Project Topic: Implementation and study of the Decision Tree (CART) Model (Option 1)

Project Topic and AI Acknowledgement:

Project Topic: The implementation and study of the **Decision Tree Model** with a focus on Option 1.

This project was carried out with the assistance of **AI**, specifically **ChatGPT**. The project involved leveraging AI to gain insights, clarify technical concepts, and implement various aspects of the Decision Tree algorithm from scratch. The detailed conversations and AI interactions throughout the project have been documented for transparency and reference.

ChatGPT conversation link:
https://studentutsedu-my.sharepoint.com/:w:/g/personal/muhammad_t_rizwan_student_uts_edu_au/EciM2JaPLGxOgtrW21eWVikB5_ml2st2T2MOvMUx5oqBlA?e=OwGlyQ

Google Colab Link: (for implementation)
https://colab.research.google.com/drive/1_XOvN4qwob-o1jUzl8Tjqg0cw8STI7g-?usp=sharing

Motivation

Coming from an econometrics background, I chose to study Decision Tree and Random Forest models because they are increasingly seen as valuable tools in econometric research. I wanted to gain a solid understanding of these models, as they offer a flexible approach to analysing complex relationships in data and have potential applications in future econometric studies.

Introduction

In this project, I explored the implementation and analysis of the Decision Tree model to better understand its behaviour and practical applications. Initially, I was curious about how the model actually works, so I turned to AI tools and visual resources like YouTube to break down the concepts into something more digestible. This helped me grasp how Decision Trees make decisions and why they often overfit the data. As I dug deeper, I realized that addressing these limitations was key, which led me to experiment with Random Forest, a method that combines multiple trees to solve the overfitting problem using techniques like bootstrapping. In my Random Forest implementation, I took a really simplified approach, where I just used bootstrapping when making new trees. I omitted random feature selection which is an important part of the Random Forest model. My goal throughout this study was not just to implement these models, but to learn their inner workings, strengths, and shortcomings, and see how ensemble methods can ultimately provide better results.

Definition

1. **Decision Tree**

   A Decision Tree is a supervised learning algorithm that represents decisions and their possible consequences in a tree-like structure, using branches to show options and leaves to show outcomes. It splits the data into subsets based on the feature values, making decisions at each node by evaluating specific criteria (e.g., Gini index or variance reduction). Decision Trees are intuitive and easy to interpret but can be prone to overfitting if not properly controlled.

2. **Random Forest**

   Random (Stochastic sampling) + Forest (Ensemble of decision trees)

   A Random Forest is an ensemble learning algorithm that builds multiple decision trees using different random subsets of the training data and features. Each tree independently makes predictions, and the final output is obtained by averaging (regression) or voting (classification) over all the trees. This approach reduces the variance and overfitting of individual decision trees, resulting in a more robust and accurate model. Random Forest leverages the power of randomness to create a diverse set of trees, improving generalization.

Journal:
I knew decision tree is a tree but I couldn't understand how it works. How it splits and how it chooses. I used this website to visually see how it happens (http://www.r2d3.us/visual-intro-to-machine-learning-part-1/) and I used these two YouTube videos to understand the technical parts.
1. https://www.youtube.com/watch?v=ZVR2Way4nwQ
2. https://www.youtube.com/watch?v=UhY5vPfQIrA&t=430s

Task/Objective

Decision Tree and Random Forest are types of supervised machine learning algorithms designed to solve both regression and classification problems. For this project, the focus is on using these models to predict a **continuous numeric output** (dependent variable) based on multiple input features (independent variables). The objective is to implement these models from scratch and find the best splits within the data that reduce variance, thus improving prediction accuracy. The aim is to understand how these models learn the relationships between features and target variables to make predictions on new, unseen data.

Hypothesis/Prediction Function

In machine learning, the hypothesis represents the mathematical relationship that a model learns from the training data to make predictions. For Decision Trees and Random Forests, the hypothesis is non-linear because it splits the data into different segments based on feature values rather than fitting a straight line or curve.

Hypothesis in Decision Trees:

The hypothesis in a Decision Tree is built by repeatedly partitioning the dataset based on certain features until the data is split into smaller, homogenous groups. The tree essentially learns "if-else" rules at each node that guide the splits. For example, a simple hypothesis rule learned by a Decision Tree might look like:

- **If** Feature_1 <= 5, predict Value A.
- **Else If** Feature_2 > 3, predict Value B.

This hypothesis takes the form of a tree structure, where each path from the root node to a leaf node represents a set of rules that the model uses to make predictions. Unlike linear models that assume a global relationship (like a straight line), Decision Trees learn localized relationships within different sections of the data.

Hypothesis in Random Forest:

In Random Forests, the hypothesis is a combination of multiple Decision Trees, each learning its own set of rules. Because each tree is trained on a random subset of data and features, they all generate slightly different hypotheses. During prediction, the Random Forest aggregates the results of all individual trees (e.g., by averaging in regression) to produce a final hypothesis that is more robust and less prone to overfitting.

Prediction Function:

Once the hypothesis is built, the prediction function uses the learned rules to predict the outcome for new data. For a given sample:

1. **Decision Tree**: The function traverses the tree according to the sample's feature values until it reaches a leaf node, which holds the predicted value.
2. **Random Forest**: Each tree in the forest makes its own prediction, and these predictions are combined (e.g., averaged) to obtain the final output.

In essence, the hypothesis is the set of decision rules that the model learns, and the prediction function is the process of applying these rules to make a prediction for new data.

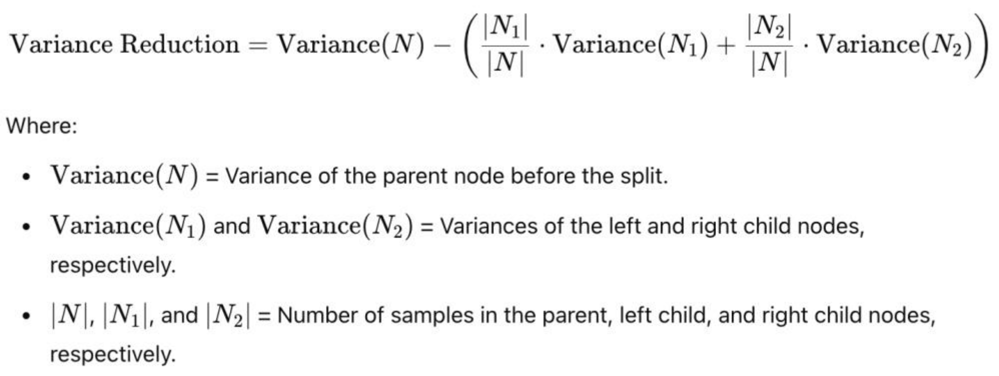
Key Components of the Model

1. **Root Node**
   The starting point of the tree, where the entire dataset is initially split based on the feature that results in the best reduction in error.

2. **Splitting Criterion**
   A measure to evaluate how well a feature and threshold split the data. For regression tasks, **Variance Reduction** is used to identify splits that minimize the variance within the child nodes.

   Variance Reduction is used to evaluate how effectively a feature, and a threshold split the data. A good split results in child nodes with lower variance compared to the parent node, indicating higher homogeneity. The objective is to select the feature and threshold that minimize the variance within the child nodes, thereby creating more distinct groups. Higher variance reduction signifies a better split and improves the model's predictive power.

From ChatGPT:



$$\text{Variance Reduction} = \text{Variance}(N) - \left(\frac{|N_1|}{|N|} \cdot \text{Variance}(N_1) + \frac{|N_2|}{|N|} \cdot \text{Variance}(N_2)\right)$$

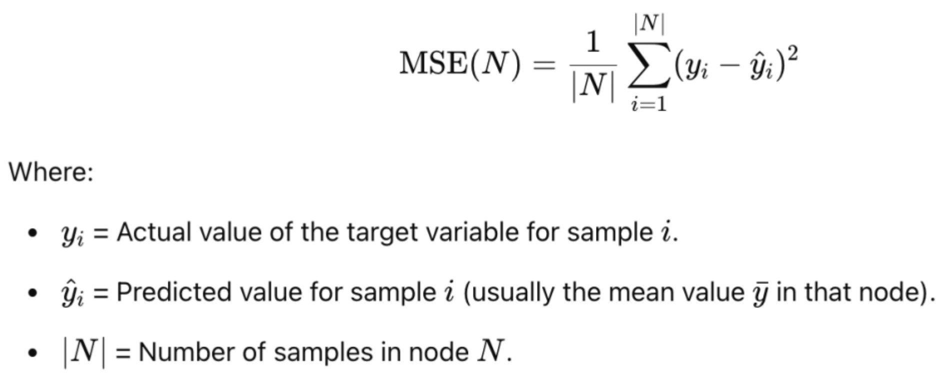
Where:

- $\text{Variance}(N)$ = Variance of the parent node before the split.
- $\text{Variance}(N_1)$ and $\text{Variance}(N_2)$ = Variances of the left and right child nodes, respectively.
- $|N|$, $|N_1|$, and $|N_2|$ = Number of samples in the parent, left child, and right child nodes, respectively.

Journal:
I was curious about how decision tree will decide what kind of split to do. I thought there might be some mathematical way or just random selection. But I was surprised to realize that the decision tree checks literally every feature with every threshold and calculates the variance reduction to see which feature and threshold is the best.

3. **Loss Function**
The Mean Squared Error (MSE) is used to quantify how well the model's predictions match the actual values. It captures the average squared differences between predicted and true values, penalizing larger errors more severely. The goal during training is to find the model parameters that minimize this value, ensuring predictions are as close as possible to the actual outcomes. A lower MSE indicates better model performance and reduced prediction errors.

From ChatGPT:



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

Where:

- y_i = Actual value of the target variable for sample i.
- $\hat{y}_i$ = Predicted value for sample i (usually the mean value $\bar{y}$ in that node).
- $|N|$ = Number of samples in node N.

Journal:
It was interesting to learn and MSE will be different based on the dataset. I thought there would be an upper and lower value, but I learnt that the value is just derived

from the predicted and actual values. Therefore, the MSE is not standard and changes with the dataset

4. **Decision Nodes (Internal Nodes)**
   Points where the data is split based on a feature and threshold. Each decision node further divides the data into smaller groups.

5. **Leaf Nodes (Terminal Nodes)**
   Nodes that store the final predicted value. For regression, the prediction is the **mean** of the target values in that node.

6. **Tree Depth**
   The number of levels in the tree. Controls the complexity of the model, with deeper trees capturing more detail but risking overfitting.

7. **Pruning**
   Technique used to remove unnecessary branches in the tree, reducing complexity and preventing overfitting by setting constraints like max_depth or min_samples_split.

8. **Ensemble Structure (for Random Forest)**
   Random Forest builds multiple decision trees on different data subsets and combines their outputs to create a stronger model, reducing overfitting and improving stability.

9. **Bootstrapping**
   Each tree in the Random Forest is trained on a unique sample of the original data (sampled with replacement), ensuring variability among the trees.

10. **Random Feature Selection**
    Randomly selecting a subset of features at each node to make each tree diverse and reduce the correlation between trees. In my implementation, I skipped this step because I wanted to make a really simple random forest model. This step can be included for further improvement.

    Journal:
    I actually tried the random feature selection, but the MSE I got was lower than the MSE without it. I didn’t investigate why this was. As my main target was decision tree I decided to move forward with just bootstrapping the data to make unique trees. I do understand that random feature selection is an important part of random forest.

11. **Aggregation (for Random Forest)**
    Combines predictions from all the trees in the Random Forest. For regression, the final prediction is the **average** of all tree outputs.

Regularization / Tree Pruning:

To prevent overfitting and ensure that the model generalizes well to new, unseen data, various techniques are applied to control the complexity of tree-based models.

1. Limiting Tree Depth (max_depth)

- Controls how deep the tree can grow by limiting the number of levels. A deeper tree captures more specific patterns but risks overfitting.
- **Purpose**: Prevents the tree from becoming too complex, making the model more robust and less sensitive to noise.
- This technique was used

2. Minimum Samples per Split (min_samples_split)

- Sets the minimum number of samples required at a node to consider splitting. Higher values prevent the model from creating branches based on small, specific subsets.
- **Purpose**: Forces the model to focus on significant patterns and avoid learning noise from smaller groups.
- This technique was used

> **Journal:**
> In the initial implementation I let the trees grow without pruning which gave me a training mse of 0 and a test mse of 5000. The model was overfitted. I decided to make a really simple tree for the initial implementation as the dataset was small.

3. Minimum Samples per Leaf (min_samples_leaf)

- Specifies the minimum number of samples required to be in a leaf node. Setting this parameter helps avoid creating overly specific leaf nodes with very few data points.
- **Purpose**: Ensures that the final predictions are based on larger groups, reducing variance and making the model more stable.
- This technique was not used

4. Randomness in Random Forest

- Uses random sampling of both data (bootstrapping) and features (random feature selection) to build each tree.
- **Purpose**: Reduces overfitting by creating a diverse set of trees that are less correlated, resulting in a more stable model.
- This technique was used but random feature selection was not implemented.

Difference between MSE and Variance Reduction

- **Mean Squared Error (MSE)**: MSE is a loss function used to evaluate a model's performance by measuring the average squared difference between the actual and predicted values. It helps in assessing how well a model fits the data and is commonly used for model evaluation after training.

- **Variance Reduction**: Variance Reduction is a **splitting criterion** used during the training phase of Decision Trees. It measures how effectively a feature can divide the dataset into homogenous subsets. Each split is selected to minimize the variance within each resulting node, ensuring that the child nodes are as pure as possible.

Types of Decision Trees

1. **CART (Classification and Regression Trees):**
   - **Purpose**: CART refers to the decision tree algorithm that can handle both classification and regression tasks. It builds a binary tree by recursively splitting the data into two subsets based on a feature and a threshold.
   - **Key Feature**: CART only creates binary splits (i.e., it splits each node into exactly two branches).
   - **Splitting Criteria**:
     - For classification: Gini impurity or entropy.
     - For regression: Variance reduction (minimizing the squared error).

2. **ID3 (Iterative Dichotomiser 3):**
   - **Purpose**: This is an early decision tree algorithm used mainly for classification tasks.
   - **Key Feature**: It chooses the split based on the feature with the highest information gain, using entropy to measure the quality of splits.
   - **Limitations**: ID3 does not handle continuous features well and can only be used for categorical target variables.

3. **C4.5:**
   - **Purpose**: A more advanced version of ID3, used for classification tasks.
   - **Key Feature**: C4.5 can handle both categorical and continuous features and is more robust than ID3. It chooses the split based on **gain ratio**, a normalized version of information gain.
   - **Pruning**: C4.5 includes pruning techniques to handle overfitting by removing parts of the tree that do not improve generalization.

Advantages and Limitations

Advantages:

1. **Interpretability**: Easy to understand and visualize, making decision rules transparent.
2. **Handles Non-linear Relationships**: Captures complex patterns without requiring feature scaling.
3. **Versatility**: Can be used for both classification and regression tasks.
4. **Randomness in Random Forest**: Reduces overfitting and improves robustness by combining multiple trees.

Limitations:

1. **Overfitting**: Decision Trees easily overfit without proper pruning.
2. **Instability**: Sensitive to small changes in data, leading to different splits.
3. **High Computational Cost**: Random Forests can be resource-intensive with many trees.
4. **Interpretability Issues**: Combined Random Forest models are harder to interpret compared to a single tree.

Hypothesis Space

The hypothesis space of a **Decision Tree** consists of all possible combinations of feature splits and threshold values that can be used to partition the data. Each unique combination of splits forms a distinct path from the root to a leaf node, representing a specific decision rule. This results in a **non-linear hypothesis** that can capture complex relationships in the data.

For **Random Forests**, the hypothesis space expands further, as it includes multiple trees with different feature subsets and random splits, creating a diverse set of hypotheses. This broader hypothesis space allows Random Forests to achieve better generalization and robustness compared to a single Decision Tree.

Training Process

Decision Tree Training:

The training starts by considering the entire dataset at the **root node**. At each node, the algorithm evaluates **all possible features and thresholds** to find the split that best reduces the error (e.g., minimizing variance in regression). Once the optimal split is found, the data is divided into two groups — the **left** and **right** child nodes. This process is repeated **recursively** for each child node, forming a tree structure.

The training stops when a **stopping condition** is met, such as:

- The **maximum depth** of the tree is reached.
- The number of samples in a node is less than a specified **minimum sample size**.
- The split no longer improves the model (e.g., no significant variance reduction).

When the training stops, the node becomes a **leaf node** that holds the **mean** of the target values in that node, which serves as the **predicted value**.

Random Forest Training:

Random Forests train multiple Decision Trees independently to create a strong ensemble model. The process includes the following steps:

- **Bootstrapping**: For each tree, a **random sample** of the training data is created with replacement (some data points may appear multiple times while others are left out).
- **Random Feature Selection**: Instead of considering all features at each split, the algorithm selects a **random subset** of features, introducing randomness in the structure of each tree. This step was omitted in this training process.
- **Tree Building**: Each tree is trained in the same manner as a standalone Decision Tree, using the bootstrapped sample and the randomly selected features.

This process is repeated for a specified number of trees (n_estimators). During **prediction**, each tree makes its own prediction, and the final output is obtained by **averaging** the predictions (for regression) from all the trees.

Challenges and Limitations

1. **Overfitting**
   One of the biggest challenges I faced was dealing with **overfitting**. When the Decision Tree grows too deep, it starts to memorize the training data instead of generalizing well. I noticed that this led to high accuracy on the training set but poor performance on the test set. Controlling this required careful tuning of the **maximum depth** and **minimum samples per split**.

2. **Computational Complexity**
   Training large trees turned out to be more computationally intensive than expected. I initially chose the California housing data however due to the dataset being big, I was unable to build the tree in a short time. This is why I chose to go with the diabetes dataset due to its smaller size.

3. **Interpretability in Random Forests**
   While single Decision Trees were easy to understand and visualize, I found that interpreting a Random Forest was almost impossible. This was frustrating because even though the model performed better, it lacked transparency, making it difficult to explain how predictions were made.

4. **Scalability Issues**
   As I increased the number of trees in the Random Forest, the model's size and training time grew significantly. This was a practical reminder that even though adding more trees can improve accuracy, it comes at a cost to scalability, making it less suitable for real-time applications.

Implementation

The implementation was carried out using AI assistance to understand and code the models from scratch. The Decision Tree was built using a recursive approach to handle splits and define leaf nodes, while the Random Forest was constructed by creating multiple decision trees and combining their results. The implementation was tested on the **Diabetes dataset** due to its moderate size and suitability for regression tasks.

Toy Dataset: Diabetes Dataset from sklearn

The **Diabetes dataset** was chosen as it is specifically designed for **regression tasks**. The dataset has 10 input variables (numeric features) that are related to health metrics, making it ideal for analysing disease progression using decision tree-based models.

- **Features**: The dataset contains 10 baseline variables:
  - Age, Sex, Body Mass Index (BMI), Average Blood Pressure, and six blood serum measurements.
- **Number of Samples**: 442 diabetes patients.
- **Target Variable**: A quantitative measure of disease progression one year after baseline.
- **Loss Function**: Mean Squared Error (MSE) is used as the primary loss function for evaluating the model.

- **Performance Metrics**: Regression metrics like MSE are used as classification metrics such as F1-score or accuracy are not suitable.
- **Optimization Method**: Due to the moderate size of the dataset, complexity control techniques such as limiting tree depth are applied to avoid overfitting.

Implementation Breakdown

Step 1: Import Necessary Libraries

Imported libraries include NumPy for data manipulation and sklearn for loading the dataset. This is the foundation for handling data operations.

Step 2: Load the Dataset

The Diabetes dataset is loaded, and the features (X) and target (y) variables are separated for further processing.

Step 3: Create the Node Class for Decision Tree Structure

Defined a Node class to represent individual decision points (splits) in the tree. Each node stores the feature index, threshold value, variance reduction, and references to child nodes (left and right).

Step 4: Create the Decision Tree Regressor Class

- **Step 4.1:** Initialize Parameters
  Parameters like min_samples_split (minimum samples required for splitting) and max_depth (maximum tree depth) are initialized to control the tree's complexity and prevent overfitting.

- **Step 4.2: Splitting the Dataset**
  Implemented a function to divide the dataset into left and right subsets based on a given feature and threshold.

- **Step 4.3: Calculate Variance Reduction**
  This function calculates the reduction in variance after a split, serving as a criterion to evaluate the quality of each split.

- **Step 4.4: Get the Best Split**
  The algorithm loops through each feature and possible threshold to identify the split that maximizes variance reduction, storing it in a best_split dictionary.

- **Step 4.5: Build the Decision Tree Recursively**
  Constructs the tree using a recursive function, stopping when the maximum depth is reached or the minimum samples per split condition is met.

- **Step 4.6: Fit the Decision Tree**
  Combines the features and target into a single dataset and calls the recursive build_tree function to construct the tree.

- **Step 4.7: Predict Single Sample**
  Traverses the tree based on feature values of a single sample until a leaf node is reached, returning the predicted value.

- **Step 4.8: Predict Multiple Samples**
  Applies the single sample prediction function to an array of samples to produce batch predictions.

Step 5: Split Data into Training and Testing Sets

Implemented a custom function to split the data into 80% training and 20% testing sets, ensuring a fair evaluation of the model's performance.

Step 6: Train and Evaluate the Decision Tree Model

- **Step 6.1: Fit the Decision Tree on Training Data**
  Trained the Decision Tree using the defined min_samples_split and max_depth parameters.

- **Step 6.2: Make Predictions on Training and Testing Data**
  Generated predictions for both training and testing data.

- **Step 6.3: Implement Mean Squared Error (MSE) Function**
  Created a function to compute MSE, which measures the average squared difference between actual and predicted values.

- **Step 6.4: Calculate and Display MSE for Training and Testing Data**
  Compared MSE for both sets to detect overfitting or underfitting.

Step 7: Implement Grid Search for Hyperparameter Tuning

- Explored a range of values for min_samples_split and max_depth.
- Trained and evaluated models for each combination.
- Recorded the MSE and identified the optimal configuration with the lowest MSE.

Step 8: Final Model Training and Evaluation

- Built a final Decision Tree using the best hyperparameters (max_depth=5 and min_samples_split=15).
- Evaluated the final model on training and testing data, observing improvements in MSE.

Step 9: Implementing Random Forest from Scratch

- Created a RandomForestRegressorScratch class to build an ensemble of decision trees using bootstrap sampling.
- Trained multiple Decision Trees on different subsets of data.
- Averaged predictions from all trees to improve model stability and reduce variance.

Step 10: Train and Evaluate the Random Forest Model

- Trained a Random Forest with 10 trees using the optimal parameters (max_depth=5 and min_samples_split=15).
- Compared the Random Forest's performance with the single Decision Tree.
- The Random Forest achieved lower MSE, indicating better generalization and less overfitting.

Final Findings

- **Initial Single Decision Tree MSE**: 3697.63 (Before tuning)
- **Final Single Decision Tree MSE**: 3415.77 (After tuning)
- **Random Forest MSE**: 2911.49

The **Random Forest** model outperformed both initial and final Decision Trees by reducing overfitting and improving generalization. This result highlights the effectiveness of ensemble methods in creating more robust models.

Conclusion

The project successfully implemented and analysed Decision Tree and (a simplified) Random Forest models using the Diabetes dataset. While Decision Trees provided a solid foundation for understanding tree-based algorithms, they showed a tendency to overfit. Random Forests improved performance by reducing overfitting and delivering more robust predictions through ensembling. This demonstrates the effectiveness of combining multiple models to achieve better generalization.

Improvements

The implementation of Random Forest in this study was more simplified than the real random forest due to the omission of random feature selection. In the future we might get better results if we carry out random feature selection in our Random Forest model and it can make the trees more varied and increase randomness, reducing noise and MSE even further.