# AdaBoost

AdaBoost, short for Adaptive Boosting, is an ensemble learning method used for classification and regression tasks. AdaBoost is particularly effective in creating a strong learner by combining the predictions of multiple weak learners. Imagine you are preparing for an exam and you ask for help from multiple classmates, each of whom might be good at different topics. AdaBoost works by combining the strengths of these multiple classmates (weak learners) to create a strong team (strong learner) that performs well.

Suppose we have the following dataset:

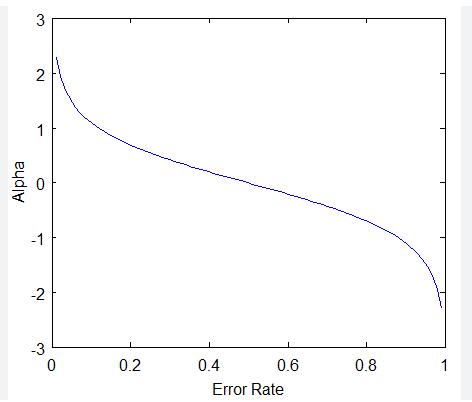
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **X1** | **X2** | **Y** | **Y\_pred** | **Initial Weight** | **Updated weights** | **Normalized**  **(To sum the**  **value as 1)** | **Range** |
| 3 | 7 | 1 | 1 | 0.2 | 0.16 | 0.16 | 0-0.166 |
| 2 | 9 | 0 | 1 | 0.2 | 0.24 | 0.25 | 0.166-0.416 |
| 1 | 4 | 1 | 0 | 0.2 | 0.24 | 0.25 | 0.416-0.666 |
| 9 | 8 | 0 | 0 | 0.2 | 0.16 | 0.16 | 0.666-0.832 |
| 3 | 7 | 0 | 0 | 0.2 | 0.16 | 0.16 | 0.832-1.0 |
|  |  |  |  |  | 0.96 |  |  |

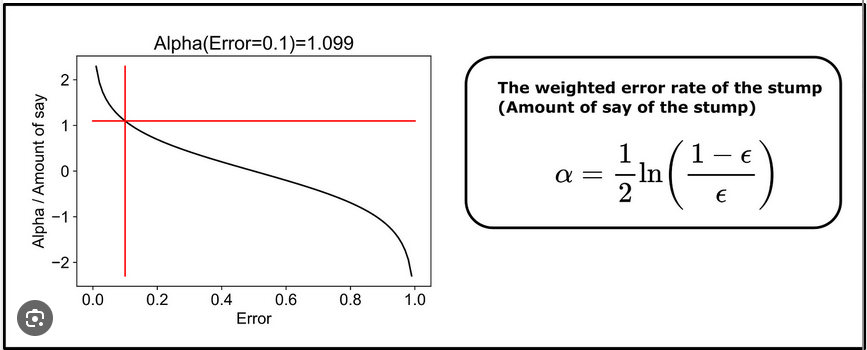
n=5 (Number of observations)

In Ada boost we assign weight to each data point. Suppose we give weight of 1/5 weight (1/n).

We train one decision tree on this, typically with one decision stump. We select a decision stump that performs well, and we calculate the alpha value (α) based on the error rate between the actual and predicted values.

We need to find the error rate in such a way that when the error is 0, the α value is positive, when the error is high, the α value is negative, and when the error is 50%, the α value is 0. The formula to get the α value in the graph format is as follows:





Error rate = 0.4 (Sum of two misclassified output)

Then alpha value = = = 0.20

We update the weight of misclassified points by adding more weight to misclassified points and reducing the weight to correctly classified points.

Weights for misclassified points:

New\_weight =

Weights for correctly classified points:

New\_weight =

Randomly generate 5 random numbers between 0 and 1

0.13, 0.43, 0.62, 0.50, 0.8

Based on the above random numbers, we select the specific row to do the up-sampling. This process is used to create the decision stump and is done iteratively until it reaches the number of stumps we have provided.

AdaBoost is a simple algorithm; hence, it has very few parameters.

Base\_estimator: The base\_estimator can be any algorithm other than a Decision Tree. By default, it is a decision tree with a max\_depth of 1 for better results. KNN is not supported.

N\_estimators: The number of base models.

Learning rate: It is the shrinkage parameter. If we have a high number of n\_estimators, there is a possibility that the model might overfit the data. In this case, if we reduce the learning rate, the weights would not change drastically, resulting in reduced overfitting.

# Gradient boosting

Gradient Boosting is an ensemble machine learning technique that sequentially combines predictions from multiple weak learners (typically decision trees). The primary objective is to enhance overall predictive performance by optimizing the weights of each model based on errors from previous iterations, progressively reducing prediction errors and improving accuracy.

## Key concepts:

1. Mean prediction (Model 1):
   * The first model produces the mean of the target variable as the initial prediction
   * The performance is evaluated using a loss function, measuring the difference between actual and predicted values.
2. Residuals and Decision trees (Model 2 and so on):
   * Subsequent models address the errors of the previous ones
   * Residuals(errors) from Model 1 are used to train the subsequent decision trees
   * Each tree corrects the errors of the previous ones by predicting the residuals
3. Learning rate:
   * A learning rate is introduced to control the impact of each model on the final prediction.
   * It avoids overfitting by reducing the weight of each model.

## Gradient boosting Regression

Fig 2.1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **IQ** | **CGPA** | **Salary** | **Mean (Pred 1)** | **Residual 1** | **Pred 2** | **Residual 2** | **Pred 3** | **Final prediction** |
| 90 | 8 | 3 | 4.8 | -1.8 | -1.8 | -1.62 | -1.62 | 4.7658 |
| 100 | 7 | 4 | 4.8 | -0.8 | -0.8 | -0.72 | -0.72 | 4.7848 |
| 110 | 6 | 8 | 4.8 | 3.2 | 3.2 | 2.88 | 2.88 | 4.8608 |
| 120 | 9 | 6 | 4.8 | 1.2 | 1.2 | 1.08 | -1.08 | 4.8012 |
| 130 | 5 | 3 | 4.8 | -1.8 | -1.8 | -1.62 | -1.62 | 4.7658 |

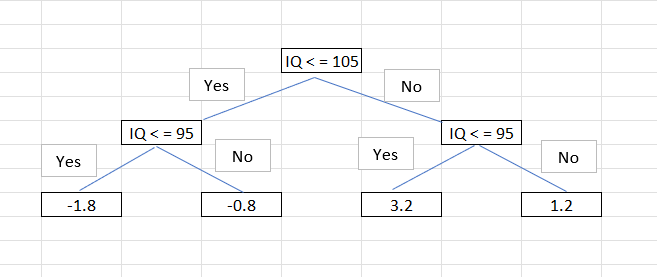
We will use three base models for understanding. We have to build model one, and in it we will take the mean of the output in regression problem. To evaluate the performance, we need loss function. Here the loss function is actual – predicted (pseudo residual). Now in model 2, we will take all the independent variable as input variable and the output variable would be the residual to predict the errors made by the model 1.

When we only have 2 model in Gradient boosting then the output will be the sum of output made by the model 1 and model 2. This is nothing but the overfitting issue. Hence, we have learning rate in place to avoid the overfitting in model. The output of the second model would be as follows:

Y = M1(output) + Learning rate \* M2

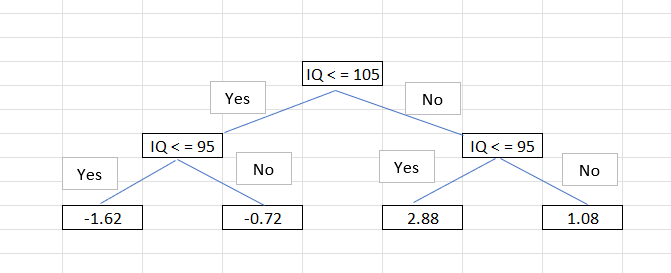
Considering learning rate as 0.1.

**Fig 2.2: M2 Decision tree:**



As we can see in the Fig 2.1 the residual is moving towards 0 in “Residual 2” column.

**Fig 2.3: M3 Decision tree**



As we reached the maximum number of iterations as 3, the final output would be as follows:

Final output = M1 (Output) + M2(Output) \* 0.1 + M3 (Output) \* 0.1

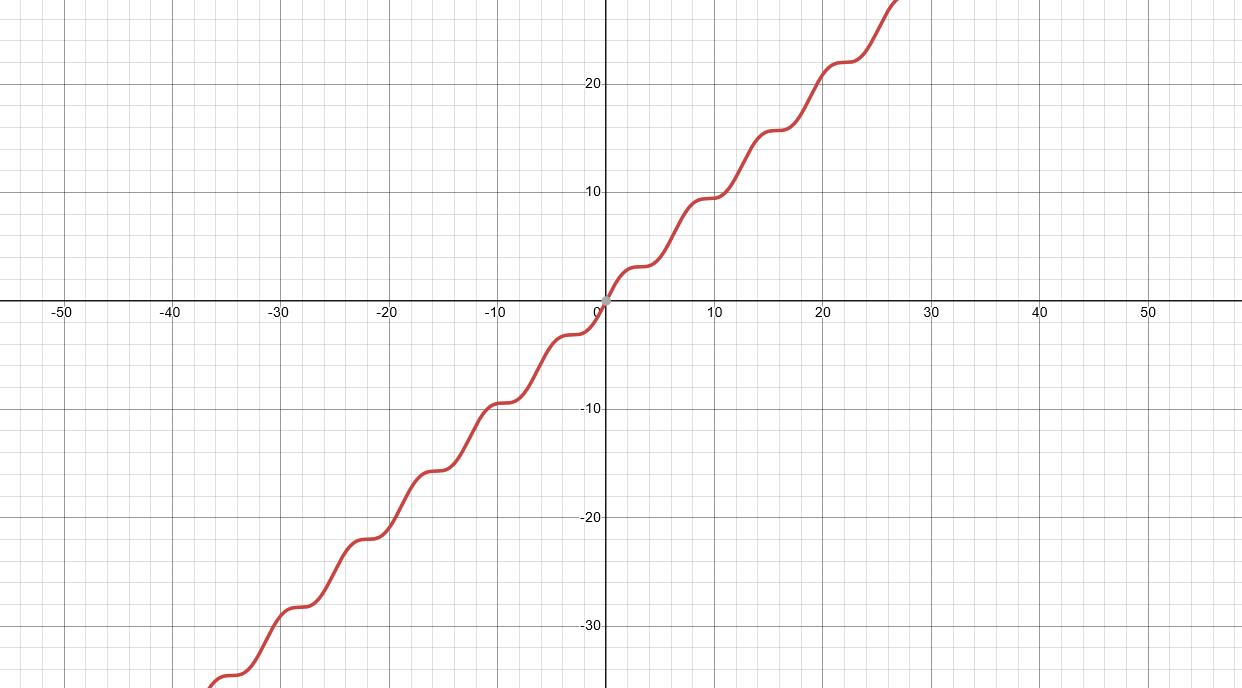
#### Comparison with AdaBoost:

1. Similar to AdaBoost, Gradient Boosting uses decision trees but with a more flexible structure (8-32 terminal nodes).
2. Learning rates are static in Gradient Boosting, unlike AdaBoost, where they dynamically adjust weights.

#### Mathematical intuition for Gradient boosting:

Additive modelling is applied stage wise in all boosting learning model. Machine learning model is a function. Let’s say we have x independent variable and y is target variable then the y=function of x

y = *f*(x)



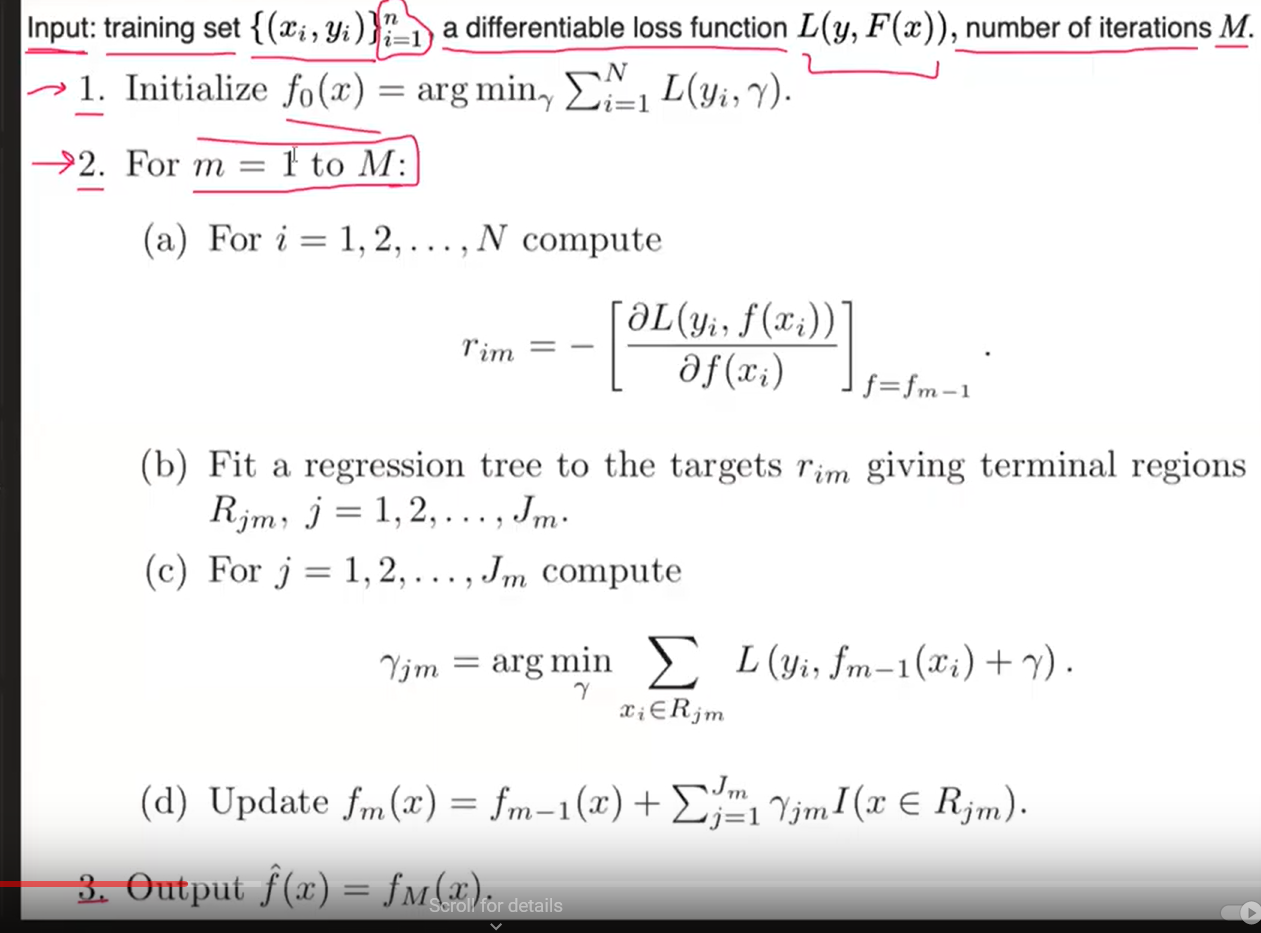
There are many instances where we would not be able to find the best fit line in regression or by providing polynomial degree to solve the regression problem. The problem is shown in above figure.

In additive modelling the function is the aggregate of smaller functions of multiple base models.

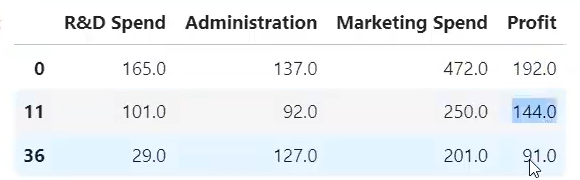
**Loss function: It’s a mathematical function which explains the error made by the model.**

In gradient boosting we can select any loss function only condition it has to be differentiable.

Here we will take loss function as (half of least square). We have taken ½ to ease the chain rule because the exponent 2 will be cleared with ½



Step 1 will build the first model which is nothing but which is nothing but We have to find the value of Gamma to find the minimum value of the expression ² (Optimization problem)



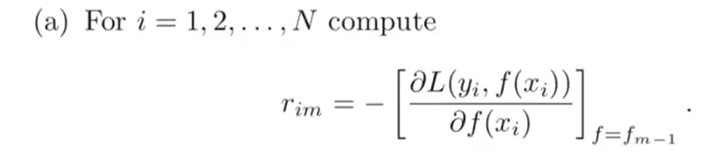
For above example the gamma value would be as follows:

(

This is nothing but the mean of the output column. Hence, its followed by AdaBoost as well as Gradient boost and XGBoost as well.

###### Step (a) of step 2:

*i* represents the row number and m represents the nth number of trees. r is nothing but the pseudo residual.

**

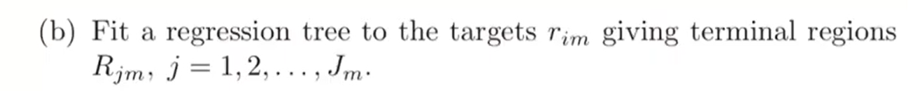
For row number 1 and Decision tree 1 it will be represented as below:

Hence, the final equation would be:

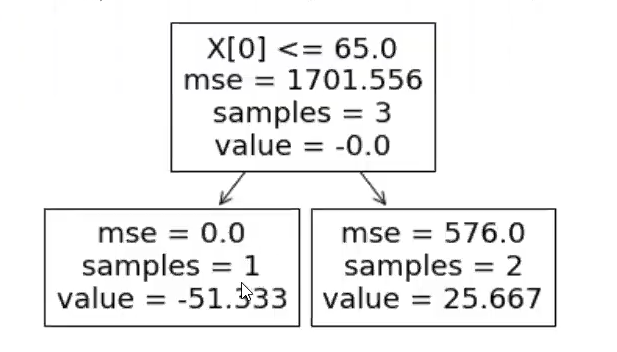




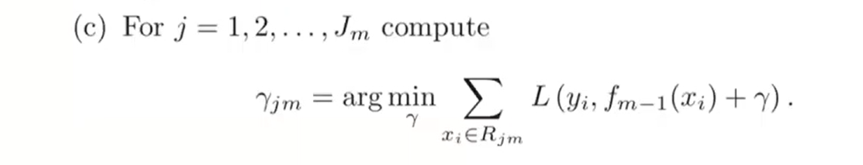
###### Step (b) of step 2:



Based on the independent variables and residual errors we have to build another decision tree. We will get the terminal regions. Terminal regions are the number of partitions or number leaf nodes present in Decision tree. Assume we get only two leaf nodes for the above dataset example then it will be represented as for first terminal and for second terminal. The values are nothing but the mean value of output received from both the terminal.



###### Step (c) of step 2:

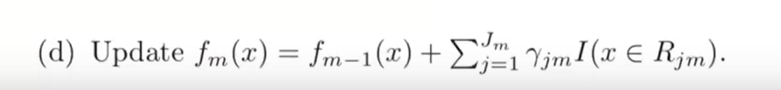


After getting the output of Decision tree, we have to calculate the output of Gradient boosting by the above formula:

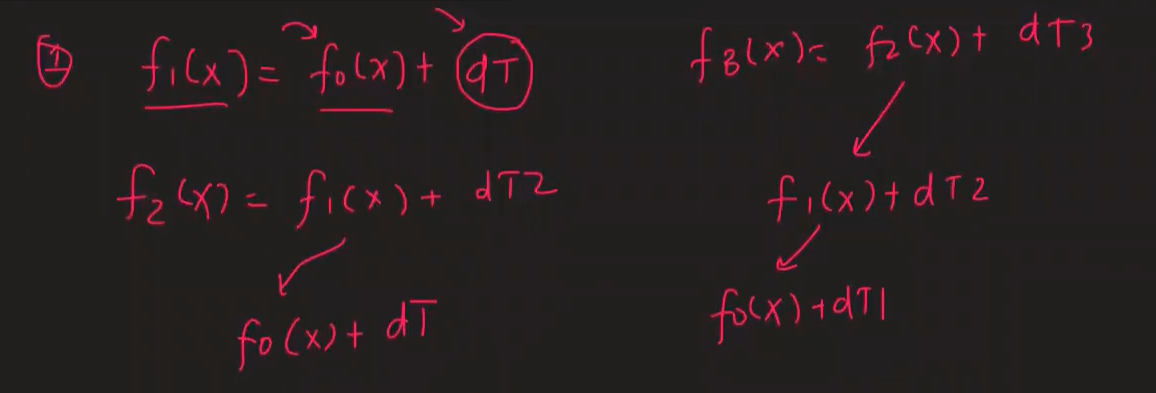
For terminal 1 the equation would be as follows:

The output of the decision tree and Gradient boosting is same the reason is the loss function. What we have used in above scenario is least square. However, if another loss function was used it could have provided different output.

###### Step (D) of Step 2:

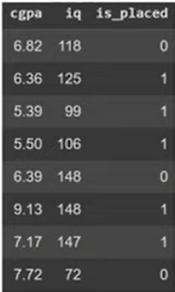


The output would be the sum of all the decision trees:

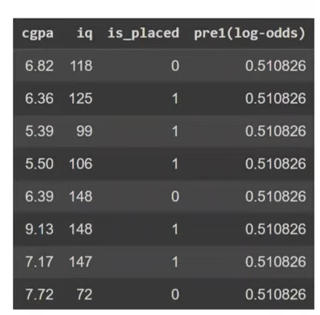


## Gradient Boosting Classification:

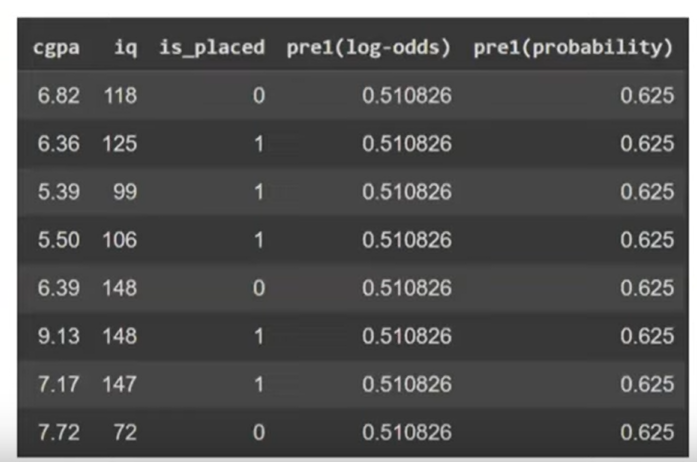
The only difference between the Gradient Boosting Regression and Classification would be the loss function. It would be working in similar fashion how regression works in gradient boosting model. In below problem, we would be building 3 model. However, In Classification problem the first model would be the simple model and the rest 2 would be decision tree. In first model, the log(odds) would be calculated based on log (odds) its is nothing but the count of 1 by count of 0.



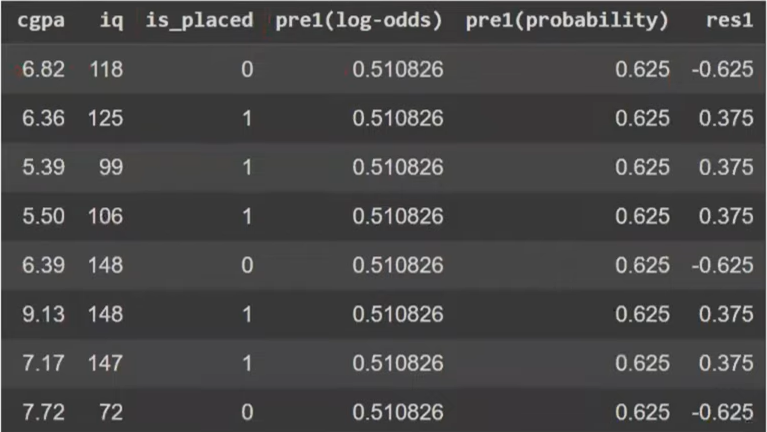
So

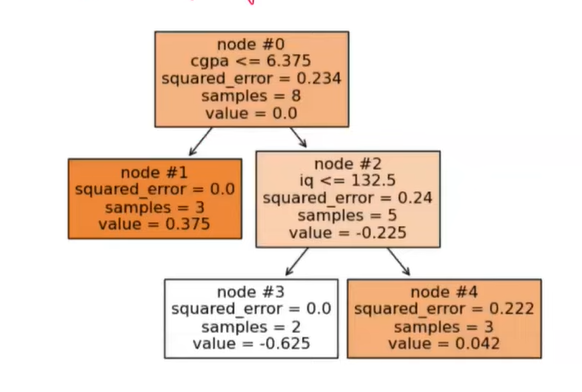


0.51 is the log(odds) from this we need to derive the probability value. The formula for probability is as follows:



Based on the above predicted values we have to find the pseudo residual. It would be the difference between the actual value – predicted value.

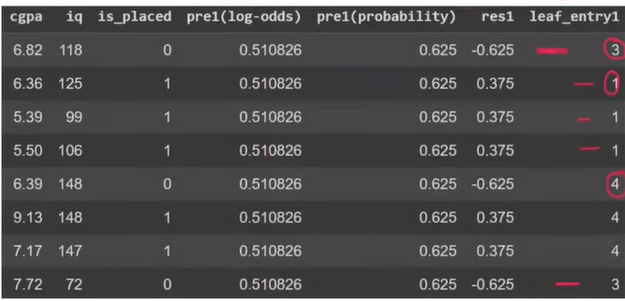




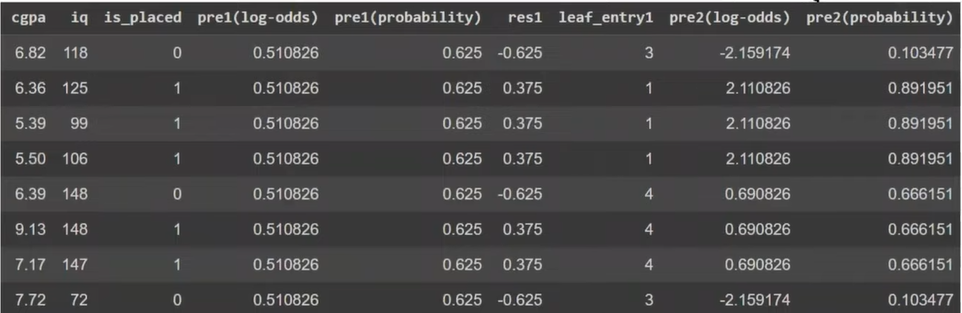
In our next model the residual would be target variable and the independent variables. The value derive in above decision tree is no the log(odds) value. Hence, we need to extract it with the help of below formula:

We will evaluate the log(odds) for the node #3 of above decision tree:

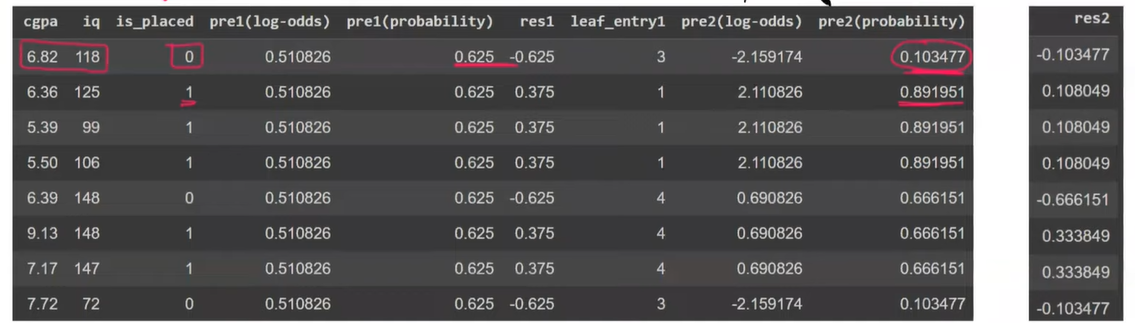
Similarly, the Log(odds) for Node #1 = 1.6, Node #2 = -2.66 & Node #3 = 0.18



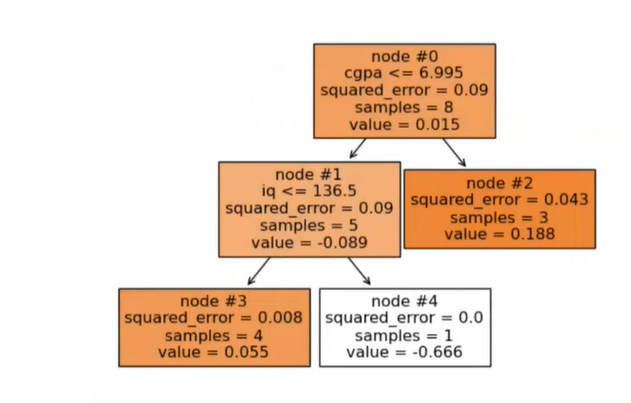
To find the combined value or the output of all the function, we have to sum the log of odds of each function of every observation. For observation #1, the log(odds) value would be 0.51 + (-2.66) = -2.51 and based on it the probability value would be



To build a second model, we have to again find the residuals based on actual values – predicted values.



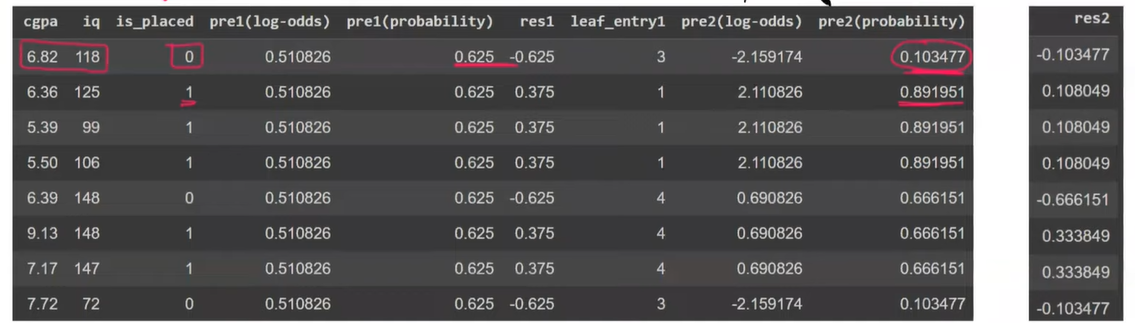
The second decision tree would be as follow:



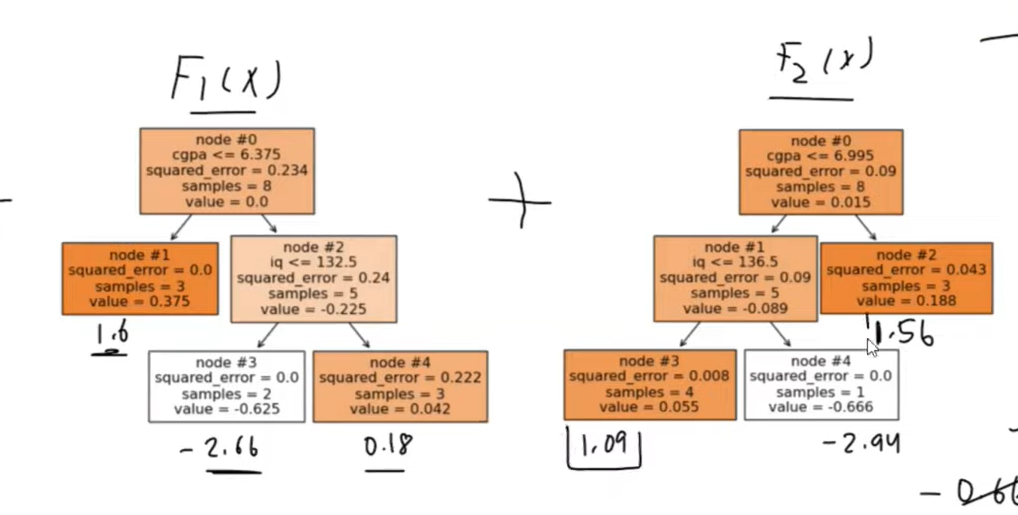
**Learning rate:** In the below image we can see the change in probability value is abruptly changed.

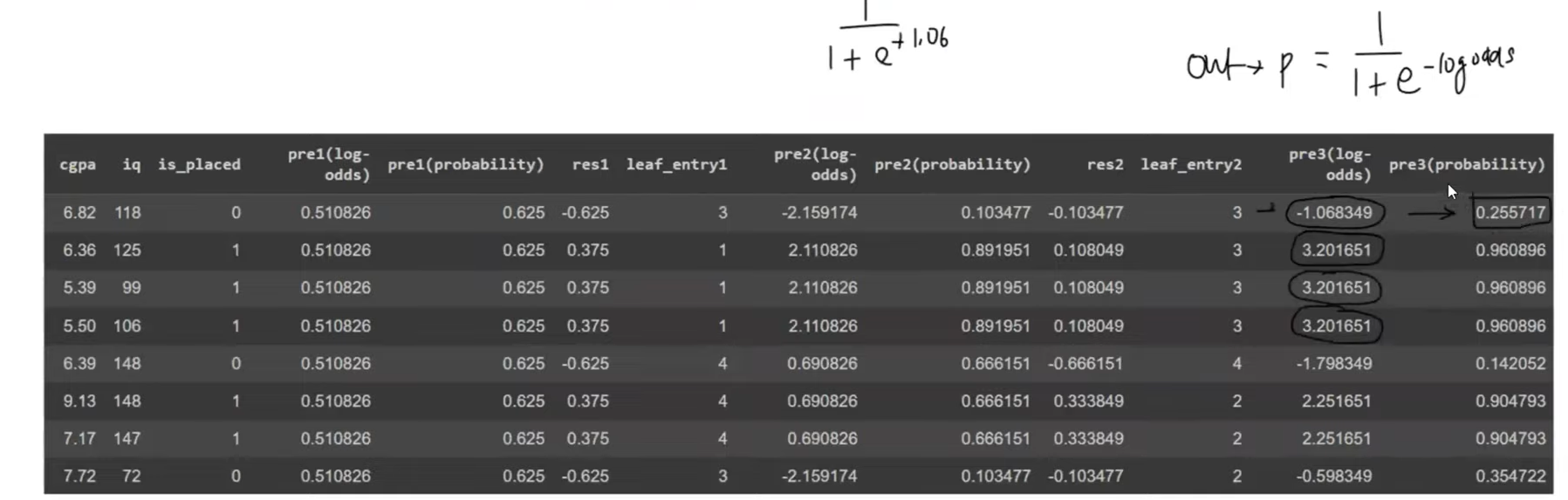
To reduce the magnitude of the difference/jump then we can add learning rate to log(odds) value derived from the nodes.

0.51



Similarly, the log(odds) value will be calculated for the second decision tree





# XGBoost

XGBoost (Extreme Gradient Boosting) is a machine learning library that provides an efficient and scalable implementation of the gradient boosting framework. It is known for its high performance and is widely used in various machine learning tasks. It was created by Tianqi Chen.

**Performance:**

* XGBoost showed superior performance compared to other algorithms. It consistently performed well in various machine learning tasks, making it a popular choice in both research and industry.
* It demonstrated effectiveness in handling structured/tabular data problems, showcasing its robustness and versatility.

**Robustness to Missing Values and Outliers:**

* XGBoost exhibited robustness to missing values and outliers, making it suitable for real-world datasets that often contain imperfections and irregularities.

**Success in Kaggle Competitions:**

* XGBoost gained popularity and recognition through its success in Kaggle competitions. Its competitive performance and ability to deliver high-quality results in data science competitions contributed to its widespread adoption.

The development of XGBoost considered three main factors: Performance, Speed, and Flexibility.

#### Flexibility

* XGBoost is designed to be flexible in terms of platform compatibility and language support. It can be trained on multiple platforms, including Windows and Linux, and supports various programming languages such as Python, Java, Scala, Ruby, and R.
* It is compatible with other popular machine learning libraries like NumPy, pandas, and scikit-learn.
* XGBoost is capable of addressing a wide range of machine learning problems, including regression, classification, time series forecasting, ranking, and anomaly detection.

#### Speed

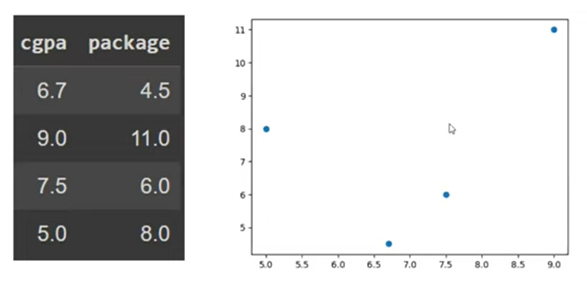
* **Parallel Processing**:
  + It uses parallel processing to build trees efficiently by simultaneously finding impurity for root nodes.
* **Optimized Data Structures**:
  + Instead of storing data in row blocks, XGBoost uses column blocks, improving speed and efficiency.
* **Cache Awareness**:
  + Utilizing CPU cache memory enhances computational speed.
* **Out-of-Core Computing**:
  + To handle large datasets that exceed available memory (RAM), XGBoost trains sequentially in batches, enabled by the 'tree\_method' hyperparameter set to 'hist.'
* **Distributed Computing**:
  + XGBoost supports distributed computing, enabling parallel training across multiple machines or nodes.
* **GPU Support**:
  + XGBoost can leverage GPU (Graphical Processing Unit) for parallel computing, further boosting performance. The 'tree\_method' hyperparameter can be set to 'gpu-hist' to enable GPU support.

#### Performance

* **Regularized Learning Objective:**
  + XGBoost incorporates regularization techniques to prevent overfitting. Regularization is achieved through the learning rate and tree construction process, helping maintain model generalization and robustness.
* **Sparsity-Aware Split Finding:**
  + XGBoost is designed to handle sparsity effectively, particularly in scenarios with a large number of missing values. During tree training, it intelligently manages missing values with a directional sense at each node level, determining the appropriate direction to consider.
* **Efficient Split Finding (Weighted Quantile Sketch + Approximate Tree Learning):**
  + XGBoost employs efficient techniques for split finding, enhancing the overall speed and accuracy of the model.
  + Weighted Quantile Sketch: It utilizes weighted quantile sketching, a method for approximating the distribution of data. This approach aids in accurate binning and improves the precision of the algorithm.
  + Approximate Tree Learning: Instead of calculating the means of two values in observations to prepare trees, XGBoost leverages an approximate tree learning approach. This technique further enhances the accuracy of binning and the overall understanding of data distribution.
* **Tree Pruning:**
  + XGBoost provides parameters for tree pruning, with gamma being one of them. Tree pruning is a regularization technique that helps control the complexity of the tree structure. The gamma parameter determines the minimum loss reduction required to make a further partition on a leaf node, allowing for the creation of more compact and effective trees.

## XGBoost with Regression:

The problem statement is we have cgpa value and the package of based on cgpa to predict the package based on the cgpa value



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CGPA | Package | Model 1 (Average) | Residual 1 | Model 2 with ETA value 0.3 Model 1 prediction + (learning rate \* Leaf node) | Residual 2 |
| 6.70 | 4.50 | 7.38 | -2.88 | 6.76 | 0.62 |
| 9.00 | 11.00 | 7.38 | 3.63 | 8.485 | -1.11 |
| 7.50 | 6.00 | 7.38 | -1.38 | 6.76 | 0.62 |
| 5.00 | 8.00 | 7.38 | 0.63 | 7.585 | -0.21 |

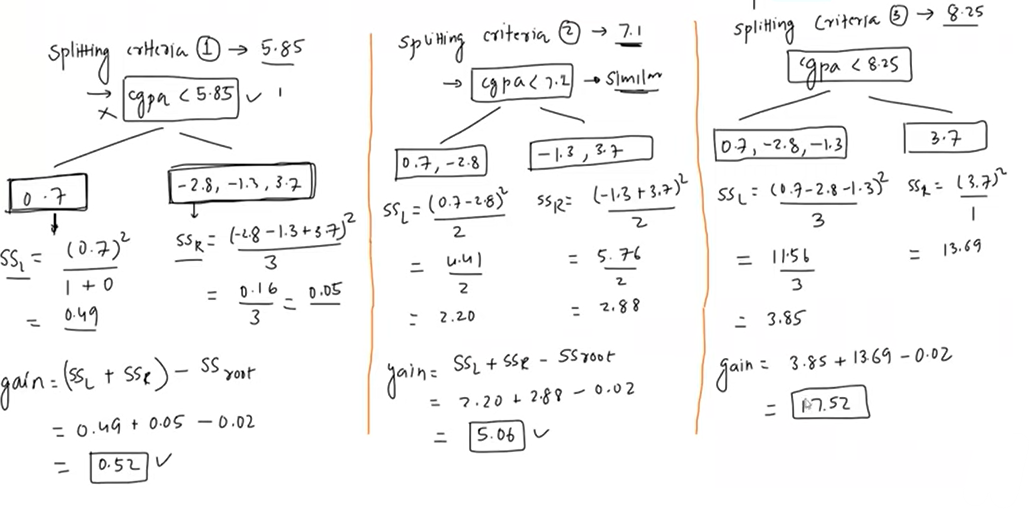
XGBoost works in similar fashion how the Gradient Boosting works. In first model, it will take the mean value in regression problem. In above table the mean would be **7.5** and the residual will be evaluated to find the cost/loss function of the model. Basis on the residual, the next model will be build and we will again predict the

It’s similar to Gradient boosting only the twist is the decision tree is build little differently compared to Gradient boosting algorithm. We have to prepare leaf nodes based on the residual and we have to calculate the similarity score which is as below:

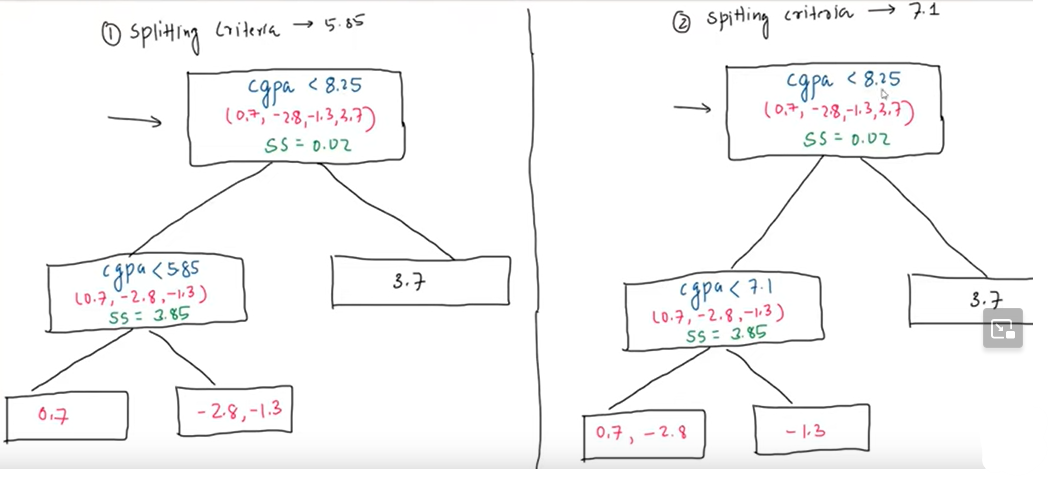
is a regularization parameter.

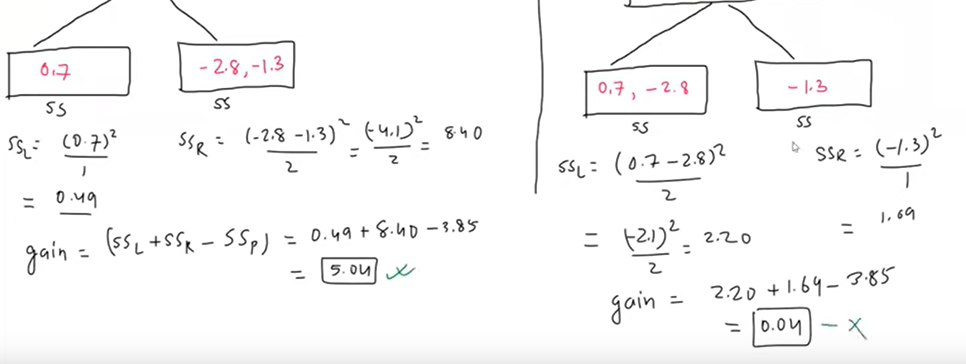
We have to do split the residual based on independent variable CGPA

First, we need to sort it and then we have to find the best splitting criteria.



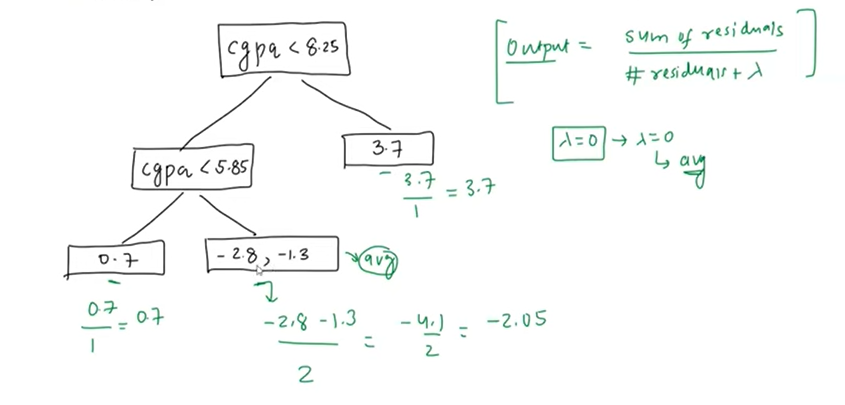
Basis on similarity score calculated above, splitting criteria 3 would be beneficial because the similarity score is 17.52 and the node containing values 0.7, -2.8 and -1.3 would be further splitted based on similarity score.





Based on the above calculation Splitting criteria is decided and further its split.

Output for all the leaf node will be calculated based on below formulae:



## XGBoost with classification

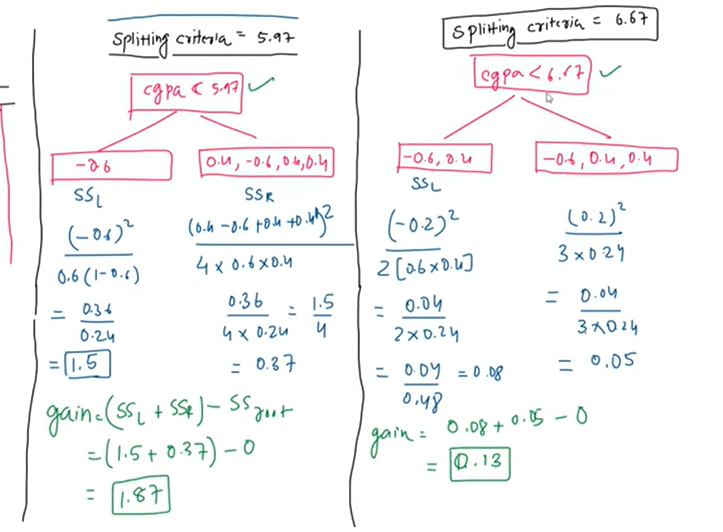
In the context of XGBoost with classification, the process involves building decision trees based on similarity scores for the log-odds of positive class probabilities. The following steps outline a simplified version of this process:

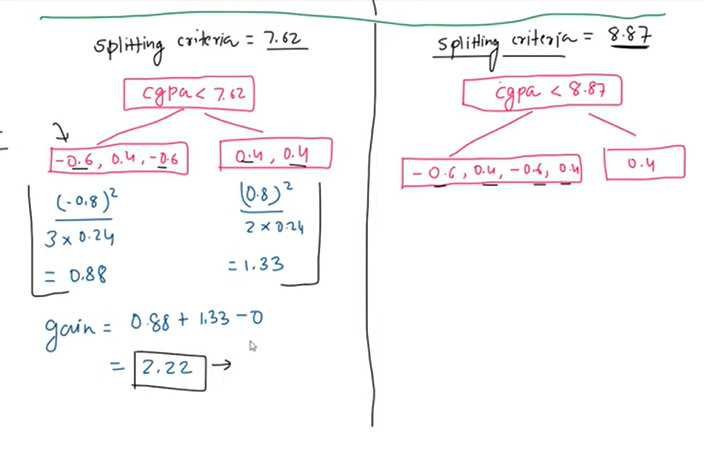
1. **Initialize with Probabilities:**
   * The first model (tree) is initialized with the probabilities of log-odds for the positive class. These probabilities are converted into log-odds using the logistic function.
2. **Calculate Pseudo Residuals:**
   * Pseudo residuals are calculated as the difference between the true class labels and the predicted probabilities. For binary classification, the residuals are calculated as True Label−Predicted ProbabilityTrue Label−Predicted Probability.
3. **Build Decision Trees:**
   * Decision trees are constructed to predict the pseudo residuals. The goal is to capture the patterns or errors made by the initial model.
4. **Update Model:**
   * The output of the new decision tree is combined with the previous model's output. The learning rate is applied to control the contribution of each tree to the final model. The learning rate helps prevent overfitting.
5. **Repeat the Process:**
   * Steps 2-4 are repeated for a specified number of boosting rounds, each time building a new decision tree to correct the errors made by the existing ensemble.
6. **Final Model:**
   * The final model is an ensemble of decision trees that collectively improve the predictive performance on the classification task.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | CGPA | Package | Model 1 (Average predictions with Log(odds) | Probability | Residual 1 | eta value = 0.3 Pred 2 [Log(odds)] | Prediction 2 Probability | Residual 2 |
|  | 5.70 | 0.00 | 0.405 | 0.6 | -0.60 | 0.072 | 0.518 | -0.52 |
| 5.98 | 6.25 | 1.00 | 0.405 | 0.6 | 0.40 | 0.072 | 0.518 | 0.48 |
| 6.68 | 7.10 | 0.00 | 0.405 | 0.6 | -0.60 | 0.072 | 0.518 | -0.52 |
| 7.63 | 8.15 | 1.00 | 0.405 | 0.6 | 0.40 | 0.903 | 0.712 | 0.29 |
| 8.88 | 9.60 | 1.00 | 0.405 | 0.6 | 0.40 | 0.903 | 0.712 | 0.29 |

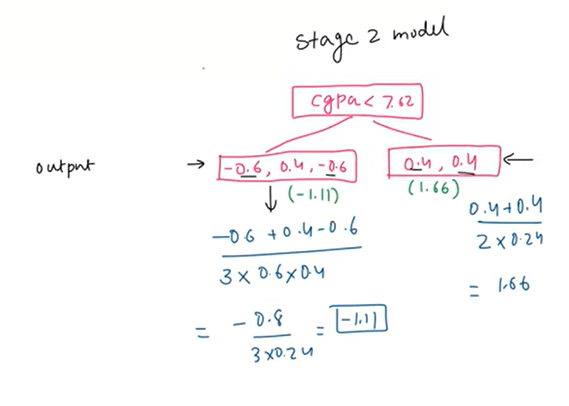
Step 1: The probability of log(odds) has to be calculated for the positive classes and then we have to find the residual between the target and probability values.

Step 2: Find the best split for CGPA value to build decision tree based on similarity scores. Splitting criteria 1 = 5.97 is selected because it has maximum gain derived based on similarity score.





Step 3: Calculate the output of the leaf node with below equation



Step 4: Similarly we can re-iterate with the above steps to reduce the residuals.

## XGBoost Mathematical intuition

**Regression:**

**Classification:**

In Gradient boosting the loss function, we can choose any loss function dynamically to reduce the difference between the actual and predicted value. In regression the loss function is same we can add any mathematical function like MAE, MSE. However, in XGB the loss function is slightly different, there is a regularization parameter added to the loss function.

Regularization parameter

T is number of leaf nodes in tree

W is the weight of leaf

Gamma and lambda are regularization parameter to reduce the loss function.

Regularization is only applied to last model only because the previous model is already considered in the previous model. Hence, the loss function can be written as:

We want such value that will minimize the objective function. Previously, we have already used in Linear regression where we use to find the value of slope and intercept value that the loss =0. We used to differentiate the loss function by and . But its little tricky to solve the optimize for

Since, in linear regression it’s a straight line the objective (parabolic shape) is convex and smooth. Whereas on other hand, XGBoost uses decision trees which has non-convex and non-smooth and non-differentiable objective which would help us to find the optimum value to reduce the loss function.

We can achieve this with Taylor series:

Any complex function, it can approximate, with polynomial. Taylor series uses nth order of derivatives to convert the complex function into polynomial.