Università
Ca'Foscari
Venezia

Master's degree in Economics and Finance

Models and Methods of Quantitative Economics - EMJMD QEM

Final Thesis

Directional movement prediction of stock returns using LSTM and Tree-Based models

Supervisors:

Prof. Roberto Casarin

Prof. Marco Corazza

Graduand:

Abhishek Pokhrel

Matriculation Number

888376

Academic Year

2021/2022

ABSTRACT

In a binary classification task, this research investigates the performance of machine learning models in predicting stock return movements of the 30 Dow Jones Industrial Average (DJIA) stocks. We apply two ensemble-of-trees algorithms, namely XGBoost, and BART, and one recurrent neural network algorithm, namely LSTM in order to forecast daily returns depending on target classifications. Daily stock return lags are used as inputs to the machine learning algorithms. We look at two scenarios: one to evaluate the direction of stock returns, and the other to evaluate a stock's ability to outperform the market average. In the first scenario, we estimate whether stock returns will be negative or positive next period, and in the second, we predict if a stock return would exceed the cross-sectional median of all stock returns in the index.

ACKNOWLEDGEMENTS

The author is grateful to Prof. Roberto Casarin and Prof. Marco Corazza for formulating the dissertation topic and their continued guidance and support. I also express immense gratitude to my family for their enduring encouragement and support.

TABLE OF CONTENTS

Chapter 1

Introduction

In his work Statistical Modeling: The Two Cultures, published in 2001, Leo Breiman bemoaned the fact that the community of statisticians had grown unduly dependent on stochastic data models, resulting in "irrelevant theory" and "questionable conclusions" (Breiman, 2001b). Simultaneously, a new breed of data analysts following the machine learning approach were successfully employing black-box, algorithmic models. Among this were random forests and AdaBoost, which were ensemble models of thousands of decision trees which had no obvious relationship to classical statistics. And thus, statisticians remained on the sidelines while new, more effective algorithmic tools supplanted model-based linear regression methods.

Since then, the situation has drastically altered, as indicated by the introduction of statistical machine learning (Hastie et al., 2009). Optimization served as a crucial link between machine learning's "black-boxes" and stochastic data models, allowing statisticians to explain these algorithms in maximum likelihood terms. Using AdaBoost to fit an additive regression model in a stage wise approach is a classic example of this link (Friedman et al., 2000). In early 2000s, ensembles of trees were considered as the black-box methods in the realm of data analysis. Now, with the ability to work with millions of parameters, neural networks are considered as the black-box methods of today. That being the case, these models are still explainable in words of regularization and deviance, i.e. in the language that statisticians are familiar with.

1.1 An Overview of Function Estimation

Many applications call for predicting the value of an unknown variable based on measurements that are known. For example, one would want to forecast the direction of a stock's movement based on the stock's fundamentals. A problem of such nature may be regarded as a function estimation problem. We can represent it mathematically as:

$$y_i = f(x_i) + \epsilon_i \qquad ; \text{where } \mathbb{E}(\epsilon_i) = 0$$

Here, y_i represents the target variable of interest (stock increase) of stock i.
x_i represents the input variables which is a set of features affecting the target variable of interest

There are numerous approaches to estimate f due to the universality of the functional form. The structure of the underlying function f, as well as the distribution of x_i and e_i, have a significant impact on the effectiveness of a method. Using the foundation of statistical learning is one approach. Numerous machine learning algorithms have their theoretical basis provided by the statistical learning theory (Von Luxburg and Schölkopf, 2011). The process of constructing predictive models under this paradigm is known as supervised learning and they are basically predictive functions meant to generate accurate predictions on novel, unseen data from the joint distribution of $\mathbf{X}, \mathbf{Y}$.The estimator is a mapping from the training data (x_i, y_i) to a set of functions represented by $\mathcal{F}$. Every estimate $f \in \mathcal{F}$ is a $\mathbf{X} \to \mathbb{R}$ mapping which produces prediction $\hat{y}_i = \hat{f}(x_i)$ given data x.

A loss function $L(y_i, \hat{y}_i)$ is used to evaluate the accuracy of the function's predictions by measuring the deviation between the predicted and actual outputs. We examine the generalization error, which evaluates $L(\tilde{y_i}, \hat{f}(\tilde{x_i}))$ using unseen data $(\tilde{x_i}, \tilde{y_i})$. It's possible to evaluate $L(f(\tilde{x_i}), \hat{f}(\tilde{x_i}))$ if we know the true underlying function f. In reality, we don't know the true underlying nature of f, whether it is linear or non-linear, single index or multi index. Therefore, non–parametric approaches are used in contemporary statistics to estimate f flexibly. Support

Vector Machines (Cortes and Vapnik, 1995) and Multivariate Adaptive Regression Splines (Friedman, 1991) achieve flexibility by expanding the basis. By introducing non–linearity, neural networks obtain flexibility. Recursive partitioning on $\mathbf{X}$ utilizing its covariates is how tree-based approaches acquire flexibility. Summary statistics from the last partitions are used to make predictions.

The methods presented above are capable of representing complex functions. They have the advantage of being able to learn the non-linearities present in functions. However, they run the risk of overfitting to data noise and creating results that aren't generalizable, which is a drawback. As a result, it's helpful to think about the Bias-Variance tradeoff while evaluating an estimator's performance.

1.2 Bias-Variance Tradeoff

As outlined before, in regression problems, every estimate $f \in \mathcal{F}$ is a $\mathbf{X} \to \mathbb{R}$ mapping which produces prediction $\hat{y}_i = \hat{f}(x_i)$ and we use a loss function to evaluate the accuracy of the function's prediction by measuring the deviation between the predicted and actual outputs. Mean Squared Error (MSE) is one type of loss function that is typically used in the problem of this nature. Mathematicaly, it's defined as:

$$MSE_{Y|x} = \mathbb{E}[(Y - \hat{Y})^2]$$

For interpretability, we can simply decompose the above into:

$$\begin{aligned}\mathbb{E}[(Y - \hat{Y})^2] &= [Y - \mathbb{E}(\hat{Y})]^2 + Var(\hat{Y}) \\ &= Bias(Y, \hat{Y})^2 + Var(\hat{Y})\end{aligned}$$

The duality between consistency in prediction is represented by this breakdown. Low bias, high variation can give rise to very accurate yet inconsistent forecasts leading to data overfitting. On the other hand, low variance, high bias can result in a relatively consistent yet erroneous prediction leading to data underfitting.

The combination that minimizes Mean Squared Error (by construction) is the best compromise between bias and variation. The purpose of cross validation is to use data to discover the best combination of Bias and Variance (Hastie et al., 2009).

Chapter 2

Introduction to Tree-based Models

As a result of their great capacity to yield better forecasts in a wide range of classification and regression tasks, ensembles of trees algorithms have taken center stage in machine learning. Traditional algorithms like random forests and gradient boosting produce data-fits using procedures based on algorithms while the recent algorithms like Bayesian Additive Regression Trees (BART) produce data-fits using an underlying Bayesian probabilistic model.

2.1 Background

Decision Trees

A decision tree model is a model of computation that may be broadly described as repeatedly partitioning the sections of the predictor variables and therefore building a tree-based model that can predict the response. We can trace the origin of this model all the way back to the early 1960s.

With the introduction of the Classification and Regression Tree (CART) algorithm by Breiman et al. (1984), tree-based models have acquired tremendous ground in numerous areas of study. The CART algorithm divides the input variable space into numerous sections that are mutually exclusive. This results in a layered hierarchy of branches that resembles a tree structure, with each branch (or separation) labelled as a node. Each of the decision tree's bottom nodes, known as terminal nodes, has a distinct path for data to reach the region. Following the construction of the decision tree, paths can be used to determine the area or terminal node to which a new collection of predictor variables will belong.

The decision tree model can be separated into two types based on the objective of predictive modeling: regression trees and classification trees.

Ensemble Methods of Decision Tree

Early decision tree approaches had the potential drawback of producing inconsistencies as a result of overfitting. However, these original tree models were improved and their use extended with the development of algorithms such as grandient boosting and random forests. This led to tree based models being adopted for numerous predictive modelling tasks. Random forest simply is a method of creating ensembles of trees from a group of fully grown unpruned trees. Basically, the trees are created using a bootstrap sampling method applied to the source data and using subsample of input variables on every split. Breiman (2001a) demonstrated that using random forests improved prediction accuracy significantly.

Boosting algorithms are able to achieve a sound balance in the Bias-Variance tradeoff which has led this algorithm to become increasingly popular in machine learning. Boosting algorithms create trees in a sequential manner, so that a tree grows with each subsequent iteration. This model integrates weak learners to create strong ones. As explained by Friedman (2002), early approaches of gradient boosting trees used optimization predicated on gradient descent algorithms, which then coined the terminology *gradient boosting*.

Uses of Tree-Based Models

In recent years, tree-based models have become increasingly popular for classification and regression tasks. It's a supervised learning technique with a lot of benefits, especially when it comes to analyzing financial data.

For starters, tree-based models are non-parametric, which means no distribution assumptions are required. Secondly, tree-based models can be utilized as a useful approach for dealing with missing data. This is significant since it is usual for

some information to be missing or unrecorded in real-world databases. Moreover, tree-based models can deal naturally with categorical variables which are advantageous because standard statistical modeling makes it difficult to manage several category levels.

Furthermore, tree-based models can discover non-linear effects and potential links between input factors automatically. Conventional linear models, on the other hand, often capture just linear effects, necessitating additional analysis to uncover non-linearity and interactions.

Finally, by measuring the relative relevance of explanatory factors, tree-based models can provide a variable selection mechanism.

However, because of the variations in their rules and principles, a comparison of the prediction accuracy of conventional linear models with tree-based models may be inappropriate. In practice, however, this comparison may be made to better assess model quality. See, for instance, Thuiller et al. (2003), Maroco et al. (2011).

2.2 Traditional Tree-based Models

Classification and Regression Trees

The Classification and Regression Trees (CART) algorithm (Breiman et al., 1984) constructs a tree structure via a greedy search technique termed recursive binary partitioning. Traditionally, the algorithm's output trees are referred to as classification trees whenever the response variable is categorical, otherwise, they're referred to as regression trees whenever the response variable is continuous. For regression, the following loss function is used to determine the optimal tree:

$$L = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

The estimate $\hat{y}_i$ is a function of x_i Thus, given partitions of data $P_1, ..., P_K$, the

estimate is:

$$\hat{y} = \hat{f}(x) = \sum_{k=1}^{K} \mu_k I(x \in P_k)$$

For every k, we need:

$$arg\min_{\hat{\mu_k}} \left[\sum_{i=1}^{N} (y_i - \sum_{k=1}^{K} \mu_k I((x \in P_k))^2 \right] = \frac{1}{N_k} \sum_{x_i \in P_k} y_i$$

$$N_k = \text{No. of observations of } p_k$$

To discover partitions $P_1, ..., P_K$, CART identifies binary partitions of the data on variable j with cutpoint c in a recursive and greedy manner such that: $P_{left}(j, c) = X|X_j \leq c$ and $P_{right}(j, c) = X|X_j > c$ which minimizes:

$$arg\min_{j,c} \left[\sum_{x_i \in P_{left}(j,c)} (y_i - \mu_{left})^2 + \sum_{x_i \in P_{right}(j,c)} (y_i - \mu_{right})^2 \right]$$

Then, using cost-complexity pruning, CART prunes the fully developed tree. So, it determines which subtree T_α of the entire tree T_{full} minimizes the following:

$$C_\alpha(T) = \sum_{k=1}^{|T|} = \sum_{X_i \in P_k} (y_i - \mu_k)^2 + \alpha|T|$$

$$|T| = \text{No. of Terminal Nodes}$$

Individual decision trees have a large variance. Tiny changes in the input data may cause significant effect on the end model and the forecast. Additionally, tree forecasts are not smooth functions due to their discrete fitting procedure. Particularly, only 2^d (d = maximum depth) potential values are available for each given tree . Here, d is maintained quite low as a result of the pruning described above.

Bagging and Random Forests

Random Forests is based on a method referred to as "bagging" (Breiman, 1996), where numerous variants of a base learner are generated autonomously using

various bootstrapped data samples and the predictions of each learners are subsequently aggregated. Trees, when grown deep enough, are considered low-bias, high-variance learners, making them ideal candidates for bagging. Aggregating over numerous instances can help decrease variability and lessen errors in generalization.

Random Forests improves on bagging by taking it a step further in order to further minimize the variance. The important innovation is examining a portion of predictor variables rather than all variables while searching for an optimal divide at every partitioning phase. Choosing $\bar{m} \leq p$ factors introduces an extra source of unpredictability into the tree-growing process. This unpredictability de-correlates decision trees over the ensembles which thus reduces their variance. There are two major tuning parameters in Random Forests. First being the number of decision trees to be used in the ensemble. The number should be chosen such that the model is computationally stable and doesn't overfit the data. The second major parameter is the number of predictor variables $\bar{m}$ to evaluate at every node split. Owing to empirical evidence, typically $\bar{m}$ is selected to be $p/3$ for regression problems, and $\sqrt{p}$ for classification problems.

Algorithm 1 Random Forest (Hastie et al., 2009)

Input: Training dataset $\mathbf{X}^*, y, B$

for *b= 1,....,B* **do**

  Using the training data, draw a bootstrap sample $\mathbf{X}^*$ of size *sampsize*;
  Grow a decision tree T_b on the bootstrap sample $\mathbf{X}^*$ with recursive binary splitting, and randomly select $\bar{m}$ variables from the p predictor variables with best split criterion *nodesize*;

end

Return the ensemble of trees $\{T_b\}_{b=1}^{B}$;

Average $f_B(\mathbf{x}^*) = \frac{1}{B}\sum_{b=1}^{B} T_b(\mathbf{x}^*)$ {For Regression}

Take Majority Vote $C_B(\mathbf{x}^*)$= *majority vote* $\{C_b(\mathbf{x}^*)\}_{b=1}^{B}$ {For Classification}

Gradient Boosted Trees

Adaptive Boosting (AdaBoost) (Freund and Schapire, 1997) is an ensemble strategy that iteratively fits weak learners. Weak learners are models that perform marginally better than random. Usually in boosting, they are decision trees. Ad-

aBoost, which was initially designed for classification tasks, has over the years been generalized to Gradient Boosting (Friedman, 2001), which operates on any loss function that is differentiable. Gradient Boosted Trees, successively fits decision trees to the residuals of the preceding trees. By utilizing the residual, the model compels the subsequent tree to discover data variations not explained by the preceding trees. This method decreases the total predictive bias.

Individual trees depend on one another via the residual, making parallelization difficult. However, Gradient Boosting often produces smaller trees than Random Forests. In addition, rapid and scalable implementations of gradient boosting algorithms, such as open source XGBoost (Chen and Guestrin, 2016), Microsoft's open source LightGBM (Ke et al., 2017), have discovered ways to parallelize other aspects of the computation. These algorithms have become increasingly popular due to their quick fitting and prediction times, excellent performance on heterogeneous data, and considerable transparency in contrast to black–box models such as neural networks.

Algorithm 2 Gradient Boosted Trees (Hastie et al., 2009)

Input: Training dataset $\mathbf{X}, \mathbf{y}$; learning rate $\boldsymbol{\lambda}$; depth $\mathbf{K}$; subsampling rate $\mathbf{p}$

Initialize a constant $\hat{f}(x) = \arg\, min_{\alpha} \sum_{i=1}^{N} L(y_i, \alpha)$

for *j= 1,....,J* **do**

Compute the negative gradient as working response

$$z_i = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\bigg|_{f(x_i)=\hat{f}(x_i)}$$

Randomly select $\mathbf{p} \times N$ cases from the dataset

Fit a regression tree with K terminal nodes, $h(\mathbf{x}) = E(z|\mathbf{x})$ using the randomly selected cases

Compute the optimal terminal node predictions, $\alpha_1, ..., \alpha_K$, as

$$\alpha_k = arg \min_{\alpha} \sum_{\mathbf{x}_i \in S_k} L(y_i, \hat{f}(\mathbf{x}_i) + \alpha)$$

where S_k is the set of $\mathbf{x}$ that define terminal node k.

Update $\hat{f}(\mathbf{x})$ as

$$\hat{f}(\mathbf{x}) \leftarrow \hat{f}(\mathbf{x}) + \lambda \alpha_{k(\mathbf{x})}$$

where $k(\mathbf{x})$ denotes the index of the terminal node in which an observation with features $\mathbf{x}$ would fall.

end

Boosting algorithms have influenced Bayesian Additive Regression Trees (BART).

Chapter 3

The Bayesian Methodology

3.1 Bayesian CART

Chipman et al. (1998) with Bayesian CART model put forward a Bayesian approach to searching for CART models. It begins by specifying the prior on the tree space and a prior on the conditional distribution which are determined at the terminal nodes of every tree. After, it specifies a Metropolis-Hastings algorithm to explore the posterior and thereby sampling trees from the distribution.

Prior

The CART model is characterized by $\Theta = \mu_1, ...\mu_M$, and the tree structure T. Bayesian analysis of the problem begins with the specification of a prior probability distribution $p(\Theta, T)$. Compositionally, the prior can be defined as:

$$p(\Theta, T) = p(\Theta|T)p(T)$$

The tree prior $p(T)$ is specified by a tree-generating random process. Every tree realization is regarded as a random draw from $p(T)$. The process of drawing from the prior begins with a single root node tree. The tree subsequently grows by randomly splitting the terminal nodes which entails the assignment of splitting rules, and right and left children nodes. The specification of the two functions $p_{split}(\eta, T)$ and $p_{rule}(\rho|\eta, T)$ determines the growth process. Given an intermediate tree T, $p_{split}(\eta, T)$ denotes the probability that the terminal node η is split, and if split, $p_{rule}(\rho|\eta, T)$ denotes the probability of assigning splitting rule ρ to η. Probability p_{rule} is used to pick the variables and cut–points to use for splitting.

Commonly, a predictor x_i is chosen uniformly at random, followed by a cut–point c chosen uniformly at random from the observed values of x_i.

$p_{split}(\eta, T)$ allows us to determine the size and shape of trees generated by the growth process. A general form of $p_{split}(\eta, T)$ which gives greater control for the shape and size of the trees is given as:

$$p_{split}(\eta, T) = \alpha(1 + d_\eta)^{-\beta}$$

where, d_η is the depth of the node η, and $\beta \geq 0$. α and β modeling parameters whose values can be set using cross validation.

The follows algorithm follows from the Chipman et al. (1998):

Algorithm 3 Tree Prior Generation Process

```
Begin by setting T to be the trivial tree consisting of a single root (and terminal) node denoted η
Procedure GROW(η, T)
    Split the terminal node η with probability p_split(η, T)
    if SPLIT then
        Assign splitting rule ρ from distribution p_rule(ρ|η, T) and create the left and right children nodes.
        GROW(η_r, t) and GROW(η_l, t)
    else
        return
```

$p(\Theta|T)$ is the specification of the parameter prior. In practice, we utilize the following zero–centered priors to center and scale the data:

$$\mu_j \mid T \overset{iid}{\sim} N(0, \tau)$$

$$\sigma^2 \sim IG(\nu/2, \nu\lambda/2)$$

Metropolis-Hasting Algorithm

The Metropolis–Hasting algorithm is used to generate draws from the tree posterior by simulating a Markov Chain sequence of trees $T^0, T^1, T^2, ...$

It begins with an initial tree T^0 after which it iteratively simulates the transitions from T^i to T^{i+1} with the steps as outlined in the Algorithm below:

Algorithm 4 Metropolis-Hastings Search Algorithm

1 Generate a candidate value T with probability distribution $q(T^i.T^*)$.
2 Set $T^{i+1} = T^*$ with probability

$$min\left[\frac{q(T^*, T^i}{q(T^i, T^*}\frac{p(Y|X, T^*)p(T^*)}{p(Y|X, T^i)p(T^i)}, 1\right]$$

3 Otherwise, set $T^{i+1} = T^i$

The transition kernel $q(T, T^*)$ is specified as a kernel which generates a new tree T^* from the existing tree T by randomly selecting among the following four steps:

1. GROW: : Randomly pick a terminal node and split it into two new ones by assigning it a splitting rule randomly based on the p_{rule} used in the prior.

2. PRUNE: Randomly pick a parent of two terminal nodes and turn it into a terminal node by collapsing the nodes below it.

3. CHANGE: Randomly select an internal node, and reassign it a splitting rule according to p_{rule}.

4. SWAP: Randomly select a parent-child pair which are both internal nodes. Swap their splitting rules unless the other child has the identical rule. In which case, swap the splitting rule of the parent with that of both children.

Observe, GROW and PRUNE restrain each other, whereas CHANGE and SWAP are counterparts of each other. In other words, if a chain is caught in a minima, the algorithm permits it to retrace and consider alternative solutions. However, once a suitable tree is discovered, the Markov Chain tends to become sticky. Although this issue remains, trees grown using the Bayesian CART approach outperform those grown using the conventional CART method.

3.2 Bayesian Additive Regression Trees (BART)

Bayesian Additive Regression Trees− BART (Chipman et al., 2010) is a "sum-of-trees" model which works by defining a regularization prior for the trees and subsequently drawing samples from a posterior distribution using an iterative Bayesian Backfitting MCMC algorithm. Motivated in particular from Gradient Boosting, BART defines an ensemble method of Bayesian CART. BART has a better mix than Bayesian CART and in many applications, has shown to yields superior results to Random Forests and Gradient Boosting. The model is thus expressed as:

$$Y = \sum_{j=1}^{m} g(x; T_j, M_j) + \epsilon \qquad ; \epsilon \sim N(0, \sigma^2)$$

Every $g(x; T_j, M_j)$ corresponds to a single tree generated with Bayesian CART method. T_j corresponds to the j^{th} binary regression tree and $M_j = \{\mu_{1,j}, ...\mu_{b,j}\}$ are its corresponding terminal node parameters.

Prior

BART uses regularization prior which are spread across numerous trees, used to create weak learners. This approach compels every tree to acquire a more nuanced understanding of the desired function.

The regularization prior is specified as:

$$\begin{aligned} p((T_1, M_1), ..., (T_m, M_m), \sigma) &= \left[\prod_j p(T_j, M_j)\right] p(\sigma) \\ &= \left[\prod_j p(T_j|M_j)p(T_j)\right] p(\sigma) \end{aligned}$$

where,

$$p(M_j|T_j) = \prod_i p(\mu_{ij}|T_j)$$

For every $T_j, p(T_j)$, and $p(M_j|T_j)$, the priors are specified as in Bayesian CART.

Bayesian Backfitting MCMC Algorithm

Given observed data y, the Bayesian Backfitting MCMC algorithm specifies a tree sampling process from the following posterior distribution:

$$p((T_1, M_1), ..., (T_m, M_m), \sigma|y)$$

Defining m trees, it locally modifies individual trees sequentially using the residual trees.

Defining $T_{(j)}$ to be the set of all trees in the sum excluding T_j. Also, define $M_{(j)}$ in similar manner. $T_{(j)}$ will be a collection of $m-1$ trees, with $M_{(j)}$ being the corresponding terminal node parameters. The draws are defined to fit on the residual by:

$$(T_j, M_j)|T_{(j)}, M_{(j)}, \sigma, y$$

$$\sigma|T_1, ..., T_m, y$$

We draw σ using the inverse gamma conjugate. To obtain m tree draws, we must first determine the vector of partial residuals R_j that are based on a fit which excludes the jth tree. The equation is given by:

$$R_j \equiv y - \sum_{k \neq j} g(\mathbf{X}; T_k, M_k)$$

This allows for a more compact representation of the tree's posterior as:

$$(T_j, M_j)|R_j, \sigma$$

As M_j employs a conjugate prior, we can carry out every draw from the tree's

posterior in the two sequential steps as:

$$T_j|R_j, \sigma$$

$$M_j|T_j, R_j, \sigma$$

The draw of T_j can be obtained using the Metropolis−Hastings algorithm as defined for Bayesian CART model. The draw of M_j can be obtained using the simple Gaussian conjugate for every μ_{ij}.

The described backfitting algorithm is ergodic, and generates a series of draws of $(T_1, M_1), ..., (T_m, M_m), \sigma$ which converges in distribution to the posterior $p((T_1, M_1), ..., (T_m, M_m), \sigma|y)$ i.e. $p(f|y)$ of "true" $f(.)$. Thus, to obtain sample from the posterior, an obvious choice is to use the average of the after burn-in sample $f_1^*, f_2^*, ..., f_K^*$:

$$\frac{1}{K}\sum_{k=1}^{K} f_k^*(x)$$

The solution is not confined to the average of the posterior distribution, but rather the posterior can accommodate any function of interest, as is the case with all Bayesian analyses. With this algorithm, it's also possible to extract the median or a desired credible intervals from the posterior. This is a great advantage of using this algorithm as opposed to other traditional ensembles algorithms. Nevertheless, there is a computational burden in using BART. It fits relatively slowly and, by contemporary standards, it is not feasible to fit on big datasets.

Chapter 4

Neural Networks

4.1 General Features

Second half of the 20th century saw an enormous advancements in computer technology which would subsequently inspire researchers to make advances in artificial intelligence. Artificial Neural Networks (ANN) are a mathematically complicated idea whose inspiration came from the human brain and its neural cells. ANNs are a subtype of machine learning methods known as supervised learning. Under this form of learning, the neural networks require instances of complete sets of inputs and outputs when training to understand the relationships between them.

McCulloch and Pitts (1943) designed the first neural network with the notion that neurons may be utilized to perform logical binary operations depending on the binary inputs gathered. Output depended on whether or not the neuron was fired which subsequently depended on whether or not a certain threshold was fulfilled. Later, Rosenblatt (1958) introduced weights to the individual inputs, enabling more complicated computations. It was the earliest neural network to consist of a single perceptron.

Once perceptrons (often referred to as neurons) existed, it was desirable for them to approximate all mathematical functions and handle classification problems. Block (1970) demonstrated that a single perceptron cannot anticipate the output of the XOR (exclusive OR) function given two inputs. Their proposed approach was thus to expand the structure of a single layer into a multi-layer model of neurons connecting to each neuron in the subsequent layer. This new model was

dubbed a multi-layer perceptron (MLP).

Now we outline the fundamental characteristics of neural networks in general:

Layers: The layers of Neural Networks (NN) are classified into three types: Input Layer, Hidden Layer, and Output Layer. Serving as a starting point, the input layer is where the researchers establish what shape the input should take based on the type of the problem. As an example, in classifying handwritten digits, the input layer typically contains the number of neurons necessary to recognize the number of pixels in the image. The input layer neurons are linked to the first hidden layer neurons. It may be difficult to determine the number of hidden layers and the number of neurons. These layers are essential to the nonlinear modeling of output, but the optimal amount need not be extremely high. Employing the parsimony principle serves to avoid model overfitting and a lengthy procedure. The output layer is dependent on the nature of the problem. As an example, if the output is one among the ten recognized digits, then the number of output neurons will be ten. An outcome of the process is delivered by the one with the highest value of output. An output layer may consist of a single neuron with an activation function in order to decide whether the value of the output (0 or 1).

Activation Function: In Neural Networks, each neuron forms a summing function which is a weighted mathematical sum of its inputs and an added bias.

$$y = \sum_{i=1}^{n} (w_i x_i) \quad + b$$

where y is an output, x_i are n inputs with respective weights w_i, and b is the bias.

The neuron's final output computes the summing function first, then uses the result as input to the activation function. An example of a frequently used activation function is the sigmoid function, given as:

$$f(x) = \frac{1}{1 + e^{-x}}$$

It's features (non-linear, differential, decreasing, bounded) enable sigmoid to integrate inputs in a non-linear manner and enhance the model's training capability.

Optimizers: These are algorithms that change the weights of neurons in order to minimize the loss to its maximum extent. Gradient descent is the most popular of the optimizers. This optimizer searches for the loss minima by changing weights based on the first-order derivative of the loss function. The loss is propagated backwards through the layers by a process called backpropagation. Then, the output and loss function are computed, accompanied by an update to the selected weights. The fundamental drawback of the Gradient descent optimizer is the local minima trap. For big datasets, the method may struggle to converge to the minima. We employ the Adam optimizer in our models invented by Kingma and Ba (2014). Adam is more cost-effective than other optimizers in terms of training time. Adam permits us to configure an initial learning rate, exponential decays for the first and second-moment estimations, and an epsilon, a small value that prevents zero division throughout the optimization process. In the major libraries, these options are configured by default. Initial weights are small random numbers. Given the nature of algorithms, this characteristic enables stochastic algorithms to investigate the set of possible solutions more effectively.

Loss Function: A goal of the optimizer is to adjust weights so as to reduce a loss function. The loss is a metric that indicates how well our model accomplishes its goal. The goal is to reduce a model's loss, as a lower loss indicates that the model is performing better. There are several loss functions to pick from, and which one to use relies significantly on the

task for which the neural network is being utilized. In this thesis, we are dealing with a binary classification problem, and therefore we use the widely regarded binary cross-entropy loss function.

Dropout: Dropout is a regularization method proposed by Srivastava et al. (2014) that is applicable to all neural network types. The dropout seeks to prevent the model from becoming overfitted. Here, randomly selected neurons are eliminated during the training phase, resulting in "thinned networks" that are immune to over co-adaptation and faster to calculate. In the test phase, all "thinned networks" are approximated to their original form and made ready for the evaluation of test data. During the building of the model in Keras , the likelihood of a unit being dropped can be specified and applied to either the input or hidden layers.

4.2 Advantages of Neural Networks

The complicated computational technique of neural networks serves primarily to identify nonlinear relationships in the data. Furthermore, the multi-layer design permits dimension reduction without knowledge of the association between variables. Researchers can also discover novel relationships between factors that appear to be unrelated and drastically reduce forecasting errors. Moreover, if a substantial amount of data is available, neural networks become more effective than other (conventional) learning techniques.

4.3 Recurrent Neural Network

Recurrent Neural Networks (RNN) are characterized by their recurrent cells, which provide the capacity to recall information from prior stages of computations. RNNs are advantageous for voice recognition models because they can process strings of texts. From the 2010s, recurrent neural networks have been utilized successfully for the development of music, dialogue, images, and molec-

ular design. They were even utilized to create a script for a film, which was subsequently cast with real actors. In addition, they are a valuable instrument for time series forecasting and anomaly identification, such as with financial data. Kamruzzaman et al. (2006) lists a variety of uses of neural networks in financial and manufacturing challenges, including trading and forecasting, future price estimate, portfolio selection, foreign exchange rate forecasting, bankruptcy prediction, and fraud detection.

Assume we have an observed time series sequence, $(\overrightarrow{\sigma_1}, \overrightarrow{\sigma_2}, ..., \overrightarrow{\sigma_n})$. To predict $\overrightarrow{\sigma_{i+1}}$ with an RNN cell, we suppose it is a function of $\overrightarrow{\sigma_i}$ and $\overrightarrow{h_i}$, the hidden state of the preceding RNN cell:

$$\overrightarrow{\sigma_{i+1}}, \overrightarrow{h_{i+1}} = RNN\left(\overrightarrow{\sigma_i}, \overrightarrow{h_i}\right)$$

Here, it is anticipated that the RNN function will compute an updated hidden state, and then use it to predict the new output, returning both. The hidden state is updated using an activation function.

$$\overrightarrow{h_{i+1}} = tanh\left(\overrightarrow{W_o}\overrightarrow{o_i} + \overrightarrow{W_h}\overrightarrow{h_i} + B\right)$$
$$\overrightarrow{o_{i+1}} = \sigma\left(\overrightarrow{W_{yh}}\overrightarrow{h_i} + B_y\right)$$

With time, however, as a result of the vanishing gradient problem induced by a technique employed during training, the RNN's memory of previous steps fades in favor of more recent ones. Although RNNs have a memory, the typical models have a limited one.

4.4 Long Short-Term Memory (LSTM)

There are numerous subtypes of RNN, one of them being the Long Short-Term Memory (LSTM) neural network which was introduced by Hochreiter and Schmidhuber (1997). Their goal was to solve the major RNN problem, the vanishing

gradient, where there is rapid loss of information over time. To combat this issue, recurring cells were updated into memory cells which had completely different characteristics. This network has feedback connections and every neuron with a memory cell has three gates: input gate, output gate, and forget gate.

The forget gate uses the last observed output, applies a sigmoid function to the current input, and multiplies the result by the internal state to determine, on a scale from 0 to 1, how much information will be passed on. The input gate combines the most recently observed output with the fresh input and passes them to the activation function. The subsequent layer returns a vector to be added to the cell's internal state using a hyperbolic tangent. The proportion of this new vector is added to the existing state and multiplied by the forget gate before being sent to the output gate. This implies that in addition to the weights of the neurons, we must additionally consider the current state of the network. Throughout this process, LSTM is learning how much historical data to retain and how much to discard. These requirements predestine LSTM to be highly advantageous for any sort of time series prediction. Particularly for data where a certain degree of dependence on past observations is envisaged. They are well-equipped for identifying, analyzing, and predicting time-dependent data.

Due to the structures of so-called gates, LSTMs are distinguished by their extensive memory. Input, output, and a forget gate comprise the three gates in the network's architecture. These gates determine what information should be added to memory and what should be forgotten. There are two states in the LSTM: the cell state and the hidden state. The initial cell state is employed for information storage, whereas the concealed state is used for formulating predictions. As noted previously, the forget gate is a highly crucial gate of the LSTM. While the cell state stores and collects all information in its memory, the forget gate determines the information to forget and maintain.

This forget gate is governed by a basic mathematical sigmoid function with a result of 0 when we wish to totally clear information from memory and a result

of 1 when the knowledge should be retained, or a result in between (the sigmoid function is however, always positive). The sigmoid function contains a sum of the input multiplied by the weights from the input to the hidden layer, the previous state of the hidden layer multiplied by its weights, and the bias for this gate.

Output vector from forget gate : $\overrightarrow{f_t}$

Previous hidden state : $\overrightarrow{h_{t-1}}$

Current input : $\overrightarrow{x_t}$

Weight vector for forget gate : $\overrightarrow{W_f}$

Bias for forget gate : $\overrightarrow{B_f}$

$$\overrightarrow{f_t} = \sigma\left(\overrightarrow{W_f}.\left[\overrightarrow{h_{t-1}}, \overrightarrow{x_t}\right] + \overrightarrow{B_f}\right)$$

After the forget gate vector has been computed, the input gate comes into play. The input gate determines whether or not to add fresh information to the memory. It is composed of two neuron layers, the sigmoid and tanh layers. The sigmoid layer determines which information to update, while the tanh layer produces a vector of temporary cell state containing candidates which can thus be utilized to update the actual cell state.

Input gate vector : $\overrightarrow{i_t}$

Candidate cell state values : $\overrightarrow{C_t}$

The input gate vector and candidate cell state are determined as follows:

$$\overrightarrow{i_t} = \sigma\left(\overrightarrow{W_i}.\left[\overrightarrow{h_{t-1}}, \overrightarrow{x_t}\right] + \overrightarrow{B_i}\right)$$
$$\overrightarrow{C_t} = tanh\left(\overrightarrow{W_c}.\left[\overrightarrow{h_{t-1}}, \overrightarrow{x_t}\right] + \overrightarrow{B_c}\right)$$

Given that we now have the forget and input gate output vector as well as the candidate state, we update the old cell state $\overrightarrow{C_{t-1}}$. We begin by multiplying $\overrightarrow{C_{t-1}}$ with $\overrightarrow{f_t}$. Meaning that if a number in the forget gate output vector is 0, the corresponding number in the old cell state becomes 0 and is forgotten, however if the number is 1, the cell state's number remains unaltered. Subsequently, we multiply the input gate output vector by the candidate cell state vector and add the resulting product to the initial product. The new cell state is thus given as:

$$\overrightarrow{C_t} = \overrightarrow{f_t} * \overrightarrow{C_{t-1}} + \overrightarrow{i_t} * \tilde{C}_t$$

The final component is the output gate, via which the new hidden state h_t is obtained. This state produces a filtered rendition of the new cell state as its output. Similar to the input gate, it comprises a sigmoid and tanh layer. The sigmoid layer uses the previous hidden state as well as the input to determine which elements of the cell state will be included in the output. The cell state is then passed across the tanh layer. Finally, the product of these layers is multiplied to produce the final output.

$$\overrightarrow{o_t} = \sigma\left(\overrightarrow{W_o}\left[\overrightarrow{h_{t-1}}, \overrightarrow{x_t}\right] + \overrightarrow{B_o}\right)$$
$$\overrightarrow{h_t} = \overrightarrow{o_t} * tanh\left(\overrightarrow{C_t}\right)$$

Chapter 5

Experiments and Results

In this empirical analysis, we have implemented three machine learning algorithms, XGBoost, BART, and LSTM to predict the one year daily directional movements for the 30 stocks of the DJIA. The analysis was done using the **R** programming language. The methodology that we used are outlined in the following sections.

5.1 Data Preprocessing

Following Krauss et al. (2017), we use three years of daily stock data to train our models, and one year of daily stock data to test our models. We select the period from 2018-01-01 until 2020-12-31 for training, and the period from 2021-01-01 until 2021-12-31 for testing. We download the daily adjusted closing price of each of the 30 stock constituents of the Dow Jones Industrial Average (DJIA). To this end, the current stock constituents of DJIA were scraped from Wikipedia by using the **rvest** package, and the closing price were downloaded using the **quantmod** package. We then calculate the daily returns of stock s at time t, denoted as R_t^s from the adjusted closing price using the formula:

$$R_t^s = \frac{AdjClosingPrice_t^s}{AdjClosingPrice_{t-1}^s} - 1$$

For model fitting and testing, we consider a lag period of 7 to be used as our input. Therefore, for stock s at time t, we define the feature vector by $X_t^s = (R_{t-1}^s, ..., R_{t-7}^s)$.

5.2 Model Targets

We then calculatd two target criterias for our models, outlined as follows:

Criteria 1

Following Fischer and Krauss (2018), we use the cross-sectional median returns of all DJIA stocks at a particular time t to define this target criteria. Taking the DJIA as the market proxy, we try to predict whether a particular stock outperformed the market i.e. the average of the DJIA stocks. Thus, we define:

$$Y_t^s = \begin{cases} 1 & \text{if } R_t^s > M_t \\ 0 & \text{else} \end{cases}$$

where,

Y_t^s is the target for stock s at time t.

R_t^s is the returns for stock s at time t.

$M_t = median(R_t^1, R_t^2, ..., R_t^{30})$ is the cross-sectional median of the returns of all 30 stocks at time t.

Note: Target criteria can still be 1 even if the individual returns is negative, and 0 even if returns is positive.

Criteria 2

Following Basak et al. (2019), we use the directional price movements of a particular stock s at a particular time t to define this target criteria. Here, we try to predict whether the price of a particular stock goes up or down. Thus, we define:

$$Y_t^s = \begin{cases} 1 & \text{if } R_t^s > 0 \\ 0 & \text{else} \end{cases}$$

5.3 Model Evaluation

Given the input features and their respective target criteria, the machine learning models calculated a target probability $\hat{f}(X_t^s) \in (0,1)$. We then transformed these probability values into a target class $\hat{Y}_t^s \in (0,1)$ based on:

Criteria 1:

Here, we transform the predicted probability to class 1 if it's greater than the predicted cross-sectional median.

$$\hat{Y}_t^s = \begin{cases} 1 & \text{if } \hat{f}(X_t^s) > \hat{M}_t \\ 0 & \text{else} \end{cases}$$

where,
$\hat{M}_t = median\left(\hat{f}(X_t^1), ..., \hat{f}(X_t^{30})\right)$ is the cross-sectional median of the predicted probabilities of all 30 stocks at time t.

Criteria 2:

Here, we transform the predicted probability to class 1 if it's greater than 0.5 i.e. positive return is predicted with probability greater than 0.5.

$$\hat{Y}_t^s = \begin{cases} 1 & \text{if } \hat{f}(X_t^s) > 0.5 \\ 0 & \text{else} \end{cases}$$

Prediction Accuracy:

The prediction accuracy of the models was calculated by averaging the total number of correct classifications. That is,

$$Accuracy = \frac{1}{30N} \sum_{s=1}^{30} \sum_{t=1}^{N} 1_{[y_t^s = \hat{y}_t^s]}$$

where, N is the total number of trading days, y_t^s is the target class observed and

$\hat{y}_t^s$ is the target class predicted for stock s at time t, 1 is the indicator function and 30 is the total number of stocks.

5.4 Model Implementation

The models were implemented in R and we describe the method for each model used.

XGBoost

We used the package **xgboost** (Chen et al., 2015) to implement this model. The function **xgb.DMatrix**() was used to first convert the training and testing data to matrix form as this structure was needed to run the model. For our binary classification model, we set the argument objective to "*binary : logistic*", and the booster is set as "*gbtree*".

We further tuned the parameters and used the following parameter settings to run the model:

```
#For Criteria 1

params_list <- list(
  booster = "gbtree",
  objective = "binary:logistic",
  eta = 0.01,
  max_depth = 1,
  subsample = 0.4,
  eval_metric = "mlogloss")
```

```
#For Criteria 2

params_list <- list(
  booster = "gbtree",
  objective = "binary:logistic",
  eta = 0.01,
  max_depth = 9,
  subsample = 0.5,
  eval_metric = "mlogloss")
```

Also, after tuning, we thus used 478 trees while training for Criteria 1 model, and 731 trees while training for Criteria 2 model.

BART

We used the package **bartMachine** (Kapelner and Bleich, 2013) to implement this model. We use the default parameter setting to implement the model, where the number of trees is set to 50. After running the model, we observe that the

MCMC sampling reached convergence at around 200 iterations, as shown in the convergence diagnostic plot below:

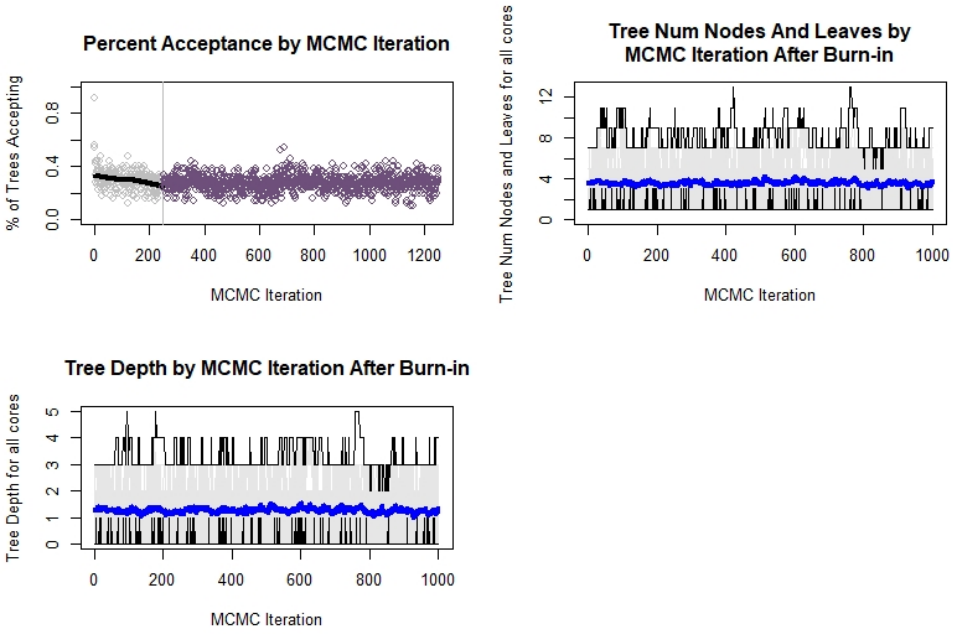

Figure 5.1: Plot Convergence Diagnostics - Criteria 1

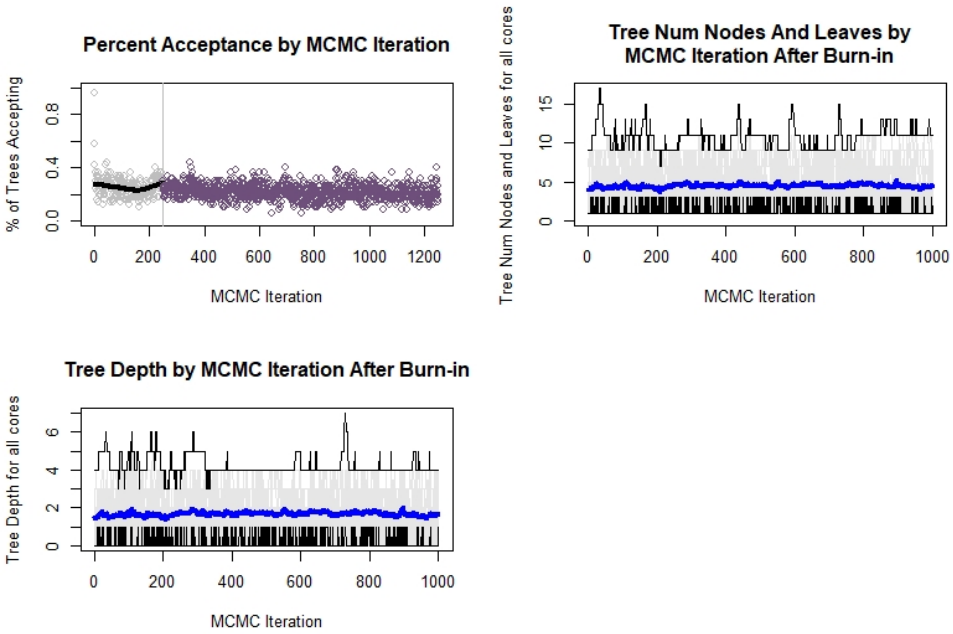

Figure 5.2: Plot Convergence Diagnostics - Criteria 2

LSTM

We used the package **keras** to implement this model, using the *keras_model_sequential* function. In our model, we used the 7 period lags as inputs in the input layer. Our model also had 2 hidden layers each comprising of h= 10 neurons, while the output layer comprised of 1 neuron. ReLU was the activation function that we used in all layers besides the output layer, for which we used the sigmoid activation function. Motivated by Fischer and Krauss (2018), we also apply dropout regularization within the hidden layers, using a relatively low dropout value of 0.1. Consequently, the danger of overfitting is decreased and generalization is enhanced as a result of the random removal of a proportion of the input units at each training update, both at the input gates and the recurrent connections. To further reduce overfitting risk, we use early stopping to derive the number of epochs dynamically.
Subsequently, we train the model on the training data for all stocks and then use it to predict our testing data.

5.5 Results

When applied to our test data, we observe a prediction accuracy of about 50% with considerably low volatility suggesting that our models, XGBoost, BART, and LSTM are no better than random chance in predicting the directional movement of returns. Thus, this implies that the features used, i.e. the 7 period returns lags, contains no information that aids in our prediction.

Observing the prediction accuracy of BART and LSTM models for Criteria 2 which are slightly above 0.5, we might say they predict slightly better than random. However, the lower confidence interval only marginally edges above 0.5 giving no evidence of improved prediction accuracy above 0.5. Also, a thing to note is that the LSTM prediction for criteria 2 has almost zero specificity with respect to target value 1. This means that the model completely fails at predicting

the target class 0 and is therefore not a model that can be generalized.

The following figures shows the prediction accuracy of the models on the testing dataset for both Criteria 1 and 2.

```
               Accuracy : 0.5054
                 95% CI : (0.4941, 0.5168)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 0.1758

                  Kappa : 0.0108

 Mcnemar's Test P-Value : 1.0000

            Sensitivity : 0.5053
            Specificity : 0.5056
         Pos Pred Value : 0.5054
         Neg Pred Value : 0.5054
             Prevalence : 0.5000
         Detection Rate : 0.2526
   Detection Prevalence : 0.4999
      Balanced Accuracy : 0.5054
```

(a) Criteria 1

```
               Accuracy : 0.5054
                 95% CI : (0.4941, 0.5168)
    No Information Rate : 0.6851
    P-Value [Acc > NIR] : 1

                  Kappa : -0.0041

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.5186
            Specificity : 0.4767
         Pos Pred Value : 0.6831
         Neg Pred Value : 0.3128
             Prevalence : 0.6851
         Detection Rate : 0.3553
   Detection Prevalence : 0.5201
      Balanced Accuracy : 0.4977
```

(b) Criteria 2

Figure 5.3: XGBoost

```
               Accuracy : 0.5054
                 95% CI : (0.4941, 0.5168)
    No Information Rate : 0.6851
    P-Value [Acc > NIR] : 1

                  Kappa : -0.0041

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.5186
            Specificity : 0.4767
         Pos Pred Value : 0.6831
         Neg Pred Value : 0.3128
             Prevalence : 0.6851
         Detection Rate : 0.3553
   Detection Prevalence : 0.5201
      Balanced Accuracy : 0.4977
```

(a) Criteria 1

```
               Accuracy : 0.5131
                 95% CI : (0.5018, 0.5244)
    No Information Rate : 0.789
    P-Value [Acc > NIR] : 1

                  Kappa : 0.003

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.5210
            Specificity : 0.4834
         Pos Pred Value : 0.7904
         Neg Pred Value : 0.2125
             Prevalence : 0.7890
         Detection Rate : 0.4111
   Detection Prevalence : 0.5201
      Balanced Accuracy : 0.5022
```

(b) Criteria 2

Figure 5.4: BART

```
               Accuracy : 0.5095
                 95% CI : (0.4982, 0.5209)
    No Information Rate : 0.5009
    P-Value [Acc > NIR] : 0.06895

                  Kappa : 0.019

 Mcnemar's Test P-Value : 0.93456

            Sensitivity : 0.5094
            Specificity : 0.5096
         Pos Pred Value : 0.5086
         Neg Pred Value : 0.5104
             Prevalence : 0.4991
         Detection Rate : 0.2542
   Detection Prevalence : 0.4999
      Balanced Accuracy : 0.5095
```

(a) Criteria 1

```
               Accuracy : 0.52
                 95% CI : (0.5086, 0.5313)
    No Information Rate : 0.9999
    P-Value [Acc > NIR] : 1

                  Kappa : -3e-04

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.5200
            Specificity : 0.0000
         Pos Pred Value : 0.9997
         Neg Pred Value : 0.0000
             Prevalence : 0.9999
         Detection Rate : 0.5200
   Detection Prevalence : 0.5201
      Balanced Accuracy : 0.2600
```

(b) Criteria 2

Figure 5.5: LSTM

Chapter 6

Concluding Remarks

In the realms of academic and applied research, there have been extensive use of supervised machine learning algorithms. By learning through experience with the use of comprehensive data sets, these algorithms have been able to produce accurate predictions over an extensive number of applications.

In this study, we applied the XGBoost, Bayesian Additive Regression Trees (BART), and a memory-based LSTM network to a financial prediction task on the Dow Jones Industrial Average (DJIA) stocks. The results of the three models where then compared in terms of their prediction accuracy. The primary source motivating our analysis was the paper by Fischer and Krauss (2018) where they concluded that memory-based LSTM networks can outperform memory-free classification models when predicting directional movements of stock prices.

Thus, undertaking this empirical study, we found that the frequently used machine learning models that are widely recognized as the best performing in terms of financial market predictions, were unable to predict the price movements better than random chance. Our findings suggests that either these algorithms are incapable of identifying the pattern underlying stock return or that return lags are white noise.

We should note that in the past few years, the stock market was very unstable. Fueled by COVID-19, the market was highly uncertain and volatile, which perhaps led to the poor model performances of our machine learning models. To combat this, sentiment analysis can be incorporated into financial market predictions. There are several studies which show that the sentiments expressed through

social medias, and news stories can provide insights into stock price trends and thus improve prediction accuracy. See for instance, Li et al. (2020), Ren et al. (2018). Further research on sentiment analysis into financial markets in combination with the use of state-of-the-art machine learning models may lead to a more promising results. ‘

REFERENCES

Basak, S., S. Kar, S. Saha, L. Khaidem, and S. R. Dey (2019). Predicting the direction of stock market prices using tree-based classifiers. *The North American Journal of Economics and Finance 47*, 552–567.

Block, H. (1970). A review of "perceptrons: An introduction to computational geometry. *Information and Control 17*(5), 501–522.

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

Breiman, L. (2001b). Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science 16*(3), 199–231.

Breiman, L., J. Friedman, R. Olshen, and C. Stone (1984). Classification and regression trees (chapman y hall, eds.). *Monterey, CA, EE. UU.: Wadsworth International Group*.

Chen, T. and C. Guestrin (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794.

Chen, T., T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, et al. (2015). Xgboost: extreme gradient boosting. *R package version 0.4-2 1*(4), 1–4.

Chipman, H. A., E. I. George, and R. E. McCulloch (1998). Bayesian cart model search. *Journal of the American Statistical Association 93*(443), 935–948.

Chipman, H. A., E. I. George, and R. E. McCulloch (2010). Bart: Bayesian additive regression trees. *The Annals of Applied Statistics 4*(1), 266–298.

Cortes, C. and V. Vapnik (1995). Support vector machine. *Machine learning 20*(3), 273–297.

Fischer, T. and C. Krauss (2018). Deep learning with long short-term memory networks for financial market predictions. *European journal of operational research 270*(2), 654–669.

Freund, Y. and R. E. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences 55*(1), 119–139.

Friedman, J., T. Hastie, and R. Tibshirani (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics 28*(2), 337–407.

Friedman, J. H. (1991). Multivariate adaptive regression splines. *The annals of statistics 19*(1), 1–67.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189–1232.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational statistics & data analysis 38*(4), 367–378.

Hastie, T., R. Tibshirani, J. H. Friedman, and J. H. Friedman (2009). *The elements of statistical learning: data mining, inference, and prediction*, Volume 2. Springer.

Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. *Neural computation 9*(8), 1735–1780.

Kamruzzaman, J., R. Begg, and R. Sarker (2006). *Artificial neural networks in finance and manufacturing*. IGI Global.

Kapelner, A. and J. Bleich (2013). bartmachine: Machine learning with bayesian additive regression trees. *arXiv preprint arXiv:1312.2171*.

Ke, G., Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems 30*.

Kingma, D. P. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krauss, C., X. A. Do, and N. Huck (2017). Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the s&p 500. *European Journal of Operational Research 259*(2), 689–702.

Li, X., P. Wu, and W. Wang (2020). Incorporating stock prices and news sentiments for stock market prediction: A case of hong kong. *Information Processing & Management 57*(5), 102212.

Maroco, J., D. Silva, A. Rodrigues, M. Guerreiro, I. Santana, and A. de Mendonça (2011). Data mining methods in the prediction of dementia: A real-data comparison of the accuracy, sensitivity and specificity of linear discriminant analysis, logistic regression, neural networks, support vector machines, classification trees and random forests. *BMC research notes 4*(1), 1–14.

McCulloch, W. S. and W. Pitts (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics 5*(4), 115–133.

Ren, R., D. D. Wu, and T. Liu (2018). Forecasting stock market movement direction using sentiment analysis and support vector machine. *IEEE Systems Journal 13*(1), 760–770.

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

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research 15*(1), 1929–1958.

Thuiller, W., M. B. Araújo, and S. Lavorel (2003). Generalized models vs. classification tree analysis: predicting spatial distributions of plant species at different scales. *Journal of Vegetation Science 14*(5), 669–680.

Von Luxburg, U. and B. Schölkopf (2011). Statistical learning theory: Models, concepts, and results. In *Handbook of the History of Logic*, Volume 10, pp. 651–706. Elsevier.