

TABLE OF CONTENT

S.NO	TITLE	PAGE.NO
1.	Abstract	
	Introduction	1
2.	System analysis	
	2.1 Existing system	3
	2.2 Proposed system	4
3.	System specification	
	3.1 Hardware specification	5
	3.2 Software specification	5
4.	Software description	
	4.1 Front End	6
	4.2 Back End	7
5.	Project description	
	5.1 Overview of the project	9
	5.2 Modules Description	9
	5.3 Data Flow Diagram	11
	5.4 Database Design	14
6.	System testing	
	6.1 System Testing	36
	6.2 Unit Testing	36
	6.3 Black box Testing	36
	6.4 White box Testing	36
	6.5 Integration Testing	37
	6.6 Validation Testing	37
7.	System implementation	38
8.	Conclusion And Future Enhancement	
	8.1 Conclusion	40
	8.2 Future enhancement	41
9.	Appendix	
	9.1 Source code	43
	9.2 Screenshot	54
10.	References	58

CHAPTER 1

ABSTRACT

Gold price prediction is vital in financial analysis and investment strategies due to its intrinsic value and role as a hedge against economic instability. This study introduces a robust methodology for forecasting gold prices using machine learning, focusing on the M5P model tree-based algorithm. Known for handling nonlinear relationships and offering interpretability, the M5P model is well-suited for modeling the complexities of gold price fluctuations. The approach involves collecting extensive historical data influenced by macroeconomic indicators, geopolitical events, market sentiment, and prior gold prices. Advanced feature engineering is applied for data preprocessing, ensuring critical factors are emphasized. The M5P algorithm then identifies patterns and relationships in the data, enabling accurate predictions. Model performance is evaluated using metrics like Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), ensuring robustness and generalization. Rigorous validation with out-of-sample data confirms the model's adaptability to varied market conditions. The results demonstrate the M5P model's ability to deliver reliable forecasts, providing stakeholders with essential insights for decision-making and risk management in the dynamic financial landscape.

INTRODUCTION

Gold price prediction is a crucial task in the realm of financial analysis and investment strategy formulation, owing to the metal's intrinsic value, historical significance, and its role as a hedge against economic uncertainty. In this study, we propose a comprehensive methodology for forecasting gold prices utilizing machine learning techniques, with a specific focus on the M5P model tree-based algorithm. The M5P algorithm, renowned for its adaptability to nonlinear relationships and interpretability akin to decision trees, emerges as a promising tool for modeling the intricate dynamics of gold price fluctuations.

Our methodology entails the aggregation of extensive historical data encompassing a spectrum of factors influencing gold prices, including macroeconomic indicators, geopolitical developments, market sentiment, and historical gold price movements. Leveraging feature engineering methodologies, the collected data undergoes meticulous preprocessing to extract salient features instrumental for predictive modeling. Subsequently, the M5P model is employed to harness the extracted features and discern underlying patterns and relationships, thereby enabling accurate gold price predictions.

Performance evaluation of the proposed framework is conducted rigorously, employing a suite of metrics to gauge prediction accuracy, robustness, and generalization capability. Furthermore, extensive validation exercises are undertaken utilizing out-of-sample data to ascertain the model's efficacy across diverse market conditions and temporal contexts. The empirical findings underscore the efficacy of the M5P model in furnishing reliable gold price forecasts, thus furnishing stakeholders with invaluable insights crucial for informed decision-making and risk management strategies in the financial domain.

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

- Reliance on Linear Models
- Limited Feature Representation
- Susceptibility to Data Noise
- Limited Adaptability to Market Dynamic

2.1.1

DRAWBACKS

The existing system has following disadvantages,

Reliance on Linear Models: Traditional approaches to gold price prediction often rely on linear regression models, which may oversimplify the inherently nonlinear relationships present in financial time series data. Linear models may struggle to capture complex patterns and dynamics, leading to suboptimal prediction accuracy, particularly in volatile market conditions.

Limited Feature Representation: Conventional models may utilize a limited set of features derived from economic indicators or historical price data, overlooking other influential factors such as geopolitical events, investor sentiment, and macroeconomic trends. This restricted feature representation may constrain the predictive capabilities of the model, resulting in incomplete and potentially biased forecasts.

Susceptibility to Data Noise: Conventional regression-based approaches are susceptible to noise and outliers in the training data, which can distort the model's learned relationships and compromise prediction accuracy. In financial markets characterized by high volatility and irregular patterns, the presence of noise poses a significant challenge to the reliability and robustness of predictive models.

Limited Adaptability to Market Dynamics: Linear regression models may exhibit limited adaptability to changing market dynamics and emerging trends, as they are based on fixed assumptions and linear relationships. In rapidly evolving financial markets, where sentiment, speculation, and external shocks play pivotal roles, the rigidity of conventional models may impede their ability to capture and respond to novel patterns and phenomena.

2.2 PROPOSED SYSTEM

Nonlinear Relationship Modeling: The proposed system leverages the M5P model tree-based algorithm, which excels in capturing nonlinear relationships and complex interactions inherent in financial time series data. Unlike traditional linear regression models, the M5P algorithm offers enhanced flexibility and adaptability to the nonlinear dynamics of gold price fluctuations, thereby improving prediction accuracy and reliability.

Interpretability and Explainability: The M5P model tree-based algorithm combines the interpretability of decision trees with the accuracy of regression models, providing stakeholders with clear and intuitive insights into the factors influencing gold price movements. By visualizing the hierarchical structure of the decision tree, users can readily interpret the model's predictions and understand the underlying drivers of gold price forecasts, enhancing transparency and facilitating informed decision-making.

Robust Feature Representation: The proposed system incorporates a diverse array of features encompassing economic indicators, geopolitical events, market sentiment, and historical gold price data. By leveraging comprehensive feature engineering techniques, the model captures the multifaceted nature of gold price determinants, resulting in a more nuanced and robust predictive framework capable of generating accurate forecasts across various market conditions.

Adaptability to Market Dynamics: The M5P model tree-based algorithm demonstrates superior adaptability to changing market dynamics and evolving trends, enabling it to effectively respond to novel patterns and phenomena in the financial markets. By dynamically adjusting the decision tree structure based on incoming data, the model can adapt to shifts in investor sentiment, geopolitical developments, and macroeconomic factors, enhancing prediction accuracy and resilience to market volatility.

CHAPTER 3

SYSTEM SPECIFICATION

3.1 HARDWARE CONFIGURATION

Processor	:	Intel Core i3
RAM	:	4 GB
Hard Disk	:	90 GB

3.2 SOFTWARE SPECIFICATION

Operating System	:	Windows 10
Middleware	:	ANACONDA(JUPYTER NOTEBOOK)
Back-End	:	Python

CHAPTER 4

SOFTWARE DESCRIPTION

4.1 ANACONDA

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system. The Anaconda distribution includes data-science packages suitable for Windows, Linux, and MacOS.

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, MacOS and Linux.

4.1.1 JUPYTER NOTEBOOK

"Jupyter" is a loose acronym meaning Julia, Python, and R. These programming languages were the first target languages of the Jupyter application. As a server-client application, the Jupyter Notebook App allows you to edit and run your notebooks via a web browser. The application can be executed on a PC without Internet access, or it can be installed on a remote server and it can access through the Internet.

A kernel is a program that runs and introspects the user's code. The Jupyter Notebook App has a kernel for Python code. "Notebook" or "Notebook documents" denote documents that contain both code and rich text elements, such as figures, links, equations. The mix of code and text elements, these documents are the ideal place to bring together an analysis description, and can be executed to perform the data analysis in real time. Jupyter Notebook contains two components such as web application and notebook documents.

A web application is a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output. Notebook documents is a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media

representations of objects. Structure of a notebook document The notebook consists of a sequence of cells. A cell is a multiline text input field, and its contents can be executed by using Shift-Enter, or by clicking either the “Play” button the toolbar, or Cell , Run in the menu bar. The execution behavior of a cell is determined by the cell’s type. There are three types of cells namely code cells, markdown cells, and raw cells. Every cell starts off being a code cell, but its type can be changed by using a drop-down on the toolbar.

Code cells

A code cell allows you to edit and write new code, with full syntax highlighting and tab completion. The programming language you use depends on the kernel, and the default kernel (IPython) runs Python code.

Markdown cells

Document the computational process in a literate way, alternating descriptive text with code, using rich text. In Python this is accomplished by marking up text with the Markdown language. The corresponding cells are called Markdown cells. The Markdown language provides a simple way to perform this text mark-up, to specify which parts of the text should be emphasized (italics), bold, form lists, etc.

Raw cells

Raw cells provide a place in which you can write output directly. Raw cells are not evaluated by the notebook. When passed through nbconvert, raw cells arrive in the destination format unmodified.

4.2 PYTHON

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Python Features

Python has few keywords, simple structure, and a clearly defined syntax. Python code is more clearly defined and visible to the eyes. Python's source code is fairly easy-to-maintaining. Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh. Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Extendable

It allows to add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

Databases

Python provides interfaces to all major commercial databases.

GUI Programming

Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Object-Oriented Approach

One of the key aspects of Python is its object-oriented approach. This basically means that Python recognizes the concept of class and object encapsulation thus allowing programs to be efficient in the long run.

Highly Dynamic and open source

Python is one of the most dynamic languages available in the industry today. There is no need to specify the type of the variable during coding, thus saving time and increasing efficiency.

Extensive Array of Libraries

Python comes inbuilt with many libraries that can be imported at any instance and be used in a specific program.

Microsoft Excel

Microsoft Excel is a spreadsheet developed by Microsoft for Windows, MacOS, Android and iOS. It features calculation, graphing tools, pivot tables and a macro programming language called Visual Basic for applications.

CHAPTER 5

PROJECT DESCRIPTION

5.1 OVERVIEW OF THE PROJECT

The project, "Gold Price Prediction Using Machine Learning," leverages the M5P model tree algorithm to forecast gold prices effectively. Gold, known for its intrinsic value and role as a hedge against economic uncertainty, requires advanced prediction methods due to the complexity of factors influencing its price, such as geopolitical events, market sentiment, and macroeconomic indicators. The M5P model combines decision tree interpretability with the precision of linear regression, making it a robust solution for capturing nonlinear relationships in gold price data.

This project involves multiple stages, including data collection, preprocessing, feature engineering, model training, and evaluation. It utilizes historical data enriched with relevant economic and geopolitical features, processed to ensure accuracy and consistency. The model's flexibility enables adaptation to changing market dynamics, while its interpretability provides clear insights for stakeholders. Rigorous performance evaluation and validation across diverse market conditions underscore its reliability for informing investment strategies, risk management, and financial decision-making. By addressing the limitations of traditional linear models, the M5P approach represents a significant advancement in financial forecasting.

5.2 MODULES DESCRIPTION

DATASET COLLECTION

A data set is a collection of data. Departmental store data has been used as the dataset for the proposed work. Sales data has Item Identifier, Item Fat, Item Visibility, Item Type, Outlet Type, Item MRP, Outlet Identifier, Item Weight, Outlet Size, Outlet Establishment Year, Outlet Location Type, and Item Outlet Sales.

HYPOTHESIS DEFINITION

This is a very important step to analyse any problem. The first and foremost step is to understand the problem statement. The idea is to find out the factors of a product that creates an impact on the sales of a product. A null hypothesis is a type of hypothesis used in statistics that proposes that no statistical significance exists in a set of given observations.

An alternative hypothesis is one that states there is a statistically significant relationship between two variables.

DATA EXPLORATION

Data exploration is an informative search used by data consumers to form true analysis from the information gathered. Data exploration is used to analyse the data and information from the data to form true analysis. After having a look at the dataset, certain information about the data was explored. Here the dataset is not unique while collecting the dataset. In this module, the uniqueness of the dataset can be created.

DATA CLEANING

In data cleaning module, is used to detect and correct the inaccurate dataset. It is used to remove the duplication of attributes. Data cleaning is used to correct the dirty data which contains incomplete or outdated data, and the improper parsing of record fields from disparate systems. It plays a significant part in building a model.

DATA MODELLING

In data modelling module, the machine learning algorithms were used to predict the Wave Direction. Linear regression and K-means algorithm were used to predict various kinds of waves. The user provides the ML algorithm with a dataset that includes desired inputs and outputs, and the algorithm finds a method to determine how to arrive at those results.

Linear regression algorithm is a supervised learning algorithm. It implements a statistical model when relationships between the independent variables and the dependent variable are almost linear, shows optimal results. This algorithm is used to show the direction of waves and its height prediction with increased accuracy rate.

K-means algorithm is an unsupervised learning algorithm. It deals with the correlations and relationships by analysing available data. This algorithm clusters the data and predict the value of the dataset point. The train dataset is taken and are clustered using the algorithm. The visualization of the clusters is plotted in the graph.

FEATURE ENGINEERING

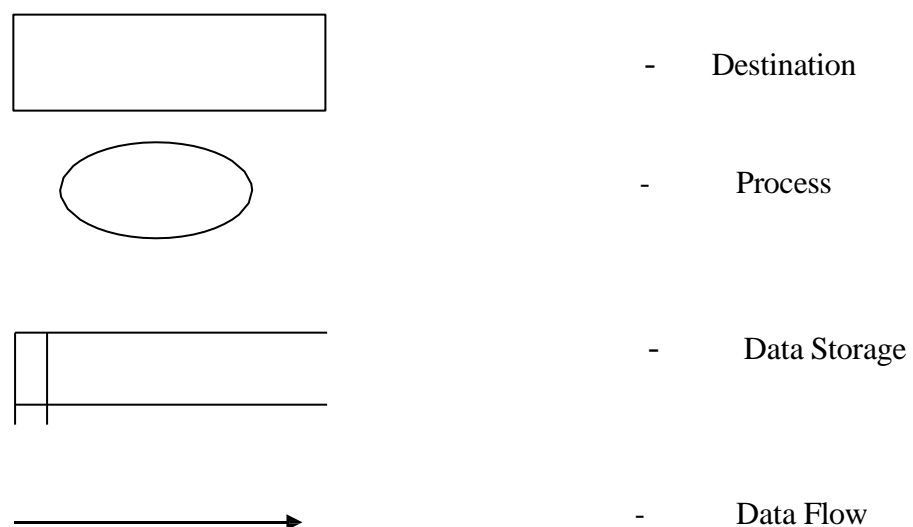
In the feature engineering module, the process of using the import data into machine learning algorithms to predict the accurate directions. A feature is an attribute or property shared by all the independent products on which the prediction is to be done. Any attribute could be a feature, it is useful to the model.

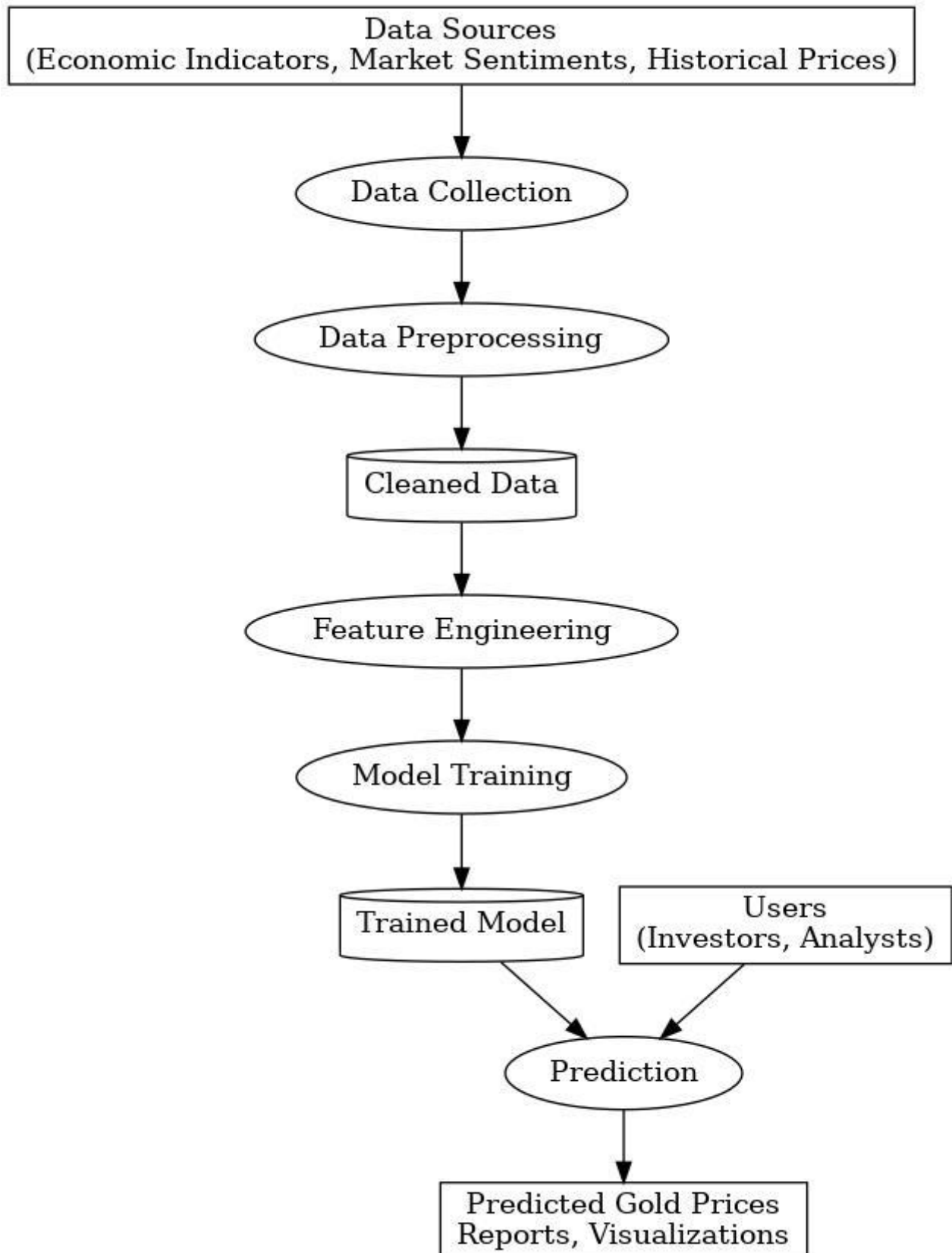
5.2 DATA FLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the “Flow” of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design). A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will stored.

DFD shows how the information moves through the system and how it’s modified by a series of transformations. It is a graphical technique that depicts information flow and the transformation that are applied as data moves from input to output .The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system

Basic DFD Notations





ENTITY RELATIONSHIP DIAGRAM

The relation upon the system is structured through a conceptual ER- Diagram, which not only specifies the existential entities but also the standard relations through which the system exists and the cardinalities that are necessary for the system state to continue. The Entity Relationship Diagram (ERD) depicts the relationship between the data objects. The ERD is the notation that is used to conduct the data modeling activity. The attributes of each data object noted in the ERD can be described as a data object description.

The set of primary components that are identified by the ERD are

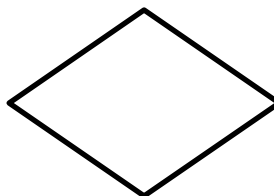
- i. Data object
- ii. Relationships
- iii. Attributes
- iv. Various types of indicators.

ER-DIAGRAM SYMBOL

Entity



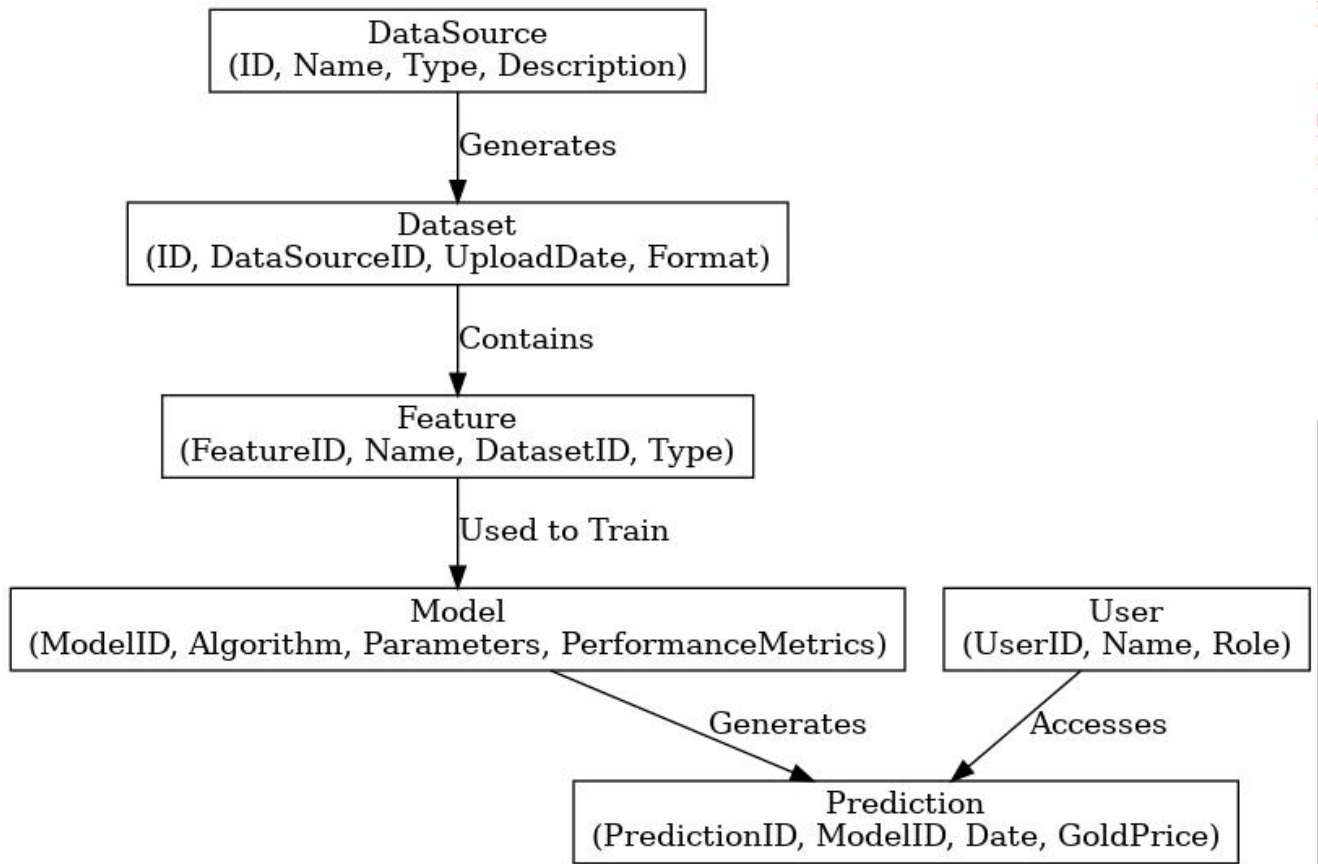
Relationship



Flow



ER DIAGRAM



5.3 DATABASE DESIGN

This phase contains the attributes of the dataset which are maintained in the database table. The dataset collection can be of two types namely train dataset and test dataset.

- System Design
- Input design
- Output design

SYSTEM DESIGN

The degree of interest in each concept has varied over the year, each has stood the test of time. Each provides the software designer with a foundation from which more sophisticated design methods can be applied. Fundamental design concepts provide the necessary framework for “getting it right”. During the design process the software requirements model is transformed into design models that describe the details of the data structures, system architecture, interface, and components. Each design product is reviewed for quality before moving to the next phase of software development.

INPUT DESIGN

The design of input focus on controlling the amount of dataset as input required, avoiding delay and keeping the process simple. The input is designed in such a way to provide security. Input design will consider the following steps:

- The dataset should be given as input.
- The dataset should be arranged.
- Methods for preparing input validations.

OUTPUT DESIGN

A quality output is one, which meets the requirement of the user and presents the information clearly. In output design, it is determined how the information is to be displayed for immediate need.

Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that the user will find the system can be used easily and effectively.

TREE BASED ALGORITHM

Tree-based algorithms are a class of machine learning algorithms that utilize decision tree structures to make predictions or decisions based on input features. These algorithms are widely used for both classification and regression tasks due to their simplicity, interpretability, and ability to handle nonlinear relationships in the data. Here's a description of tree-based algorithms:

Decision Trees: Decision trees are the fundamental building blocks of tree-based algorithms. They consist of nodes representing decision points and branches representing possible outcomes or decisions. At each node, the algorithm selects the best feature to split the data based on certain criteria, such as entropy or Gini impurity for classification tasks and mean squared error for regression tasks.

Random Forests: Random Forest is an ensemble learning technique that combines multiple decision trees to improve predictive performance and reduce overfitting. It constructs a multitude of decision trees using random subsets of the training data and random subsets of features at each node. The final prediction is made by averaging or voting the predictions of individual trees. Random Forests are robust to noise and outliers and can handle high-dimensional data effectively.

Gradient Boosting Machines (GBM): GBM is another ensemble learning method that builds a strong predictive model by sequentially adding weak learners (decision trees) to the ensemble. Unlike Random Forests, GBM trains trees sequentially, with each subsequent tree learning from the residuals (errors) of the previous trees. This allows GBM to focus on the most challenging examples, leading to improved predictive performance. Popular implementations of GBM include XGBoost, LightGBM, and CatBoost.

AdaBoost (Adaptive Boosting): AdaBoost is an ensemble learning algorithm that combines multiple weak learners (typically shallow decision trees) to create a strong learner. It sequentially trains weak learners on modified versions of the training data, with each subsequent learner focusing more on the misclassified examples from the previous iterations. AdaBoost assigns weights to each weak learner based on its performance, with higher weights given to more accurate learners.

M5P Model Tree: The M5P algorithm is a decision tree-based regression model that incorporates linear regression models at the leaf nodes to improve prediction accuracy. It recursively constructs a decision tree where each leaf node contains a linear regression model, allowing for piecewise linear approximations of the target function. M5P is particularly effective for capturing nonlinear relationships and interactions in the data while maintaining interpretability.

5.4 ALGORITHM

Machine Learning Algorithms Used:

- 1. MP5**
- 2. Gradient Booster Classifier**
- 3. Random Forest Classifier**
- 4. Decision Tree Classifier**

M5P

The M5P algorithm, short for M5 Model Tree with Linear Regression Models at the Leaves, is a decision tree-based regression algorithm that combines the interpretability of decision trees with the accuracy of linear regression models. It was proposed by Quinlan in 1992 as an extension of the M5 algorithm, primarily designed for regression tasks.

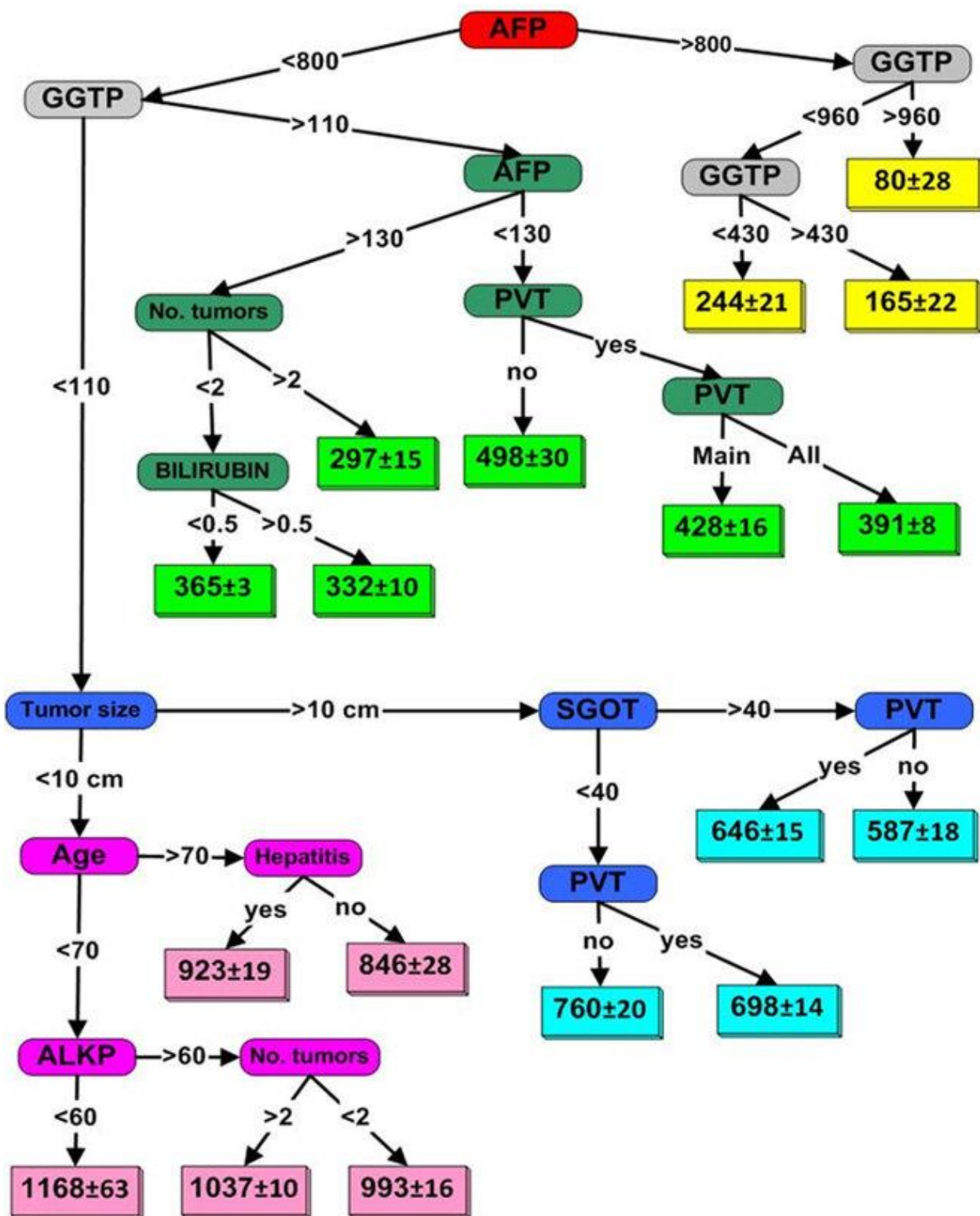
Here's how the M5P algorithm works:

Construction of the Decision Tree: Similar to traditional decision trees, the M5P algorithm recursively splits the training data based on the values of input features to create a tree-like structure of decision nodes. At each node, the algorithm selects the best feature and split point based on certain criteria, such as variance reduction or mean squared error reduction.

Linear Regression Models at the Leaves: Unlike standard decision trees, where each leaf node contains a constant value representing the predicted outcome, the M5P algorithm fits a linear regression model at each leaf node. This allows for more flexible and accurate predictions, as the linear regression models can capture the nuances and complexities of the relationships between features and the target variable.

Pruning and Model Simplification: After constructing the initial decision tree, the M5P algorithm performs pruning and model simplification to prevent overfitting and improve generalization performance. This involves collapsing branches of the tree and replacing them with linear regression models, reducing the complexity of the model while maintaining predictive accuracy.

Prediction: To make predictions for new instances, the M5P algorithm traverses the decision tree from the root node to a leaf node, following the path determined by the values of the input features. At the leaf node, it applies the corresponding linear regression model to compute the final prediction.



DECISION TREE CLASSIFIER

“Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too”.

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by **learning simple decision rules** inferred from prior data(training data).

In Decision Trees, for predicting a class label for a record we start from the **root** of the tree. We compare the values of the root attribute with the record’s attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

Types of Decision Trees

Types of decision trees are based on the type of target variable we have. It can be of two types:

1. **Categorical Variable Decision Tree:** Decision Tree which has a categorical target variable then it called a **Categorical variable decision tree**.
2. **Continuous Variable Decision Tree:** Decision Tree has a continuous target variable then it is called **Continuous Variable Decision Tree**.

Example:- Let’s say we have a problem to predict whether a customer will pay his renewal premium with an insurance company (yes/ no). Here we know that the income of customers is a significant variable but the insurance company does not have income details for all customers. Now, as we know this is an important variable, then we can build a decision tree to predict customer income based on occupation, product, and various other variables. In this case, we are predicting values for the continuous variables.

Important Terminology related to Decision Trees

1. **Root Node:** It represents the entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called the decision node.
4. **Leaf / Terminal Node:** Nodes do not split is called Leaf or Terminal node.
5. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning.

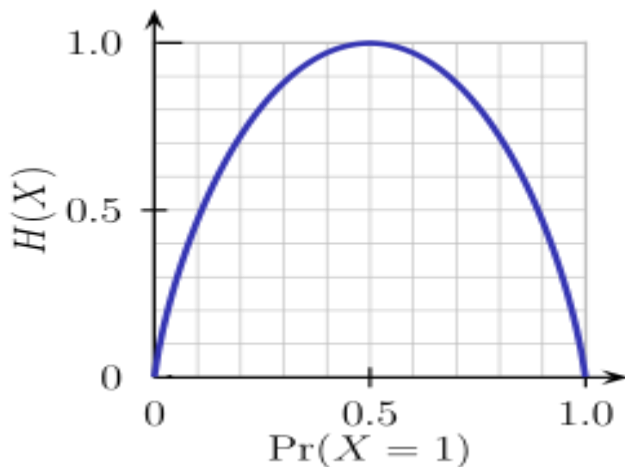
You can say the opposite process of splitting.

6. **Branch / Sub-Tree:** A subsection of the entire tree is called branch or sub-tree.

7. **Parent and Child Node:** A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.

1. Entropy

“Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information. Flipping a coin is an example of an action that provides information that is random”.



From the above graph, it is quite evident that the entropy $H(X)$ is zero when the probability is either 0 or 1. The Entropy is maximum when the probability is 0.5 because it projects perfect randomness in the data and there is no chance if perfectly determining the outcome.

ID3 follows the rule — A branch with an entropy of zero is a leaf node and A brach with entropy more than zero needs further splitting. Mathematically Entropy for 1 attribute is represented as:

$$E(S) = \sum_{i=1}^c -p_i \log_2 p_i$$

Where

$S \rightarrow$

Play Golf	
Yes	No
9	5



$$\begin{aligned} \text{Entropy(PlayGolf)} &= \text{Entropy}(5,9) \\ &= \text{Entropy}(0.36, 0.64) \\ &= -(0.36 \log_2 0.36) - (0.64 \log_2 0.64) \\ &= 0.94 \end{aligned}$$

Current state, and $P_i \rightarrow$ Probability of an event i of state S or Percentage of class i in a node of state S .

Mathematically Entropy for multiple attributes is represented as:

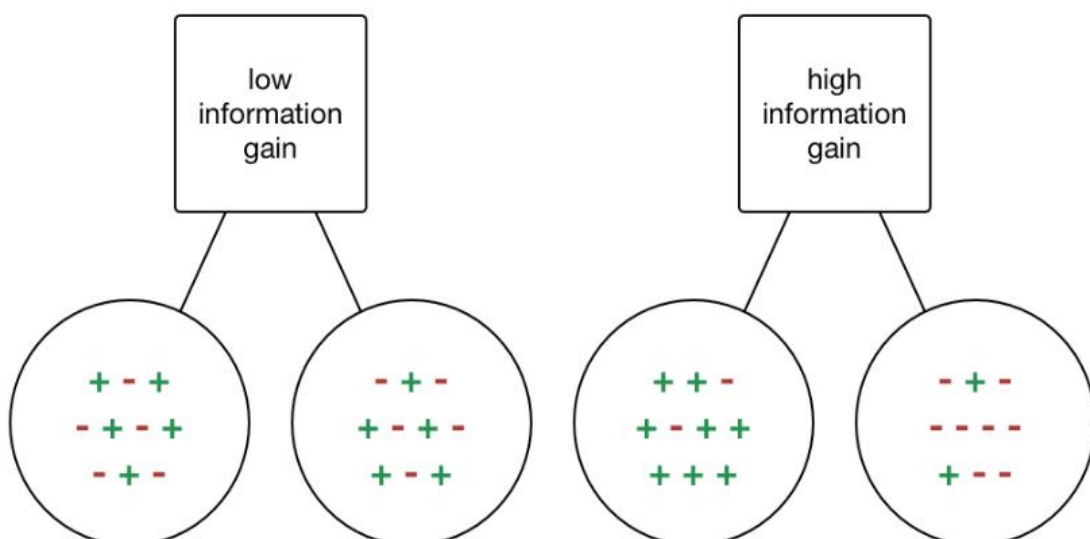
$$E(T, X) = \sum_{c \in X} P(c) E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14



$$\begin{aligned}
 E(\text{PlayGolf, Outlook}) &= P(\text{Sunny}) * E(3,2) + P(\text{Overcast}) * E(4,0) + P(\text{Rainy}) * E(2,3) \\
 &= (5/14) * 0.971 + (4/14) * 0.0 + (5/14) * 0.971 \\
 &= 0.693
 \end{aligned}$$

where $T \rightarrow$ Current state and $X \rightarrow$ Selected attribute **Information Gain**
 Information gain or **IG** is a statistical property that measures how well a given attribute separates the training examples according to their target classification. Constructing a decision tree is all about finding an attribute that returns the highest information gain and the smallest entropy.



Information Gain

Information gain is a decrease in entropy. It computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values. ID3 (Iterative Dichotomiser) decision tree algorithm uses information gain.

Mathematically, IG is represented as:

$$\text{Information Gain}(T,X) = \text{Entropy}(T) - \text{Entropy}(T, X)$$

$$\begin{aligned}\text{IG}(\text{PlayGolf}, \text{Outlook}) &= E(\text{PlayGolf}) - E(\text{PlayGolf}, \text{Outlook}) \\ &= 0.940 - 0.693 \\ &= 0.247\end{aligned}$$

In a much simpler way, we can conclude that:

$$\text{Information Gain} = \text{Entropy}(\text{before}) - \sum_{j=1}^K \text{Entropy}(j, \text{after})$$

Information Gain

Where “before” is the dataset before the split, K is the number of subsets generated by the split, and (j, after) is subset j after the split.

“You can understand the Gini index as a cost function used to evaluate splits in the dataset.

It is calculated by subtracting the sum of the squared probabilities of each class from one. It favors larger partitions and easy to implement whereas information gain favors smaller partitions with distinct values”.

$$Gini = 1 - \sum_{i=1}^C (p_i)^2$$

Gini Index

Gini Index works with the categorical target variable “Success” or “Failure”. It performs only Binary splits.

Higher the value of Gini index higher the homogeneity.

Steps to Calculate Gini index for a split

1. Calculate Gini for sub-nodes, using the above formula for success(p) and failure(q) (p^2+q^2).
2. Calculate the Gini index for split using the weighted Gini score of each node of that split.

CART (Classification and Regression Tree) uses the Gini index method to create split points.

Gain

ratio

“Information gain is biased towards choosing attributes with a large number of values as root nodes. It means it prefers the attribute with a large number of distinct values”.

C4.5, an improvement of ID3, uses Gain ratio which is a modification of Information gain that reduces its bias and is usually the best option. Gain ratio overcomes the problem with information gain by taking into account the number of branches that would result before making the split. It corrects information gain by taking the intrinsic information of a split into account.

Let us consider if we have a dataset that has users and their movie genre preferences based on variables like gender, group of age, rating, blah, blah. With the help of information gain, you split at ‘Gender’ (assuming it has the highest information gain) and now the variables ‘Group of Age’ and ‘Rating’ could be equally important and with the help of gain ratio, it will penalize a variable with more distinct values which will help us decide the split at the next level.

$$Gain\ Ratio = \frac{Information\ Gain}{SplitInfo} = \frac{Entropy\ (before) - \sum_{j=1}^K Entropy(j, after)}{\sum_{j=1}^K w_j \log_2 w_j}$$

Gain Ratio

Where “before” is the dataset before the split, K is the number of subsets generated by the split, and (j, after) is subset j after the split.

Reduction in Variance

“Reduction in variance is an algorithm used for continuous target variables (regression problems). This algorithm uses the standard formula of variance to choose the best split”. The split with lower variance is selected as the criteria to split the population:

$$Variance = \frac{\sum (X - \bar{X})^2}{n}$$

Above X-bar is the mean of the values, X is actual and n is the number of values.

Steps to calculate Variance:

1. Calculate variance for each node.
2. Calculate variance for each split as the weighted average of each node variance.

Chi-Square

“The acronym CHAID stands for *Chi*-squared Automatic Interaction Detector. It is one of the oldest tree classification methods. It finds out the statistical significance between the differences between sub-nodes and parent node. We measure it by the sum of squares of standardized differences between observed and expected frequencies of the target variable”.

It works with the categorical target variable “Success” or “Failure”. It can perform two or more splits. Higher the value of Chi-Square higher the statistical significance of differences between sub-node and Parent node.

It generates a tree called CHAID (Chi-square Automatic Interaction Detector).

Mathematically, Chi-squared is represented as:

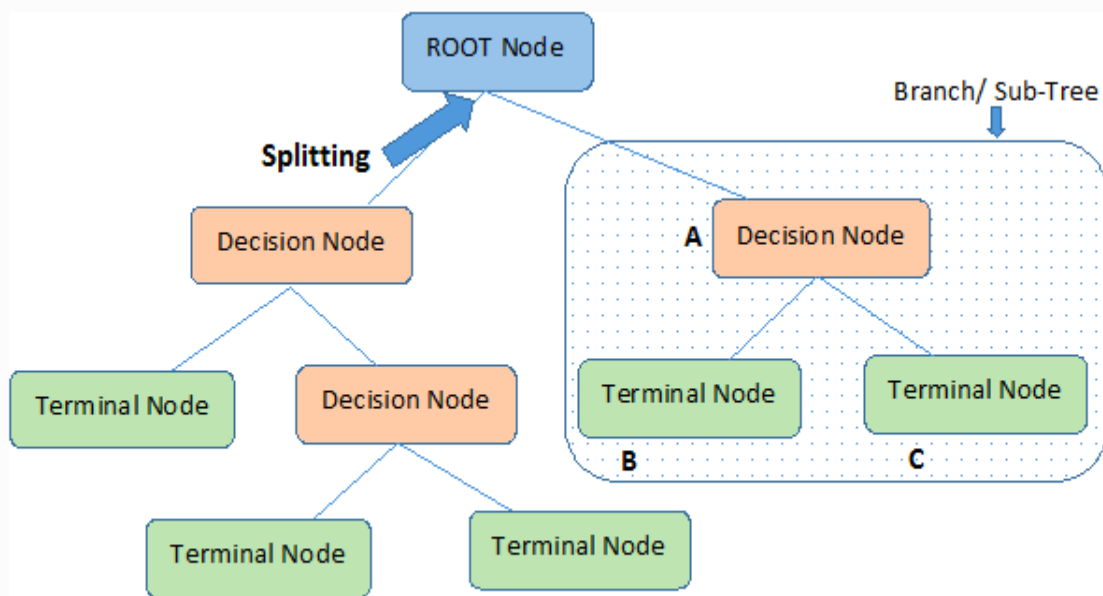
$$\chi^2 = \sum \frac{(O-E)^2}{E}$$

Where:

χ^2 = Chi Square obtained
 \sum = the sum of
 O = observed score
 E = expected score

Steps to Calculate Chi-square for a split:

1. Calculate Chi-square for an individual node by calculating the deviation for Success and Failure both
2. Calculated Chi-square of Split using Sum of all Chi-square of success and Failure of each node of the split



Note:- A is parent node of B and C.

Decision trees classify the examples by sorting them down the tree from the root to some leaf/terminal node, with the leaf/terminal node providing the classification of the example. Each node in the tree acts as a test case for some attribute, and each edge descending from the node corresponds to the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new node.

Assumptions while creating Decision Tree

Below are some of the assumptions we make while using Decision tree:

- In the beginning, the whole training set is considered as the **root**.
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- Records are **distributed recursively** on the basis of attribute values.
- Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

Decision Trees follow **Sum of Product (SOP)** representation. The Sum of product (SOP) is also known as **Disjunctive Normal Form**. For a class, every branch from the root of the tree to a leaf node having the same class is conjunction (product) of values, different branches ending in that class form a disjunction (sum).

The primary challenge in the decision tree implementation is to identify which attributes do we need to consider as the root node and each level. Handling this is known as the attributes selection. We have different attributes selection measures to identify the attribute which can be considered as the root node at each level.

How do Decision Trees work?

“The decision of making strategic splits heavily affects a tree’s accuracy. The decision criteria are different for classification and regression trees”.

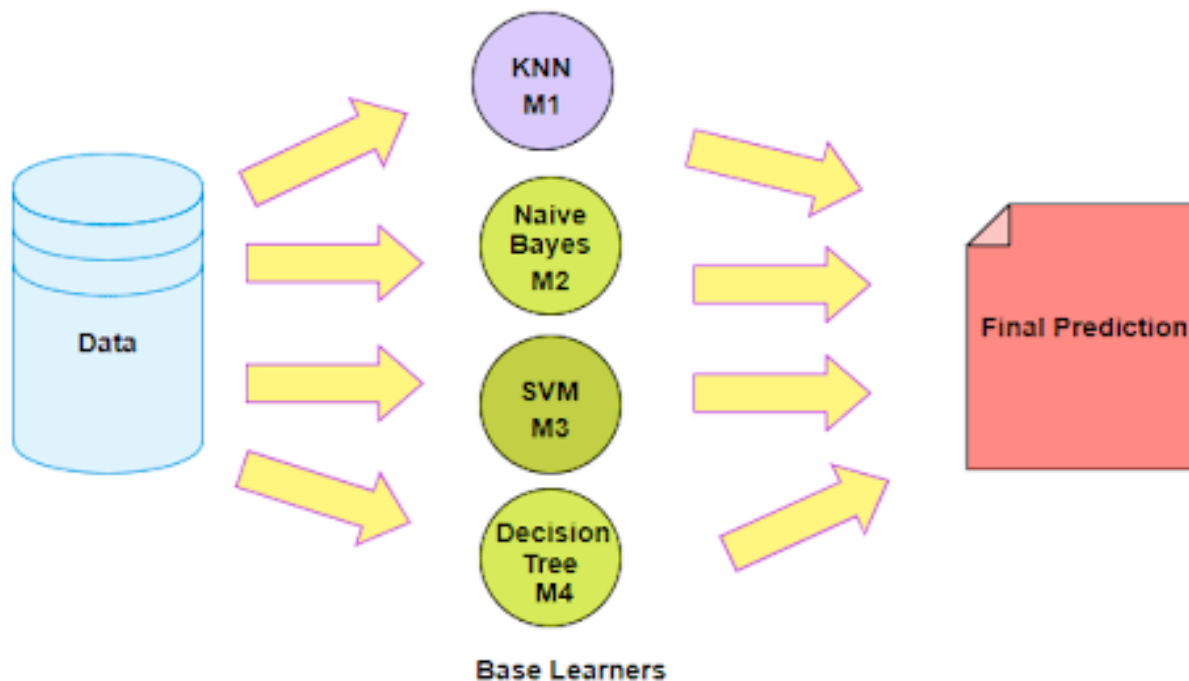
Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that the purity of the node increases with respect to the target variable. The decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

RANDOM FOREST

Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because of its simplicity and diversity (it can be used for both classification and regression tasks). In this post we'll learn how the random forest algorithm works, how it differs from other algorithms and how to use it.

Ensemble Learning

An Ensemble method is a technique that **combines the predictions from multiple machine learning algorithms** together to make more accurate predictions than any individual model. A model comprised of many models is called an **Ensemble model**.



Types of Ensemble Learning:

1. Boosting.
2. Bootstrap Aggregation (Bagging)

1. Boosting

Boosting refers to a group of algorithms that utilize weighted averages to make weak learners into stronger learners. Boosting is all about “teamwork”. Each model that runs, dictates what features the next model will focus on.

In **boosting** as the name suggests, one is learning from other which in turn **boosts** the learning.

2. Bootstrap Aggregation (Bagging)

Bootstrap refers to **random sampling with replacement**. Bootstrap allows us to better **understand the bias and the variance** with the dataset. Bootstrap involves random sampling of

small subset of data from the dataset.

It is a general procedure that can be used to **reduce the variance** for those **algorithm that have high variance, typically decision trees**. Bagging makes each model run independently and then **aggregates the outputs at the end without preference to any model**.

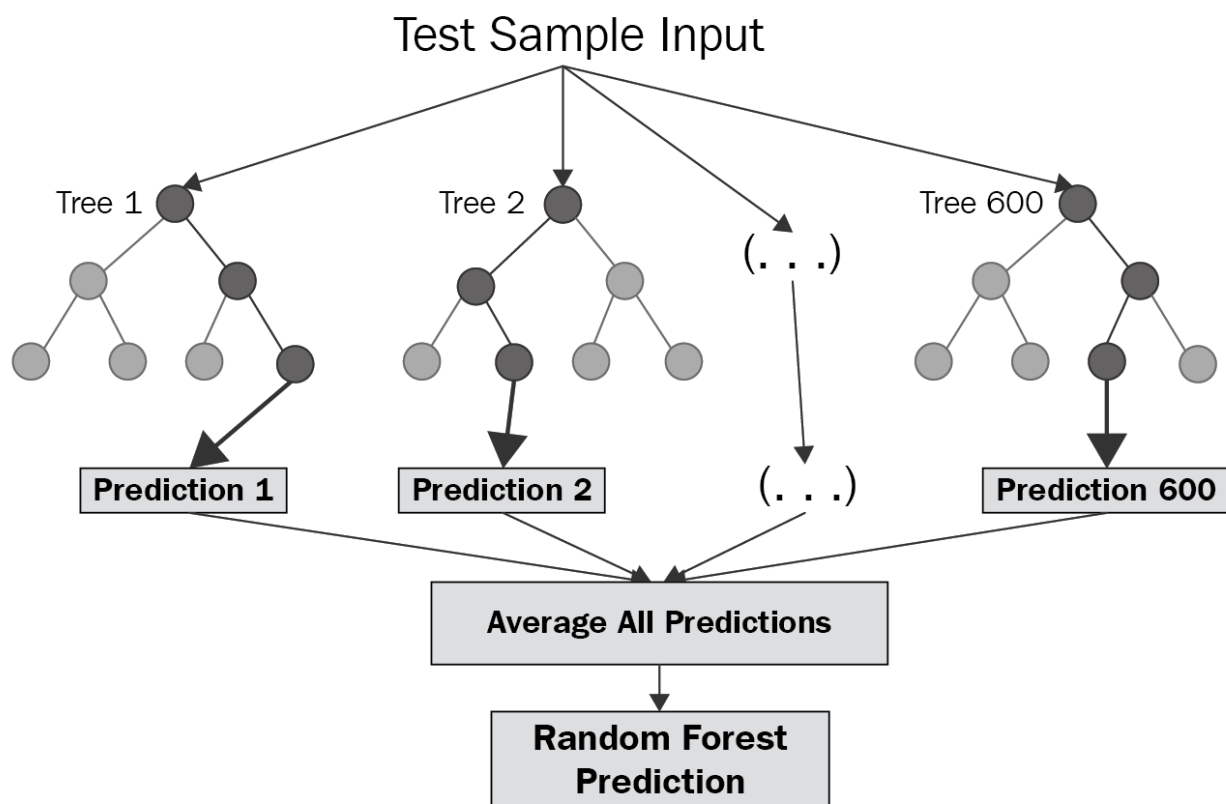
Problems with Decision Trees

Decision trees are sensitive to the specific data on which they are trained. If the training data is changed the resulting decision tree can be quite different and in turn the **predictions can be quite different**.

Also Decision trees are **computationally expensive to train**, carry a big risk of **overfitting**, and tend to find local optima because they can't go back after they have made a split.

To address these weaknesses, we turn to Random Forest :) which illustrates the power of combining many decision trees into one model.

Random Forest



Random forest is a **Supervised Learning algorithm** which uses ensemble learning method for **classification and regression**.

Random forest baggingnot a boosting random forests

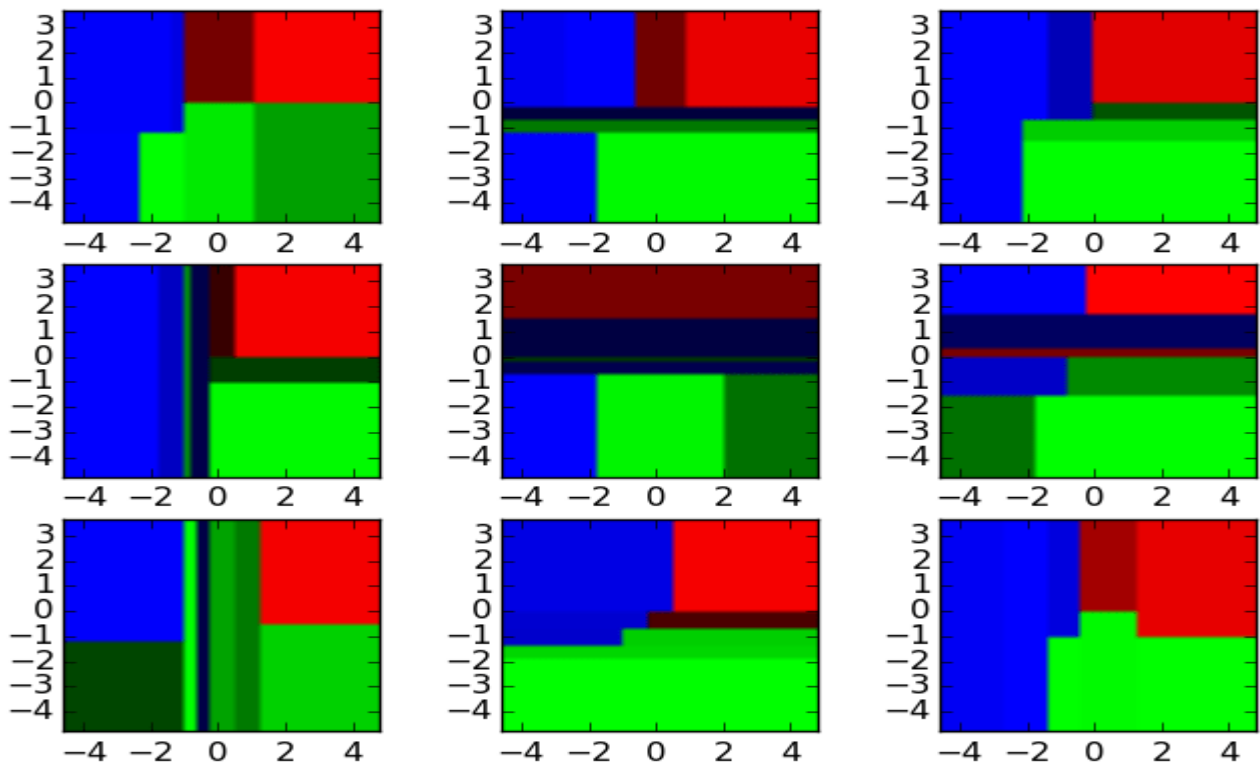
It operates by constructing a multitude of decision trees at training time and outputting the class that is the **mode** of the **classes (classification)** or **mean prediction (regression)** of the individual trees.

A random forest is a meta-estimator (i.e. it combines the result of multiple predictions) which **aggregates many decision trees**, with some helpful modifications:

1. The number of features that can be split on at each node is limited to some percentage of the total (which is known as the **hyperparameter**). This ensures that the ensemble model **does not rely too heavily on any individual feature**, and makes **fair use of all potentially predictive features**.
2. Each tree draws a random sample from the original data set when generating its splits, adding a further element of randomness that prevents **overfitting**.

The above modifications help prevent the trees from being too highly correlated.

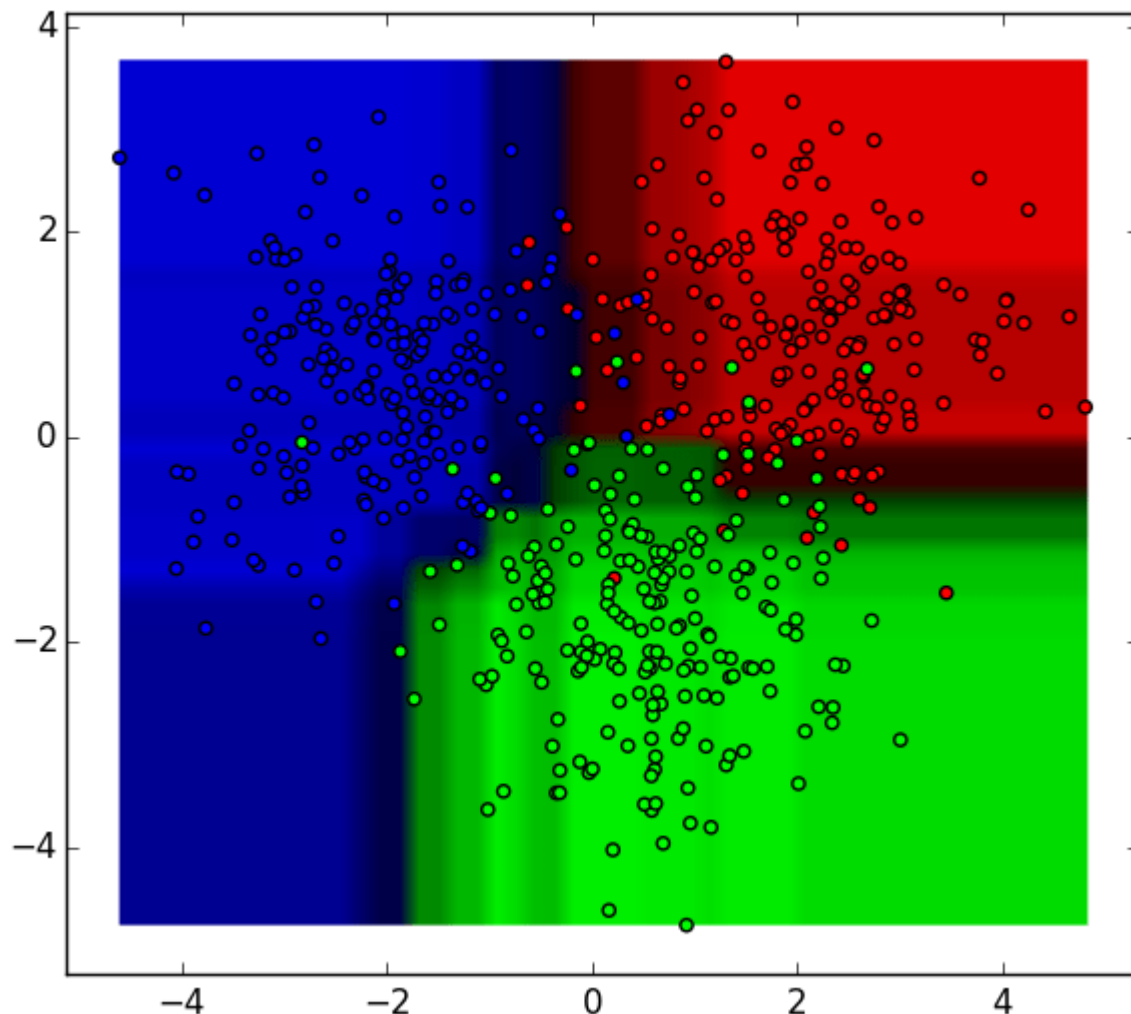
For Example, See these nine decision tree classifiers below :



These decision tree classifiers can be aggregated into a random forest ensemble which **combines their input**. Think of the horizontal and vertical axes of the above decision tree outputs as features x_1 and x_2 . At certain values of each feature, the decision tree outputs a classification of “blue”, “green”, “red”, etc.

These above **results are aggregated**, through model votes or averaging, into a single ensemble model that ends up outperforming any individual decision tree’s output.

The aggregated result for the nine decision tree classifiers is shown below :



Feature and Advantages of Random Forest :

1. It is one of the most accurate learning algorithms available. For many data sets, it produces a **highly accurate classifier**.
2. It runs efficiently on large databases.
3. It can **handle thousands of input variables** without variable deletion.

4. It gives estimates of what variables that are important in the classification.
5. It generates an internal **unbiased estimate of the generalization error** as the forest building progresses.
6. It has an **effective method for estimating missing data** and maintains accuracy when a large proportion of the data are missing.

Disadvantages of Random Forest :

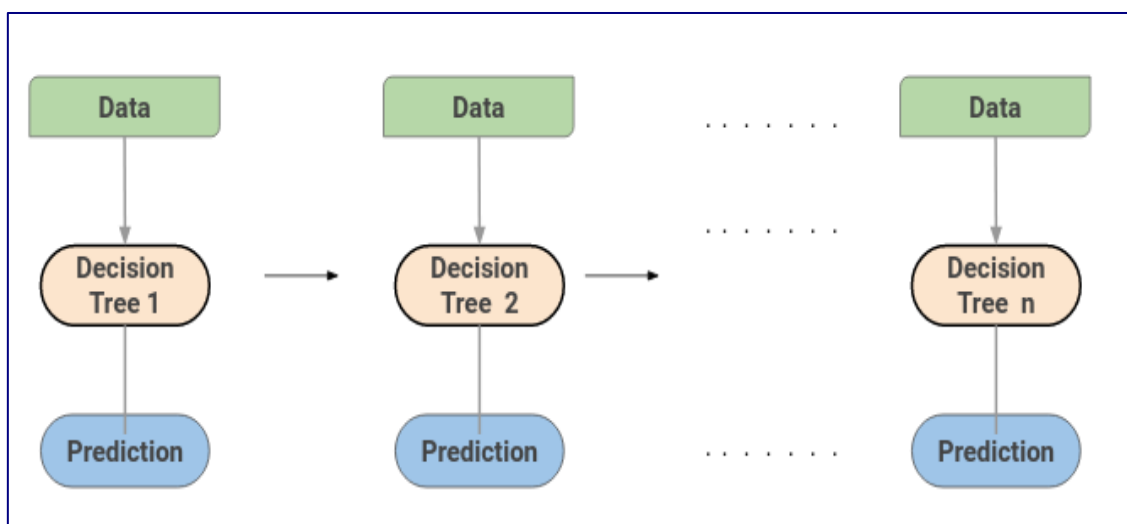
1. Random forests have been observed to **overfit for some datasets** with noisy classification/regression tasks.
2. For data including categorical variables with different number of levels, **random forests are biased in favor of those attributes with more levels**. Therefore, the variable importance scores from random forest are not reliable for this type of data.

GRADIENT BOOSTING MACHINE(GBM)

A Gradient Boosting Machine or GBM combines the predictions from multiple decision trees to generate the final predictions. Keep in mind that all the weak learners in a gradient boosting machine are decision trees.

But if we are using the same algorithm, then how is using a hundred decision trees better than using a single decision tree? How do different decision trees capture different signals/information from the data?

Here is the trick – the nodes in every decision tree take a different subset of features for selecting the best split. This means that the individual trees aren't all the same and hence they are able to capture different signals from the data.



Additionally, each new tree takes into account the errors or mistakes made by the previous trees. So, every successive decision tree is built on the errors of the previous trees. This is how the trees in a gradient boosting machine algorithm are built sequentially.

When we try to predict the target variable using any machine learning technique, the main causes of difference in actual and predicted values are noise, variance, and bias. Ensemble helps to reduce these factors (except noise, which is irreducible error)

An ensemble is just a collection of predictors which come together (e.g. mean of all predictions) to give a final prediction. The reason we use ensembles is that many different predictors trying to predict same target variable will perform a better job than any single predictor alone. Ensembling techniques are further classified into Bagging and Boosting.

Bagging:

Bagging is a simple ensembling technique in which we build many independent predictors/models/learners and combine them using some model averaging techniques. (e.g. weighted average, majority vote or normal average). We typically take random sub-sample/bootstrap of data for each model, so that all the models are little different from each other. Each observation is chosen with replacement to be used as input for each of the model. So, each model will have different observations based on the bootstrap process. Because this technique takes many uncorrelated learners to make a final model, it reduces error by reducing variance. Example of bagging ensemble is Random Forest models.

Boosting:

This technique employs the logic in which the subsequent predictors learn from the mistakes of the previous predictors. Therefore, the observations have an unequal probability of appearing in subsequent models and ones with the highest error appear most. (So the observations are not chosen based on the bootstrap process, but based on the error). The predictors can be chosen from a range of models like decision trees, regressors, classifiers etc. Because new predictors are learning from mistakes committed by previous predictors, it takes less time/iterations to reach close to actual predictions. But we have to choose the stopping criteria carefully or it could lead to overfitting on training data. Gradient Boosting is an example of boosting algorithm.

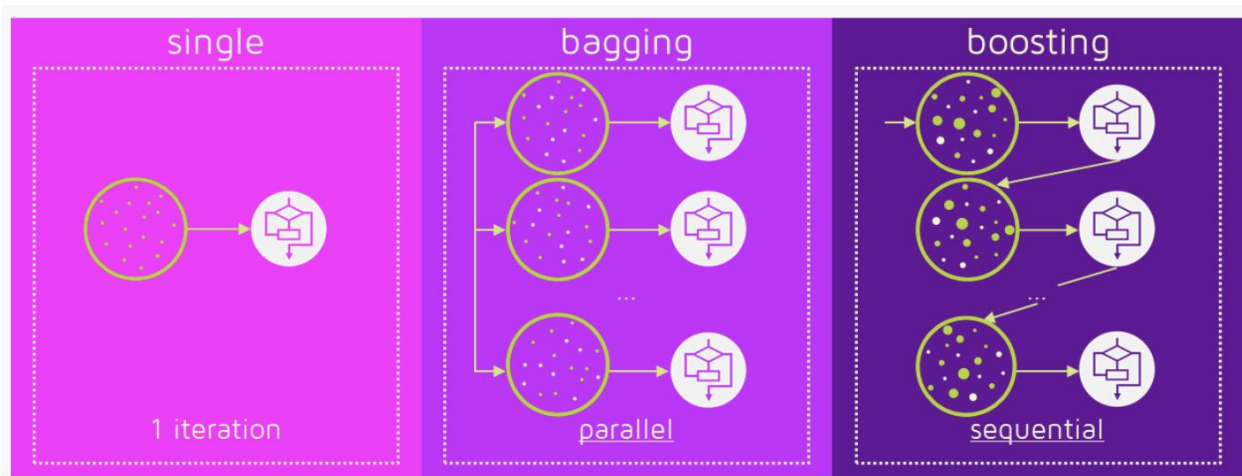


Fig 1. Bagging (independent models) & Boosting (sequential models).

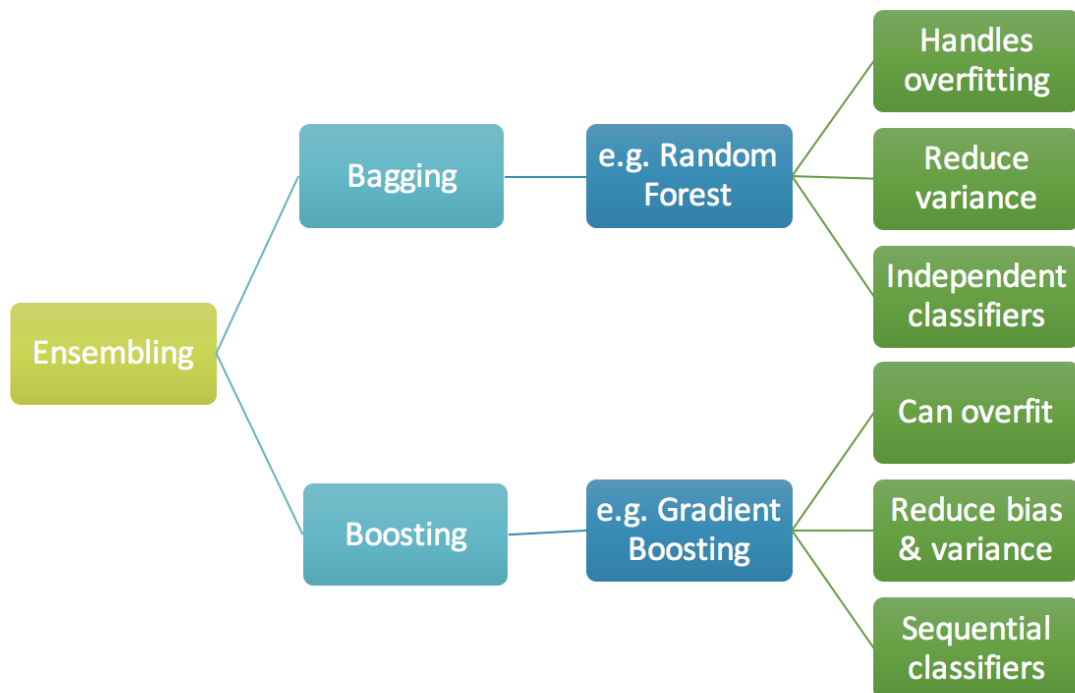


Fig 2. Ensembling

Gradient Boosting algorithm

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

The objective of any supervised learning algorithm is to define a loss function and minimize it.

Let's see how maths work out for Gradient Boosting algorithm. Say we have mean squared error (MSE) as loss defined as:

$$Loss = MSE = \sum (y_i - y_i^p)^2$$

where, y_i = ith target value, y_i^p = ith prediction, $L(y_i, y_i^p)$ is Loss function

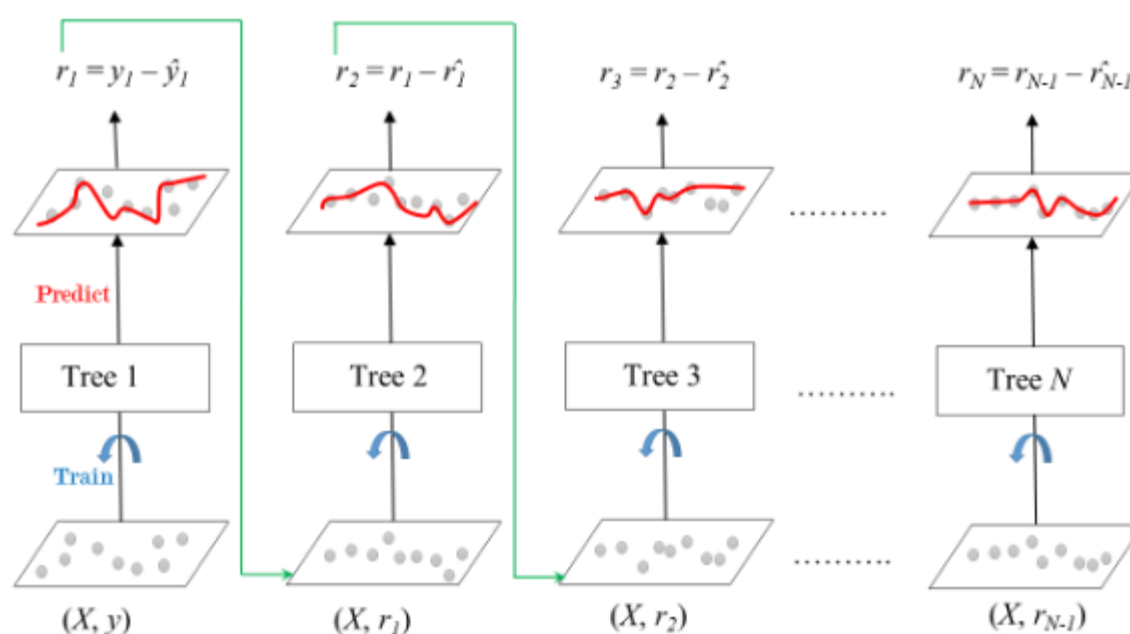
Predictions is based on, such that our loss function (MSE) is minimum. By using gradient descent and updating our predictions based on a learning rate, we can find the values where MSE is minimum.

$$y_i^p = y_i^p + \alpha * \delta \sum (y_i - y_i^p)^2 / \delta y_i^p$$

which becomes, $y_i^p = y_i^p - \alpha * 2 * \sum (y_i - y_i^p)$

where, α is learning rate and $\sum (y_i - y_i^p)$ is sum of residuals

So, we are basically updating the predictions such that the sum of our residuals is close to 0 (or minimum) and predicted values are sufficiently close to actual values.



Intuition behind Gradient Boosting

The logic behind gradient boosting is simple, (can be understood intuitively, without using mathematical notation). I expect that whoever is reading this post might be familiar with simple linear regression modeling.

A basic assumption of linear regression is that sum of its residuals is 0, i.e. the residuals should be spread randomly around zero

Now think of these residuals as mistakes committed by our predictor model. Although, tree-based models (considering decision tree as base models for our gradient boosting here) are not based on such assumptions, but if we think logically (not statistically) about this assumption, we might argue that, if we are able to see some pattern of residuals around 0, we can leverage that pattern to fit a model.

So, the intuition behind gradient boosting algorithm is to repetitively leverage the patterns in residuals and strengthen a model with weak predictions and make it better. Once we reach a stage that residuals do not have any pattern that could be modeled, we can stop modeling residuals (otherwise it might lead to overfitting). Algorithmically, we are minimizing our loss function, such that test loss reach its minima.

CHAPTER 6

SYSTEM TESTING

6.1 SYSTEM TESTING

System testing is the stage of implementation that is aimed at ensuring that the system works accurately and efficiently before live operation commences. Testing is vital to the success of the system. System testing makes logical assumption that if all the parts of the system are correct, then the goal will be successfully achieved. System testing involves user training system testing and successful running of the developed proposed system. The user tests the developed system and changes are made per their needs. The testing phase involves the testing of developed system using various kinds of data. While testing, errors are noted and the corrections are made. The corrections are also noted for the future use.

6.2 UNIT TESTING:

Unit testing focuses verification effort on the smallest unit of software design, software component or module. Using the component level design description as a control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and the errors those uncover is limited by the constrained scope established for unit testing. The unit test focuses on the internal processing logic and data structures within the boundaries of a component. This is normally considered as an adjunct to the coding step. The design of unit tests can be performed before coding begins.

6.3 BLACK BOX TESTING

Black box testing also called behavioural testing, focuses on the functional requirement of the software. This testing enables to derive set of input conditions of all functional requirements for a program. This technique focuses on the information domain of the software, deriving test cases by partitioning the input and output of a program.

6.4 WHITE BOX TESTING

White box testing also called as glass box testing, is a test case design that uses the control structures described as part of component level design to derive test cases. This test case is derived to ensure all statements in the program have been executed at least once during the testing and that all logical conditions have been exercised.

6.5 INTEGRATION TESTING

Integration testing is a systematic technique for constructing the software architecture to conduct errors associated with interfacing. Top-down integration testing is an incremental approach to construction of the software architecture. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module. Bottom-up integration testing begins the construction and testing with atomic modules. Because components are integrated from the bottom up, processing required for components subordinate to a given level is always available.

6.6 VALIDATION TESTING

Validation testing begins at the culmination of integration testing, when individual components have been exercised, the software is completely assembled as a package. The testing focuses on user visible actions and user recognizable output from the system. The testing has been conducted on possible condition such as the function characteristic conforms the specification and a deviation or error is uncovered. The alpha test and beta test is conducted at the developer site by end-users.

CHAPTER 7

SYSTEM IMPLEMENTATION

Implementation is the stage in the project where the theoretical design is turned into a working system. The most crucial stage is achieving a successful new system & and giving the user confidence that the new system will work efficiently & and effectively in the implementation stage.

The stage consists of

- Testing the developed program with simple data.
- Detections and correction of errors.
- Creating whether the system meets user requirements.
- Testing whether the system.
- Making necessary changes as desired by the user.
- Training user personnel.

Implementation Procedures

The implementation phase is less creative than the system design. A system project may be dropped at any time before implementation, although it becomes more difficult when it goes to the design phase.

The final report to the implementation phase includes procedural flowcharts, record layouts, report layouts, and a workable plan for implementing the candidate system design into an operational one. Conversion is one aspect of implementation

System Maintenance

The maintenance phase of the software cycle is the time in which a software product performs useful work. After a system is successfully implemented, it should be maintained in a proper manner. System maintenance is an important aspect in the software development life cycle. The need for system maintenance is to make adaptable and some changes in the system environment. There may be social, technical and other environmental changes, which affect a system, that is implemented. Software product enhancements may involve providing new functional capabilities, improving user displays and mode of interaction, upgrading the performance of the characteristics of the system.

Maintenance phase identifies if there are any changes required in the current system. If the changes are identified, then an analysis is made to identify if the changes are really required. Cost benefit analysis is a way to find out if the change is essential. System maintenance conforms the system to its original requirements and the purpose is to preserve the value of software over the time. The value can be enhanced by expanding the customer base, meeting additional requirements, becoming easier to use, more efficient and employing newer technology.

CHAPTER 8

CONCLUSION

In conclusion, the application of the M5P model tree-based algorithm for gold price prediction represents a significant advancement in the realm of financial forecasting. Through the utilization of machine learning techniques, particularly the integration of decision trees with linear regression models at the leaves, this approach offers a compelling solution to the challenges associated with traditional forecasting methods. The efficacy of the M5P model lies in its ability to capture complex nonlinear relationships and interactions inherent in financial time series data, while maintaining interpretability and transparency. By leveraging historical data on gold prices alongside a diverse array of economic indicators, geopolitical events, and market sentiment, the M5P model can generate accurate and reliable predictions, providing valuable insights for investors, financial institutions, and policymakers.

Furthermore, the interpretability of the M5P model facilitates a deeper understanding of the factors driving gold price movements, enabling stakeholders to make informed decisions regarding investment strategies, risk management, and portfolio diversification. While the M5P algorithm may not be immune to challenges such as sensitivity to outliers and model complexity, its benefits in terms of prediction accuracy and flexibility outweigh these limitations.

In essence, the adoption of the M5P model tree-based algorithm for gold price prediction represents a promising approach to addressing the complexities and uncertainties of financial markets. As advancements in machine learning continue to evolve, the M5P model stands as a testament to the power of data-driven modeling techniques in enhancing decision-making and unlocking new opportunities in the field of finance.

FUTURE ENHANCEMENT

Integration of Additional Features: Incorporate additional features such as global economic indicators, geopolitical events, inflation rates, and currency exchange rates. These factors can significantly influence gold prices and may improve the accuracy of the prediction model.

Advanced Machine Learning Models: Explore more sophisticated machine learning algorithms beyond the M5P model tree, such as ensemble methods (e.g., Random Forest, Gradient Boosting), deep learning architectures (e.g., LSTM, CNN), or hybrid models. Experiment with different model architectures and hyperparameters to improve prediction performance.

Time-Series Analysis: Implement time-series analysis techniques to capture temporal patterns and seasonality in gold price data. This could involve using techniques like ARIMA (Auto Regressive Integrated Moving Average) models, Prophet, or Fourier transforms to extract and model periodic trends in the data.

Sentiment Analysis: Integrate sentiment analysis of news articles, social media posts, and market reports related to gold. Sentiment analysis can provide insights into market sentiment and investor behavior, which may impact gold prices. Natural language processing (NLP) techniques can be applied to analyze textual data and extract sentiment features.

Feature Engineering: Conduct extensive feature engineering to create new features or transform existing ones that better capture the underlying patterns in the data. This could involve techniques such as polynomial features, interaction terms, or domain-specific transformations tailored to the gold market.

Cross-Validation and Hyperparameter Tuning: Perform rigorous cross-validation and hyperparameter tuning to optimize the performance of the prediction model. Utilize techniques like grid search, random search, or Bayesian optimization to search the hyperparameter space efficiently and find the best model configuration.

Ensemble Learning: Explore ensemble learning techniques to combine predictions from multiple models or model variants. Ensemble methods such as stacking, bagging, and boosting can often yield better performance than individual models by leveraging the diversity of different learners.

Real-Time Prediction: Develop a real-time prediction system that continuously monitors incoming data streams and updates the prediction model dynamically. Implement streaming data processing techniques and online learning algorithms to handle large volumes of data and adapt the model in real-time.

Deployment on Cloud Infrastructure: Deploy the prediction model on cloud infrastructure to improve scalability, reliability, and accessibility. Utilize cloud platforms like AWS, Google Cloud, or Microsoft Azure to deploy and manage the prediction system, allowing for easy scaling and integration with other services.

User Interface and Visualization: Build an interactive user interface and visualization dashboard to present the predicted gold prices and related insights in a user-friendly manner. Incorporate features for data exploration, scenario analysis, and historical trend visualization to empower users to make informed decisions.

CHAPTER 9

APPENDIX

9.1 SOURCE CODE

```
!pip install pandas
!pip install numpy
!pip install matplotlib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as figure
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.dates as mdates
from sklearn import linear_model
from sklearn.model_selection import TimeSeriesSplit
from sklearn.svm import SVR
import os
x=pd.read_csv("FINAL_USO.csv",na_values=['null'],index_col='Date',parse_dates=True,infer_datetime_format=True)
x.head()
x.shape
x.describe()
x.isnull().values.any()
GLD = x['Adj Close']
SPY = x['SP_Ajclose']
DJ = x['DJ_Ajclose']
df_p = pd.DataFrame({'GLD':GLD, 'SPY':SPY, 'DJ':DJ})
```

```

# df_ax = df_p.plot(title='Effect of Index prices on gold rates',figsize=(15,8))

# df_ax.set_ylabel('Price')
# df_ax.legend(loc='upper left')
# plt.show()
palette = sns.color_palette("rocket_r")
sns.relplot(df_p,kind="line",height=5, aspect=.75, facet_kws=dict(sharex=False),palette=palette)
def compute_daily_returns(df):
    """Compute and return the daily return values."""
    # TODO: Your code here
    # Note: Returned DataFrame must have the same number of rows
    daily_return = (df / df.shift(1)) - 1
    daily_return[0] = 0
    return daily_return
GLD_adj_close = x['Adj Close']
SPY_adj_close = x['SP_Ajclose']
DJ_adj_close = x['DJ_Ajclose']
EG_adj_close = x['EG_Ajclose']
USO_Adj_close = x['USO_Adj Close']
GDX_Adj_close = x['GDX_Adj Close']
EU_price = x['EU_Price']
OF_price = x['OF_Price']
OS_price = x['OS_Price']
SF_price = x['SF_Price']
USB_price = x['USB_Price']
PLT_price = x['PLT_Price']
PLD_price = x['PLD_Price']
rho_price = x['RHO_PRICE']
usdi_price = x['USDI_Price']
GLD_daily_return = compute_daily_returns(GLD_adj_close)
SPY_daily_return = compute_daily_returns(SPY_adj_close)
DJ_adj_return = compute_daily_returns(DJ_adj_close)
EG_adj_return = compute_daily_returns(EG_adj_close)
USO_Adj_return = compute_daily_returns(USO_Adj_close)

```

```

GDX_Adj_return =compute_daily_returns(GDX_Adj_close)
EU_return      = compute_daily_returns(EU_price)
OF_price       =compute_daily_returns(OF_price)
OS_price       =compute_daily_returns(OS_price)
SF_price       =compute_daily_returns(SF_price)
USB_price      =compute_daily_returns(USB_price)
PLT_price      =compute_daily_returns(PLT_price)
PLD_price      =compute_daily_returns(PLD_price)
rho_price      =compute_daily_returns(rho_price)
USDI_price     =compute_daily_returns(usdi_price)
df_d = pd.DataFrame({'GLD':GLD_daily_return, 'SPY':SPY_daily_return, 'DJ':DJ_adj_return,
'EG':EG_adj_return, 'USO':USO_Adj_return,
                    'GDX':GDX_Adj_return,'EU':EU_return, 'OF':OF_price,'SF':SF_price,'OS':OS_price,
'USB':USB_price, 'PLT':PLT_price, 'PLD':PLD_price,
                    'RHO':rho_price,'USDI':USDI_price})
# daily_ax = df_d[-100:].plot(title='Last 100 records of daily return of all features',figsize=(15,8))
# daily_ax.set_ylabel('Daily return')
# daily_ax.legend(loc='lower left')
# plt.show()
palette = sns.color_palette("rocket_r")
sns.relplot(df_d,kind="line",height=5, aspect=.75, facet_kws=dict(sharex=False),palette=palette)
df_s = pd.DataFrame({'GLD':GLD_daily_return, 'SPY':SPY_daily_return, 'DJ':DJ_adj_return})
# daily_ax = df_s[-100:].plot(title='Last 100 records of daily return of Stock Indexes',figsize=(15,8))
# daily_ax.set_ylabel('Daily return')
# daily_ax.legend(loc='lower left')
# plt.show()
palette = sns.color_palette("rocket_r")
sns.relplot(df_s,kind="line",height=5, aspect=.75, facet_kws=dict(sharex=False),palette=palette)
x.head()
df_d
# df_d.plot(kind='scatter', x='SPY', y='GLD')
X=df_d['SPY']
Y=df_d['GLD']
sns.jointplot(x=X,y=Y,kind='resid',color="#d9670f")
# df_d.plot(kind='scatter', x='DJ', y='GLD')

```

```

sns.set_theme(style="ticks")

data={
    'x':df_d['DJ'],
    'y':df_d['GLD']
}

sns.histplot(
    data,edgecolor=".3",
    linewidth=.5,palette="light:m_r",
    multiple="stack"
)

# df_d.plot(kind='scatter', x='EG', y='GLD')
# df_d.plot(kind='scatter', x='EG', y='GLD')

sns.set_theme(style="dark")
X=df_d['EG']
y=df_d['GLD']
data={
    'x':df_d['EG'],
    'y':df_d['GLD']
}

f, ax = plt.subplots(figsize=(6, 6))
sns.scatterplot(x=X,y=Y, s=5, color=".15")
sns.histplot(x=X,y=Y, bins=50, pthresh=.1,cmap='mako')
sns.kdeplot(x=X,y=Y, levels=5, color="w", linewidths=1)

# df_d.plot(kind='scatter', x='USO', y='GLD')

sns.set_theme(style="whitegrid")
X=df_d['EG']
y=df_d['GLD']
data={
    'x':df_d['EG'],
    'y':df_d['GLD']
}

sns.displot(
    data=data,
    kind="kde",
    height=6,

```

```

    multiple="fill",
    clip=(0, None),
    palette="ch:rot=-.25,hue=1,light=.75",
)
# df_d.plot(kind='scatter', x='USB', y='GLD')
sns.set_theme(style="dark")
X=df_d['EG']
y=df_d['GLD']
data={
    'x':df_d['EG'],
    'y':df_d['GLD']
}
sns.displot(
    data,
    kind="kde", aspect=.75, linewidth=2,height=6,
    multiple="stack",palette="ch:rot=-.25,hue=1,light=.75")
# df_d.plot(kind='scatter', x='EU', y='GLD')
sns.set_theme(style="dark")
X=df_d['EG']
y=df_d['GLD']
data={
    'x':df_d['EG'],
    'y':df_d['GLD']
}
sns.relplot(x=X, y=Y,
            sizes=(40, 400), alpha=.5, palette="muted",kind='scatter',
            height=6, data=data,color='#131326')
# df_d.plot(kind='scatter', x='PLT', y='GLD')
X=df_d['EG']
y=df_d['GLD']
data={
    'x':df_d['EG'],
    'y':df_d['GLD']
}
sns.swarmplot(data=data, x=X, y=Y,color='black')

```



```
FS_RMSE_scores = {}
```

```
def fs_model_review(models):
```

```
    fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(16, 16))
```

```
    #plot benchmark model
```

```
    benchmark_dt_predicted = benchmark_dt_fs.predict(feature_selected_validation_X)
```

```
    benchmark_RSME_score = np.sqrt(mean_squared_error(validation_y, benchmark_dt_predicted))
```

```
    FS_RMSE_scores['Benchmark'] = benchmark_RSME_score
```

```
    axes[0,0].plot(validation_y.index, benchmark_dt_predicted,'y', label='Predict')
```

```
    axes[0,0].plot(validation_y.index, validation_y,'b', label='Actual')
```

```
    axes[0,0].xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
```

```
    axes[0,0].xaxis.set_major_locator(mdates.MonthLocator())
```

```
    axes[0,0].set_ylabel('Price')
```

```
    axes[0,0].set_title("Benchmark Predict's RMSE Error: "
```

```
    + "{0:.2f} ".format(benchmark_RSME_score))
```

```
    axes[0,0].legend(loc='upper right')
```

```
    #plot block
```

```
    ax_x = 0
```

```
    ax_y = 1
```

```
    #plot solution model
```

```
    for name, model in models.items():
```

```
        predicted = model.predict(feature_selected_validation_X)
```

```
        RSME_score = np.sqrt(mean_squared_error(validation_y, predicted))
```

```
        R2_score = r2_score(validation_y, predicted)
```

```
        axes[ax_x][ax_y].plot(validation_y.index, predicted,'y', label='Predict')
```

```
        axes[ax_x][ax_y].plot(validation_y.index, validation_y,'b', label='Actual')
```

```
        axes[ax_x][ax_y].xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
```

```
        axes[ax_x][ax_y].xaxis.set_major_locator(mdates.MonthLocator())
```

```
        axes[ax_x][ax_y].set_ylabel('Price')
```

```

axes[ax_x][ax_y].set_title(name + "'s RMSE Error: " + "{0:.2f}".format(RSME_score))
axes[ax_x][ax_y].legend(loc='upper right')
FS_RMSE_scores[name] = RSME_score
if ax_x <=2:
    if ax_y < 2:
        ax_y += 1
    else:
        ax_x += 1
        ax_y = 0
plt.show()

fs_model_review(feature_selected_solution_models)

fs_model_names = []
fs_model_values = []
for name, value in FS_RMSE_scores.items():
    fs_model_names.append(name)
    fs_model_values.append(value)

fs_model_values = np.array(fs_model_values)
fs_model_names = np.array(fs_model_names)

fs_indices = np.argsort(fs_model_values)
fs_columns = fs_model_names[fs_indices[:8]]
fs_values = fs_model_values[fs_indices[:8]]
origin_values = model_values[fs_indices[:8]]

fig = plt.figure(figsize = (16,8))
plt.bar(np.arange(8) - 0.2 , origin_values ,width = 0.4, align="center", color = '#b2b2ff', label =
"Original")
plt.bar(np.arange(8), fs_values ,width = 0.4, align="center", color = '#3232ff', label = "Feature
Selected")
plt.xticks(np.arange(8), fs_columns)
plt.xlabel('Model')
plt.ylabel('RMSE')

```

```
plt.title('RMSE compare after feature selection')
plt.legend(loc = 'upper center')
plt.show()
```

```
# Choosing the top three performing models to ensemble them
ensemble_solution_models = [lasso_clf_feat, bay_feat, ridge_clf_feat]
```

```
class EnsembleSolution:
```

```
    models = []
```

```
    def __init__(self, models):
```

```
        self.models = models
```

```
    def fit(self, X, y):
```

```
        for i in self.models:
```

```
            i.fit(X, y)
```

```
    def predict(self, X):
```

```
        result = 0
```

```
        for i in self.models:
```

```
            result = result + i.predict(X)
```

```
        result = result / len(self.models)
```

```
        return result
```

```
print("Ensemble Solution Model with Original features")
```

```
EnsembleModel = EnsembleSolution(ensemble_solution_models)
```

```
validate_result(EnsembleModel, 'EnsembleSolution')
```

```
ensemble_solution_model_fs = [lasso_clf_fs, bay_feat_fs, linear_svr_clf_fs]
```

```
print("Ensemble Solution Model with Selected features")
```

```
EnsembleModel_fs = EnsembleSolution(ensemble_solution_model_fs)
```

```
feature_selected_validate_result(EnsembleModel_fs, 'EnsembleSolution with FS')
```

```
def train_reg_multiplentimes(model, times):
```

```
    total_rmse = 0
```

```
    total_r2 = 0
```

```
    for i in range(times):
```

```

reg = model
for train_index, test_index in TimeSeriesSplit(n_splits=i+2).split(feature_minmax_transform):
    X_train, X_test = feature_minmax_transform[:len(train_index)],
feature_minmax_transform[len(train_index): (len(train_index)+len(test_index))]
    y_train, y_test = target_adj_close[:len(train_index)].values.ravel(),
target_adj_close[len(train_index): (len(train_index)+len(test_index))].values.ravel()
    reg.fit(X_train, y_train)
    predicted = reg.predict(validation_X)
    rmse, r2 = print_result(validation_y, predicted, [0,len(validation_y)])
    total_rmse += rmse
    total_r2 += r2
return total_rmse / times, total_r2 / times

```

```

def print_result(actual, predict, index):
    RMSE_score = np.sqrt(mean_squared_error(actual, predict))
    print('From { } to { }'.format(index[0],index[-1]))
    print('RMSE: ', RMSE_score)
    R2_score = r2_score(actual, predict)
    print('R2 score: ', R2_score)
    print('-----')
    return RMSE_score, R2_score

```

```

print('Benchmark')
t_multiple_benchmark_RMSE,t_multiple_benchmark_R2 = train_reg_multiplentimes(benchmark_dt,
7)
print('RMSE: { } // R2: { }\n'.format(t_multiple_benchmark_RMSE, t_multiple_benchmark_R2))

```

```

print('LSVR')
t_multiple_LSVR_RMSE,t_multiple_LSVR_R2 = train_reg_multiplentimes(linear_svr_clf_feat, 7)
print(' RMSE: { } // R2: { }'.format(t_multiple_LSVR_RMSE, t_multiple_LSVR_R2))
print('Lasso')
t_multiple_lasso_RMSE,t_multiple_lasso_R2 = train_reg_multiplentimes(lasso_clf_feat, 7)
print(' RMSE: { } // R2: { }'.format(t_multiple_lasso_RMSE, t_multiple_lasso_R2))
print('Ridge')
t_multiple_ridge_RMSE,t_multiple_ridge_R2 = train_reg_multiplentimes(ridge_clf_feat, 7)

```

```

print(' RMSE: {} // R2: {}'.format(t_multiple_ridge_RMSE, t_multiple_ridge_R2))
print('BayRidge')
t_multiple_bayridge_RMSE,t_multiple_bayridge_R2 = train_reg_multiplentimes(bay_feat, 7)
print(' RMSE: {} // R2: {}'.format(t_multiple_bayridge_RMSE, t_multiple_bayridge_R2))

print('Ensemble')
t_multiple_ensemble_RMSE,t_multiple_ensemble_R2 =
train_reg_multiplentimes(EnsembleSolution(ensemble_solution_models), 7)
print(' RMSE: {} // R2: {}'.format(t_multiple_ensemble_RMSE, t_multiple_ensemble_R2))
def cross_validate(model, ts_split):
    clf = model
    total_rmse = 0
    total_r2 = 0
    count = 0
    for train_index, test_index in ts_split.split(validation_X):
        X_test1, X_test2 = validation_X[:len(train_index)], validation_X[len(train_index):
(len(train_index)+len(test_index))]
        y_test1, y_test2 = validation_y[:len(train_index)].values.ravel(), validation_y[len(train_index):
(len(train_index)+len(test_index))].values.ravel()
        predicted_test1 = clf.predict(X_test1)
        temp1_RMSE, temp1_R2 = print_result(y_test1, predicted_test1, train_index)

        predicted_test2 = clf.predict(X_test2)
        temp2_RMSE, temp2_R2 = print_result(y_test2, predicted_test2, test_index)

        total_rmse += temp1_RMSE + temp2_RMSE
        total_r2 += temp1_R2 + temp2_R2
        count += 2
    return total_rmse / count, total_r2 / count
timeseries_cv = TimeSeriesSplit(n_splits=10)
test_bench__RMSE, test_bench_R2 = cross_validate(benchmark_dt,timeseries_cv)
test_lsrv__RMSE, test_lsrv_R2 = cross_validate(lsrv_grid_search_feat,timeseries_cv)

test_ridge__RMSE, test_ridge_R2 = cross_validate(ridge_clf_feat,timeseries_cv)
test_lasso__RMSE, test_lasso_R2 = cross_validate(lasso_clf_feat,timeseries_cv)

```

```

test_bay_RMSE, test_bay_R2 = cross_validate(bay_feat, timeseries_cv)
test_ensemble_RMSE, test_ensemble_R2 =
cross_validate(EnsembleSolution(ensemble_solution_models), timeseries_cv)
def cross_validate(model, ts_split):
    clf = model
    total_rmse = 0
    total_r2 = 0
    count = 0
    for train_index, test_index in ts_split.split(validation_X):
        X_test1, X_test2 = validation_X[:len(train_index)], validation_X[len(train_index):
(len(train_index)+len(test_index))]
        y_test1, y_test2 = validation_y[:len(train_index)].values.ravel(), validation_y[len(train_index):
(len(train_index)+len(test_index))].values.ravel()
        predicted_test1 = clf.predict(X_test1)
        temp1_RMSE, temp1_R2 = print_result(y_test1, predicted_test1, train_index)

        predicted_test2 = clf.predict(X_test2)
        temp2_RMSE, temp2_R2 = print_result(y_test2, predicted_test2, test_index)

        total_rmse += temp1_RMSE + temp2_RMSE
        total_r2 += temp1_R2 + temp2_R2
        count += 2
    return total_rmse / count, total_r2 / count

```

9.2.SCREENSHOTS

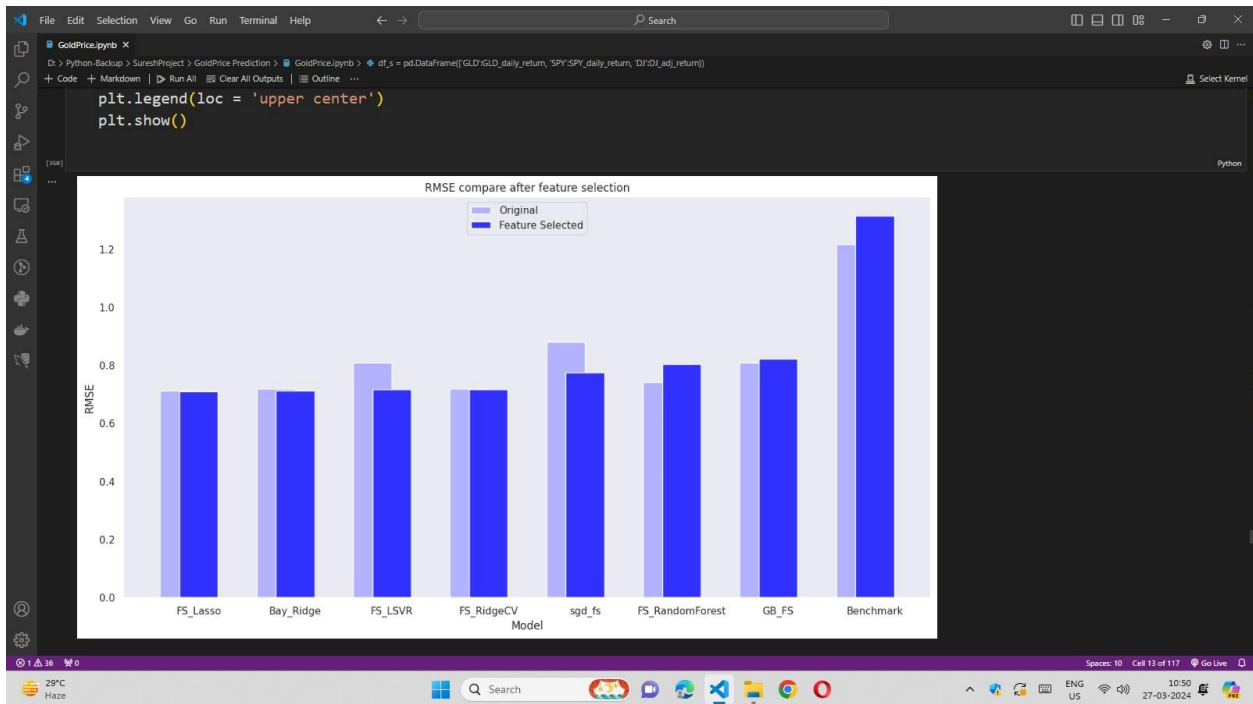


Figure 9.1 Bar Graph

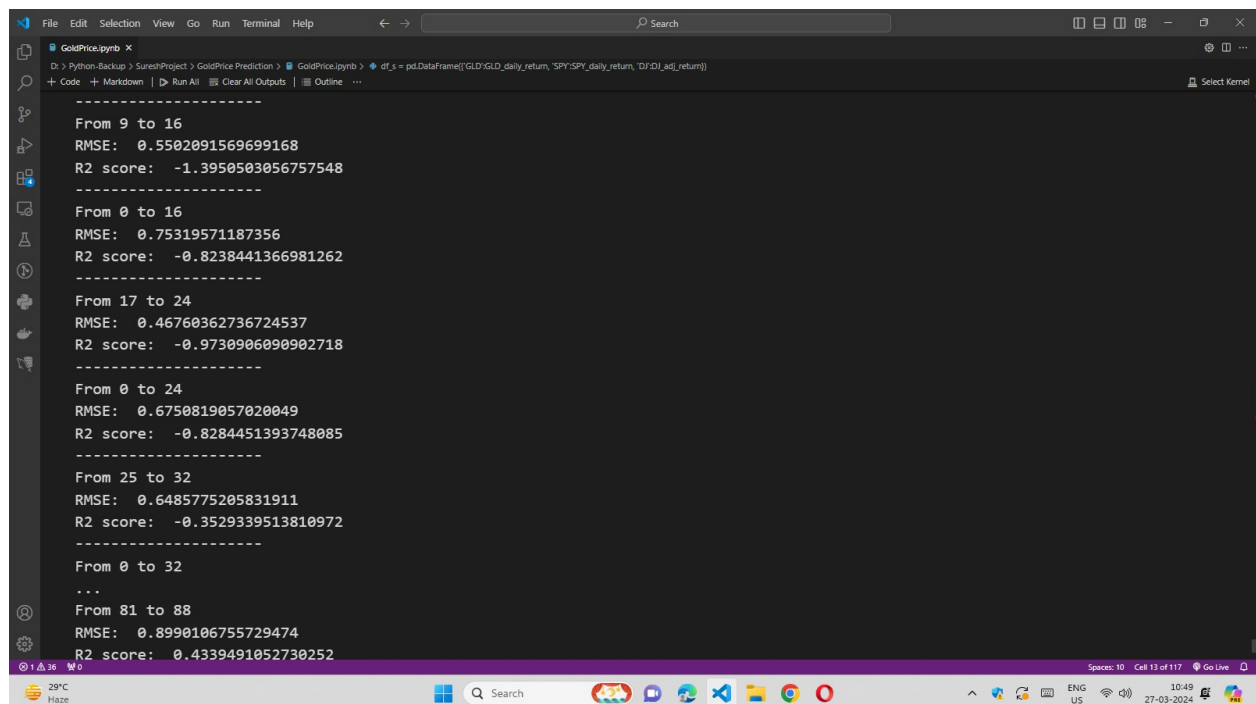


Figure 9.2 Prediction

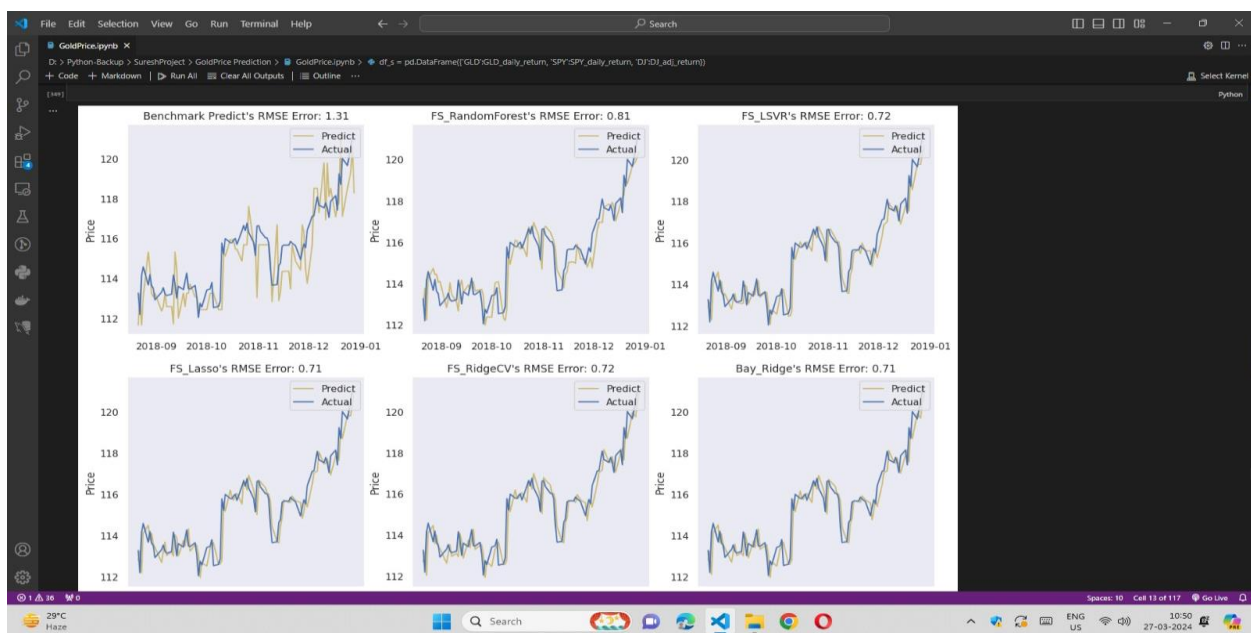


Figure 9.3 line chart

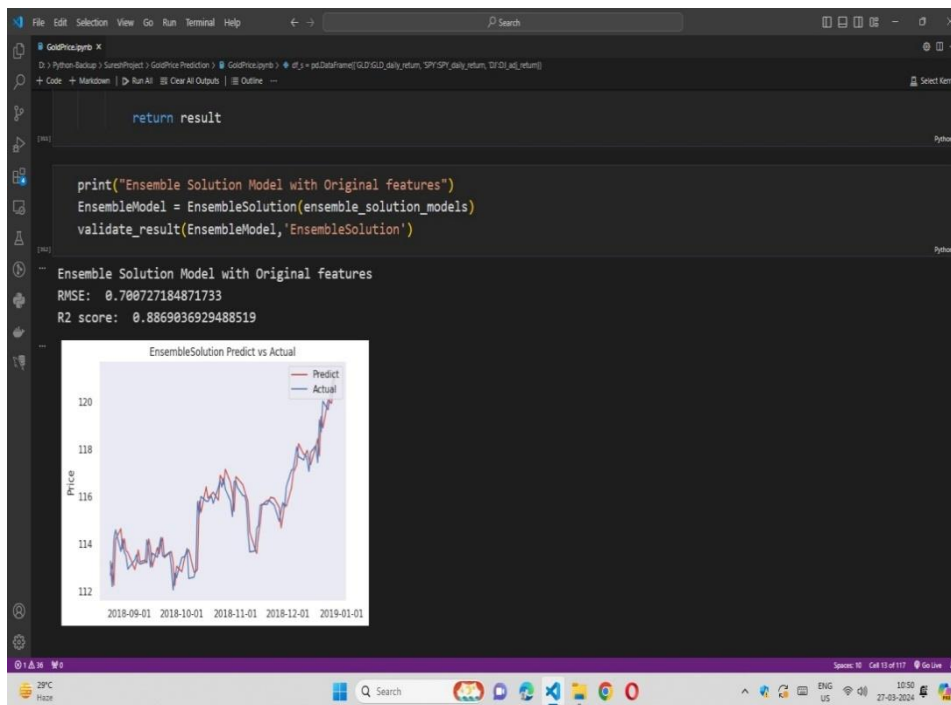


Figure 9.4. Ensemble solution

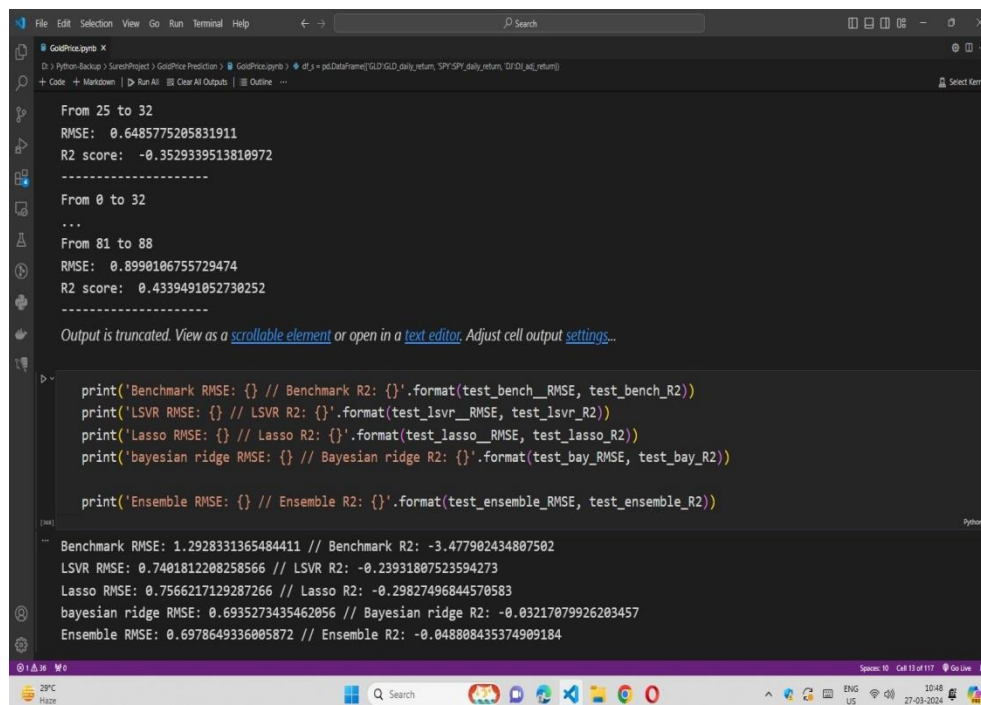
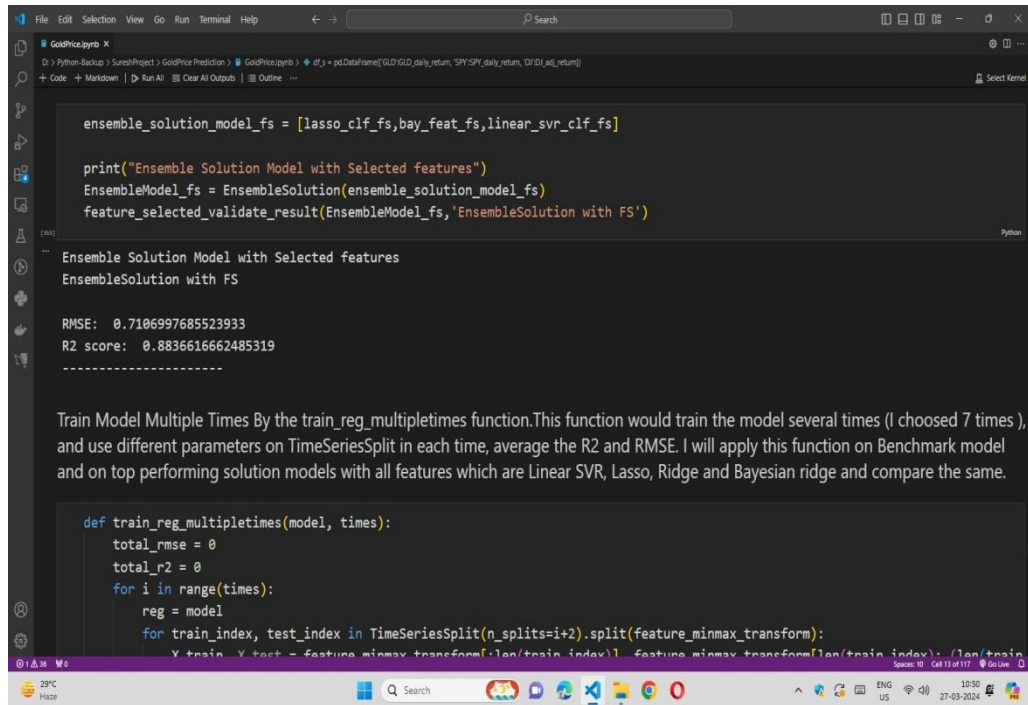


Figure 9.5. Output



```
ensemble_solution_model_fs = [lasso_clf_fs, bay_feat_fs, linear_svr_clf_fs]

print("Ensemble Solution Model with Selected features")
EnsembleModel_fs = EnsembleSolution(ensemble_solution_model_fs)
feature_selected_validate_result(EnsembleModel_fs, 'EnsembleSolution with FS')

...
Ensemble Solution Model with Selected features
EnsembleSolution with FS

RMSE: 0.7106997685523933
R2 score: 0.8836616662485319
-----

Train Model Multiple Times By the train_reg_multiplentimes function. This function would train the model several times (I chose 7 times),
and use different parameters on TimeSeriesSplit in each time, average the R2 and RMSE. I will apply this function on Benchmark model
and on top performing solution models with all features which are Linear SVR, Lasso, Ridge and Bayesian ridge and compare the same.

def train_reg_multiplentimes(model, times):
    total_rmse = 0
    total_r2 = 0
    for i in range(times):
        reg = model
        for train_index, test_index in TimeSeriesSplit(n_splits=i+2).split(feature_minmax_transform):
            X_train, X_test = feature_minmax_transform[:len(train_index)], feature_minmax_transform[len(train_index):len(train_index)+len(test_index)]
            y_train, y_test = feature_minmax_transform[:len(train_index)], feature_minmax_transform[len(train_index):len(train_index)+len(test_index)]
```

Figure 9.6. Training Model

CHAPTER 10

REFERENCES

10.1 REFERENCES

1.Hastie, T., Tibshirani, R., & Friedman, J. (2009). "The Elements of Statistical Learning: Data Mining, Inference, and Prediction". Springer.

2.This book provides comprehensive coverage of machine learning algorithms, including decision trees and ensemble methods, which are relevant to gold price prediction using the M5P model.

Quinlan, J. R. (1993). "C4.5: Programs for Machine Learning". Morgan Kaufmann.

3.This book introduces the C4.5 algorithm, which is a precursor to the M5P algorithm and provides insights into decision tree-based algorithms and their applications.

Web:

4."Machine Learning Algorithms: A Review". (n.d.). Retrieved from <https://www.mdpi.com/2073-8994/11/11/1337>

5.This review article provides an overview of various machine learning algorithms, including decision trees and ensemble methods, which are pertinent to gold price prediction using the M5P model.

6."M5P Model Tree Algorithm Explained". (n.d.). Retrieved from <https://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart>

7.This resource provides an explanation of the M5P model tree algorithm and its implementation in the scikit-learn library, offering practical insights into its usage for regression tasks.

Journals:

8.Peng, Y., & Leung, C. S. (2015). "Model tree as a novel hybrid decision tree technique for time series prediction". *Expert Systems with Applications*, 42(7), 3594-3606.

9.This journal article discusses the application of model trees, including the M5P algorithm, for time series prediction tasks, which is relevant to gold price forecasting.

10.Zhang, J., Zhang, C., & Liu, B. (2014). "A survey of recent achievements in time series prediction with machine learning". *Annals of Information Systems*, 14(1), 53-80.

11.This survey article provides an overview of recent advancements in time series prediction techniques, including machine learning approaches such as the M5P model, which can be applied to gold price prediction tasks.