hat{\imath}_t^2 I(v(t) = \ell). \tag{8}$$

In a binary splitted tree with J terminal nodes are there $J-1$ internal nodes (splitting points). The measure in (8) is defined as the squared relevance of the input variable X_ℓ. The indicator function $I(\cdot)$ within the sum determines whether variable X_ℓ was used in the split number t. Because t is an internal node, $v(t)$ is used to denote the index of the splitting variable. The value of $\hat{\imath}_t^2$ is simply the reduction in squared error due to the split t. For the ensemble of B trees, this measure is generalised to be the average of (8) applied to all trees, giving

$$F_\ell^2 = \frac{1}{B} \sum_{b=1}^{B} F_\ell^2(T_b).$$

This measure F_ℓ^2 is definitely more accurate than the corresponding measure $F_\ell^2(T)$ for a single tree. Note that the tree T_b is in the setting of Algorithm 2 equivalent to $\{R_{jb}\}_{j=1}^J$. From here on it is straightforward to compare the variables, commonly by scaling such that the variable with the largest value is scaled to 100 and the others accordingly. One must only remember to take the square roots of the relevance values before comparing them.

2.4.3 Stochastic gradient boosting

A useful addition to the boosting method that exploits the main advantage of bagging is to sample a fraction η without replacement from the training data in each iteration and use that subsample to grow the next tree, see [6]. This is called stochastic gradient boosting and it will be utilised in the analysis section of this report. It can be shown to reduce variance with the same type of reasoning as for bagging, and it also has the benefit of reducing the computing time with the same fraction η. Nothing else is changed in the algorithm. The standard value is $\eta = \frac{1}{2}$, but if the data set is large the fraction could be chosen much lower. One downside with this modification is that one more parameter η needs to be determined in the training process. Returning to Algorithm 2, we would have to sample ηN data points in each iteration and change N for ηN in line 2. (a) and (b) in order to implement this idea.

2.5 Time series models

The following models are common, or at least extensions of common models, in financial time series analysis. They have been developed to explain some of the patterns that such time series usually follow. Financial time series could for example be asset returns. Less common to model is the actual price of an asset. The models share the common feature that current values express dependency on previous values in the series. This means that future values can be predicted using a couple of recently observed values, which is called forecasting. Data will be simulated from these models to compare least squares boosting with random forests.

2.5.1 AR models

We start off with a simple model that is linear and lays the foundation of other more complex models, the autoregressive (AR) model. The information is gathered from [8] (pages 37-46). An AR(1) model has only one lag, that is, it only depends on its last observed value and is defined as

$$r_t = \phi_0 + \phi_1 r_{t-1} + a_t. \tag{9}$$

Here, ϕ_0 and ϕ_1 are constants and r_{t-1} is the previous value clarified from its index. The chock at time t is denoted by a_t and is assumed to be a random variable with mean 0 and variance σ_a^2. The series $\{a_t\}$ is called *white noise* and deals with the randomness that usually appears in any type of regression. It is also assumed that a_{t-i} and a_t are independent for any $i \neq 0$. Note that we can rewrite (9) as $r_t = \phi_0 + \phi_1(\phi_0 + \phi_1 r_{t-2} + a_{t-1}) + a_t$ provided that r_{t-2} exists. This model can be viewed as a simple linear regression model in which r_t is the response variable and r_{t-1} is the explanatory variable.

It is straightforward to generalise an AR(1) model to an AR model with an arbitrary number of lags, yielding the AR(p) model

$$r_t = \phi_0 + \phi_1 r_{t-1} + \ldots + \phi_p r_{t-p} + a_t, \tag{10}$$

see [8] (page 38, equation 2.9). This model can be interpreted in analogy with (9). It is important to bear in mind that $\mathrm{Var}(r_t) \neq \sigma_a^2$ unless $\phi_1, \ldots, \phi_p = 0$, which in the AR(1) case follows from the fact that there is variance in r_{t-1} as well if it is unknown. In the AR(p) case, the different lags are correlated and account for another source of variation. However, this is not of our concern since we will only make one step forecasts. Hence, all the lags will be given in our simulation study, so that

$$\mathrm{Var}(r_t | r_{t-1}, \ldots, r_{t-p}) = \sigma_a^2.$$

It could be helpful to know the mean of r_t as it is generally not equal to ϕ_0. Taking the expectation of (10) and using the fact that $E(a_t) = 0$ yields

$E(r_t) = \phi_0 + \phi_1 E(r_{t-1}) + \ldots + \phi_p E(r_{t-p})$. We know that $E(r_t) = E(r_{t-l})$ for any l, yielding

$$E(r_t) = \frac{\phi_0}{1 - \phi_1 - \ldots - \phi_p}.$$

The past results are based on the assumption that the series is *weakly stationary*, see [8] (page 30) for details. This essentially requires that the solutions to the polynomial equation

$$1 - \phi_1 x - \phi_2 x^2 - \ldots - \phi_p x^p = 0$$

for x should be greater than 1 in modulus according to [8] (page 46).

2.5.2 Threshold AR models

The threshold autoregressive (TAR) model is nonlinear and was originally developed to deal with common behaviours of certain asset returns, cf. [8] (pages 179-180). For example, the outcome of a series at time t might have different dependencies whether the last observed value was negative or positive. On some markets, the expected value of a time series at time t given the value at $t-1$ could be positive if the last value was either negative or positive, in contrast to the behaviour of the simple AR(1) model. As an illustration one can employ the following 2-regime TAR(1) model, presented in [8] (equation 4.8),

$$r_t = \begin{cases} 0.5 r_{t-1} + a_t & \text{if } r_{t-1} \geq 0, \\ -1.5 r_{t-1} + a_t & \text{if } r_{t-1} < 0. \end{cases}$$

This model is piecewise linear and has threshold value 0. The two regimes have different coefficients which makes the increasing and decreasing pattern asymmetric. The interpretation is that a negative value of r_t makes it likely for the next value of the series to be positive, whilst a positive value of r_t is again likely to produce a positive value of r_{t+1}, but with lower probability due to the less explosive coefficient 0.5. A discussion of properties of this model is outside the scope of this report and only models with one threshold value will be considered. We can formally define the 2-regime TAR model with threshold value c as

$$r_t = \begin{cases} \phi_0^{(1)} + \phi_1^{(1)} r_{t-1} + \ldots + \phi_p^{(1)} r_{t-p} + a_t & \text{if } r_{t-1} \geq c, \\ \phi_0^{(2)} + \phi_1^{(2)} r_{t-1} + \ldots + \phi_p^{(2)} r_{t-p} + a_t & \text{if } r_{t-1} < c. \end{cases}$$

A final note is that the mean of the series is generally not zero even if the constant terms are zero, see [8] (page 179). Also, in the case with only one lag, the model is stationary if and only if $\phi_1^{(1)}, \phi_1^{(2)} < 1$ and $\phi_1^{(1)} \cdot \phi_1^{(2)} < 1$. It is possible to consider many variants of this model. For instance, the regime condition that depends on r_{t-1} could instead depend on r_{t-d} for any $d > 0$, making the model even more complex.

2.5.3 GARCH-M models

This model is nonlinear in mean as well as variance and is described in [8] (page 142, equation 3.23 in particular). It is an extension of the GARCH model which in itself is a generalisation of the well-known ARCH model. It was proposed to deal with certain financial time series where the return seemed to correlate with its volatility, with the volatility evolving over time. The GARCH(1,1)-M model depends on the latest chock and the last measure of volatility and it is the only model of this kind that will be considered in this report. It is defined as

$$r_t = \mu + c\sigma_t^2 + a_t, \qquad a_t = \sigma_t \epsilon_t,$$
$$\sigma_t^2 = \alpha_0 + \alpha_1 a_{t-1}^2 + \beta_1 \sigma_{t-1}^2$$

where $\{\epsilon_t\}$ is a sequence of independent identically distributed variables with mean 0 and variance 1. The parameter c determines how much and whether the return of the series is positively or negatively correlated with its volatility. It is called the risk premium parameter.

Simulating from GARCH-series in general requires extra caution. For the unconditional variance to be finite, we set the constraint $\alpha_1 + \beta_1 < 1$, as suggested in [8] (page 132). Also, we set $\alpha_0 > 0$ and $\alpha_1, \beta_1 \geq 0$ to make sure that the variance is positive. With $c = 0$ and $\beta_1 = 0$ this model reduces to an ARCH(1) model. It could be useful to know that

$$E(a_t^2) = \frac{\alpha_0}{1 - \alpha_1 - \beta_1}$$

for this particular model.

3 Simulation study

Three main simulations are performed in this study using the time series models described previously. In the first experiment, we simulate from an AR model. Many lags will be considered so that variable importance and selection can be investigated in an appropriate way. Variable importance is a vital tool in interpretation of boosting and random forest algorithms. With that tool, we can say something about the models they provide and compare them in a way that is complementary to predictive power. This will also make it possible to examine the ability of the methods to handle more input variables. We will include some lags that have no effect on the response variable in the training process of the algorithms to test if these variables properly get neglected.

In the second experiment, a rather simple TAR model will be considered because the main focus is not to test variable importance. Instead, we are primarily interested in comparing forecast accuracy given this particular

nonlinear structure. It is also interesting to see how well a linear model, such as the AR model, predicts in this case. This comparison could therefore serve as justification to whether it is motivated to use the more complex algorithms in the first place.

Together with AR and TAR, the third model captures a great variety of established time series models. The GARCH-M model will test the abilities of the algorithms to capture a different kind of nonlinearity. As the variance of the white noise series for this model is not constant, the algorithms need to distinguish what patterns are related to the mean of the series.

3.1 Preliminaries

This study only concerns one-step ahead forecasts. Multistep ahead forecasts will use the previous forecasts to predict the next value and such forecasts will not be considered here. For certain stationary models, this means that the multistep ahead forecast will approach the mean of the time series as the forecast horizon increases. In this study, we will instead forecast time series where all the relevant inputs are given, which can be viewed as a typical regression problem.

In terms of software, R will be used with the specific packages `gbm` (gradient boosted models) and `randomForest`. If a single tree is grown for comparison, the package `tree` is used. The seed will be set to 1999, that is, we use the command `set.seed(1999)` in R to make sure that the simulations are reproducible.

To predict time series in the perspective of supervised learning, suppose that we have observations $y = r_t$ for some consecutive values of t. Then the first input variable x_1 is simply $x_1 = r_{t-1}$ and the second input variable is $x_2 = r_{t-2}$ and so on. An example is provided in the following table.

x_1	r_t
?	1
1	2
2	3
3	4
4	5
5	?

In the table above, we have five observations of r_t and five observations of r_{t-1}. Note that the rows containing question marks cannot be used, so there are only four input-output pairs. Therefore, in experiments with $x_1 = r_{t-1}$ as input we simulate $r_0, r_1, r_2, \ldots, r_T$ for some specified T and discard r_0, leaving a total of T observations for training and testing. If p lags are included in the model we have to simulate $T + p$ observations and discard the p first values of the series. In this case we will still denote our reworked

series by $r_1, r_2, \ldots, r_T$ for simplicity. Hence, r_{10} will always denote the tenth value irregardless of the number of input variables.

As an example, assume that we have observations from the simple AR(1) model (9) for $t = 0, 1, \ldots, 100$. We chose to divide the data such that $\{r_t\}_1^{80}$ is used for training and $\{r_t\}_{81}^{100}$ for testing. An algorithm is trained using the input-output pairs (r_{t-1}, r_t) for $t = 1, 2, \ldots, 80$. When the training process is over we end up with a model $\hat{f}(x)$. Now, we predict r_{81} using $\hat{f}(r_{80})$ and record the error. The prediction of r_{82} is then unaffected by $\hat{r}_{81}$, since we use the true value r_{81} in $\hat{f}(r_{81})$ to predict r_{82}. This relates to the fact that we only make one-step forecasts. The purpose of this example is to clarify the procedure as working with time series and lags can be confusing.

3.1.1 Error measurements

The predictive performance measure used in this report has extensively been the mean squared error, defined as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

where $\hat{y}_i$ is the prediction of observation i. See [9] (section 2.1). This is the main measure of prediction in this report and it will be used if nothing else is stated. In previous sections, the object was to attain the least squares, which is an equivalent problem as to minimise MSE. It is for example used to determine the best splits in regression trees. In the context of time series, MSE ought to be a good measure since it puts heavier weight on large errors. Time series are notoriously noisy, meaning that one expects small errors and the aim is generally to avoid making predictions far from the true value. MSE is also easy to compare because of its relation to variance. It is possible to obtain a lower bound of the MSE if the variance is known.

For robustness reasons it is still a good idea to consider other performance measures than MSE. Also, it might not be fair to only consider MSE because LS_Boosting is specifically based on minimising that criterion. The mean absolute error is defined as

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

and it puts heavier weight on errors smaller than 1 compared to MSE, according to [9]. This is a good alternative which is less affected by outliers. If MSE in some sense relates to the mean, MAE instead relates to the median. MAE will be considered as the second most important measure in this report.

Another measure of prediction error is mean absolute percentage error,

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|.$$

MAPE might be intuitive, since the error's size in relation to the size of the observation is taken into consideration. However, it can be misleading, especially for the simulated data sets used here. Observations close to zero will cause the measure to be unstable. One can also realise that negative prediction errors are given higher weights. These issues are discussed in [9] (section 2.2), where the definition of MAPE includes multiplying by a factor of 100 in contrast to the definition above.

If the individual predictions are not of interest, but only the total level of the prediction error, one may consider the mean bias,

$$\text{Bias} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i).$$

This performance measure can be used to see if a predictor systematically provides too low or too high estimates. It is actually an approximation of bias as it is defined in section 2.1. Squared bias could explain high or low MSE to some extent according to (1). If MSE is much larger than $\text{Var}(\epsilon)$ whilst squared bias is low, we know that the variance of the estimator must be high. Remember however that these measures are just estimates of the true MSE and bias. There exist many other measures of prediction. An example is mean absolute scaled error, which is generally applicable to time series. It will not be used in this study, but its definition and supportive arguments can be found in [9] (section 3).

3.1.2 LS_Boost vs random forest

This subsection summarises various ways in which LS_Boost and random forests differ, based on material presented in the theory section. It additionally presents some practical differences between the two methods, not discussed yet, as a preparation for the simulation study. A fundamental difference between random forests and boosting is that random forests reduces variance while boosting reduces bias. As mentioned earlier, this implies that random forests work best on large regression trees and boosting generally performs best on very small trees. Remember that small trees have low variance and high bias because they usually underfit data, while the opposite is true for large trees that are prone to overfitting.

Another major difference is that random forests are indeed random and will produce slightly different results from new attempts. Further, a random forest model is trained using a new bootstrapped dataset in each iteration.

On the contrary, LS_Boost adjusts the next dataset according to the residuals of the current model in each iteration and thereby puts more weight on places where the predictions in the previous step went wrong. This does again relate to the different underlying objectives that the algorithms are designed for.

Similar to bagging, boosting is a meta algorithm, which means that it can be applied to many statistical learning methods to create an ensemble. Random forests does not possess this feature and is hence more restricted. However, LS_Boost as it is defined here is also restricted to regression trees. This does nevertheless imply that gradient boosting in general is a more flexible algorithm. It is also easy to switch loss functions within the algorithm to deal with other types of response variables.

When it comes to overfitting, random forests are more robust. With limited computational power, a random forest model is impossible to overfit. This also has the advantage of not needing to regulate the size of the forest as long as it is not too small. LS_Boost needs a moderate number of iterations in order not to overfit nor underfit, although regulation can be made in many ways so that overfitting is seldom a cause of concern with boosting. There is no shrinkage in random forest implying that there is one less hyperparameter to tune. Note also that we tune the depth J of each tree for boosting instead of the terminal node size $n_{\min}$. The main difference is, as the size of the data set varies, that the same J produces equally many final regions.

One important question is which algorithm is faster to train. Some early attempts show that the LS_Boost algorithm is much faster to run through in each of the three experiments when the same number of trees are used. It might as well be an artifact of the implementation of the particular packages, but it seems reasonable to be the case in general. That much smaller trees are fitted for LS_Boost is the main explanation of its computational advantage. However, if we take into account the extended tuning process for boosting and the fact that less trees are probably required for random forests to reach its potential, this computational time difference may even out. It is therefore difficult to reach a conclusion here as to which method is computationally preferable, without performing a more detailed study.

3.2 AR simulation

We simulate from the following model,

$$\begin{aligned} r_t = 0.9r_{t-1} - 0.8r_{t-2} + 0.7r_{t-3} - 0.6r_{t-4} + 0.5r_{t-5} - 0.4r_{t-6} + 0.3r_{t-7} \\ - 0.2r_{t-8} + 0.1r_{t-9} - 0.1r_{t-10} + 0.1r_{t-12} - 0.1r_{t-15} + a_t \end{aligned}$$

where $a_t \sim N(0,1)$. This series is stationary since all the solutions, both real-valued and complex, of the equation

$$1 - 0.9x + 0.8x^2 - 0.7x^3 + 0.6x^4 - 0.5x^5 + 0.4x^6 - 0.3x^7 + 0.2x^8 \\ - 0.1x^9 + 0.1x^{10} - 0.1x^{12} + 0.1x^{15} = 0$$

are greater than one in modulus. The equation only has one real root and 14 complex ones. We thus know that the series is weakly stationary and has mean 0. From this model, 2000 observations are simulated with an initial value $r_0 \sim N(0,1)$, of which 1600 are devoted to training the algorithms and 400 to testing them out. Specifically, $\{r_t\}_{t=1}^{1600}$ is the training set and $\{r_t\}_{t=1601}^{2000}$ is the testing set. The lags r_{t-11}, r_{t-13} and r_{t-14} are also provided to the algorithms as input variables in this experiment even though they have no influence.

Main comparison

To begin with default parameters are used with one exception. We manually set $\eta = 1$, because stochastic gradient boosting is actually the default choice for the implementation in `gbm` with $\eta = 0.5$. The default parameters in the random forest function is $m = 5$ $(p/3 = 15/3)$ for the number of variables to consider in each split and again $n_{\min} = 5$ for the maximum node size in the terminal regions. For the `gbm` function the shrinkage parameter is by default $\upsilon = 0.1$ and the number of splits for the trees is one $(J = 2)$, that is, stumps are the default types of trees used. Letting the number of trees vary, we obtain the result of Figure 2.

In Figure 2, the black line at the bottom corresponds to the magnitude of the irreducible error, that is, at least in theory the lowest attainable test error. Test error here denotes the MSE calculated on test data. It is of course possible to reach lower test error occasionally since it is an estimate of the true MSE and we have a finite amount of data. The black line has intercept 1 because the noise has variance 1. One should not think of adjacent points in the plot as adding 5 trees to the existing ensemble. Completely new ensembles are instead created at each point of the plot. This does not matter for LS_Boost though unless stochastic gradient boosting is used.

We can see that LS_Boost outperforms random forests in Figure 2, although for small ensembles, random forest achieves lower error. We can also see that the random forest stabilises early, while the MSE for the boosted model continues to decrease. Note further the behaviour of the different curves. Random forests have scattered test results for small ensembles and the error oscillates up and down even for the larger ensembles. Meanwhile, LS_Boost essentially improves in each step as the algorithm does not involve randomness and apparently does not begin to overfit either. It seems like boosting potentially can do even better with more iterations.

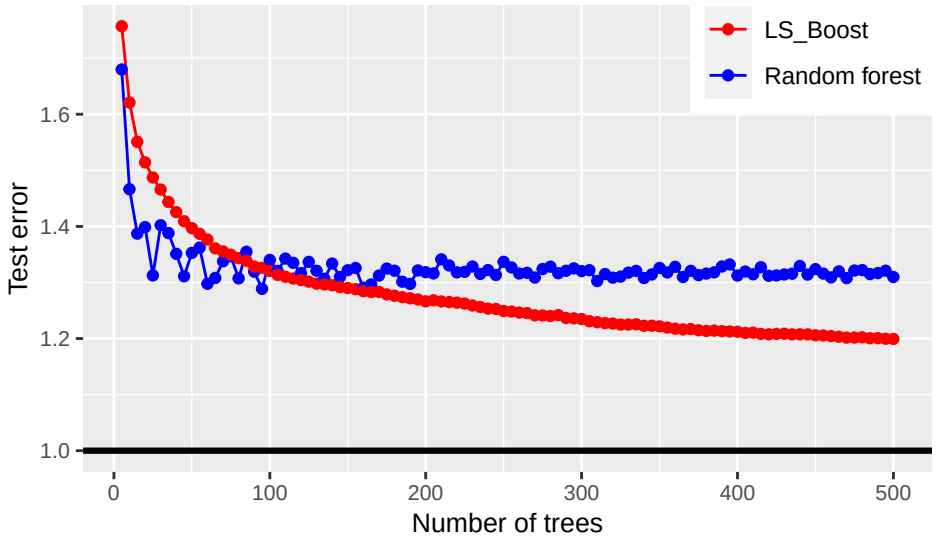

Figure 2: Random forest vs LS_Boost with default parameters, varying the number of trees in the ensemble. The grid for the number of trees starts at 5 and grows by 5 for each point in the plot.

Now, we tune the parameters for both algorithms by further dividing the training data into 1200 observations for pure training and 400 for validation. This means that we train models on $\{r_t\}_{t=1}^{1200}$ using different parameterizations and record the MSE of each model on $\{r_t\}_{t=1201}^{1600}$. We then choose the model achieving the lowest MSE. By doing this, we avoid overfitting the test data. The procedure for trying different parameterizations is to vary one parameter at a time and keep the others fixed.

After the tuning process we ended up with the following random forest model: $m = 9$, $n_{\min} = 15$ and we use $B = 4000$ trees for safety matters. Note already now that this model is simpler than the default one, which makes sense because of the relatively simple structure of the data. For the boosted model the default value $J = 2$ was the best, again due to simple structure of data. Regarding the shrinkage parameter and the number of iterations, we got $\upsilon = 0.05$ and $B = 3000$ respectively. These two parameters should not be varied individually in the tuning process and were therefore regulated together; raising B generally implies reducing υ. Stochastic gradient boosting did not achieve lower validation error so η was still kept at a value of 1. The following results were obtained on the test data.

	MSE	MAE	MAPE	Bias
Random forest	1.27	0.90	1.50	-0.018
LS_Boost	1.17	0.86	1.69	-0.006
Single tree	1.38	0.93	1.93	-0.014

In the table above, we see that boosting wins according to all measures except MAPE. It is however a bit weird that MAE is lower whereas MAPE is higher for LS_Boost, which showcases the deficiency of MAPE. The differences are however quite small between the two methods, especially according to the MAE criterion. According to the bias measure, random forests on average predicts 0.018 lower than the actual value in this case. A single regression tree was also grown as comparison. The validation set was used to find a suitable value for the penalising parameter $\alpha = 25$. It does not perform much worse than random forests, in line with the theory of bagging.

The training error was 1.20 and 0.72 for random forest and LS_Boost respectively for the tuned models. The much lower training error for boosting might be concerning, but is not a matter of overfitting according to the validation error. It is probably a consequence of the algorithm itself and we did indeed use many trees in relation to the shrinkage parameter. It can be added that the validation error was higher than the test error for both models, so the true MSE of the algorithms might be slightly underestimated.

In Figure 3, we see the variable importance measures for the tuned models. Although testing errors showed promising results, here the downside of the algorithms is clearly exposed. The lack of interpretation has been discussed earlier, but both algorithms are unable to estimate the importance of variables correctly. It might be because there is high correlation between the input variables. Lags adjacent to each other have about 0.5 correlation. Anyhow, the stacks would be around 90 for $x2 = r_{t-2}$ and slightly lower than 80 for $x3 = r_{t-3}$ and so on if the variable importance measures had been correctly estimated. The importance of irrelevant lags are also incorrectly estimated compared to the less important but significant lags. To nominate a winner, both LS_Boost and Random forest node do relatively well compared to random forest OOB. I would argue that LS_Boost wins again because without knowledge of the true importance, it is difficult to know what measure to use for random forests. I also think that LS_Boost would achieve better results with fewer trees in this regard.

Alternative methods

Since the data is generated from a simple linear model, it is of course better to use linear estimation methods. A Linear regression model created with the `lm` command in R obtains 1.00 in test error. It does a great job with parameter estimation as well by correctly neglecting the three irrelevant variables as insignificant. All the other parameters are significant and overall close to the true value.

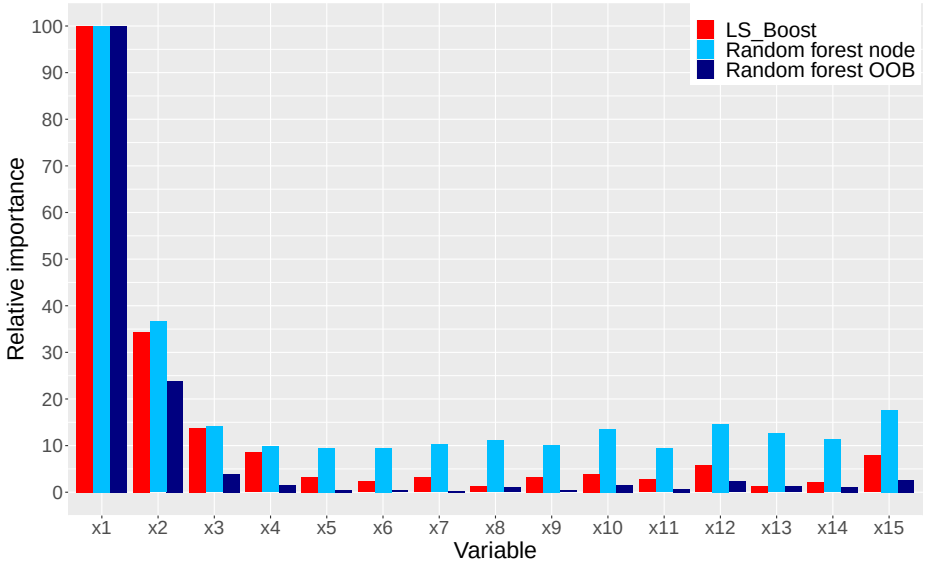

Figure 3: Variable importance. LS_Boost and Random forest node use the same method for estimation, described in the LS_Boost section, whilst Random forest OOB uses the decrease in accuracy, described in the random forests section.

Employing an AR model to the data using the **ar** function from the stats package in R is the best option. This function uses AIC to determine the suitable amount of lags and then estimates the coefficients of the chosen model by means of ordinary least squares. A model of order 15 is correctly chosen and coefficients are estimated close to their true values, yielding a test error of 0.99. This method has thus the advantage of not needing to specify the amount of lags in advance. If we do not employ any model at all, the best guess is to predict new values with the mean of the series, which gives a test error of 2.02.

3.3 TAR simulation

TAR models can be made very complicated with many lags and regimes. However, it is difficult to evaluate the properties of such models. It is possible that weak stationarity is not fulfilled, meaning that the model is not even a TAR model in the strict sense. We therefore simulate from a similar model as the example in the theory section but with more explosive coefficients, namely

$$r_t = \begin{cases} 0.7r_{t-1} + a_t & \text{if } r_{t-1} \geq 0, \\ -3r_{t-1} + a_t & \text{if } r_{t-1} < 0, \end{cases} \tag{11}$$

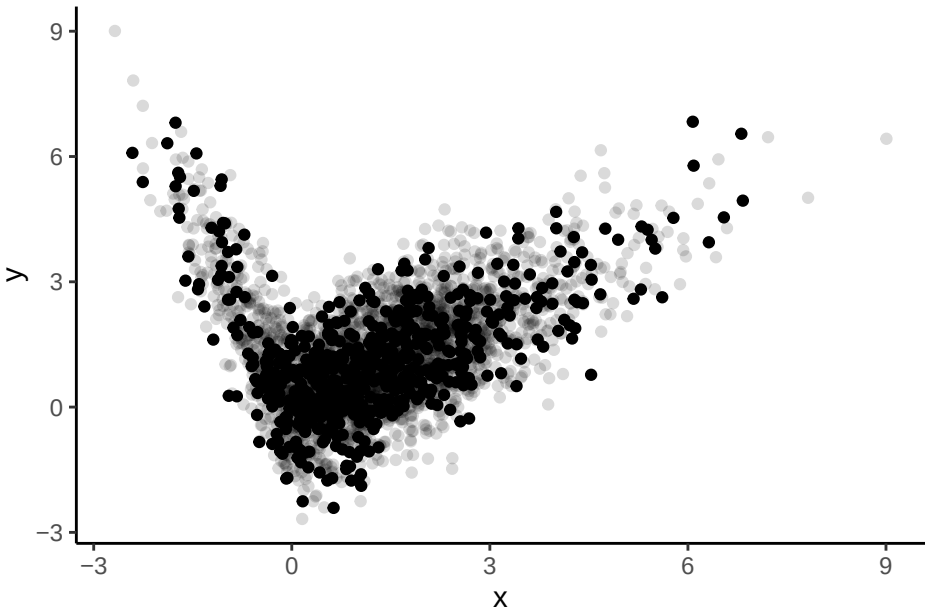

Figure 4: Plot of the simulated data from the TAR model of equation (11). Training data has been made transparent.

where $a_t \sim N(0, 1)$. This model is weakly stationary according to the discussion about TAR models. A large trial simulation of 10 million data points shows that the mean of the series is 1.33 after rounding. We can use this to set the first value of the series equal to $r_0 = 1.33 + a_0$, after which 3000 observations are simulated accordingly. Again, the proportion of 80-20 is used for training and testing, so that $\{r_t\}_{t=1}^{2400}$ observations are devoted to training and the rest for testing. The single input variable is r_{t-1}.

In Figure 4 we see the structure of the data. Note the choice of axes, we do not plot $y = r_t$ against t but rather $x = r_{t-1}$ to be able to observe the influence of the covariate. The plot clearly shows the nonlinearity of data, where $x = 0$ serves as a threshold value.

Main comparison

First off, we make a similar comparison as for the AR simulation, letting the number of trees vary. The default parameters are chosen as before. Looking at Figure 5, we see that boosting outperforms again, but now in an even larger fashion, although random forests once more is better for very small ensembles. Random forests seem to stabilise already at 30 trees and the test error for LS_Boost starts to decrease very slowly at 100 trees. It is remarkable that the MSE of LS_Boost even goes below the expected irreducible error

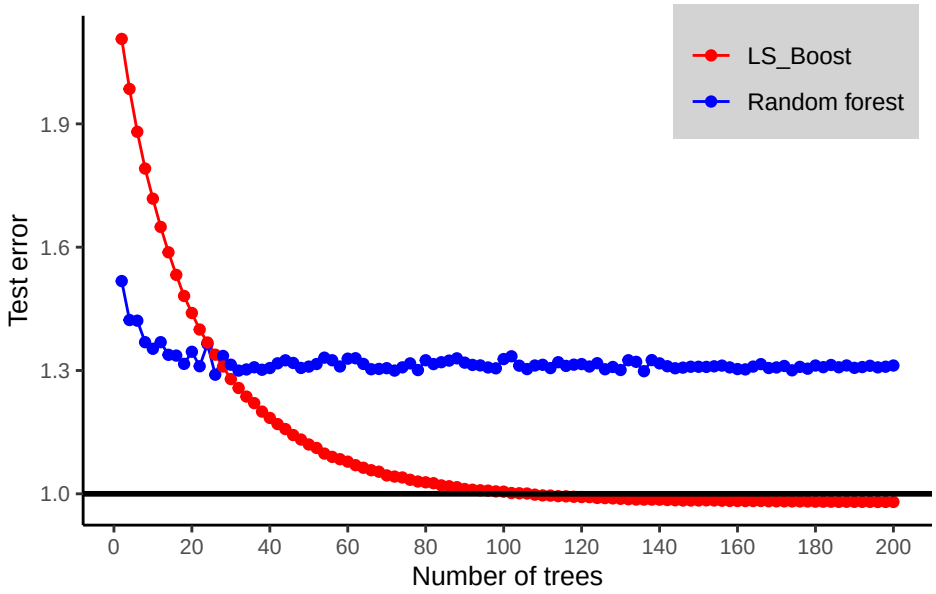

Figure 5: Test error (MSE) comparison for models of varying size, based on simulated data from the TAR model. The evaluation starts at 2 trees and increases by the same number of trees in each step.

at 1, indicating that the model predicts almost perfectly. Taking the mean squares of the error terms for the test observations however gives the value 0.96. This corresponds to taking

$$\frac{1}{600}\sum_{t=2401}^{3000} a_t^2 = 0.96.$$

So if our models had perfect forecast accuracy they would obtain 0.96 in mean squared error for this experiment. Boosting does nevertheless a great job here. The promising results with a small amount of trees may possibly be explained by the fact that there is a single input variable.

Further, we tune the algorithms in the same fashion as before, despite that boosting probably cannot improve significantly. Note that there is one less parameter m to regulate for random forest since there is only one input variable for this dataset. A high value of $n_{\min} = 60$ for the maximal node size was found to be optimal for the random forest model. This does not necessarily mean that few splits are made, the data set happens to be large and some regions may include many observations that do not need to be splitted further, for example observations close to zero. Also, $B = 2000$ trees were grown.

It was indeed difficult to improve the boosted model, changing the parameters does not change the predictive performance very much. The decision to try stochastic gradient boosting was made so that a fraction of $\eta = 0.7$ of the data is drawn in each iteration without replacement. The shrinkage parameter was set to 0.005 and 5000 trees were grown. Stumps, with one single split, is again the best option regarding tree size. We also employ an AR model to the data using the same technique as described earlier. The following results are obtained.

	MSE	MAE	MAPE	Bias
Random forest	1.04	0.82	2.06	0.11
LS_Boost	0.98	0.79	1.93	0.10
AR model	2.06	1.06	3.03	0.07

Above, we can see that LS_Boost wins in all cases. The differences are nevertheless small and it is fair to say that random forests can compete with LS_Boost in this case. The linear alternative, AR, did not perform well, besides the fact that the total error is low according to the bias measure. This method barely turns out to be better than guessing, as expected. It can be mentioned that the validation error was higher for all methods compared to test MSE. Still, the results are remarkable and will be studied more closely. We can also conclude that random forests did improve a lot with some tuning.

In Figure 6 we illustrate how well the algorithms perform in relation to the expected value $E(r_t|r_{t-1})$. In some sense, the expected value is the best prediction. This is the only estimate $\hat{r}_t$ that satisfy

$$E(\hat{r}_t - r_t) = 0.$$

Bear in mind that $E(r_t|r_{t-1})$ can only be calculated if we know the underlying model, which we do in this case. This exhibits one of the benefits of working with simulated data. Note that the expected value for this time series model is never negative. In spite of this, LS_Boost does extremely well, since the two lines of Figure 6 a) are almost identical in most places. Random forests also predicts well, as can be seen by looking at b), but in this case it is easier to separate the two lines from each other. One way to interpret the plots is that more green is worse as it preferably should be covered by the black line. The most important thing to observe is whether the predicted and expected values go in different directions. Overall, this figure explains the low test errors we obtained earlier.

Further investigation

To make things slightly more complicated, we tried another simulation with one additional independent variable. Keeping r_t defined as in (11) we now

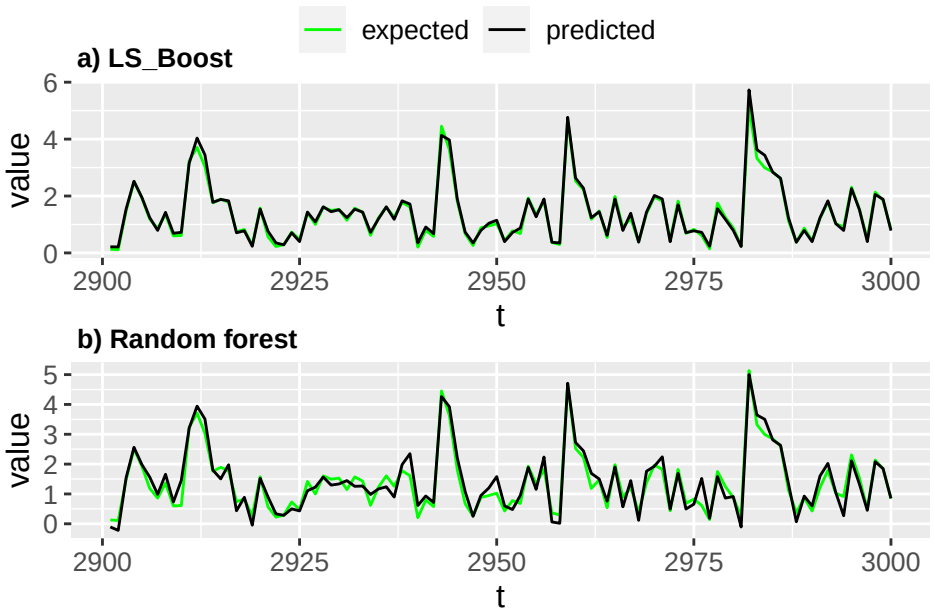

Figure 6: Expected values and predicted values for data from the TAR model, based on model fits based on the a) LS_Boost method, b) random forests method. The green curve is thus the same in both plots. Only the last 100 predictions are included.

simulate from

$$r_t + 3 \cdot \sin\left(\frac{2\pi \cdot t}{365}\right).$$

This means that time (of the year) affects the return of the series. A third input variable r_{t-2} will also be provided in an attempt to confuse the algorithms. The three input variables are thus r_{t-1}, r_{t-2} and t. The setup is otherwise the same as before, except that every fifth observation will serve as test set instead of the last 600. Hence, $\{r_5, r_{10}, r_{15}, \ldots, r_{3000}\}$ is the test set. We would otherwise forecast values where t is greater than all t in the training set, which both algorithms have difficulties to handle (see discussion).

Instead of tuning the methods' parameters we use the same settings as before with some exceptions. The extra input variables motivate more terminal regions for LS_Boost, so we use $J = 4$ instead of 2. For random forests the default $m = 1$ is used along with $n_{\min} = 20$ instead of 60 due to increased complexity in the data set. The following results are obtained.

	MSE	MAE
Random forest	1.11	0.85
LS_Boost	1.01	0.81

In the table above, we can see that every error measure is higher in contrast to the previous investigation on the original data set. The errors are still very low though. We also notice that LS_Boost handles the extra complexity marginally better than random forests. The results are in conclusion reasonable since the variance of the white noise has not changed.

A quick look at variable importance measures shows that LS_Boost does a better job at neglecting the irrelevant variable, while random forests seems to estimate the importance of t more accurately. The second statement has not been properly examined, but LS_Boost estimates that t has around 16% influence in relation to r_{t-1}. This estimate may seem low considering that the mean of the series is increased by roughly 35% compared to the previous data set. This might be due to the fact that t is used as input variable instead of $\sin(t)$, but this is not the main point of the study and will not be further investigated. The corresponding relative influence of t for random forests is 22% or 32% depending on estimation method.

Now, we repeat the past experiment with two different distributions for the white noise series $\{a_t\}$. In the first case we make the modification $a_t \sim N(0, 2)$ to scatter the data while keeping the residuals normally distributed. For the second case we set $a_t \sim t(3)$, that is, the white noise is t-distributed with 3 degrees of freedom. The t-distribution is heavier tailed and should yield more extreme values compared to the normal distribution. Results are provided in Table 1.

Distribution	$a_t \sim N(0,2)$		$a_t \sim t(3)$	
Error measure	MSE	MAE	MSE	MAE
Random forests	2.07	1.16	3.94	1.30
LS_Boost	1.89	1.11	3.91	1.28

Table 1: Test results for simulations from the TAR-model with time as an added covariate, and different types of noise distributions.

Interestingly enough, LS_Boost still wins, for both measures of prediction, as can be seen in Table 1. This means that despite increased noise LS_Boost appears to be preferable in forecasting this particular time series model one step ahead. However, random forests is even more competitive for the t-distributed error terms, which has the highest variance of the models considered so far. Compared with previous results, we conclude that random forests responds better to increased variance relative to the case when $a_t \sim N(0, 1)$. This is realised by comparing the relative margins at which LS_Boost wins for the different experiments.

3.4 GARCH-M simulation

Since GARCH(1,1) models are most common among GARCH models used in practice so we simulate from the following GARCH(1,1)-M series,

$$r_t = 2 * \sigma_t^2 + a_t, \qquad a_t = \sigma_t \epsilon_t,$$
$$\sigma_t^2 = 0.01 + 0.1 a_{t-1}^2 + 0.89 \sigma_{t-1}^2,$$

where $\epsilon_t \sim N(0,1)$ for all t. With prior knowledge of the parameters it is possible to calculate

$$E(a_t^2) = \frac{0.01}{1 - 0.1 - 0.89} = 1$$

unconditional on t. Unconditional on t means that we have no information about past chocks or variances. Also, since $\sigma_t^2 > 0$, it holds that the expected value of r_t is positive for all t because

$$E(r_t) = 2E(\sigma_t^2) + E(a_t) = 2E(\sigma_t^2) + E(\sigma_t \epsilon_t) = 2E(\sigma_t^2) > 0.$$

We used $E(\sigma_t \epsilon_t) = E(E(\sigma_t \epsilon_t | r_{t-1})) = E(\sigma_t E(\epsilon_t)) = 0$ above. Observe that the conditional variance σ_t^2 is much more dependent on the previous volatility σ_{t-1}^2 than the previous shock a_{t-1}^2 based on the coefficients of the model. Conditional variance means that we know the previous chocks and variances at time t. This is intentional because it prevents the series from reaching undesirable states where too large or small values occur. The comparison then becomes unfair since one observation can determine the outcome. Anyhow, 1000 observations are simulated, of which 20% corresponding to $\{r_t\}_{t=801}^{1000}$ are devoted to testing as per usual. At $t = 0$ we set $\sigma_0^2 = 1$ and hence $a_0 \sim N(0,1)$ to generate r_0, after which the rest of the observations are simulated accordingly.

In Figure 7 we can see a plot of the data as it occurs in time. The local peaks are apparent because of the return's dependence on the conditional variance, indicating temporarily high variance. Around time $t = 800$ there are some really high returns in the figure. It is usually hard for other time series models to exhibit similar patterns when plotted against t, but in this case the moving average has a clear maximum within the time period of high returns.

Main comparison

Now, using default parameter values, we make comparison of the algorithms' predictive performance in a similar fashion as before. The default number of trees is $B = 100$ for LS_Boost and $B = 500$ for random forests. Even though r_t does not explicitly depend on r_{t-1}, we still use it as the input variable. We also fit a linear regression model to the data, as for the AR simulation, and obtain the following results.

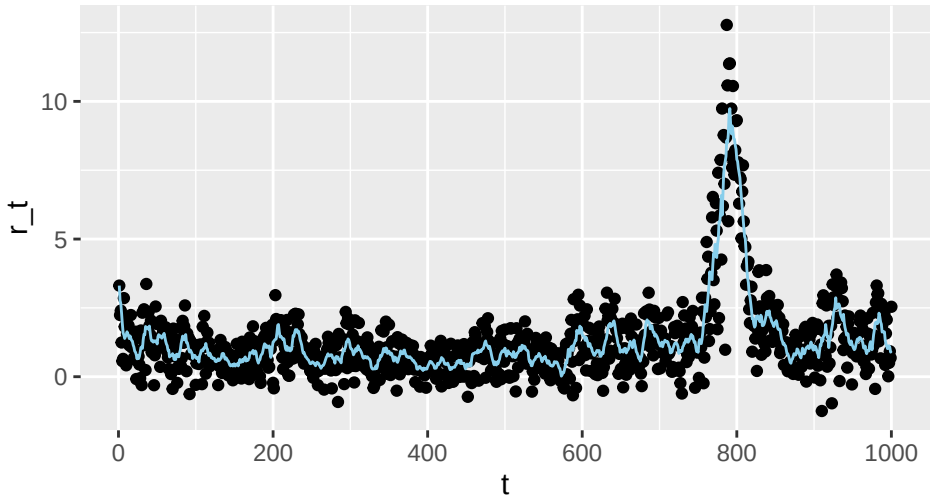

Figure 7: Plot of the simulated data from the GARCH-M model. The blue line represents a moving average of 9 values.

	MSE	MAE	MAPE	Bias
Random forest	1.63	0.97	1.55	-0.16
LS_Boost	1.03	0.82	1.33	-0.21
Linear regression	1.11	0.81	1.51	-0.13

Once again, boosting wins according to the most relevant measures MSE and MAE, compared to random forests, as we can see in the table above. If we take the average of the variance and the squared white noise series we get

$$\frac{1}{200}\sum_{t=801}^{1000}\sigma_t^2 = 0.85 \quad \text{and also} \quad \frac{1}{200}\sum_{t=801}^{1000}a_t^2 = 0.69$$

respectively. The summations were taken over the test observations only, so that the values can be compared with the test results. Note that the mean of a_t^2 for these particular simulated data points is much lower than its unconditional expected value. In this regard, random forests does a terrible job, which needs more investigation. The linear regression fit indicates that the results of LS_Boost are not that promising either. LS_Boost does indeed provide worse forecasts compared to previous simulations if the smaller squared error terms are taken into account. For the other simulations, it holds that $\text{mean}(a_t^2) \approx 1$ over sufficiently many t. In other words, it should be easier

to make predictions for this particular simulation, because the white noise series happened to have lower variance.

Changing the seed is one way to validate the results. Different seeds give similar test results, although LS_Boost performs better relatively speaking than linear regression when the variance of the white noise series is high. After experimenting with seeds, we can conclude that the size of the fluctuations in data vary a lot for each simulation. Ideally, we would like a much larger simulation, but it is not common in practice to possess such data sets. Still, since the results are similar on different data sets, there should be ways to improve the results on the current data set.

One option in attempting to achieve lower test errors is to tune the models. It is always possible to reach a lower test error by making small adjustments without a validation set, so we only try a few parametrizations to avoid overfitting the test data. For the LS_Boost algorithm, we try $B = 3000$, $J = 5$, shrinkage $\upsilon = 0.001$ and stochastic data selection with $\eta = 0.5$. This gives a slight improvement with MSE = 1.00 and MAE = 0.80. For random forests, we use $B = 4000$ and $n_{min} = 200$ to obtain MSE = 1.13 and MAE = 0.84.

Further investigation

A final attempt to achieve better results is to change input variables. The output variable r_t depends on its previous value through the shock a_{t-1}. Meanwhile σ^2_{t-1} is also important because it is highly correlated with the return. A large value of r_t thus means that σ^2_{t-1} is likely to be large and so is r_{t-1} accordingly. In the light of this, we define one input variable $(r_{t-1} - r_{t-2})^2$ to account for a_{t-1} and a second input variable r^2_{t-1} to account for σ^2_{t-1}. Keeping the same parameter tuning for the algorithms as before, with the addition $m = 1$ for random forests, yields the following results.

	MSE	MAE	MAPE	Bias
Random forest	1.00	0.79	1.40	-0.18
LS_Boost	0.90	0.76	1.36	-0.19
Linear regression	1.22	0.84	1.34	-0.47

In the table above, we can see some improvement. The overall test errors are now lower for LS_Boost and random forest, while random forest seems to capitalise slightly more from the new inputs. Still, boosting wins in a small but clear fashion. Linear regression is now distinctly worse than the other algorithms. In conclusion, it is a difficult task to forecast this time series model, as further displayed in Figure 8.

The conditional variance of the time series as a function of time along with the squared prediction errors of LS_Boost is shown in Figure 8 for the test set. For instance, at $t = 900$ the red curve shows $(r_{900} - \hat{f}_B((r_{899} - r_{898})^2, r^2_{899})^2$ if $\hat{f}_B$ denotes the boosting estimator. In the plot we can see

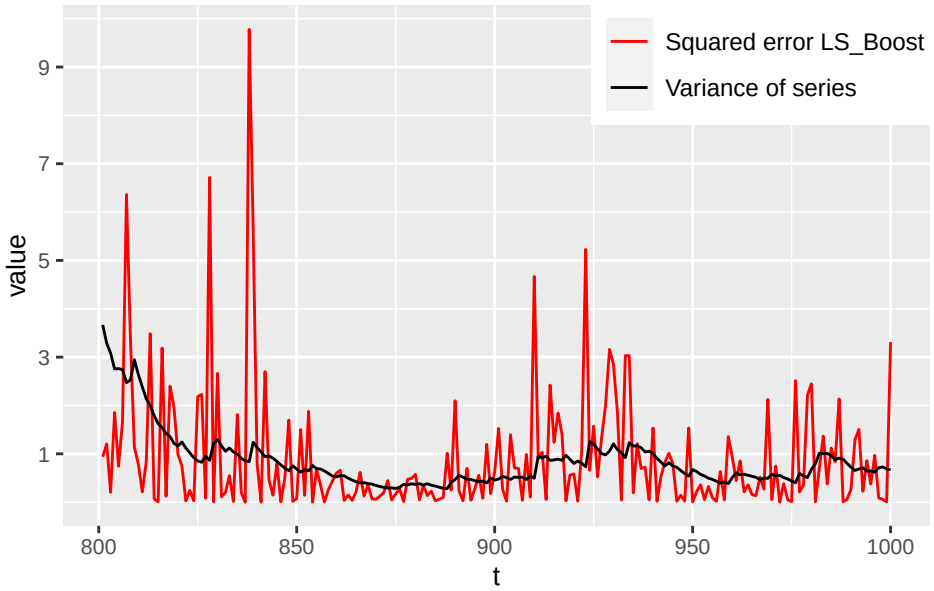

Figure 8: The variance σ_t^2 and squared forecast error for LS_Boost plotted together for the GARCH-M time series.

some individual extremely large and small errors, where the errors are naturally larger when the variance is high. The corresponding plot for random forests looks similar and hence it is not included in the report. The short leaps in the black curve for the variance σ_t^2 exhibit large values of a_{t-1}^2. These leaps seem to cause especially large forecast errors. It indicates that there are still flaws in our choice of input variables, but also that neither of the algorithms are able to provide solid forecasts for this simulated GARCH-M dataset.

The past experiment was repeated with 100,000 observations to check its robustness. The outcome was quite different in absolute terms, but identical if the algorithms are ranked in terms of their MAE. Boosting achieved much lower MSE than the other two methods. However, it could be considered as an unstable measure here because the time series contains values as large as 100 for this simulation. As a by-product, bias was approximately 0 for all methods, suggesting that none of them have prediction errors with a systematic component.

3.5 Summary of results

In the first experiment, we simulated from an AR model. With default parameters LS_Boost had lower MSE than random forests when the number of iterations B was larger than 100. Random forests had lower MSE though for smaller ensembles, with one reason being its use of larger trees compared to LS_Boost. Increasing B from this point on did not improve random forests whilst the test error for LS_Boost continued to decrease. At $B = 500$ the values of MSE was about 1.2 and 1.3 for LS_Boost and random forests respectively. Tuning the parameters of the algorithms did not change much as boosting still won with 1.17 in MSE and 0.86 in MAE compared to 1.27 and 0.90 respectively for random forests. A single regression tree was also grown for comparison and was not much worse than random forests. Since an AR model is linear in its parameters, we found that linear methods such as linear regression were superior, achieving an MSE around 1.00.

In the second experiment, we simulated from a TAR model and afterwards a modification of it. The results before tuning were similar in relative terms as in the AR simulation. Random forests did however improve a lot with some tuning, even though LS_Boost still had slightly lower MSE and MAE. The MSE was close to 1 for both algorithms, which means that variance as well as squared bias were close to zero according to (1). This was confirmed since we estimated the squared bias to be around 0.01. We also fitted an AR model to this data for comparison. It did not perform well as expected given the nonlinear structure of data, obtaining a MSE of 2.06. Adding time as an explanatory variable did not confuse the algorithms very much, although LS_Boost handled these modifications better than the other methods. After increasing the variance of the white noise series to 2, LS_Boost still won with approximately the same relative margin. However, after changing to a heavier tailed distribution for the white noise series we found that random forests came even closer at reaching the prediction error levels of LS_Boost.

In the third experiment, we simulated from a GARCH-M model and afterwards an extended version of it to validate the results. Different seeds were also tried since the variance series σ_t^2 varied quite a lot for each simulation. Boosting did once again outperform random forests according to both criteria MSE and MAE. The difference in MSE was 0.60 with default parameters, but dropped to 0.13 with a moderate amount of tuning. We also evaluated the performance of linear regression for this data as comparison. In fact, the linear estimator achieved lower MAE than LS_Boost and lower MSE than random forests. After changing input variables, both boosting and random forests had lower MSE and MAE, while linear regression changed in the opposite direction. Still, LS_Boost performed the best.

4 Discussion

Both algorithms random forests and LS-Boost provided accurate forecasts in this study. It is easy to see why they are popular to use in practice when prediction error is the most important concern. However, this does not mean that they are perfect. One issue with the algorithms is that they do poorly on new observations that are larger or smaller than the values used to train them. This can be realised by studying the structure of a regression tree. Unlike linear regression for example, a regression tree does not adapt to new values. A much higher value will still be predicted by the mean of some smaller values. However, it is generally not advisable anyway to predict outside the scope of what the training data covers.

Statistical significance of our results has not been investigated in this report, although we checked for robustness of the results from the GARCH-M model by simulating much more data. Another way to validate the results could be to repeat the experiment for say 10,000 different seeds and count the number of wins for each algorithm according to MSE. If these counts weigh heavily in the favour of either algorithm, say at least 95% wins, we can be more certain of the results. This procedure does not prove statistical significance in the strict sense, but is definitely an alternative to more formal tests. One could assume that $P(\text{LS\_Boost wins}) = P(\text{random forests wins}) = 0.5$ is the null hypothesis and use the counts to test an alternative hypothesis that the two methods have different probabilities of winning a comparison. This is possible to test using a binomial distribution for the number of times (say) LS_Boost wins. Yet another option is to estimate the prediction variances of the models on unseen data and test whether they are equal or not.

Why does LS_Boost win?

The better algorithm to make one-step ahead forecasts in this report is LS_Boost. There might be many answers to why this is the case. For all simulations, the time series were strongly dependent on their explanatory variables. We did for example exclude intercept terms in all simulated models because they obscure the influence of lags. One could claim that there are more to gain from reducing bias than variance in these simulations, hence the better performance of LS_Boost. This is because it is advantageous to fit the data closely with the use of input variables. One way to test this is by reducing the influence of lags and including an intercept term.

Another reason why boosting performed best in our study might be explained by the relatively large data sets. Smaller data sets are more likely to vary as the experiment is repeated, making LS_Boost easy to overfit. Tuning the models also becomes more difficult as there are too few observations for a validation and testing set. Further, similar studies indicate that random forests perform better when the data is sparse, see related work in the sub-

section below. Assuming that this is true, one could apply the algorithms on real life situations in the following way. Random forests might be better for forecasting returns on stocks of small companies, since data of such stock prices probably are more noisy. Meanwhile boosting should be better when forecasting stock prices of large, less volatile companies. This is of course not shown in the report, but could be an interesting topic of further investigation. Recall however what we saw in Table 1 that random forests deteriorated relatively less than LS_Boost for a more heavy tailed error term distribution.

Improvement of random forests

One way to improve an estimator could be to use random forests in conjunction with boosting. An idea is to apply random forests and then use its output $\hat{f}_r(x)$ as the first tree of the LS_Boost algorithm instead of using $f_0(x) = \bar{y}$. Such an estimator would in theory achieve lower bias than random forests alone. However, it would probably yield higher variance due to tighter adaptation to existing observations. It is also a bit tricky though to determine if the proposed method would improve LS_Boost. In most cases, it holds that $\text{Bias}(\hat{f}_r) < \text{Bias}(f_0)$, but we would expect that $\text{Var}(\hat{f}_r) > \text{Var}(f_0)$. The second inequality is somewhat uncertain as both sides could possibly have very low variance. Nevertheless, it may be too optimistic to believe that the estimator would improve significantly after running through the iterations of the LS_Boost algorithm.

Another idea, which is inspired by boosting, is that random forests might improve if the probabilities are adjusted in each iteration for every observation to be chosen. Preferably in a way that gives observations where the prediction went most wrong a much higher probability to be selected in the subsequent tree. This means that the first tree in the random forest ensemble would still be grown on a bootstrap sample where the probabilities are $1/N$ for all observations, but the next bootstrap sample would be drawn from another probability distribution. One option for adjusting the probabilities could be

$$P(\text{selecting } z_i) = \frac{(T_j(x_i) - y_i)^2}{\sum_{n=1}^{N}(T_j(x_n) - y_n)^2},$$

where T_j is the current output of the algorithm (the average of j trees) and $z_i = (x_i, y_i)$. This can be interpreted as follows. After j iterations we adjust the probability of observation $z_i, i = 1, \ldots, N$ to be its residual in the current output relative to the total sum of residuals. This would make it more likely for inaccurately predicted observations to have more influence in the next tree. If the sample size remains N and selection is made with replacement, then this new algorithm would definitely need some type of shrinkage. It would probably work best with shrinkage anyway, since slow learning tends to yield better results.

Limitations of this study

This is a small study including few simulations. Much space is devoted to theory and we have not proved anything. We have only shown some examples of time series models where LS_Boost seems to be slightly superior to random forests in terms of one-step ahead forecast accuracy. If the study was enlarged many more simulations would be included from other types of time series models. Another extension could be to include multivariate time series, which requires a definition of multiple output regression trees. There are also too few theoretical aspects presented, especially concerning gradient boosting. Hence, based on this study alone we cannot make firm recommendations of situations in which the algorithms are suitable, or if they should be used on time series at all.

4.1 Further reading and related work

For more information and details on regression trees, see [3]. A more nuanced description is provided there, along with statistical properties. Regarding random forests, see for example [10] to get a deeper insight. This is a chapter in a book about ensemble learning methods and should be excellent for interested readers. To learn more about boosting, one should read [11], the article where this method was formally introduced. For more theory and applications of different time series models, see [8].

A more comprehensive study is performed in [12], although the focus is on classification problems rather than regression. This study compares bagging, Adaboost and randomization. Random forests was yet to be introduced when the article was published, although the algorithm is almost identical to randomization. He finds that boosting is superior in a situation with little or no classification noise. Bagging had however better accuracy for noisy data sets. Another similar but larger study can be found in [13], where multiple machine learning algorithms are compared in terms of forecasting time series data. For a more recent study, see for example [14]. Random forests and boosted regression trees are compared here along with other machine learning algorithms to predict stock market prizes. One can also scan through [2] for several examples of boosting versus random forests.

References

[1] James, G., Witten, D., Hastie, T. & Tibshirani, R. (2017). *An Introduction to Statistical Learning*. Eighth edition. New York: Springer. E-book.

[2] Hastie, T., Tibshirani, R. & Friedman, J. (2008). *The Elements of Statistical Learning*. Second edition. New York: Springer.

[3] Breiman, L., Friedman, J., Olshen, R. & Stone, C. (1984). *Classification And Regression Trees*. First edition. Wadsworth and Brooks.

[4] Breiman, L. (1996a). Bagging predictors. *Machine Learning* **24**, 123-140.

[5] Breiman, L. (2001). Random Forests. *Machine Learning* **45**, 5-32.

[6] Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics and Data Analysis* **38**(4), 367–378.

[7] Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics* **29**(5), 1189–1232.

[8] Tsay, R. S. (2010). *Analysis of financial time series*. Third edition. Hoboken, New Jersey: John Wiley & Sons, Inc.

[9] Hyndman, R. J. & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting* **22**(4), 679-688.

[10] Cutler, A., Cutler, D. R. & Stevens, J. R. (2012). Random Forests. In: Zhang C., Ma Y. (eds) *Ensemble Machine Learning*. Boston: Springer, MA.

[11] Freund, Y. & 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.

[12] Dietterich, T.G. (2000). An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning* **40**, 139-157.

[13] Ahmed, N., Atiya, A., El Gayar, N. & El-Shishiny, H. (2010). An Empirical Comparison of Machine Learning Models for Time Series Forecasting. *Econometric Reviews* **29**(5-6), 594-621.

[14] Singh, S., Madan, T. K., Kumar, J. & Singh, A. K. (2019). Stock Market Forecasting using Machine Learning: Today and Tomorrow. Kannur, India: *2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, 738-745.