# Ensemble Methods

Tanishq Kulthe

Submitted for the Degree of Master of Science in

MSc Data Science and Analytics


Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

December 17, 2022

# Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

**Word Count**: 12064

**Student Name**: Tanishq Kulthe

**Date of Submission**: 17th December, 2022

**Signature**: Tanishq kulthe

# Abstract

Machine Learning plays an important role in created user tailored experiences in many websites and applications. Ensemble methods represent one of the best Machine Learning algorithms to implement such features. This project aims at implementing multiple ensemble methods along with their underlying algorithms from scratch for both classification and regression-based problems. These methods are applied to relevant datasets and their performances are compared.

At the core of this project are Decision Trees and Stumps. This project implements Decision Trees for both classification and regression. A Decision Stump can be created by setting the height of the Decision Tree as one. The ensemble methods implemented for this project are Bagging, Random Forest, and ADABoost for classification, and Bagging, Random Forest, and Gradient Boost for regression. All of the aforementioned algorithms are implemented as python libraries. The ensemble methods generate forest of trees. The algorithm for generation of trees along with their predictions is specific to the ensemble method.

These libraries are utilized in four jupyter notebooks, two for regression, and two for classification. Time consuming and hardware intensive task like estimating losses and parameter tuning are performed in one file using GridSearchCV, and analysis and plotting are performed in another. In the end the estimators are compared on the basis of losses and visualized using learning curves.

# Contents

# 1 Introduction

## 1.1 Machine Learning

Machine learning (ML), a subfield of artificial intelligence, focuses on how AI learns from experience and develops predictions based on that experience. Machine learning is the study of statistical models and algorithms that utilise patterns and inferences as opposed to explicit instructions to carry out a certain set of activities by computer systems (ML). As a subset of artificial intelligence, it is understood. ML algorithms generate a mathematical model based on samples. Without specialized programming, the task can be accomplished using data that has been identified as training data to produce hypotheses or judgments.

AI calculations are created using a model-building informational gathering. When fresh data is familiar with a machine learning computation, a forecast is made using the model. The process of a machine learning application is described in Figure. In order to build a model that accurately predicts outcomes, machine learning (ML) starts by reading and monitoring the training data to find relevant insights and patterns. The test information set is then used to deliberate the model's effectiveness. This approach is used up until the point where the computer learns to automatically map the input to the desired output without any intervention from humans.

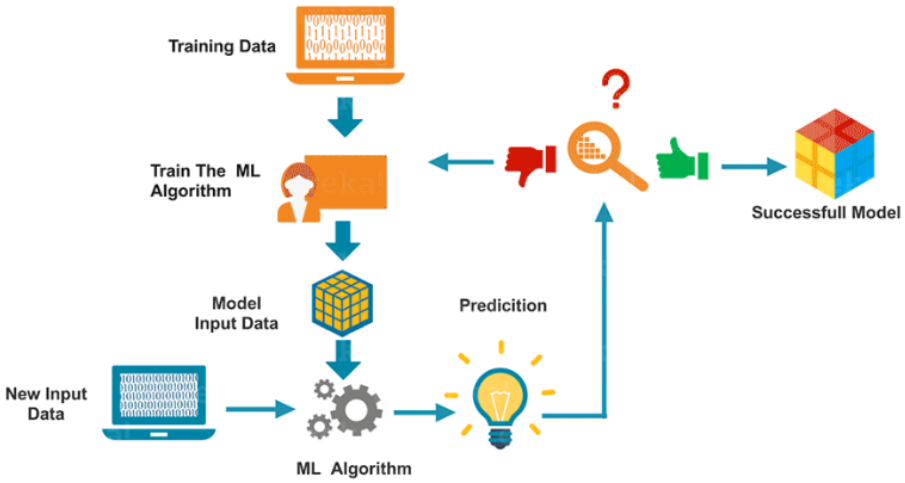



Figure 1.1 how Machine Learning works [1]

Arthur Samuel, an IBM computer scientist and pioneer in artificial intelligence and computer games, is credited with coining the term "machine learning." Samuel created a checkers-playing computer programmer. The more the programmer was used, the more it used algorithms to forecast outcomes and learned from experience. Machine learning is a field that studies the research and creations of algorithms that can studied and predict data.

The study of the production and research in machine learning because it can solve issues at a speed and scale that cannot be matched by the human mind alone, ML has shown to be useful. Machines can be trained to recognize patterns in and relationships between incoming data by putting large amounts of processing power behind a single activity or a number of focused tasks. This allows machines to automate repetitive tasks.

Data Is Vital: Machine learning algorithms are essential to its success. Without being expressly taught to do so, ML algorithms construct a mathematical model from sample data, frequently referred to as "training set of data," in order to generate predictions or decisions. This can highlight patterns in the data that organizations can utilize to enhance decision-making, maximize productivity, and collect meaningful data at scale. The Purpose Is AI: AI systems that automate workflows and find solutions to data-based business challenges on their own are built on top of machine learning (ML). It enables

businesses to supplement or replace specific human competencies. Chabot's, self-driving cars, and speech recognition are examples of common machine learning applications you could encounter in daily life.

## 1.2 History of Machine Learning

In order to fully utilize artificial intelligence (AI) technology, machine learning (ML) is a crucial technique. Machine learning is frequently referred to as AI due to its capacity for learning and decision-making, but in truth, it is a subset of AI. Up to the late 1970s, it was a stage in the development of AI. It then split off to continue evolving on its own. Machine learning is now employed in many cutting-edge technologies and has become a crucial response tool for cloud computing and ecommerce. For many firms today, machine learning is an essential component of contemporary business and research. It helps computer systems perform better over time by using algorithms and neural network models.Without being explicitly instructed to make certain judgments, machine learning algorithms use sample data, commonly known as "training data," to automatically create a mathematical model. A model of how brain cells interact is one of the foundations of machine learning. The Organization of Behavior, written by Donald Hebb in 1949, contains the model (PDF). Hebb's thoughts on neuronal transmission and excitement are presented in the book.Hebb stated, "When one cell repeatedly assists in firing another, the first cell's axon generates synaptic knobs in contact with the second cell's soma, or enlarges them if they already exist." When two neurons or nodes are triggered simultaneously, their association becomes stronger than when they are activated separately. These associations are referred to as "weights," and nodes/neurons that tend to be both positive and negative are said to have "strong positive weights." Strong negative weights are developed by nodes that typically have opposite weights (for example, 11=1, -1x-1=1, and 11=-1).Some of the biggest technological achievements of the last few years can now be attributed to machine learning. It is employed in both the emerging field of self-driving cars and in galactic exploration since it aids in the discovery of exponents.

## 1.3 Why use Ensemble Learning?

Simply put, we employ ensemble learning models because they frequently produce superior outcomes. It is unfair to place the burden of obtaining the greatest outcome on just one model. Even if the model performs well on one dataset, we could observe inconsistent performance on other datasets. You have both excellent and poor professors. Let's imagine you want to get the best possible grades in discrete mathematics. There are three instructors: the best instructor who teaches and provides the best video lectures, the best instructor who teases exam questions, and the best instructor who provides the best notes. Choose any of these educators to maximize your grades in discrete mathematics; the best course of action is to blend the solutions provided by all three teachers and use ensemble machine learning techniques to this situation. The same principles apply to ensemble models. Now that you know why we utilize ensemble learning, you can see how it often outperforms individual models. Okay, it's been interesting getting a handle on ensemble learning. Let's get down to business now. Although I dislike utilizing jargon to master data science, it is the preferred.[2]

Let's discuss the many ensemble learning techniques available in Python. We will first discuss the basic models before moving on to the more advanced ensemble learning strategies used in machine learning. Let's start!

Basic Ensemble Techniques in Machine Learning, The maximum voting, averaging, and weighted average approaches are covered in the sections below. These approaches are simple but incredibly effective. Theoretical reading and comprehension of data science is enjoyable.[3]

## 1.4 Applications of Machine Learning

Machine learning is becoming increasingly popular because to modern technology, and it is growing swiftly. We utilize machine learning every day in programmers like Google Maps, Google Assistant, Alexa, etc. without even realizing it. The top machine learning real-world applications are summarized in the list below:

### 1.4.1 **Image Recognition:**

One of the most often utilized applications of machine learning is image identification. It's employed to label things like digital images, people, places, and objects. A frequent application of facial recognition and image recognition is automatic buddy suggestion. Facebook provides us with the option of automatic buddy tagging recommendations. Whenever we invite our friends to view a photo on Facebook.

### 1.4.2 **Speaking Recognition:**

We may "Search by voice" on Google, which is a well-known machine learning application that falls under speech recognition. The act of converting spoken orders into writing is referred to as speech recognition, often known as "Speech to text" or "Computer speech recognition." It is methods are currently widely used in speech recognition applications. Alexa, Google Assistant, Siri, Cortana, and Microsoft Cortana all employ speech recognition technologies to carry out voice instructions.

### 1.4.3 **Self-driving cars:**

Autonomous automobiles are among the most fascinating applications of machine learning. It is a key component of self-driving vehicles. Tesla, the most well-known automaker, is working on a self-driving car. Unsupervised learning was utilized to teach the automobile models to identify people and objects while they were moving.

# 2 Background Research

## 2.1 Types of Machine Learning

It is possible to further categories the various machine learning techniques. Below is a list of the two most important types of machine learning: unsupervised and supervised learning.

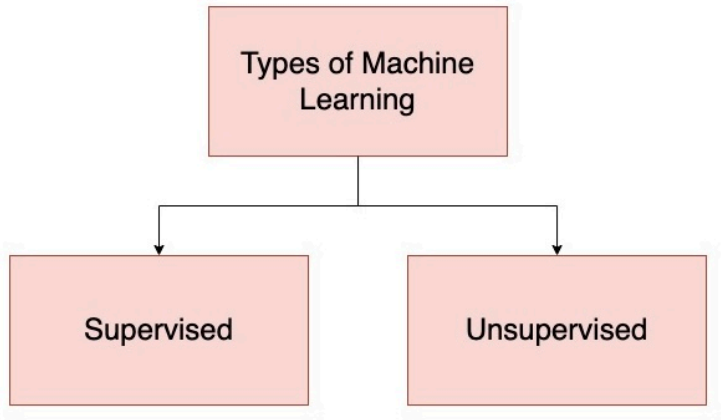



Figure 2.1Types of Machine Learning

## 2.2 On the basis of Training

### *2.2.1* **Supervised Learning:**

The dataset for supervised learning consists of a set of labeled data with input variables (x), output variables (y), and guidelines for using an algorithm to learn the mapping function from input to output. F(x)=y.

You must precisely approximation the mapping function in order to anticipate the output variables (y) for that data when you receive new input variable (x) Our goal is to more clearly characterize the supervised learning problem as researching a feature h: x y in order to make h(x) a helpful analyst for the value of y, given a training set. This quality is referred to as a hypothesis. It is illustrated in this way practically in the image below.

When attempting to predict a continuous target variable, such as housing, the challenge of learning a regression is well recognised. If y can only take on a limited set of distinct values, such as the living area, it can be said to be a classification problem. For example, given the living area, it can be predicted if a building is, say, a home or an apartment.

Problems with classification and regression can also be distinguished from supervised learning concerns. There is a categorization issue when the output variable is divided into the categories of "red," "blue," "illness," or "no disease." There is a regression issue when the output variable is a real number, such as "lira" or "weight." [4]

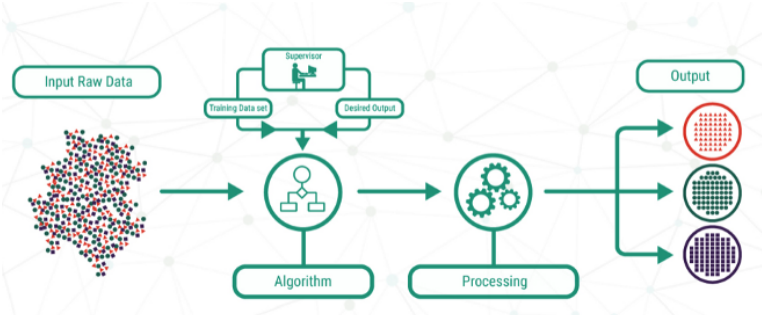



Figure 2.2.1 Schematic of Supervised learning

### 2.2.2 **Unsupervised Learning:**

In this type of machine learning, the region where there are no dependent variables is non-marked or supervised learning, and the outcomes are unambiguously labeled with conceptual conditions.

The first stage in this technique is to start with the points as independent clusters. There is only one cluster left in the pair of clusters closest to the current step in each subsequent step as it advances. There are three main types of unlabeled unknown data in that system, and because unsupervised learning lacks a defined knowledge set, the majority of the difficulties go unrecognized in the findings. Despite having a sizable and faultless logical operation to direct it, the system is made even more complex by the obtainability of adequate algorithms for input and output. Simply put, the AI system and ML objective are blinded when the operation goes as anticipated.

Unsupervised learning is as remarkable as it sounds, it uses input set of data and the binary logic mechanism present in all

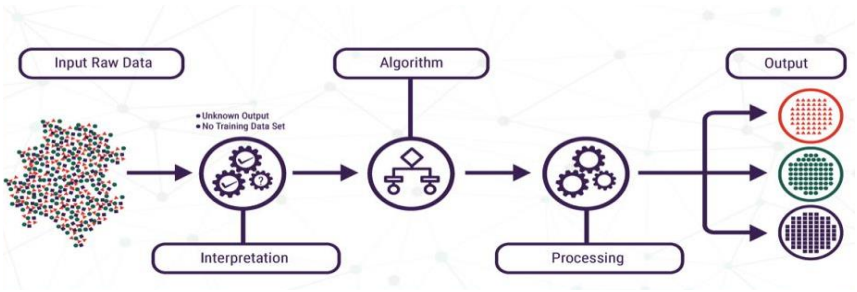



*Figure 2.2.2 Schematic of* Unsupervised *learning [5]*

computer systems to comprehend and solve an endless number of issues. For the model, there are no reference data available.

### 2.2.3 On the basis of Predictions

Techniques for classification and regression are examples of supervised learning approaches. Both techniques are used in machine learning for prediction and make use of labeled datasets. They diverge from one another, though, when it comes to how they are used to solve certain machine learning issues. As opposed to classification algorithms, which are used to predict or classify discrete values such as Male or Female, True or False, Spam or Not Spam, etc., regression algorithms are used to predict or classify continuous variables such as price, wage, age, etc.

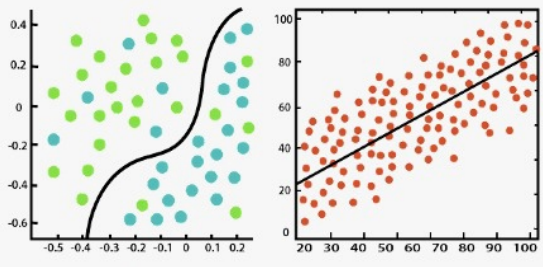



Figure 2.2.3 Classification & Regression[14]

#### 2.2.3.1 Classification

Finding a function to divide the dataset into groups based on several criteria is the task of classification. In classification, the data is divided into various classifications by a computer programmer that has been trained on the training dataset. The classification method seeks to identify the mapping function to transform the input (x) to the discrete output (y).

#### 2.2.3.2 Regression

Finding links between dependent and independent variables is the process of regression. It facilitates the forecasting of continuous variables such as market movements, housing values, and other things. Finding the mapping function to convert the continuous output variable (y) to the input variable (x) is the goal of the regression algorithm (y).

## 2.3 Machine Learning Algorithms

### 2.3.1 KNN

It is more frequently employed to address classification issues. It is a simple method that preserves all of the existing cases before sorting new examples with the approval of at least k of their neighbors. The case is then given to the class that it most closely resembles. This calculation is made using a distance function. Before choosing the K Nearest Neighbors algorithm, take the following into account:

- The computational cost of KNN is high.
- Higher range variables should be standardized to prevent algorithm bias.
- Preprocessing of the data is still necessary.

### 2.3.2 Decision Trees

One of the most widely used machine learning algorithms nowadays is the decision tree algorithm, a supervised learning method for categorizing situations. It can classify dependent variables that are categorical or continuous. In accordance with the most important characteristics or independent variables, this strategy separates the population into two or more homogeneous groupings.

### 2.3.3 Linear Regression

One of the most fundamental and well-liked Machine Learning methods is linear regression. It is a technique for performing statistically-based predictive analysis. For continuous/real/numeric

variables like sales, salary, age, and product price, among others, linear regression creates projections. The linear equation serves as a representation for this line, often known as the regression line.

$$Y=a*X+b$$

One or more independent variables (y) and a dependent variable are shown to have a linear relationship by the linear regression process, commonly known as linear regression (y). Linear regression can be used to find out how the value of the dependent variable changes over time as a function of the value of the independent variable because there is a linear relationship between the two variables. The link between the variables is represented by a sloping straight line in the linear regression model. Look at the illustration below:

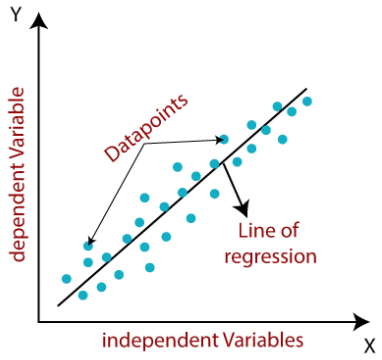



Figure 2.3.1 Linear Regression[6]

### 2.3.4 Logistic Regression

Logistic regression is used to estimate discrete values (typically binary values like 0/1) from a collection of independent variables. It helps predict the likelihood of an event by converting the data to a logit function. Sometimes referred to as logit regression. The following methods are widely used to improve logistic regression models:

- The terms of interaction
- Exclude characteristics
- Using a non-linear model
- Regularization technique

## 2.4 Data Sets

### 2.4.1 Training set

A set of data based on examples is used to fit the parameters (like weights) of something like a classifier. The training data set is examined by a supervised learning algorithm to identify the optimum variable codes that will provide a reliable predictive model for classification tasks. A trained (fitted) model ought to be very generalizable to brand-new, suspect data. We evaluate the fitted model's precision in categorising new data using "new" samples from the datasets that have been kept out (validation and test datasets). To lower the possibility of problems like over-fitting, it is not advised to train the model using examples from the validation and test datasets. Most methods for discovering empirical relationships in training data have a tendency to overfit the data, which enables them to find and take advantage of relationships in training data that appear to exist but are false.

### 2.4.2 Validation set

It is a set of examples used to modify the hyperparameters of a classifier (i.e., architecture). It is frequently referred to as a "dev set" or "development set." The quantity of hidden units in each layer

is an illustration of a hyper parameter for an artificial neural network. It ought to adhere to the same probability distribution as the training and testing sets of data.

Over fitting must be prevented by stacking a validation data set on top of the training and test datasets when a classifying parameter needs to be changed. After numerous candidate classifiers have been trained using the training data set, validation data set, and test data set, respectively, the test data set is utilised to collect performance metrics including accuracy, sensitivity, specificity, and F-measure. Due to its inclusion of testing-related training data that is neither a component of low-level training nor of final testing, the validation data set performs as a hybrid. The simplest way to use a validation data set is as one of the training, validation, and test data sets when choosing a model. It goes as follows: The easiest method for comparing different networks is by evaluating the error function with data that is different from the training data. This allows us to compare the performance of each network using new data. By decreasing an acceptable error function made in respect to a training data set, various networks are trained. The network with the lowest error in respect to the validation set is chosen after analysing the performance of the networks using an independent validation set to evaluate the error function.

This tactic is known as the holdout method. Because this approach may lead to some over fitting to the validation set, it is advisable to evaluate the effectiveness of the selected network using a third independent set of data known as a test set. Early stopping candidate models are iterations of the same network that are trained until the validation set error increases, at which point the training is stopped and the previous model is applied.

### 2.4.3 Testing set

With the same probability distribution as the training data set, it is a different set of data. If a model correctly fits the training data set and the test data set, there hasn't been considerable over fitting (see figure below). Over fitting is evident since this data set frequently fits the model better than the test data set.

A test set, or collection of cases, is consequently used to assess the effectiveness (or generalization) of a fully described classifier. These predictions from the final model are used to categories the cases in the test set. To determine the model's accuracy, these predictions are compared to the cases' actual classifications.

The final model chosen during the validation step is commonly evaluated using the test dataset when both the validation and test datasets are utilized. When the initial data set is divided into training and test datasets, the model may only be evaluated once on the test dataset. Be aware that certain websites discourage using this strategy. However, two divisions may be adequate and productive when utilizing a technique like cross-validation, where results are averaged after numerous cycles of model training and testing, aiding in the elimination of bias and variability.

## 2.5 Data Preparation

### 2.5.1 Data Analysis

Data analysis entails converting, visualizing, and manipulating data in order to derive insightful conclusions from the findings. On the basis of these insights, people, companies, and even governments frequently make decisions. Data analysts may use basic linear regression to forecast consumer behaviour, stock prices, or insurance claims. They may use graphs to illustrate the portfolio of a financial technology business to acquire some impact knowledge, or they could use classification and regression trees (CART) to produce homogeneous groups. Human analysts were indispensable when it comes to identifying patterns in data until the last decades of the 20th century. However, machines can and do handle most of the analytical work today. They are still crucial when it comes to giving the correct kind of data to learning algorithms and deriving meaning from algorithmic output.

### 2.5.2 **Feature Selection**

It is a set of examples used to modify the hyper parameters of a classifier. It is frequently referred to as a "development set." The quantity of hidden units in each layer is an illustration of a hyper

parameter for an artificial neural network. It ought to adhere to the same probability distribution as the training and testing sets of data.

Over fitting must be prevented by stacking a validation data set on top of the training and test datasets when a classifying parameter needs to be changed. After numerous candidate classifiers have been trained using the training data set, validation data set, and test data set, respectively, the test data set is utilized to collect performance metrics including accuracy, sensitivity, specificity, and F-measure. Due to its inclusion of testing-related training data that is neither a component of low-level training nor of final testing, the validation data set performs as a hybrid.

The simplest way to use a validation data set is as one of the training, validation, and test data sets when choosing a model. It goes as follows: The easiest method for comparing different networks is by evaluating the error function with data that is different from the training data. This allows us to compare the performance of each network using new data. By decreasing an acceptable error function made in respect to a training data set, various networks are trained. The network with the lowest error in respect to the validation set is chosen after analysing the performance of the networks using an independent validation set to evaluate the error function.

### 2.5.3 Data Preprocessing

Preparing raw data to be acceptable for a machine learning model is known as data preparation. In order to build a machine learning model, it is the first and most important stage. Real-world data typically includes noise, missing values, and may be in an undesirable format, making it impossible to build machine learning models on it directly. Data preprocessing is necessary to clean the data and prepare it for a machine learning model, which also improves the model's accuracy and effectiveness.

## 2.6 Building a Decision Stump

CART, or Classification and Regression Trees, is an acronym that Leo Breiman created to stand for decision tree approaches that can be used to solve classification or regression predictive modeling problems. A binary tree serves as the CART model's representation. Assuming the value is numerical, a node represents a single input variable (X) and a split point on that variable. The prediction is made using the output variable (y), which is located in the leaf nodes of the tree, also known as the terminal nodes. After it has been made, a tree can be studied by adding a new row of information anytime a branch splits, all the way up until a conclusion is reached. In reality, segmenting the input space is a necessary step in the building of a binary decision tree. The space is divided using the greedy method of recursive binary splitting. As soon as all the numbers are in order, a cost function is used to study and test a number of split points. Because we are trying to reduce costs, we pick the division with the lowest costs. A greedy cost function is used to analyses and select each split point and input variable.

### 2.6.1 **Classification**

The purity of the nodes is estimated using the Gini cost function, which depends on how mixed the training data provided to each node is. A minimum number of training instances must be present at each node before splitting can stop or a maximum tree depth can be reached.

### 2.6.2 **Regression**

The sum squared error for all training samples that fall inside the rectangle is utilized to determine the split points.

## 2.7 Loss Functions

### 2.7.1 gini impurity

When all elements are precisely divided into distinct classes, the division is referred to as pure (an ideal scenario). The probability that a randomly chosen example would be wrongly classified by a particular node is predicted using the Gini impurity, which is pronounced "genie." Because it demonstrates how the model differs from a pure division, it is known as a "impurity" measure. The Gini impurity scale goes from 0 to 1, where 0 means that all of the elements belong to the same class and 1 means that there is only one class.

Let's examine the Gini impurity as follows:

$$Gini(t) = 1 - \sum_{i=1}^{j} P(i|t)^2$$

The j stands for how many classes are in the label, and
The P stands for the class ratio at the i th node.

### 2.7.2 RSS

A statistical technique called residual sum of squares (RSS) can be used to determine the degree of discrepancy in a dataset that cannot be explained by a regression model. As a result, it calculates the variance between the actual data's value and the value predicted by the regression model.

### 2.7.3 Loss functions

The predictions of the model you constructed are closely correlated with the Loss function. Therefore, if the value of your loss function is lower, your model will be producing good outcomes. It is necessary to minimise the Loss function, or perhaps I should say the Cost function, which is used to assess the model's performance.

Predicting the respective probability of each class that the problem involves is the task in classification. In contrast, the goal of regression is to forecast the continuous value for a given set of independent features for the learning process.

## 2.8 Ensemble Methods

They are Meta algorithms that blend different machine learning methods into a single prediction model to increase the predictability and stability. While some models do well when modelling one component of the data, others perform well when modelling a different feature. Learn a few simple models instead of many complex ones, then combine their output to arrive at the final choice. In ensemble learning, the cumulative model strength cancels out the biases and variances of each individual model. As a result, the accuracy of the composite is higher than the accuracy of the individual models. Techniques like boasting and stacking are employed to achieve this Bagging.

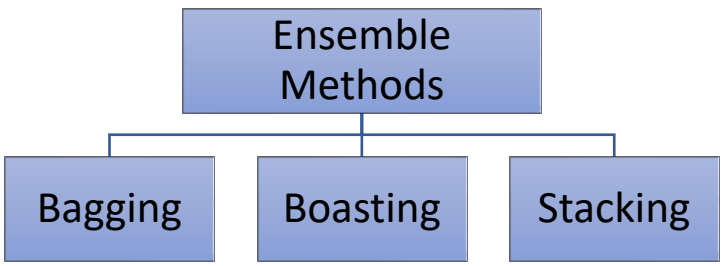



Figure 2.8 (a) Ensemble Methods

Multiple machine learning models are merged through the technique of ensemble learning to provide superior outcomes. The ideas we'll cover are simple to understand. We already have a general understanding of Ensemble models from the introduction. The fundamental tenet is that a model combination can produce results that are more accurate than any one machine learning model by itself.

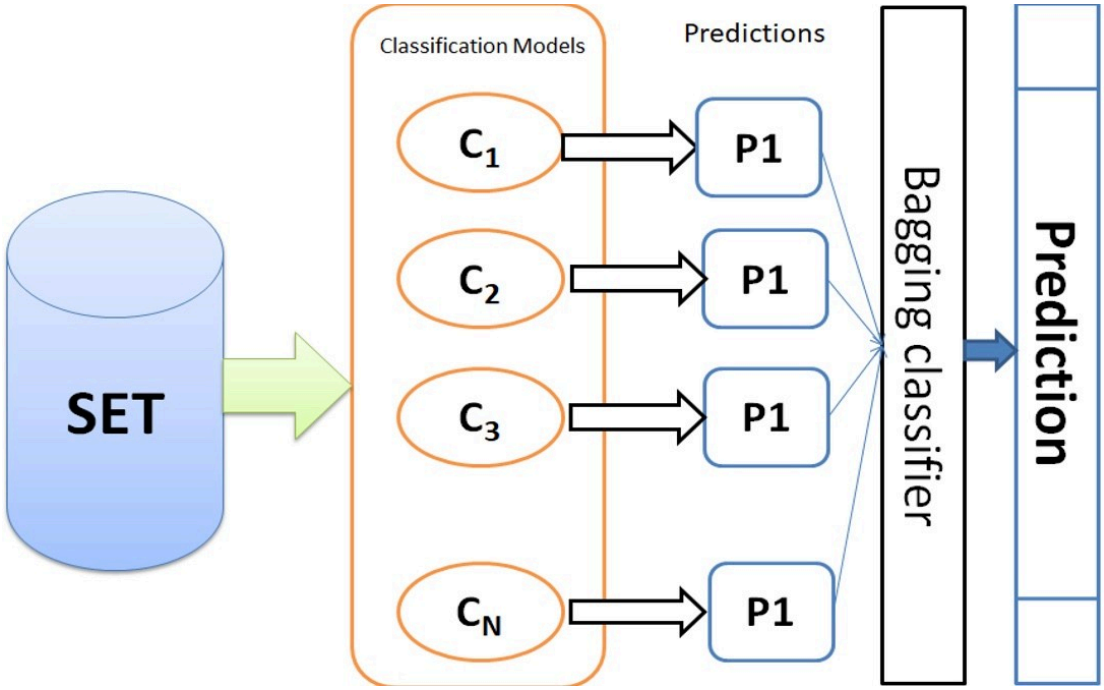



*Figure 2.8 (b) Bagging Classifier*

Let's go over the example from earlier. We were discussing the best approach to control our diets so that we might get in shape. The main goal of each of the aforementioned tactics is to increase calorie intake. Each technique has its own distinct but effective means of achieving desired results. By appropriately mixing the procedures, the activity as a whole will yield the best results. Consider a model as a person who studies patterns and makes recommendations on what to do next. Based on their prior experiences, the advice from each person in our example can be viewed as a model for the best method to get fit. Learners are another name for models. Each student is viewed as being "weak" on their own.

However, a robust model can be created when several weak learners are joined in one way or another. We shall refer to this robust model as an ensemble model. The models ought to be as varied as is practical. On various subsets of the data, each model might do well. The inadequacies of the individual models are balanced out when they are combined. The premise of ensemble models is that there is strength in multiple models over a single one.

### 2.8.1 Decision Tree vs Decision Stump

The decision stump is a single-level decision tree and a type of machine learning model. A single internal node (the root) that is directly connected to the terminal nodes makes it a decision tree (its leaves). A decision stump only considers the value of one particular input attribute while making a prediction. They are also known as 1-rules on occasion. The type of input feature will determine the various variants that can be made. In order to represent nominal features, one can either construct a tree stump with two leaves, each of which represents a different category while the other leaf represents every other category, or a stump with a leaf for each conceivable value of the feature. These two methods are equivalent for binary attributes. One could consider a missing value to be in a distinct category. Typically, a threshold feature value is used for continuous features, and the stump comprises two leaves for values below and over the threshold. Rarely, though, more than one threshold may be

selected, in which case the stump has three or more leaves. Decision stumps are frequently employed in machine learning ensemble approaches like bagging and boosting as components (sometimes known as "weak learners" or "base learners"). For instance, the AdaBoost method with decision stumps as weak learners is used in the Viola-Jones face detection system.

### 2.8.2 Bagging

Bootstrapped Aggregating is referred to as bagging. Making the most of a situation while employing available resources is known as bootstrapping. Putting our possessions into a class or cluster is aggregating. There are two basic steps in bagging: Utilize our current training data and create many instances of the data (bootstrapping); in this case, you can reuse the same training samples. The data can be sampled with replacement.From this bootstrapped data and various model outputs, create multiple models. Add together the model's findings to obtain the final outcome.

Let's say you're trying to teach fifth-graders. Consider asking them to respond to 50 True or False questions. As a result, you decide to teach the kids 40 questions before having them answer ten of them in a test. The train data in this instance consists of 40 questions, the test data of 10, and the overall dataset is made up of 50 questions.Let's say you have 25 children. One approach is to teach them all at once, and then test them all at once. We can use the 40 practise questions we have here to their full potential. It's okay to ask questions repeatedly. We are bootstrapping in this case, or making the most of the tools at our disposal. After that, the students will try to answer the 10 test questions. Here, we'll offer the kids a chance to answer each of the ten questions honestly or falsely. We'll conduct a maximum vote at the end to decide the result. It's referred to as aggregating. Last but not least, we bootstrap the readily available train data, build many models, and aggregate the results. Aggregating and bootstrapping are both referred to as bagging.[12]

*To the process of aggregating and bootstrapping.*

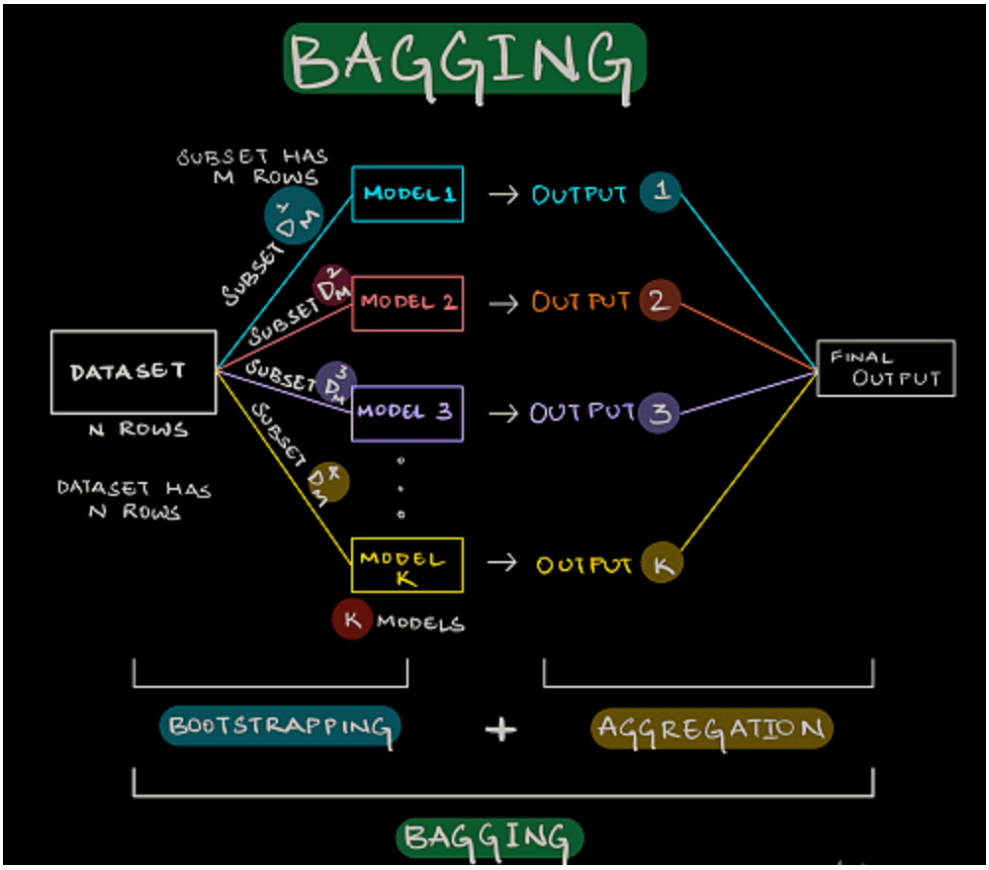



Figure: Bagging[12]

### 2.8.3 **Boosting**

This is enjoyable. Boosting is based on the idea that it expresses. A model is first run, and the output is obtained. The remainder is wrongly classified, while some points are correctly classified. We will now repeat the model and assign extra weight to the points that were erroneously categorised. What transpires? Since some points have a higher weight than others, the model is more likely to include those elements. As we keep going through this process, more models are produced, and each one fixes the flaws of the prior one.

Let's say you are instructing a young person in maths. They make mistakes when they first start, and you point them out in each session so they don't repeat them. The child has a clear understanding of how to distinguish between right and wrong after several sessions. This gradual development was accomplished by emphasising (boosting) their errors.

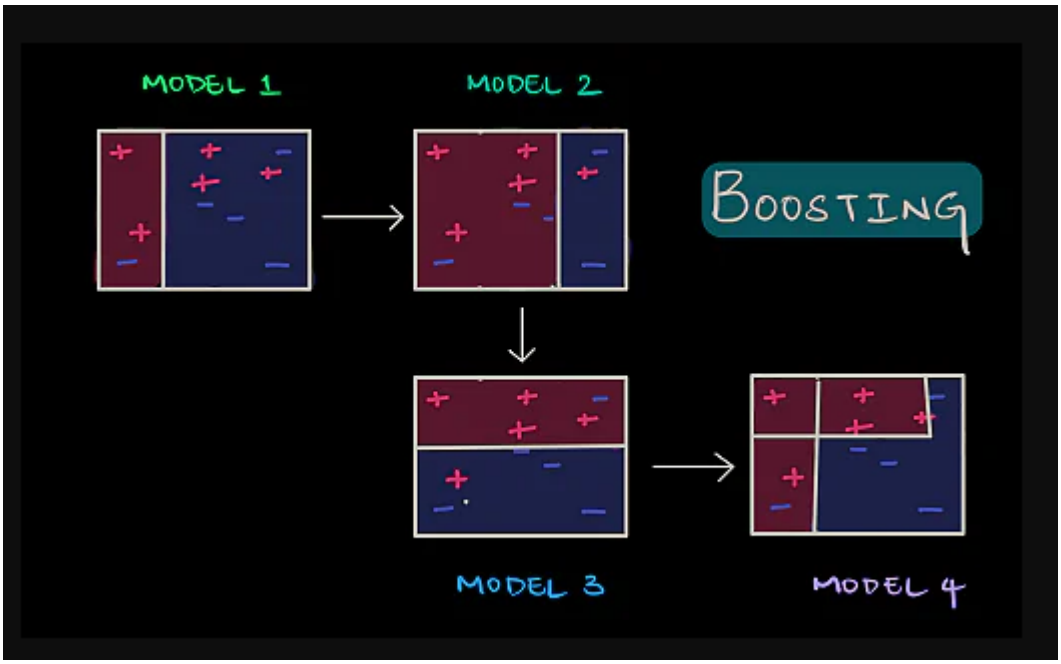



Figure: Boosting

Boosting uses forms like, Adaptive Boosting (AdaBoost), gradient boosting, and XGBoost (Extreme Gradient Boosting). [13]

#### 2.8.3.1 Adaptive boosting of AdaBoost :

Adaptive boosting is abbreviated as AdaBoost. We gave the technique the name "adaptive" because it uses the missed-classification cases from the prior model to produce a new weighted data sample where the missed-classification cases are given additional weight. Thus, the newly added model ought to be more adaptable in resolving the erroneous behaviour of the earlier model.

The Adaptive Boosting technique was created by Yoav Freund and Robert Schapire, who both received the Gödel Prize for their work. AdaBoost concentrates on topics that the average learner has trouble with. The boosting strategy is used to transform a foundation learner, which is a weak machine learning algorithm, into a strong learner. Any machine learning algorithm that accepts weights on training data can be used as a basic learner. A decision stump serves as the basic learner in the paradigm described below.

The decision stump algorithm is used to classify the randomly selected points from the training data. We fitted the decision tree stump to the entire training set after classifying the sampled points. This method is repeated until all training data fits accurately or a predetermined maximum number of estimators have been used.

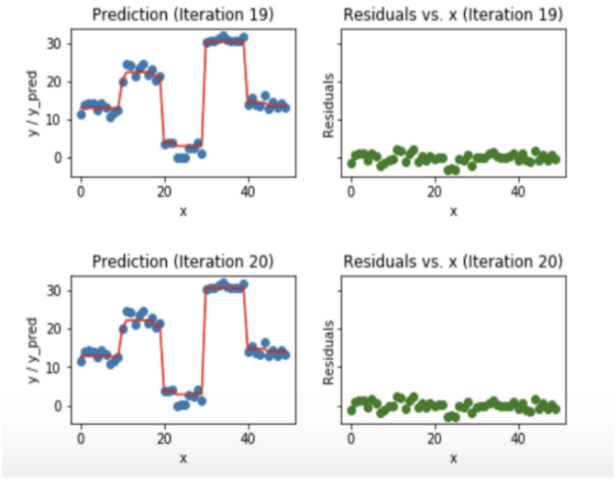
### 2.8.3.2 Gradient Boosting :

In contrast to AdaBoost, gradient boosting uses a loss function rather than penalising missed-classified cases. For regression problems, the loss function can be the mean average error, or log loss for classification issues. The gradient boosting approach additionally employs the gradient descent technique to constantly minimise the loss function in order to identify the ideal point.

Gradient boosting, another boosting approach. Gradient Boosting, which is also a boosting method .aims to develop a strong learner from a group of weak ones. Although this technique and adaptive boosting (AdaBoost) are similar, there are some key differences. By taking a loss function and attempting to optimise it, we try to view the boosting problem as an optimisation problem in this approach. Leo Breiman was the first to come up with this concept.

In order to improve performance and develop a strong learner, we start with a weak learner (in the prior scenario, it was decision stump) and add another weak learner at each stage. As a result, the loss function loses less. Each model is successively added, and the loss is calculated. The loss value is used to update the predictions in order to minimise the residuals, which are the error residual.

You can see that the model can fit the data better after three iterations. The residuals are reduced progressively until they reach zero.[7]




The model almost perfectly matches the data after 20 iterations, and the residuals are nil.

### 2.8.4 Random Forest :

It has numerous decision trees, each of which represents a different instance of how the classification of data entered into the random forest is done. The forecast chosen by the random forest technique is the one that receives the most votes after taking into account each case separately.

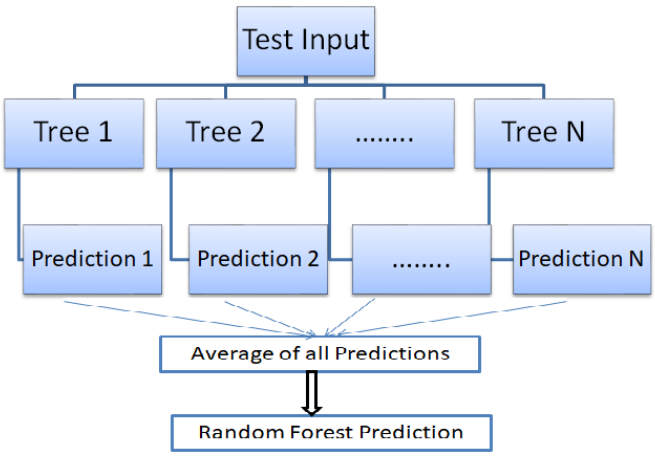



Figure: Random Forest

## 2.9 Common Libraries for Ensemble Methods

### 2.9.1 XGBoost :

Extreme Gradient Boosting, or XGBoost, refers to engineers' efforts to stretch the capabilities of the gradient boosting method's computing power. The method of gradient boosting is improved in XGBoost. First, by applying regularization, it enhances the over fitting. By leveraging parallel running to optimize sorting, it also increases runtime speed. Finally, it prunes the tree using the decision tree's maximum depth, which drastically cuts down on runtime. XGBoost is one of the most popular gradient boosting variants. It uses the gradient boosting framework and is an ensemble machine learning method based on decision trees. XGBoost can boost a machine learning model's efficiency and speed. However, decision tree-based algorithms are now regarded as best-in-class when applied to modest to moderate amounts of structured/tabular data.

XGBoost uses a histogram-based algorithm and a pre-sorted method to get the best split. The histogram-based method employs discrete bins to calculate the split value of the histogram by dividing all the data points for a feature into discrete bins. The number of terminal nodes in the trees can also change with XGBoost, and the left weights of the trees whose calculations rely on less evidence are reduced more severely.

### 2.9.2 Cat Boost :

Cat Boost is the name of a modern machine learning algorithm. Deep learning frameworks with simple user interfaces include Google's Tensor Flow and Apple's Core ML. Cat Boost works with a variety of data formats and can help organizations with a variety of issues today. Cat Boost's one hot max size enables the flexibility of providing indices for categorical columns for one-hot. Additionally, if the cat features argument is left empty, Cat Boost will consider each column as a numeric variable. Cat boost addresses categorical features. They feature discretization into a set number of bins and data processing with GPU acceleration (128 and 32).

The implementation of symmetric trees by Cat Boost distinguishes it significantly from other boosting techniques. Although it may seem strange, this aids in reducing prediction time, which is crucial in low latency environments.

### 2.9.3 LightGBM :

Lightweight gradient boosting devices are referred to as LightGBM. It filters out the data instances in order to determine a split value using a cutting-edge method called Gradient-based One-Side Sampling (GOSS). Due to its rapid speed, LightGBM is prefixed with "Light." Being able to handle enormous data sizes and using less RAM to operate makes LightGBM popular. The popularity of LightGBM is also attributed to its emphasis on the accuracy of outcomes.

## 2.10 Hyperparameters

A model's parameters serve as its representation in machine learning and deep learning. The best/optimal hyperparameters are chosen throughout a training phase, in contrast, so that learning algorithms can produce the best results. What exactly are these hyperparameters then? The user-defined parameters known as hyperparameters are those that are used to explicitly regulate the learning process, is the response.[8]

The word "hyper" in this context denotes top-level settings that are utilised to regulate the learning process. Before the learning algorithm starts training the model, the machine learning engineer chooses and sets the value of the hyperparameter. As a result, they are independent of the model and their values cannot be altered during training.

Several illustrations of hyperparameters used in machine learning

The learning rate for the K-Nearest Neighbor algorithm, which is used to train neural networks, is k.

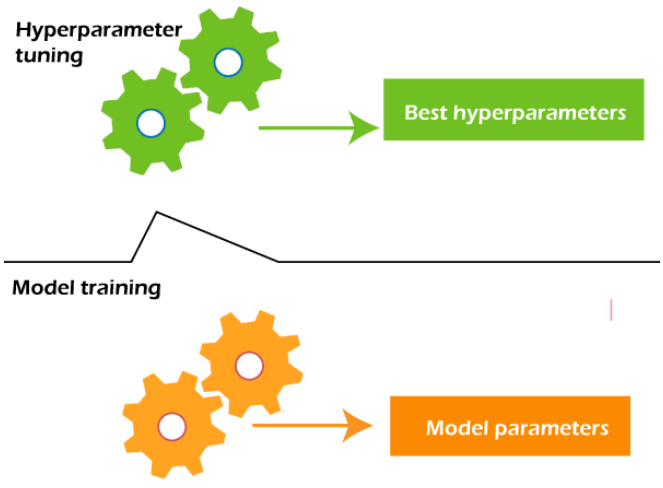
- Test-to-train ratio
- Sample Size
- Number of clusters in the Clustering
- Algorithm Number of epochs in the Decision Tree

### 2.10.1 Categories of Hyperparameters

Hyperparameters can be broadly split into the following two categories:

1) Hyperparameter for Optimization

Hyperparameter tuning, sometimes referred to as hyperparameter optimization, is the process of choosing the optimum hyperparameters to utilise. The model is optimised using optimization parameters.[9]




The following list of common optimization parameters includes

Learning Rate: The learning rate is a hyperparameter in optimization algorithms that regulates how much the model must alter each time its weights are updated in response to the predicted mistake. It is one of the most important parameters when creating a neural network, and it also controls how frequently model parameters are cross-checked. It can be difficult to choose the optimal learning rate since if it is too low, the training process may be slowed down. However, if the learning rate is too high, the model may not be optimised appropriately.

Batch Size: The training set is broken into several subsets, known as a batch, in order to speed up the learning process. Amount of Epochs: The entire cycle of training the machine learning model is referred to as an epoch.

Epoch stands for an incremental learning process. Each model has a different number of epochs, and some models have more than one epoch. The appropriate number of epochs is chosen by accounting for a validation error. As a validation error decreases, the number of epochs is raised. It is recommended to discontinue increasing the number of epochs if the reduction error does not improve over successive epochs.

2) Hyperparameter for Specific Models

Hyperparameters for certain models are those that are integral to the model's structure. Here are some of them: There are several hidden units, which are the layers of processors that make up a neural network's input and output units. Hidden units are a component of neural networks.The hyperparameter for the neural network's number of hidden units needs to be specified. It should fall somewhere between the input layer's and output layer's sizes. To be more precise, there should be as many hidden units as there are input and output layers combined.

The number of hidden units must be specified for complicated functions, but it must not overfit the model.

Number of Layers: Layers are the vertically stacked parts that make up a neural network. Input layers, hidden layers, and output layers are primarily present. The performance of a 3-layered neural network is superior to a 2-layered network. A Convolutional Neural network model performs better with more layers.

# 3 Methodology

The goal of the project is to implement ensemble methods like bagging, random forest, and boosting along with underlying algorithms for both classification and regression tasks. Once implemented their performance on a suitable dataset is to be compared, and the effect of parameters on the performance is to be analysed.

The methodology described in this section helps in performing all of the above tasks efficiently with less scope for errors.

The tasks involved in the methodology devised is same for classification and regression tasks and can be performed separately. The tasks are as follows:

**Find a suitable dataset**: The focus of this dissertation is more on evaluating ensemble methods, their underlying algorithms, and hyperparameters. Hence, simple and small datasets that involve little to no data analysis and pre-processing are considered suitable.

**Data Analysis and Pre-processing**: Perform data analysis to understand the structure of the data. For the datasets secured for this dissertation the only pre-processing step required was to convert categorical string-based columns into ordinals or dummy variables.

**Implement Decision Tree/Decision Stump**: The most fundamental elements for all ensemble methods are trees and stump. Decision Stumps can be generated by setting the height of a tree as 1. Hence, two classes for Decision Trees were implemented for regression and classification. These trees were created while considering compatibility with Random Forests.

**Evaluate Decision Tree/Decision Stump**: To ensure that the Decision Tree/Stumps work as intended, a rough jupyter notebook was created to verify proper functionality by using the datasets. As this is not within the scope of this project, these files will not be a part of the final submission.

**Implement Ensemble Methods**: Six different ensemble methods are to be implemented for this dissertation. Three for classification-based learning and three for regression-based learning.

- Bagging for Classification
- Bagging for Regression
- Random Forest for Classification
- Random Forest for Regression
- ADABoost for Classification
- Gradient Boost for Regression

**Hyperparameter Tuning:** There are various hyperparameters that can be tuned for the aforementioned ensemble methods. GridSearchCV class from the scikit-learn package is an excellent tool for finding best parameters. Apart from finding best parameters the computations performed by GridSearchCV can be stored and later used for analysing effects of a parameters on performance of ensemble methods by plotting learning curves.

**Compare Performance on Test Set:** Using the Best Parameters for the ensemble methods, compare the performance of methods separately for the test set from regression and classification data set.

**Plot Graphs to visualize effect of parameters on performance:** Utilizing the scores estimated by GridSearchCV in step 6, graphs can be plotted to visualize effects of parameters on performance.

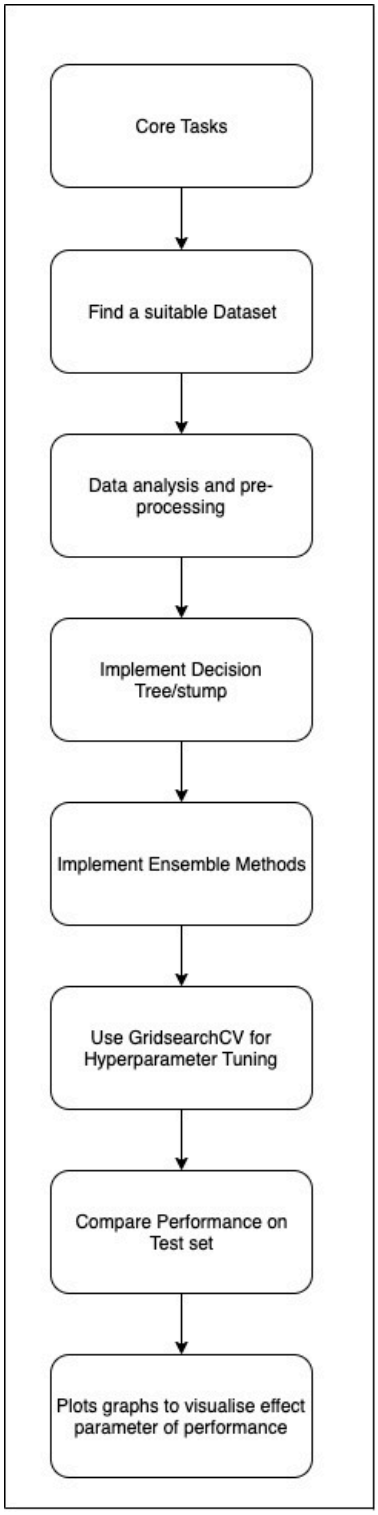



*Figure: 3-1 Methodology*

# 4 How to run the code

The code is written in such a way that is pretty easy to run it. The first step is to install the required python packages using pip. After that the relevant jupyter notebooks need to be accessed and executed depending on the purpose. Here is a short tutorial on doing so.

## 4.1 Installing dependencies

There are only four libraries utilized for this project. They can be easily installed by running the following command on the terminal.

**pip install numpy pandas matplotlib scikit-learn**

It is important to run this command in a terminal with the right virtual environment selected. The same virtual environment is required in the jupyter notebook while executing the code.

## 4.2 List of notebooks

There are four core notebook written for this project. They can serve as an entry point for going through the project. These files can be viewed in any sequence but the recommended flow of looking at the notebooks is as follows:

1. classification-tuning.ipynb: This file contains code that is used to import various ensemble methods, create a parameter grid, and estimate losses with GridSearchCV and gini impurity as a loss function. The outputs of GridSearchCV are stored in multiple files using pickle.
2. classification-results.ipynb: This file imports the results from the pickle file for tuning, analyses the results while visualizing them through plots.
3. regression-tuning.ipynb: This file contains code that is used to import various ensemble methods, create a parameter grid, and estimate losses with GridSearchCV and mean squared error as a loss function. The outputs of GridSearchCV are stored in multiple files using pickle.
4. regression-results.ipynb: This file imports the results from the pickle file for tuning, analyses the results while visualizing them through plots.

All the algorithms related to ensemble methods are implemented as .py libraries and are not required to be executed by themselves.

# 5 Data Analysis

Before applying any machine learning techniques, it is always a good idea to understand the underlying data. This section focusses on discussing multiple aspects related to the data in question like source, description, shape, missing values, etc. There are two datasets utilized for this project. The first dataset is the Iris Dataset which will be used to test ensemble methods written for classification tasks. The second dataset is the Servo Dataset which will be used to test ensemble methods written for regression tasks. Both of these datasets are retrieved from the online UCI Machine Learning Repository. These two datasets were chosen as they are simple, small, and do not require a lot of pre-processing. The focus of this project is to focus on implementing and evaluating ensemble methods, and these datasets allow to do so without the overhead of intensive data analysis and pre-processing.

The data analysis discussed in this section is based of the jupyter notebook, **data-analysis.ipynb** submitted alongside this dissertation.

## 5.1 Iris Dataset

It is one of the most popular datasets if not the most popular dataset utilized for evaluating classification-based machine learning algorithms. It is a multivariate dataset with 4 features, 1 label, and 150 records of data. The data set contains measurements of sepal and petal of an iris plant based on which a machine learning algorithm should classify it into one of the three predefined species: Iris-Setosa, Iris-Versicolour, and Iris Virginica.

### 5.1.1 Accessing the Dataset

This dataset can be downloaded from the ftp server which can be accessed from https://archive-beta.ics.uci.edu/dataset/53/iris. The two files required are **iris.data** and **iris.names**. The iris.data files contains comma separated values which can be accessed using pandas. The iris.names file contains the description of the dataset. The data can be read using the code shown in Figure 4-1 below: [10]

```
1  columns = ["sepal length (cm)","sepal width (cm)","petal length (cm)","petal width (cm)","class"]
2  df = pd.read_csv("./iris.data",names=columns)
3  df.head()
✓ 0.5s                                                                    Python
```

Figure 5-1 Importing Dataset

### 5.1.2 Dataset Statistics

| | |
|---|---|
| Number of attributes: | 4 |
| Number of records: | 150 |
| Missing Values: | 0 |
| Number of classes: | 3 |
| Non-Numeric Columns: | 1(class) |
| Number of Unique Labels: | 3 (Iris-Setosa, Iris Versicolour, and Iris Virginica) |
| Value Counts: | Iris-Setosa 50<br>Iris Versicolour 50<br>Iris Virginica 50 |

### 5.1.3 Encoding the Labels

As the **class** column of the DataFrame contains non numeric values representing the labels, this column needs to be encoded into numerical ordinals. To achieve this the LabelEncoder class from sklearn.preprocessing is used as shown below in Figure 4-2:

```
1 df["class"] = LabelEncoder().fit_transform(y=df["class"])
2 df["class"]
✓ 0.4s                                                    Python
```

Figure 5-2 Encoding labels

This assigns a value of 0 to Iris Setosa, 1 to Iris Versicolour and 2 to Iris Virginica.

This is the only preprocessing step that is required for the rest of this project, and has been carried out in each jupyter notebook related to classification.

## 5.2 Servo Dataset

This dataset has also been retrived from the UCI Machine Learning repository. This dataset can be used to evaluate ensemble methods implemented for regression. The data has 4 categorical features, 1 label, and 167 records. The dataset can be utilized to predict the rise time of a servomechanism with respect to two continuous gain settings and two discrete columns representing types of screw and motor.[11]

### 5.2.1 Accessing the Dataset

The dataset can be downloaded via the UCI machine learning repository's ftp server which can be accessed from the page https://archive-beta.ics.uci.edu/dataset/87/servo. There are two files which should be downloaded: servo.data, and servo.names. Similar to the iris dataset, the servo.data file contains comma separated values of data, and servo.names contains description of the data. There are no column names presented in the servo.data file and can be inferred from the servo.names file. The data can be imported as a pandas DataFrame using the code shown in Figure 4-3:

### 5.2.2 Dataset Statistics

```
1 columns = ["motor","screw","pgain","vgain","class"]
2 df = pd.read_csv("./servo.data",names=columns)
3 df.head()
✓ 0.5s                                                    Python
```

Figure 5-3 Importing servodataset

| | |
|---|---|
| Number of attributes: | 4 |
| Number of records: | 167 |
| Missing Values: | 0 |
| Number of classes: | 3 |
| Non-Numeric Columns: | 2(motor and screw) |

### 5.2.3 **Encoding Categorical Columns**

The motor and screw columns in the dataset represent categorical variables. These categorical columns can be encoded into multiple columns using OneHotEncoder class from sklearn.preprocessing library or get_dummies function from pandas. For this the project the get_dummies function is utilized as shown in

This is the only preprocessing step required to work with the ensemble methods.

```
1  df = pd.get_dummies(df,columns=["motor","screw"])
2  df.head()
```

✓ 0.6s Python

| | pgain | vgain | class | motor_A | motor_B | motor_C | motor_D | motor_E | screw_A | screw_B | screw_C | screw_D | screw_E |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 5 | 4 | 0.281251 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 6 | 5 | 0.506252 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 2 | 4 | 3 | 0.356251 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 3 | 2 | 5.500033 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 4 | 6 | 5 | 0.356251 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |

Figure 5-4Dummy Variables for Servo Dataset

# 6 Implementation

This section focuses on discussing the implementation details of the companion code that will be submitted alongside the dissertation. The code is organized into multiple files each of which serve a single purpose. The content of each file can be described as follows:

1. dtc.py: Decision Tree for Classification.
2. dtr.py: Decision Tree for Regression.
3. baggingclassifier.py: Bagging ensemble method for Classification.
4. baggingregressor.py: Bagging ensemble method for Regression.
5. rfclassifier.py: Random Forest ensemble method for Classification.
6. rfregressor.py: Random Forest ensemble method for Regression.
7. adaboost.py: ADABoost ensemble method for Classification.
8. gradientboost.py: Gradient Boosted ensemble method for Regression.
9. classification-tuning.ipynb: Evaluating ensemble methods for classification using GridSearchCV.
10. classification-results.ipynb: Visualizing results from GridSearchCV and plots of learning curves of ensemble methods for classification.
11. regression-tuning.ipynb: Evaluating ensemble methods for regression using GridSearchCV.
12. regression-results.ipynb: Visualizing results from GridSearchCV and plots of learning curves of ensemble methods for regression.
13. bagging-classifier.pkl: Pickle file that stores the GridSearchCV results for bagging classifier.
14. bagging-regressor.pkl: Pickle file that stores the GridSearchCV results for bagging regressor.
15. rf-classifier.pkl: Pickle file that stores the GridSearchCV results for random forest classifier.
16. rf-regressor.pkl: Pickle file that stores the GridSearchCV results for random forest regressor.
17. ada-classifier.pkl: Pickle file that stores the GridSearchCV results for adaboost ensemble method for classification
18. gbt-regressor.pkl: Pickle file that stores the GridSearchCV results for gradient boosted ensemble method for regression
19. data-analysis.ipynb: A simple data analysis of the dataset. This file was used in the previous section 4. Data Analysis.

Out of these files the two most important files are regression-results.ipynb and classification-results.ipynb. These two files analyze the results from GridSearchCV, compares training and test scores for various estimators, and plots learning curves with respect to different parameters. These files are recommended to be the starting point for the code.

## 6.1 Decision Trees

At the base of this project are Decision Trees. The Decision Trees are implemented as a class and share the same constructor for both classification and regression. They can be used as normal python libraries and are present in **dtc.py** and **dtr.py.**
Both of these classes have the same parameters which are explained below:

```
class DecisionTreeRegressor:
    def __init__(self, height=3, min_samples=0, p=0):
        """
            height is the height of the node in the decision tree
            If height is 1 then the tree will turn into a stump
            min_samples is the minimum number of samples required for a DecisionTree to split into sub trees
        """
        self.height = height
        self.min_samples = min_samples
        self.p = p
```

Figure 6.1Decision Tree Regressor

```
class DecisionTreeClassifier:
    def __init__(self, height=3, min_samples=0, p=0):
        """
            height is the height of the node in the decision tree
            If height is 1 then the tree will turn into a stump
            min_samples is the minimum number of samples required for a DecisionTree to split into sub trees
        """
        self.height = height
        self.min_samples = min_samples
        self.p = p
```

Figure 6-2Decision Tree Classifier

1. height: This is the max height for the Decision Tree.
2. min_samples: This is the minimum number of samples required in a node for splitting to occur
3. p: This is a special parameter to enable creation of trees for Random Forest. If set to zero all features are used to find optimal split at all nodes. For a positive value, p number of features are randomly sampled to find optimal split at each node.

The class actually represents a node rather than a tree. If the node is to be used as a leaf then it only has predictions and no child nodes. If splitting takes place then the node is branch with two child nodes which are instances of the same class. From parent to child the height parameter is reduced by 1.

Both these classes have a fit method. This is inspired from scikit-learn estimator conventions. The training dataset is passed to the fit method which initiates generating the tree. Gini Impurity and RSS are used as loss functions for classification and regression respectively.

```
def calcGiniImpurity(series: pd.Series):
    counts = series.value_counts()
    probs = counts/sum(counts)
    impurity = 1
    for pi in probs:
        impurity -= pi**2
    return impurity
```

Figure 6-3dtc.py Gini Impurity

```
def calcRSS(data: pd.Series):
    """
        Calculate RSS for a split in decision tree
    """
    return ((data - data.mean())**2).sum()
```

Figure 6-4dtr.py RSS

First, it makes few checks to see if splitting is required or the current node can be used as a leaf. The following checks are made for Decision Trees.

1. If the number of samples received by the constructor is less than minimum number of samples, then the node in question is converted into a leaf node.
2. If the height reaches 0 for the current node, then the node is converted into a leaf.
3. If all the labels are same for the current node, then the node is converted into a leaf.
4. If the loss for the current node is less than the total loss after splitting, then the current node is used as a leaf

```
def fit(self, X_train: pd.DataFrame, y_train: pd.Series):
    self.leaf = False
    # print(f"Samples {len(y_train)} , {self.min_samples}")
    if len(y_train) < self.min_samples or self.height == 0:
        self.leaf = True
        self.prediction = y_train.mean()
        # print("Prediction", self.prediction)
        return self
    if len(y_train.unique()) == 1:
        self.leaf = True
        self.prediction = y_train.unique()[0]
        # print("Made a leaf with prediction", self.prediction)
        return self
```

Figure 6-5Conversion to a leaf

For a leaf, the prediction is set in the following ways:

1. Mean of all labels for regression
2. Label with the highest probability for classification

```
def predict_proba(self, x):
    if self.leaf:
        return self.predict_proba
    if x[self.opt_col] > self.opt_cut:
        return self.left.predict(x)
    return self.right.predict(x)
```

Figure 6-6 Prediction probability - classification

The predict method returns the prediction for a row or set of rows of input values.

The Decision Tree classifier also supports outputting prediction probabilities through the predict_proba() method.

## 6.2 Bagging

The first ensemble method to consider is Bagging. BaggingRegressor and BaggingClassifier are both based on scikit learn's BaseEstimator class. This project utilizes GridSearchCV for finding optimal parameters and plotting learning curves. To use a custom estimator with GridSearchCV it should have some functions which can be easily applied by making our estimators inherit from the BaseEstimator class.

BaggingClassifier and BaggingRegressor are nearly the same with the only difference being in the predict function, where BaggingClassifier performs a vote from predictions from the individual trees and BaggingRegressor returns the mean.

Both of these classes have the same parameters:

1. B: Defines the number of bootstrapped datasets and hence, the number of estimators
2. h: max height for the decision trees in the forest

3. min_samples: to be passed to the individual decision trees in the forest.

```python
class BaggingRegressor(BaseEstimator):
    def __init__(self, B=100, h=3, min_samples=0):
        """
            B: number of Bootstrapped Trees to be Constructed
            h: max height of each tree
        """
        self.B = B
        self.h = h
        self.min_samples = min_samples

    def fit(self, X_train: pd.DataFrame, y_train: pd.Series):
        self.data = X_train.copy()
        features = X_train.columns
        label = "Y"
        self.data["Y"] = y_train
        datasets = [self.data.sample(len(self.data), replace=True)
                    for _ in range(self.B)]
        self.forest = [DecisionTreeRegressor(height=self.h, min_samples=self.min_samples).fit(
            df[features], df[label]) for df in datasets]
        return self

    def predict(self, X):
        if isinstance(X,pd.DataFrame):
            return [self.predict(X.iloc[i]) for i in range(len(X))]
        prediction = np.array([tree.predict(X) for tree in self.forest]).mean()
        return prediction
```

Figure 6-7Boosting Regression

```python
class RFClassifier(BaseEstimator):
    def __init__(self, B=100, h=3, min_samples=0, p=0):
        """
            B: number of Bootstrapped Trees to be Constructed
            h: max height of each tree
        """
        self.B = B
        self.h = h
        self.min_samples = min_samples
        self.p = p

    def fit(self, X_train: pd.DataFrame, y_train: pd.Series):
        if self.p == 0:
            self.p = int(np.floor(np.sqrt(len(X_train.columns))))
        self.data = X_train.copy()
        features = self.data.columns
        label="Y"
        self.data["Y"] = y_train
        datasets = [self.data.sample(len(self.data), replace=True)
                    for _ in range(self.B)]
        self.forest = [DecisionTreeClassifier(height=self.h, min_samples=self.min_samples, p=self.p).fit(
            df[features], df[label]) for df in datasets]
        return self

    def predict(self, X):
        if isinstance(X,pd.DataFrame):
            return [self.predict(X.iloc[i]) for i in range(len(X))]
        proba = self.predict_proba(X)
        return proba.keys()[proba.argmax()]

    def predict_proba(self, x):
        proba = pd.Series([tree.predict(x) for tree in self.forest])
        proba = proba.value_counts()
```

Figure 6-8Boosting Classifier

## 6.3 Random Forest

The classes for Random Forest are exactly the same as of bagging with only a single change. They sport a parameter p on top of the parameters found in Bagging.
This parameter is passed to the underlying Decision Trees which also have a parameter named p. The different in how the decision trees handle p vs how the Random Forest handles p is that if p is set to 0 in decision trees then all features are used at each and every node, while if p is set 0 in random forest, then the class sets it to either of the following values:
(Number of features)/3 for regression

1. (Number of features)/3 for regression
2. (Number of features)**0.5 for classification

```
class RFClassifier(BaseEstimator):
    def __init__(self, B=100, h=3, min_samples=0, p=0):
        """
            B: number of Bootstrapped Trees to be Constructed
            h: max height of each tree
        """
        self.B = B
        self.h = h
        self.min_samples = min_samples
        self.p = p

    def fit(self, X_train: pd.DataFrame, y_train: pd.Series):
        if self.p == 0:
            self.p = int(np.floor(np.sqrt(len(X_train.columns))))
        self.data = X_train.copy()
        features = self.data.columns
        label="Y"
        self.data["Y"] = y_train
        datasets = [self.data.sample(len(self.data), replace=True)
                    for _ in range(self.B)]
        self.forest = [DecisionTreeClassifier(height=self.h, min_samples=self.min_samples, p=self.p).fit(
            df[features], df[label]) for df in datasets]
        return self

    def predict(self, X):
        if isinstance(X,pd.DataFrame):
            return [self.predict(X.iloc[i]) for i in range(len(X))]
        proba = self.predict_proba(X)
        return proba.keys()[proba.argmax()]

    def predict_proba(self, x):
        proba = pd.Series([tree.predict(x) for tree in self.forest])
        proba = proba.value_counts()
```

Figure 6-9 RF Classifier(rf-classifier.py)

## 6.4 ADABoost

ADABoost is an example of a boosting algorithm written for classification-based tasks. It doesn't need a lot in the form of parameters as it only takes a single parameter B which corresponds to the number of weak learners to be generated. The say for each weak learner is automatically calculated and ADABoost performs prediction on the basis of voting.

```
class ADABoost(BaseEstimator):
    def __init__(self, B=100):
        self.B = B

    def fit(self, X_train: pd.DataFrame, y_train: pd.Series):
        self.df = X_train.copy()
        self.df["class"] = y_train
        self.forest = [(self.getTreeSayDF()) for _ in range(self.B)]
        return self

    def getTreeSayDF(self):
        l = len(self.df)
        weights = pd.Series(np.full(shape=(l), fill_value=1/l))
        weights.index = self.df.index
        tree = DecisionTreeClassifier(height=1).fit(
            self.df.iloc[:, :-1], self.df.iloc[:, -1])
        predictions = pd.Series(
            [tree.predict(self.df.iloc[i, :-1]) for i in range(len(self.df))])
        predictions.index = self.df.index
        correct = predictions == self.df.iloc[:, -1]
        acc = correct.sum()/len(predictions)
        err = 1 - acc
        say = 0.5 * np.log((acc/err))
        weights[~correct] *= np.exp(say)
        weights[correct] *= np.exp(-say)
        weights /= weights.sum()
        self.df = self.df.sample(len(self.df), weights=weights, replace=True)
        return tree, say

    def predict(self, X):
        if isinstance(X,pd.DataFrame):
            return [self.predict(X.iloc[i]) for i in range(len(X))]
        result = pd.DataFrame([(tree.predict(X), say)
                               for tree, say in self.forest])
        result.columns = ["Prediction", "Say"]
        return result.groupby(["Prediction"]).sum().idxmax()[0]
```

Figure 6-10ADABoost

## 6.5 Gradient Boosting

Gradient Boosting ensemble method is written for regression-based tasks. It takes in four parameters: B, h, lr, and min_samples. B,h, and min_samples have the same usage as the other estimators. lr represents the learning rate for prediction purposes.

The code uses an intermediate variable called residuals. This helps in updating residuals while using list comprehension to generate forest of trees.

```
class GBTreeRegressor(BaseEstimator):
    def __init__(self, B=100, h=3, lr=0.1):
        self.B = B
        self.h = h
        self.lr = lr

    def fit(self, X_train, y_train):
        tree = DecisionTreeRegressor(height=0).fit(X_train, y_train)
        self.residuals = y_train - y_train.mean()
        self.forest = [tree] + \
            [self.generateTree(X_train) for _ in range(self.B-1)]
        return self

    def generateTree(self, X_train):
        tree = DecisionTreeRegressor(
            height=self.h).fit(X_train, self.residuals)
        predictions = pd.Series([tree.predict(X_train.iloc[i])
                                 for i in range(len(self.residuals))])
        predictions.index = self.residuals.index
        self.residuals = self.residuals - predictions
        return tree

    def predict(self, X):
        if isinstance(X,pd.DataFrame):
            return [self.predict(X.iloc[i]) for i in range(len(X))]
        predictions = pd.Series([self.forest[i].predict(X)
                                 for i in range(self.B)])
        predictions[1:] *= self.lr
        return predictions.sum()
```

Figure 6-11 Gradient Boosting

## 6.6 Utilizing the estimators

All the ensemble methods and Decision Trees described above can be used in the same way.
Initialize the model using the constructor
Fit the training data using the fit method
Making predictions using the predict method

## 6.7 GridSearchCV

GridSearchCV is used in two files(regression-tuning.ipynb and classification-tuning.ipynb) to find best parameters for the estimators and to estimate losses for various training set proper and validation set carried out during cross validation.
The primary reason for using GridSearchCV rather than writing code from scratch to find best parameters and plot a proper learning curves of training and test data rather than training set proper and validation set was due to the n_jobs parameter found in GridSearchCV. If this parameter is set to -1 then it uses all the available cores in the computer resulting in much better speed of processing.
Various parameter grids were created for each ensemble methods and can be observed in regression-tuning and classification tuning notebooks.

# 7 Analysis and Results

This section is based on the results obtained in the jupyter notebooks: regression-results.ipynb and classification-results.ipynb.

## 7.1 Analyzing Results

The first task is to compare the performance of different emsemble methods on test set while using the best parameters using GridSearchCV. The estimation is actually performed in the notebooks regression-tuning.ipynb and classification-tuning.ipynb because the dataset was split into training and test set, and test set was only accessible in these notebook. The results can be summarized in the following table

| Ensemble Method | Task | Best Params | Best CrossVal Loss | Test Loss |
|---|---|---|---|---|
| **Bagging** | Regression | {'B': 50, 'h': 6, 'min_samples': 0} | 0.3984(MSE) | 0.205(MSE) |
| **Random Forest** | Regression | {'B': 100, 'h': 6, 'min_samples': 0, 'p': 8} | 0.3978(MSE) | 0.5918(MSE) |
| **Gradient Boost** | Regression | {'B': 50, 'h': 3, 'lr': 1, 'min_samples': 0} | 0.3723(MSE) | 0.18(MSE) |
| **Bagging** | Classification | {'B': 50, 'h': 3, 'min_samples': 0} | 0.0364(Error Rate) | 0.0263(Error Rate) |
| **Random Forest** | Classification | {'B': 50, 'h': 3, 'min_samples': 12, 'p': 2} | 0.0273(Error Rate) | 0.052(Error Rate) |
| **ADABoost** | Classification | {'B':50} | 0.135(Error Rate) | 0.316(Error Rate) |

From the above table we can see that for Regression on Servo Dataset the best result is achieved from Gradient Boost. The second-best result is from bagging and random forest performs the worst out of the three.
For classification on Iris Dataset the best result is achieved by Bagging, followed by AdaBoost and the worst result is achieved by Random Forest.
Random Forest performs the worst on both datasets.

## 7.2 Visualizing effect of parameters via Learning Curves

Learning Curves are usually drawn by plotting a parameter against a Loss function(Error Rate/MSE). They are drawn for both training and test data. As it was computationally expensive and time consuming to perform both GridSearchCV and estimate test losses, instead of using training set and test set, crossvalscores from GridSearchCV on trainingset proper and validation are used.

Initially, the learning curves were plotted by keeping all parameters except one constant. This led to an overload of figures being plotted as the number of combinations resulting from the parameter grid used for GridSearhCV were too many. The learning curves were overloaded with information, as can be seen from the example below:

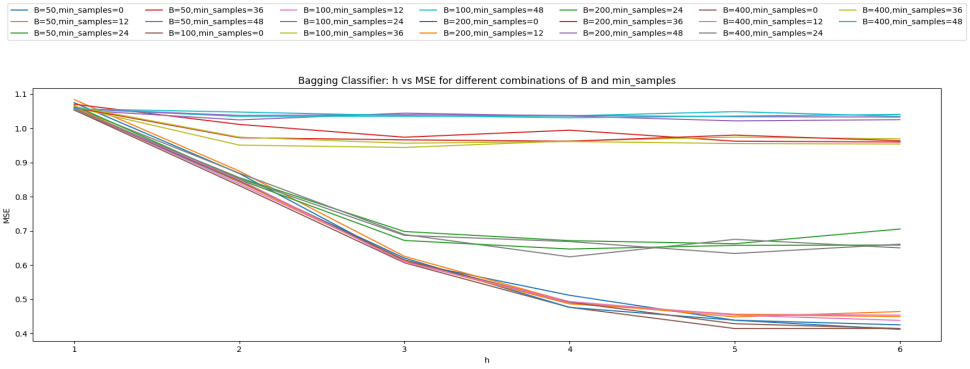



Figure 7.2.1 h vs MSE for different combinations of B and min_samples

The curves successfully show that the behavior observed by changing h for different combinations of B and min_sample is the same but with different amplitudes. With increasing h the mse decrease upto a point and then stabilizes. This is one of the best graphs in terms of presentations, but other graphs were outright ugly and non informative. To solve this issue another approach was taken in which the combination of best parameters was taken and one parameter was changed at a time observing changes in losses on training set proper and validation set. These results were not re-estimated rather extracted from stored results from gridsearchcv results using pandas as shown below:

```python
plt.figure(figsize=(20,6))
learning_curve_B = bagging[bagging["h"] ==6]
learning_curve_B = learning_curve_B[learning_curve_B["min_samples"] ==0]
learning_curve_B
plt.plot(learning_curve_B["B"],learning_curve_B["val_mse"],label="Validation MSE")
plt.plot(learning_curve_B["B"],learning_curve_B["train_pr_mse"],label="Training Set Proper MSE")
plt.xlabel("B")
plt.ylabel("MSE")
plt.title("Learning Curve of MSE vs B while keeping h and min samples as best parameters")
plt.legend()
```

Python

Figure 7.2.2Plotting Learning Curves

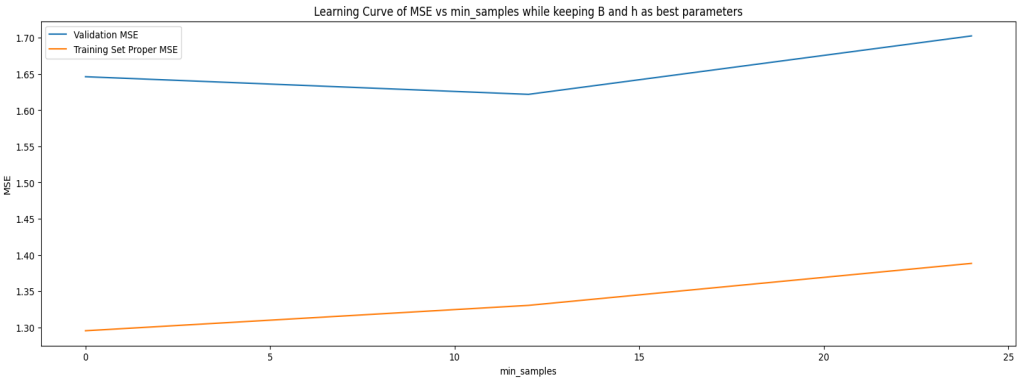



Figure 7.2.3 Plotting Learning Curve of MSE vs min_samples

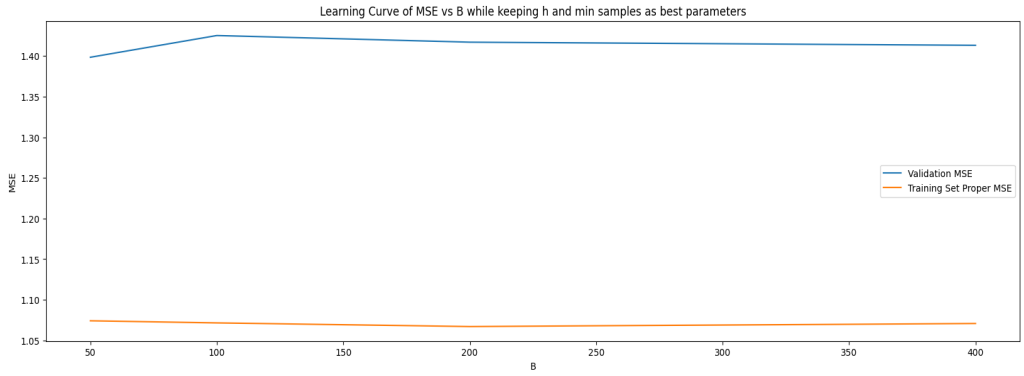



Figure 7.2.4 Learning curve of MSE vs B

## 7.3 Learning Curves: Bagging for Regression

From the above graph we can observe that the parameter B doesn't really account to a lot of changes in the MSEs. The validation error increases initially but stabilizes.
From the above graph we can observe that by increase h both the Training and Validation MSE decreases. They decrease rapidly and then kind of stabilizes.
From the above graph we can see that min_samples lead to increase in both validation and training MSEs. For validation the MSE goes down a bit at 12, but then increases. This drop in MSE gave min_samples=12 as the best parameter.

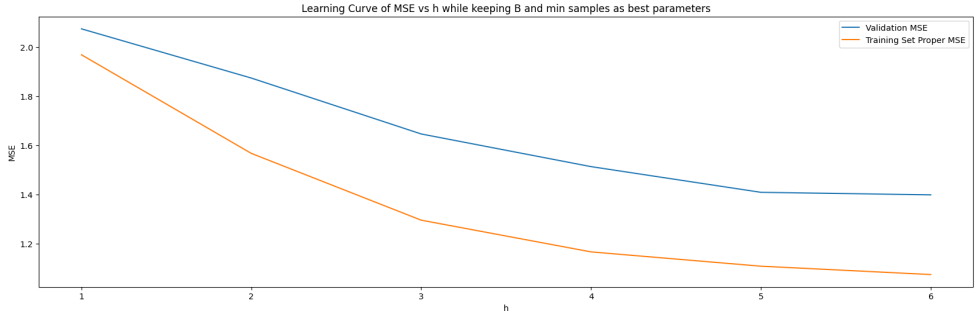



Figure 7.3.1 Learning Curve of MSE vs min_samples

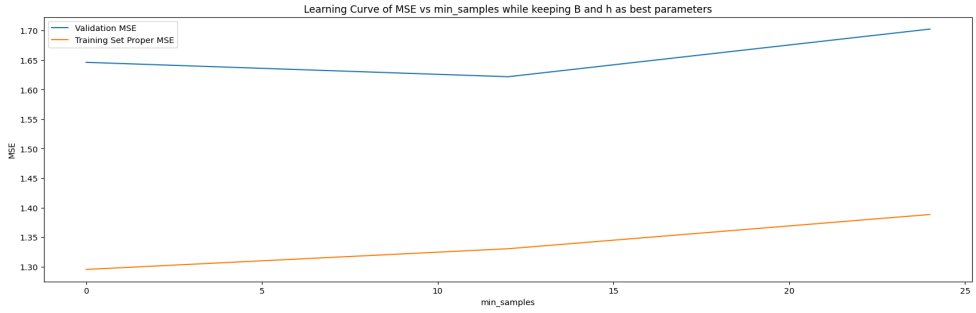
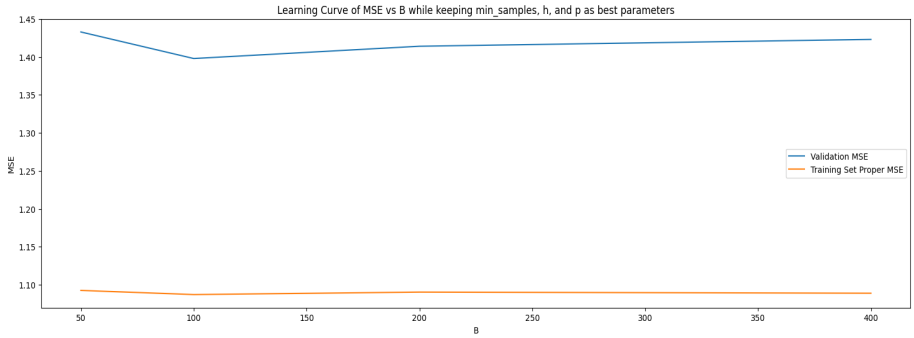
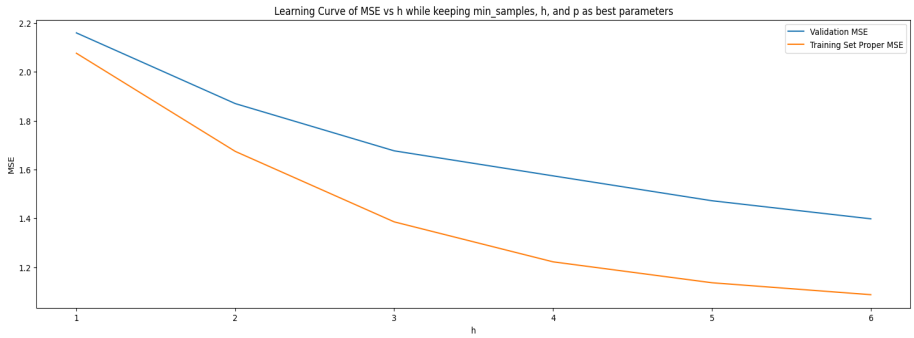



Figure 7.3.2 Learning Curve

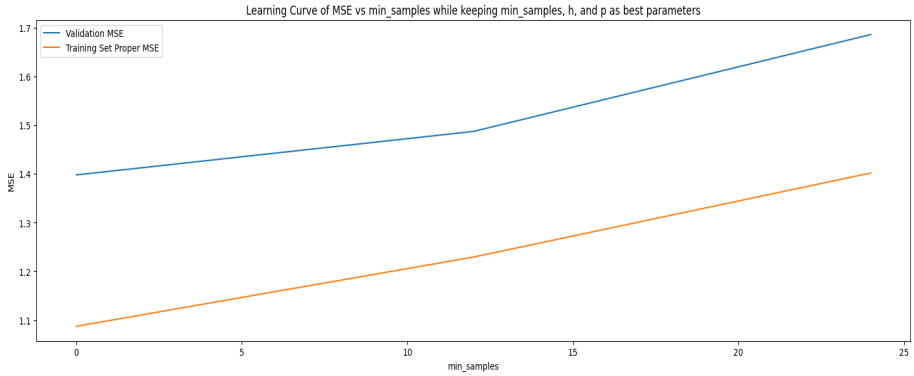
## 7.4 Learning Curves: Random Forest for Regression

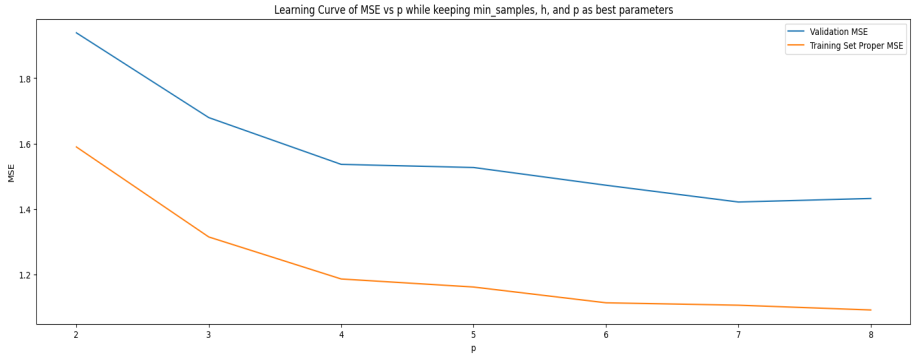



- The above graph is similar to the one in Bagging. B has little to no impact on improving the validation and training scores.



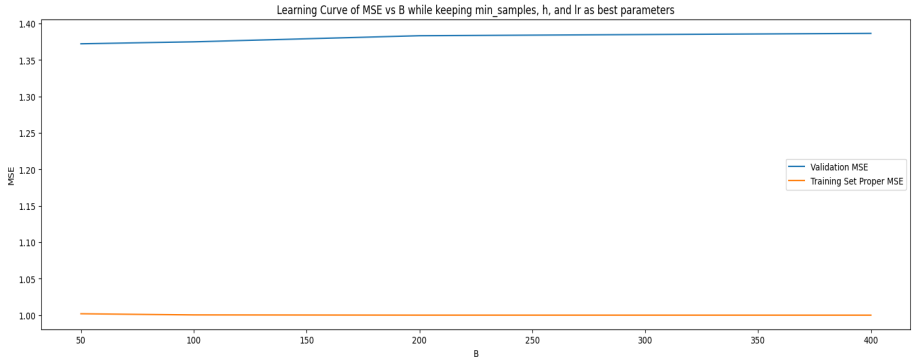

- This graph is also similar to the one in bagging. Increasing h decreases MSE which then stabilizes
- Here min_samples only cause increase in MSE



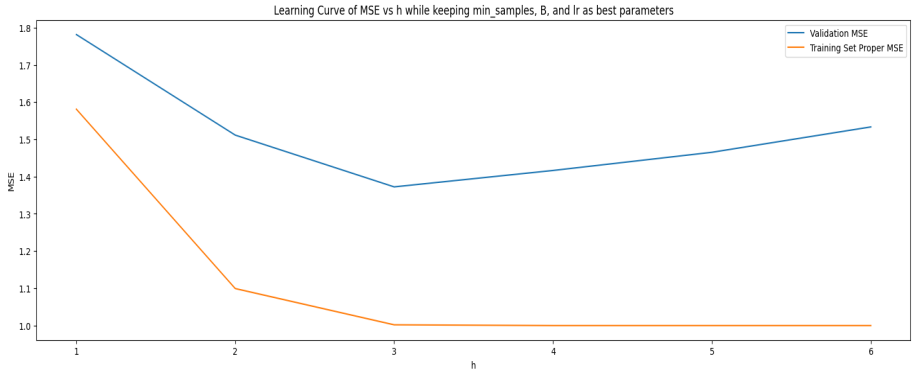



- Increasing the number of randomly sampled attributes decreases the MSE at first, but after p=7 we can see that the MSE is gradually.

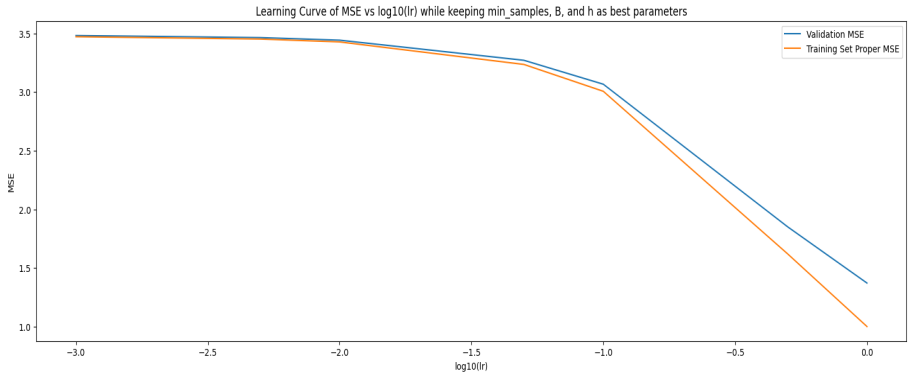
## 7.5 Learning Curve: Gradient Boost for Regression



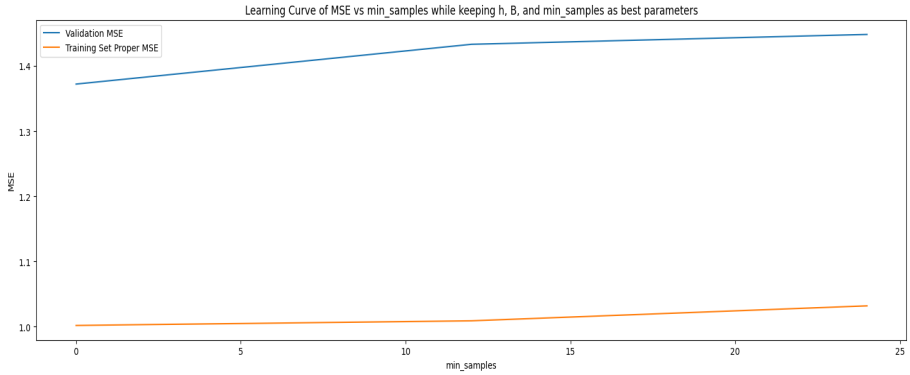

- Again, the value of B doesn't affect MSE on Servo Dataset by a lot.




- This is an interesting curve as we can see a case of overfitting. At h=3 we see a minimum for validation set, but after that the training set MSE decreases slightly but the validation set MSE keeps on increasing.

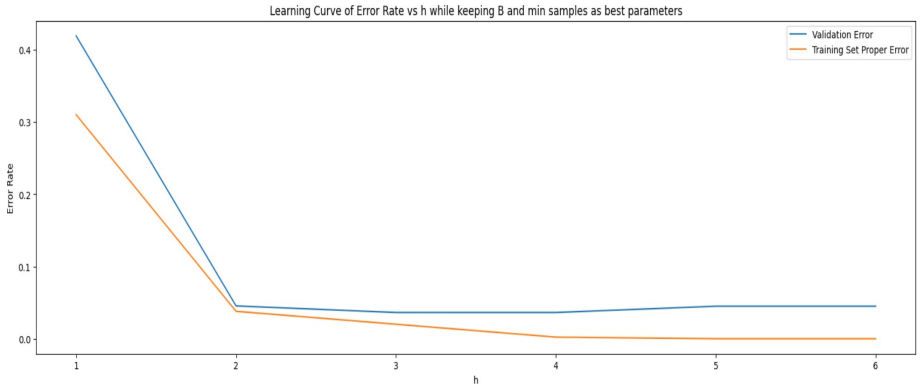



- This is a strange graph as in gradient boosting learning rate is present so that individual trees do no affect the final score by a lot. But, here the best results are achieved by putting lr=1, which means that all trees have the same say as the first tree.



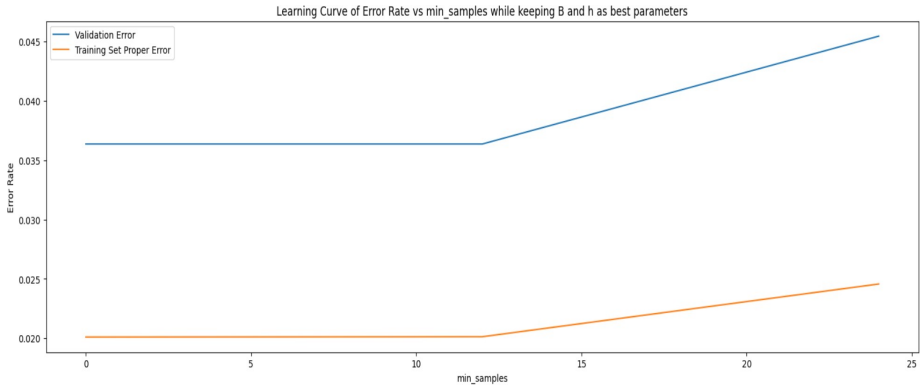

- We can see that min_samples again does not amount to a lot of
- change in the MSE.

## 7.6 Learning Curves: Bagging for classification



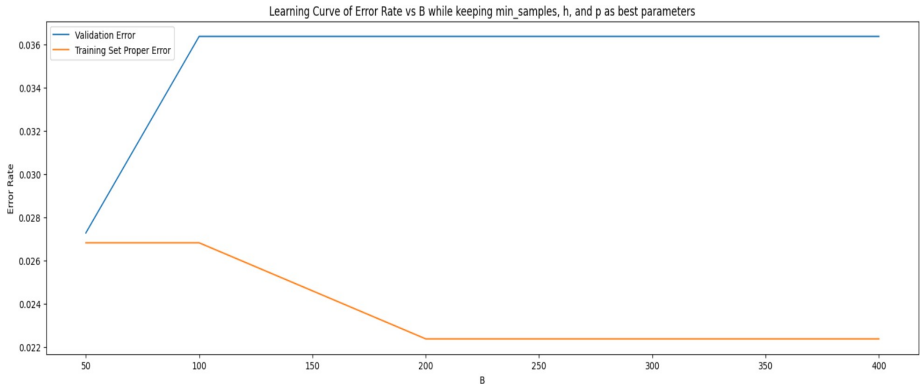
- From the above graph we can see that Training Set proper error and Validation Set decreases significantly between 1 and 2, and then Validation error stabilizes at 3, while the Training Set error keeps on slowly decreasing.

GridSearchCV uses Validation Error to find the best parameter and hence h=3 shows minimum values.

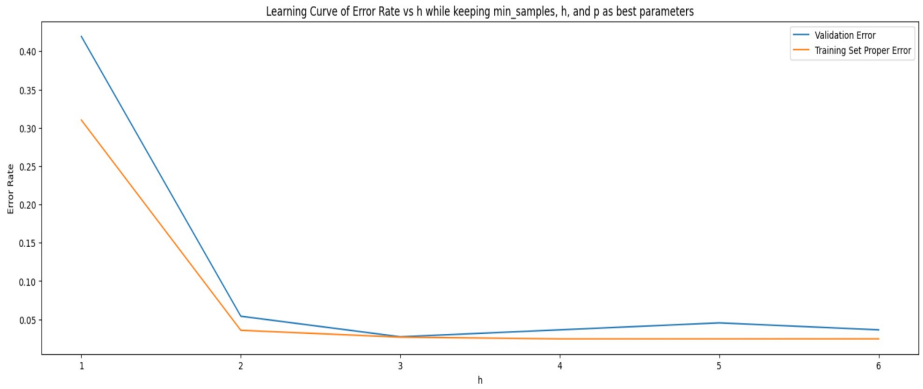



- From the above graph we can see that min_samples only leads to an increase in error. Around amin_sample of 12 both the training and validation errors go up.

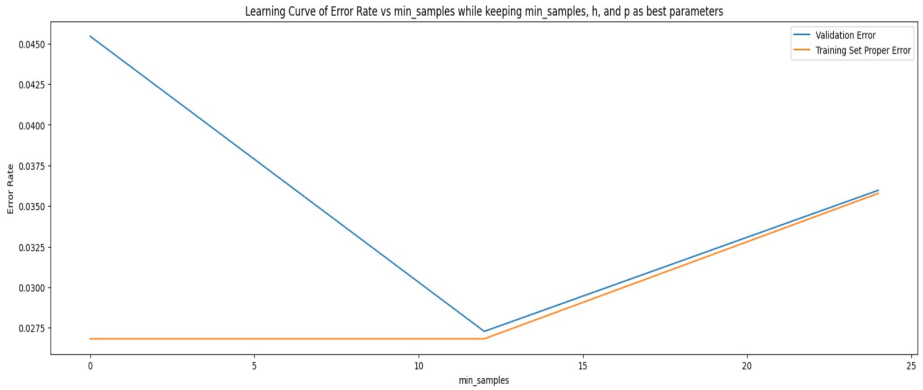
## 7.7 Learning Curves: Random Forest for Classification




- From the above graph we can see that after B=50 there are signs of overfitting as the training set errordecreases but the validation error increases.

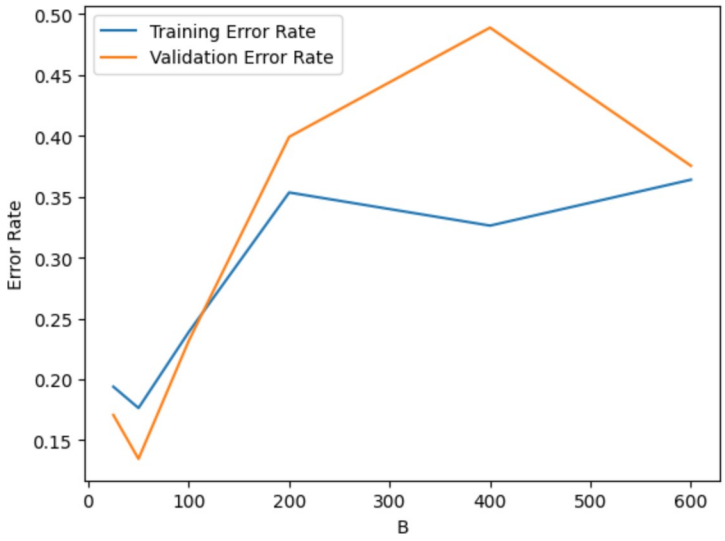



- From the above graph we can observe that both validation and training error decrease from h = 1 to h=2 and then stabilitizes with a minima at 3




- This is a change from Bagging. The validation error decreases with increase in min_samples upto avalue of 12 and then increases again.

## 7.8 Learning Curves: ADABoost For Classification




We can see that ADABoost gives best results at B=50.

# 8 Professional Issues: Using Data from public domain

Even as a masters student writing a dissertation there are various professional issues that are required to be tackled. There were various issues that were encountered, but most of them were related to choosing a dataset and its source.

In today's day and age data science has become one of the highest trending fields. With a lot of students and professionals trying to migrate to Data Science a lot of resources to learn and practice the practical aspects of Data Science have been made publicly available on the Internet. The most necessary resource to become a good data scientist is accessibility to good datasets to practice on. A lot of dummy and real-world datasets are made available on the Internet by various entities. A lot of times the uploaders of these datasets do not consider professional issues while uploading these datasets. The following two issues were observed while trying to find a suitable dataset for this project.

## 8.1 Ethical Considerations

Boston Housing Dataset is a popular dataset that is available on the public Internet. This dataset was the initial choice for applying ensemble methods for regression. Eventually it was realised that there was a feature named **B** in the dataset. This feature corresponded to the number of people with black skin colour. As the dataset involved predicting house prices, the presence of such feature felt very unethical. There were no experiments carried out to check how this feature affects the housing prices, but even if the feature led to increase or decrease in prices, including such a feature is strictly unacceptable. Hence, rather than redacting the feature, the complete dataset was replaced by the Servo motor dataset.

## 8.2 Citations

Another issue that was observed was to understand the source of the dataset and the process carried out to gather the data. The data science platform Kaggle contains thousands of datasets that are uploaded by its users. Most of these datasets do not mention how the dataset was obtained or if it was the work of someone else. Gather data and creating a dataset from it requires time and effort. It makes sense to attribute the original authors for their work.

Initially, the datasets were acquired from the scikit-learn python package and from Kaggle. Both of these sources did not outright mention the source and author of the dataset. Scikit-learn did cite the sources but it took some exploration to get to them. Finally, UCI machine learning repository was taken as the source for retrieving datasets. This website clearly mentioned the author and source of the dataset while also making it easier to cite them.

# 9 Self-Assessment

This project has served as a great learning experience. Prior to this project I lacked enough programming experience and knowledge about machine learning algorithms and practices. The biggest challenge faced during this dissertation was not knowing the next step or if I was going in the right direction. A lot of time was consumed in exploring things that did not give required results. But with persistence, constant learning, and feedback from the project supervisor things started becoming clearer.

My project turned out well, but I still have a lot of ideas that remain unimplemented. The primary reasons for this were lack of time, computational power, and code organization. The code was first organized into a single file and took few hours to execute. This was a big bottleneck as after making minor fixes I had to wait for few hours to analyse the result. This was really challenging at times because I did not know what results to expect. Later I split the code into multiple files by dividing tasks like tuning which were straightforward and time consuming into one file, and tasks like analysing the results and plotting which required frequent changes into another.

The result of this time constraint was having improper grids for parameter tuning. Some of the learning curves did not show a minima, but due to time constraints it was hard to update the grids. When the code was in a single Jupiter notebook it took multiple days to experiment with the plots. Most of these plots did not make it to the final draft as they were overloaded with information.

The best thing that I have learned during the implementation of the project is the importance of persistence, organization, managing time, and priorities. To me this project served as a good starting point for my future career.

# 10 References:


[1] "Data Science Random Trees - Enterprise AI Solutions." https://randomtrees.com/data-science (accessed Dec. 07, 2022).

[2] "Ensemble Technique | Ensemble Techniques in Machine Learning." https://www.analyticsvidhya.com/blog/2021/03/basic-ensemble-technique-in-machine-learning/ (accessed Dec. 08, 2022).

[3] "A Comprehensive Guide to Ensemble Learning Methods." https://www.projectpro.io/article/a-comprehensive-guide-to-ensemble-learning-methods/432 (accessed Dec. 08, 2022).

[4] "Ultimate Guide on How to Choose the Right Algorithm in Machine Learning." https://www.malicksarr.com/how-to-choose-the-right-algorithm-in-machine-learning/ (accessed Dec. 07, 2022).

[5] S. Manish, A. Prajakta, and D. Mali, "TECHNICAL REVIEW ON MACHINE LEARNING," *International Journal of Engineering Applied Sciences and Technology*, vol. 5, pp. 183–187, 2020, Accessed: Dec. 08, 2022. [Online]. Available: http://www.ijeast.com

[6] "Linear Regression in Machine learning - Javatpoint." https://www.javatpoint.com/linear-regression-in-machine-learning (accessed Dec. 12, 2022).

[7] "Gradient Boosting and XGBoost. Starting from where we ended, let's… | by Rohith Gandhi | HackerNoon.com | Medium." https://medium.com/hackernoon/gradient-boosting-and-xgboost-90862daa6c77 (accessed Dec. 12, 2022).

[8] "Hyperparameters in Machine Learning - Javatpoint." https://www.javatpoint.com/hyperparameters-in-machine-learning (accessed Dec. 08, 2022).

[9] "Difference between Model Parameter and Hyperparameter - Javatpoint." https://www.javatpoint.com/model-parameter-vs-hyperparameter (accessed Dec. 08, 2022).

[10] ''Iris- UC Irvine Machine Learning Repositroy." (accesed Dec. 1,2022)

[11] ''Servo – UC Irvine Machine Learning Repository." https://beta.ics.uci.edu/dataset/87/servo (accesed sed nov 29,2022).

[12] abhishek shauly 20, Decision Tree and Ensemble Algorithm in Machine Learningh, J

[13] J.A. Benediktsson, G.J. Briem, and J.R. Sveinsson. Boosting. Data from many remote sensing sources is bagged and classicalized using consensus. in Multiple Classier Systems, edited by J. Kittler and F. Roli. Pages 279–288 of Lecture Notes in Computer Science volume 2096, Second International Workshop, MCS 2001, Cambridge, UK. 2001 Springer-Verlag

[14] Udbhav Govindu, April 22,2022 , Regression vs classification (Dec 10 2022)