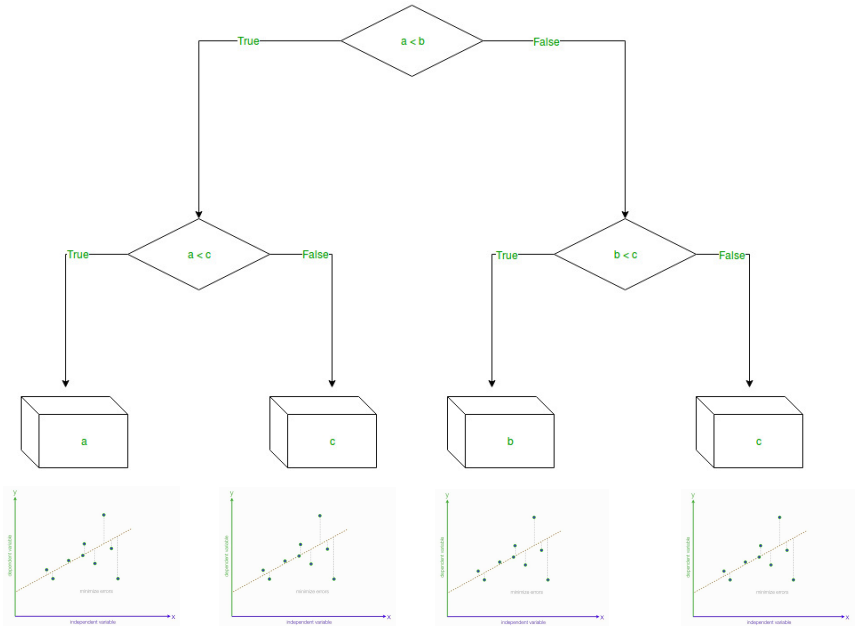
Arun Krishnavajjala
G# 01063551

Enhanced Decision Tree Regressor

Introduction:

The idea of an Enhanced Decision Tree Regressor is a derivative of the Decision tree regressor. The decision tree regressor on its own, creates a tree based on training data and then has final leaf nodes that average the values of the training data and return the estimated value in thee leaf. This is not as efficient because similar vectors, with slightly deferring features will end up in the same leaf, and as a result, return the same value through the decision tree regression. This is not efficient because multiple vectors will end up with a regression value that is the same. To solve that issue, we use a linear regression model within the leaf nodes to produce an individual unique regression value for every test input.

Diagram:



Dataset:

The dataset was obtained from Kaggle's trip duration prediction contest. The dataset had columns: *id* *vendor_id* *pickup_datetime* *dropoff_datetime* *passenger_count* *pickup_longitude* *pickup_latitude* *dropoff_longitude* *dropoff_latitude* *store_and_fwd_flag*

I only used the Vendor_id, passenger_count, pickup_longitude, pickup_latidude, dropoff_longitude, dropoff_latitude.

I used those because the id for each trip was unique so therefore, I avoided it. I avoided the time of pickup and drop off because that seemed like the prediction goal and I decided not to

use it. I did not use store_and_fwd_flag because I wanted to keep a limited feature set. I finalized on those 5 features listed above.

Training:

For training, we were used a train test split of the training data. I split it into 80% training and 20% test. I stored each in their own data frames.

Building the Decision Tree Regressor:

The first model I trained was a decision tree regressor.

```
# Now create our decision tree
dtr = DecisionTreeRegressor(maxDepth = 10, minInfoGain = 100, minInstancesPerNode = 30, maxBins = 200
).setFeaturesCol("features").setLabelCol("trip_duration")
```

This code creates the tree and sets the features as “features” and label as “trip_duration.” This is looking for specific columns to train with. It has a max depth of 10 and some other parameters that I tuned to try and minimize the RMSE score.

```
# Train the model using our training data
model = dtr.fit(trainingDF)
```

this code trains the model. By doing dtr.fit(trainingDF), it makes the tree based on the training data frame and tries to make it so that the tree created is an accurate representation of the data given.

This tree is then used to predict the values from the test data frame. The test data frame Is run only through the tree in this instance.

```
collectionTest = model.transform(testDF)
```

This code stores the test predictions in collectionTest for use later on.

Building the Linear Regressor:

To build the regressor, I first had to filter the leaf nodes from the tree. I first ran the training data through the pre-built tree and stored it in a data frame. The reason I did this was, the tree is built off the training data and has seen the exact data before. By running the training data through it, it was just going to give me all of the values predicted for each vector in each leaf node. I used the values predicted and put all the distinct values into a list. This list consisted of all the unique individual leaf nodes in the tree.

Arun Krishnavajjala
G# 01063551

```
leafNodes = []
for i in cPred:
   if i not in leafNodes:
      leafNodes.append(i)
```

This code is what I used to store all the leaf nodes values. This stored the unique values of each leaf. cPred is the data frame that had predictions of all the training data.

After storing the leaf nodes, I had to get all the vectors in each leaf to make the regression model. To do this, I used the data frame filter function to extract the feature vectors from the Training predictions that all had the same prediction/leaf node. By filtering with a certain value, I was able to get all the vectors from a specific leaf node and therefore create a regression model using that. I used the features column and the "trip_duration" column to make the linear regression model that would create a regression for the filtered test data set.
The test dataset was run through the decision tree regressor and outputted a data frame of regression elements. I then filtered the prediction column of the testDF using the leaf node prediction and ran the filtered data frame through the regression model.

Pseudo Code:

For i in leaf Nodes

FilteredTrain = Filter the training data with prediction = i
FilteredTest = Filter the test data with prediction = i

Model = linearRegression (train with FilteredTrain)
predictWithModel = model(filteredTest)

add predictWithModel to the larger dataframe, to calculate RMSE later

Regression Metrics:

For metrics, I used RMSE and MSE to determine the accuracy of the tree. My version of spark was outdated for some odd reason, so I found that I had to implement the RMSE and MSE functions by myself. To calculate the RMSE is just the square root of the MSE. The MSE was quite simple to calculate it's the difference between prediction and actual value, squared and added. Then all divided by the number of items calculated.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Arun Krishnavajjala
G# 01063551

Pseudo Code:

```
TotalSum = 0
Count = 0
Go through predictions:
        Sub = prediction – original
        Square = sub ^ 2
        TotalSum += Square
        Count += 1
MSE = TotalSum / Count
RMSE = √ TotalSum / Count
```

This is the basic manifestation of the formula and is quite simple to implement. It was time efficient and provided a good metric to make comparisons with.

Cross validation:

As my older version of spark did not allow the usage of cross validation packages, I had to create my own cross validation. I ran all the code through a loop that iterated 10 times and I averaged the MSE and RMSE scores over the 10 different runs. I did random initializations of the 80% and 20% split and did the algorithm 10 times to get an average approximation of how much better the enhanced decision tree was than the regular decision tree.

Pseudo Code:

```
totalMSE = 0
Repeat 10 times:
        Do decision tree
        Do linear regression
        Add MSE to totalMSE
totalMSE = totalMSE / 10
totalRMSE = √totalMSEE
```

Results:

The results showed that the RMSE and MSE were better for the enhanced tree. The RMSE for the enhanced decision tree regression method was always a few hundred points better than the RMSE for the normal tree. Given below is one of the runs from the program and shows that the Enhanced tree's RMSE is better by about 281.

Arun Krishnavajjala
G# 01063551

```
None
MSE TREE: 1975914.8208786102
MSE ENHANCED: 1264573.6180367402
RMSE TREE: 1405.6723732358869
RMSE ENHANCED: 1124.5326220420377

In [15]:
```

There were many runs like this and therefore this enhanced tree was better overall.

Conclusion:

This project allowed us to explore the idea of a decision tree and how to make it better using the idea of the enhanced decision tree. This taught us how to analyze a popular algorithm and make a few changes to it to make it significantly better. Given the state of the dataset, the RMSE values were destined to be very inaccurate from the start, but even through this dataset, we were able to find improvements in the model and predict a regressed value with higher accuracy.