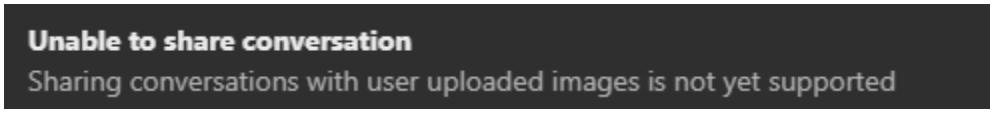
Project Topic: Implementation and Study of Decision Tree Regression Model (Option 1)

This project will discern the implementation and study of decision trees in a regression task. It shall focus primarily on low-level technical details of the model.

AI Acknowledgement: This study and implementation was undertaken with the assistance of AI, specifically ChatGPT. A link of the chat conversation cannot be provided; as uploaded images were contained throughout the conversation;



Google Colab link:
https://colab.research.google.com/drive/1QoHsKo_ruBKCZoZzNLL1NrjUJ6ow9hJo?usp=sharing

Introduction

Starting this project, I study the basics of decision trees with regards to machine learning by determining its definition, process, key components and goal. Sequentially, I then further study any techniques that could be applied in order to enhance my understanding and practicality of the technique during implementation. External materials as well as generative AI were utilised in order to build this foundational knowledge. Any material that was found confusing to me would be further clarified and studied in simpler terms to align my learning pace and ability.

Definition

Decision Tree regression is a type of supervised learning algorithm, in which the model is in the form of a tree structure with branches of choices (decisions). Within regression type models, the main concept is the prediction and computation of continuous values which are estimated from the relationship between a dependent variable and one or more independent variables.

Task/Objective

The ultimate goal of a supervised machine learning algorithm is to predict an outcome based on one or multiple inputs as accurately as possible. For classification problems, the goal is to predict a binary value, or categorical value (yes or no), whereas for regression problems (in this case), the outcome prediction is a continuous value.

Key Components of the Model

1. **Root node**

   Contains the entire dataset/sample, which will split into the further nodes

2. **Interior node**

   Represent the features of the dataset

3. **Leaf node**

   Indicates the outcomes and predictions

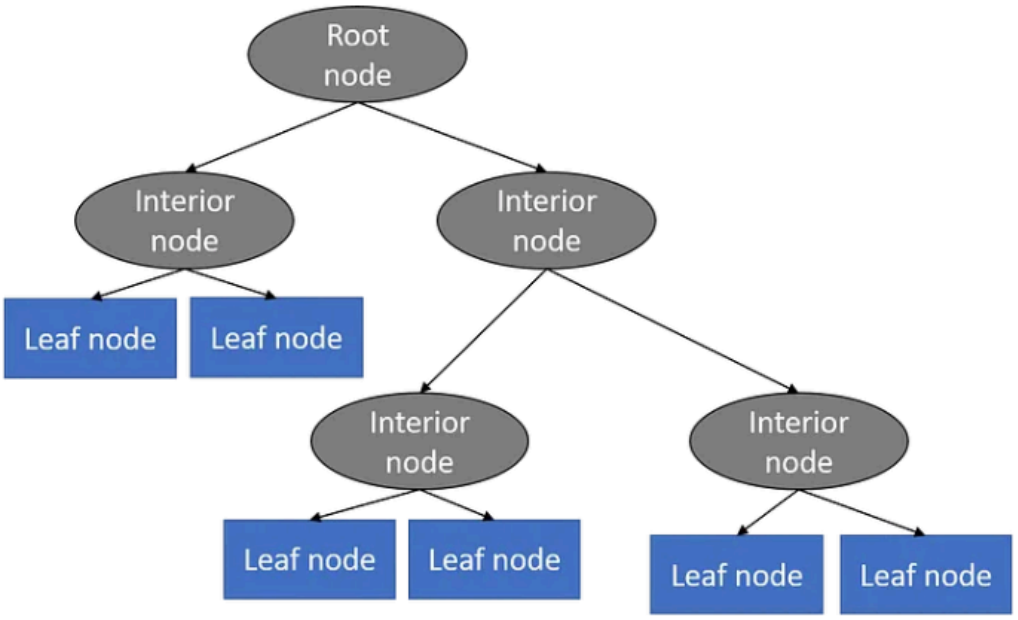
4. **Splitting criteria**

   Denote the branches which determine the rules of how the tree splits at each interior node

5. **MSE and Variance (Loss Function)**

   Mean Squared Error measures the error between the predicted and actual target value. Variance measures the spread of the actual data points from the average of the entire sample.

Naturally, the decision tree algorithm is visualised as a tree-like structure with three types of nodes; **root**, **interior**, and **leaf** nodes. Root nodes are the initial nodes containing the entire sample, which split further into the latter nodes. Interior nodes represent features of the dataset, and leaf nodes indicate the outcomes. All types of nodes are then connected by branches, which represent the decision rules.



(1)

In order to improve the outcome to be as accurate as possible, the decision tree algorithm aims to recursively partition data into subsets based on feature values based on a decision rule applied to the feature at each node. Each node splits into two more nodes until target values in each subset are as accurate as possible.

Splitting Criteria

Splitting criteria are the branches which determine the rules of how the tree splits at each interior node. Within regression problems, the two most common criteria include the aforementioned **Variance** reduction or **Mean Squared Error** (MSE), which are also known as loss functions. They measure the error between the predicted values and the target values, and the splitting criteria follows that the data should be split in a way that the loss function is minimised.

Difference between Variance and MSE

The measure of variance dictates the spread of the actual data points from the average of the dataset; this only concerns the original data itself. On the other hand, the MSE dictates the error between the predicted and target values, ultimately reflecting on the model's performance.

Goal

The goal of the decision tree algorithm is to find the best fit feature and corresponding threshold, which will separate the data into subsets that correspond to the target value as accurately as possible. This kind of data separation happens over several iterations of branching until the leaf nodes more accurately predict the target.

Minimising variance is important throughout the branches and subsets as it enables the model to provide a more reliable prediction. With lower variance, the data points within a subset would be more homogenous; this further leads to the target values being more consistent within the training. Without the minimisation of variance throughout the splitting process, the data in the subsets will remain just as diverse as within the root node. This defeats the entire purpose of the decision tree algorithm, which aims to distinguish the most meaningful relationships between data points to correctly predict a value.

In order to specifically determine the goal being reached, the measure of Mean Squared Error is used to ultimately assess the model's performance. As such, it could be stated that the goal is to reduce the Mean Squared Error as much as possible throughout the project.

Process

Throughout the machine learning process, there are two main phases;

- **Training**
    - Begins at the root node and recursively split the data based on a splitting criterion.
    - The algorithm should evaluate all possible splits for each feature, and reduce the variance as much as possible in the case of a regression problem.
    - The recursive splitting should stop once a stopping criteria is met. These stopping criteria typically rely on the maximum depth of the tree, a threshold for the variance reduction, or a minimum amount of data points left in a node.

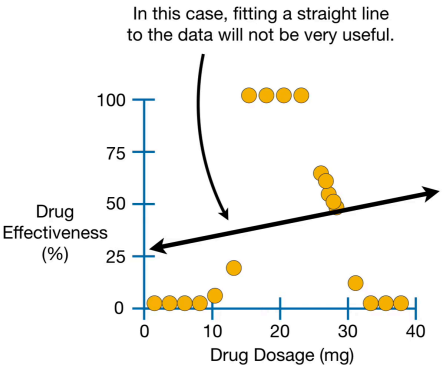
- **Validation and testing**
    - Throughout training, a subset of the training data is set aside for validating the performance of the model. This helps to tune hyperparameters to avoid overfitting, which include the maximum depth of the tree, minimum samples per leaf node, and the max amount of features to consider during splitting.
    - Furthermore, the model is then trained on an addendum testing set of data, separate from that of the training and validation data. This provides the measure of how the model performs on completely unseen data, such as in the form of Mean Squared Error (MSE).

Advantages/Limitations

Garnering a deeper understanding in a short form also requires knowledge of the benefits and downsides of the model. This ensures that any setbacks may be properly addressed during the implementation of the project.

- **Advantages**
  - One of the initial mentions throughout this journal notes the simplicity of decision trees, which provide a clear and intuitive visual model. This enhances the interpretability of the model from those with limited knowledge.

  - Decision trees can also address non-linear relationships between features and target variables, such as depicting the effectiveness of drug dosages with a bell curve.



(3)

- **Limitations**
  - Overfitting occurs when the decision tree model becomes too complex and focuses on noise, rather than the important underlying patterns within the data. Decision trees tend to grow at a staggering rate if not controlled, and split the data into increasingly smaller subsets, which can be so overly specific that the variance increases.

  - Additionally, decision trees may be biassed towards the majority class unless class weights are adjusted. One solution to this bias is to balance the dataset, so that there are no majority classes.

Hypothesis Space

The hypothesis space of decision tree regression models refer to all different possible trees the model can choose from during training. The primary concepts of the hypothesis space within the decision tree regression problem lie within the ultimate tree structure, splitting criteria (branches), node assignments, constraints/parameters and the amount of possible trees. Ultimately, the aforementioned concepts define the maximum potential solutions that the model may consider; a hypothesis space. It is then the loss function's purpose to evaluate the performance of different hypotheses, by comparing the predictions against the actual values (from model world to the real world). Such an evaluation also acts as a 'probe' to exploring the

hypothesis space, as it acts as an informant to the necessary parameter adjustments for the model to change.

Loss Function

In machine learning, loss functions play a key part in determining the quality of a model. In the case of a regression problem, we measure the Mean Squared Error of a model (error between predictions and actual target values) in order to assess the performance. However, classification tasks measure 'impurity' such as Gini Impurity or Entropy, which denote the randomness of incorrectness.

- Loss functions are important as the measures
    - Guide the model's learning during training, especially if techniques such as cross validation are implemented
    - Provide feedback on the model's performance, which is then utilised to adjust for further improvements.

Training process

Training is arguably the most important part of the machine learning process. Decision trees begin with combining all of the input data at the root node, and recursively splits the data with the goal of minimising the variance. Minimising the variance refers to minimising the spread of the values from their averages. With lower variance, the data points are closer to the average in their respective subsets, which ultimately means the data points are similar for that subset. As such, with similar values, predictions can be made more effectively in comparison to subsets with higher variance. In other words, this enables the model to focus more clearly on a particular set of inputs and outputs, rather than a more complex set, leading to less confusion.

Once the algorithm evaluates all possible splits for each input feature, the split with the best minimisation in variance is chosen. The process of splitting then repeats for the chosen split (subset) until a stopping criteria is met. Stopping criterion refers to the maximum depth of the tree being reached, or the reduction in variance being below a specified threshold, or the minimum number of data points leftover in a subset. When the tree has then stopped growing, the final nodes (leaf nodes) then create predictions, which are typically the average of the target variable from that node.

Hyperparameter Tuning

Throughout the training process, there are certain techniques that may be applied in order to provide better results and performance. One such technique is hyperparameter tuning, which is the process of finding an optimal set of parameters for the model in order to increase its performance. Typically, the performance and accuracy is measured by Mean Squared Error (MSE); hyperparameter tuning cross validates the MSE of different combinations of parameters, and chooses the best parameters which output the best result.

Pruning

Pruning is an addendum method which refers to the removal of branches and nodes. This method prevents overfitting, and also reduces the complexity of the tree given that it is performed sequentially after the tree is built. Pre pruning, or early stopping, is the concept of preventing the tree from growing once it has reached a maximum depth. This is a parameter that can be tuned prior to the tree being built, and can be adjusted accordingly. Furthermore, post-pruning refers to removing the nodes which are too complex (2). This is typically used for much larger trees, and has proven to be ineffective in my testing.

Cross Validation

Cross validation is the final method that I applied during my implementation in order to increase the performance of the model. The process follows the idea that the dataset is partitioned into subsets, some of which are used to train the model, with the remainder used to validate the model. This process is then repeated multiple times in order to obtain a more reliable estimate of the model's performance. Ultimately, this should improve the predictive performance of the model as it ensures that the model is validated thoroughly (due to the process being repeated multiple times) before final evaluation on the testing data initially allocated.

Challenges

Throughout this project, I encountered a handful of problems.

Dataset problems

First of all, finding a dataset to showcase the effectiveness of the model was an initial problem I encountered. I began using the diabetes dataset from scikit-learn's toy datasets to build the initial model and apply techniques, before switching to another dataset with much lower variance in the target variable. This made no difference to the goal or task at hand, but it made visualising the techniques and their effectiveness significantly easier. When performing hyperparameter tuning on the diabetes dataset initially, the best maximum depth of the tree was only 3; this resulted in a huge amount of error overall to the extent that any other techniques implemented to improve the performance visually showed little difference.

Overfitting

One of the most common causes of overfitting is the tree growing to extreme limits. When a model overfits, it results in its growing complexity by learning the noise of the data rather than any underlying patterns. As such, it would result in higher accuracy only on training data, but poor performance on unseen data (test or validation sets). In order to overcome this challenge, parameters such as the max_depth and min_sampes_split and min_samples_leaf were adjusted accordingly to improve the MSE measure.

Implementation

Dataset is from UC Irvine Machine Learning Repository, 'Airfoil Self-Noise' (see appendix). It concerns predicting the noise (in terms of decibels) generated by an airfoil, based on aerodynamic and physical properties.

Name	Role	Type	Units
Frequency	Feature	Integer	Hz
Attack-angle	Feature	Binary	Deg
Chord-length	Feature	Continuous	m
Free-stream-velocity	Feature	Continuous	m/s
Suction-side-displacement-thickness	Feature	Continuous	m
Scaled-sound-pressure	Target	Continuous	dB

Base decision tree model, explained in steps

Step 1

Importing necessary libraries, numpy, matplotlib, pandas, etc

Step 2

Loads the dataset and extract features, separating them into input features and target variable

Step 3

Split the data into training and testing sets (80%, 20%)
Training and validation are combined in the 80%, and 20% of that value is used as the validation set. As such;
64% for Training
16% for Validation
20% for Testing

Step 4 (Most important step)

Node class

Node class defines some variables necessary for the decision nodes; feature_index, threshold, left, right, var_red.
Value is defined for the leaf nodes.

Tree class

Tree building method, variance reduction method, fit method, predict method, getting the best split, etc

Also includes a constructor for the class, with min samples split, max depth, and min samples leaf as its variables. These are needed for the stopping criteria.

fit method

Trains the tree by calling the internal build_tree function, and combines the features and target into a single dataset for easier manipulation.

build_tree method

First, take the dataset and separate the features from the target variable. Then recursively builds the decision tree, and splits the data at each node based on the best feature and threshold. Stops building the tree if a stopping criteria is met.

get_best_split method

This method finds the best feature and threshold which results in the greatest reduction in variance.

split and variance_reduction method

As the name implies, the split method splits the dataset into subsets, where the feature value is less than or equal to the threshold in one subset, with the latter being greater than the threshold. The variance reduction method calculates the reduction of variance regarding the target variable, which is used to determine the quality of the split.

make_prediction function

Takes a single data point and takes the corresponding y value to that data point, by starting at the root node and traversing through the tree until a leaf node value is returned.

Step 5

Initialise the regressor with some parameters, fit the model on the training set, and make predictions. Then, calculate the MSE (loss function), and create some visualisations.

Adding Hyperparameter Tuning and Cross Validation in Additional Models

The model containing hyperparameter tuning requires an extra import in step 1 (GridSearch), and also features some extra methods in step 4 to get parameters from a parameter grid, and set them, before creating trees. It then compares the MSE of all trees constructed, revealing the tree with the best performing combination of parameters.

The final model additionally contains cross validation, which required importing GridSearchCV and initialising it as seen below.

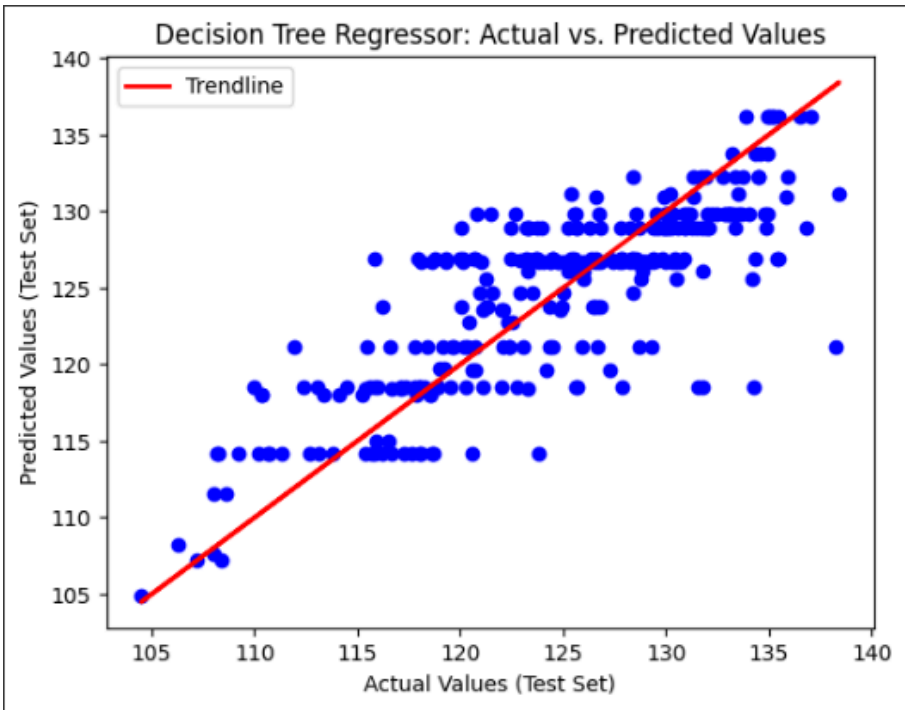
```
# Create the GridSearchCV instance with cross-validation
grid_search = GridSearchCV(estimator=regressor, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', verbose=1)
```

cv=5 specifies the 5-fold cross validation, referring to splitting the dataset into 5 subsets, with training on the first 4 sets and validation on the remaining set. This process is then repeated 5 times with each subset taking turns at being the validation set.

Results

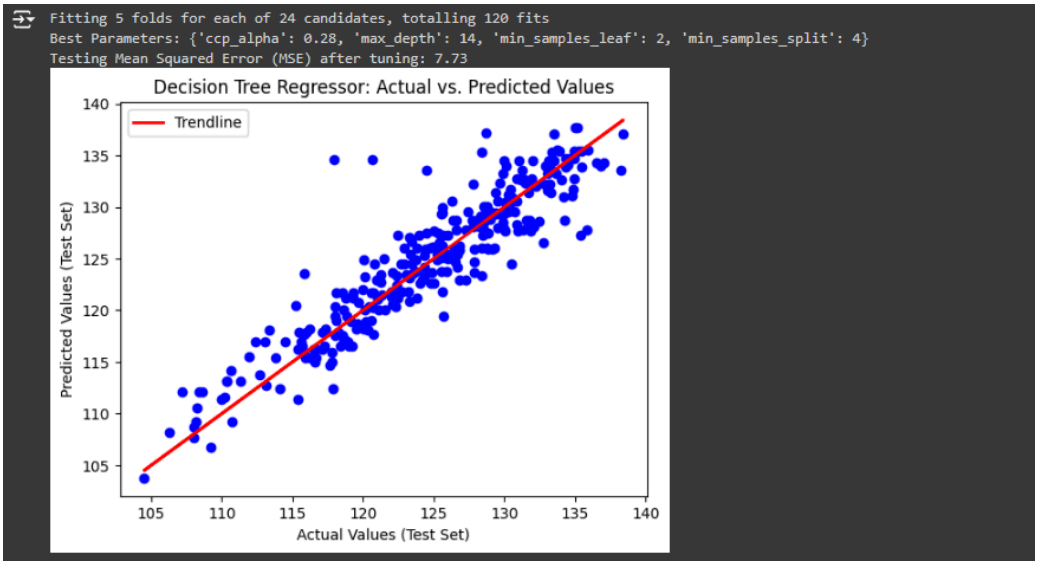
Base model

The base model had a MSE (loss function) of 17.14. The following plot line reveals the predicted values against the actual values, where the blue dots are computed against the trendline to create the loss value.



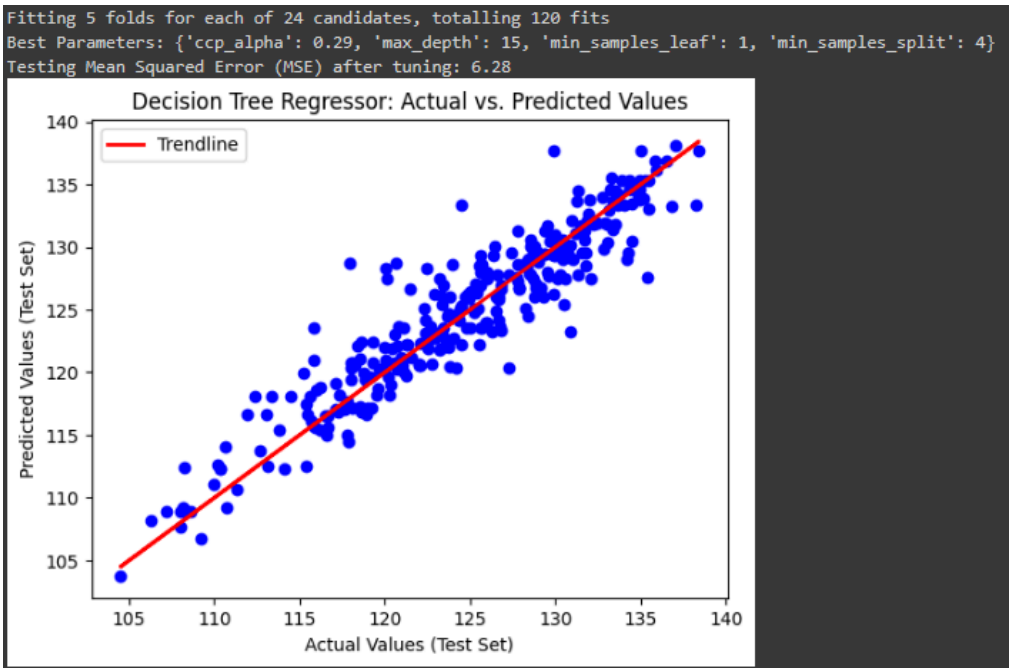
Base model + Hyperparameter Tuning

The model with hyperparameter tuning implemented scored significantly better than the previous model, with a MSE of 7.73. The best parameters found included an alpha value of 0.28, max depth of 14, 2 minimum samples per leaf, and 4 minimum samples per split.



Base model + Hyperparameter Tuning + Cross Validation

The final model scored an MSE of 6.28; performing the best out of all models. Interestingly, this model found slightly better parameters than the previous when cross-validation was implemented.



More on hyperparameters

The hyperparameters in the grid were adjusted accordingly to produce the best result I could. Each parameter started with 3 values which were adjusted with smaller variance in order to find a more ideal value. For example, max depth started with 3, 5 and 7, and I found that it performed better with a higher max depth. I then adjusted the hyperparameters to a new set of numbers until I found the best parameter to be the value in the middle, and slowly adjusted the other values to find the one that truly performed the best.

Appendix

UC Irvine - Dataset
https://archive.ics.uci.edu/dataset/291/airfoil+self+noise

References

(1) Schlagenhauf, T. (2022, February 17). *Decision trees in Python*. Python Course. https://python-course.eu/machine-learning/decision-trees-in-python.php

(2) Scikit-learn. (n.d.). *Post pruning decision trees with cost complexity pruning*. Scikit-learn. https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html

(3) Starmer, J. (2020, February 18). *Decision trees, clearly explained!* [Video]. YouTube. StateQuest with Josh Starmer. https://www.youtube.com/watch?v=g9c66TUylZ4&t=107s