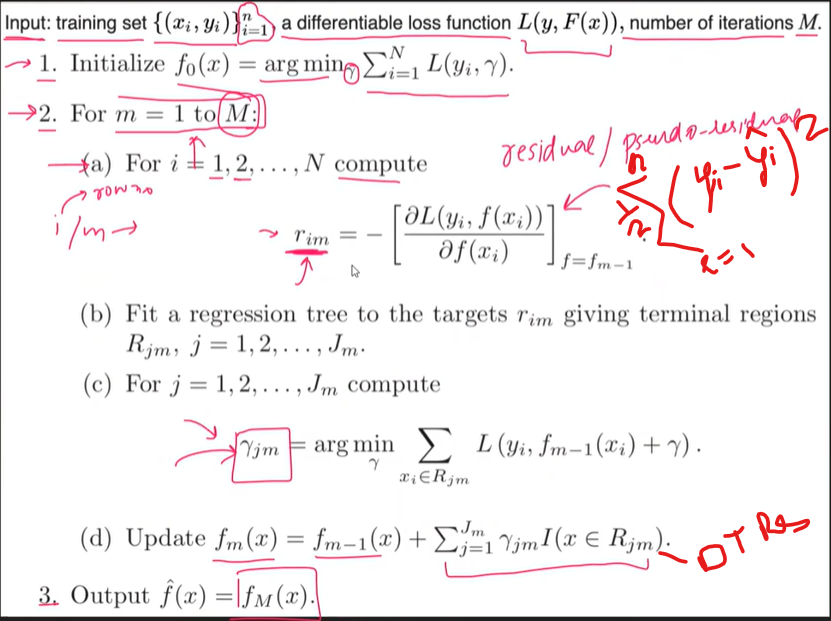
**Gradient Boosting Classifier**

<https://www.digitalocean.com/community/tutorials/gradient-boosting-for-classification>

We will use log loss function in ‘Gradient boosting classification’ instead of MSE as used in ‘Gradient boosting regressor’ while algorithm will be same given below



In regressor problem F0 simple model was calculated throw mean but here as target data will be as (1,0) so there so no meaning of calculating means instead we will calculate as below

F0 = log (odds) = log ((number of ones)/number of zeros)

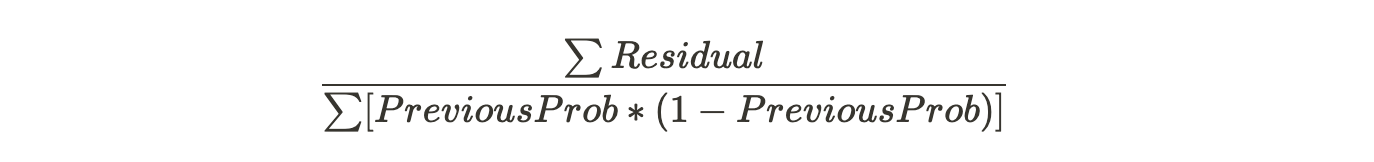
As before residual r1 was calculated using (target y – F0) but as in classification y will be zero and ones while other size F0 is calculated on probability so we can’t do the same here

We will follow other formula as

R = ( 1/(1+pow(e,-log(odds) ) ) )

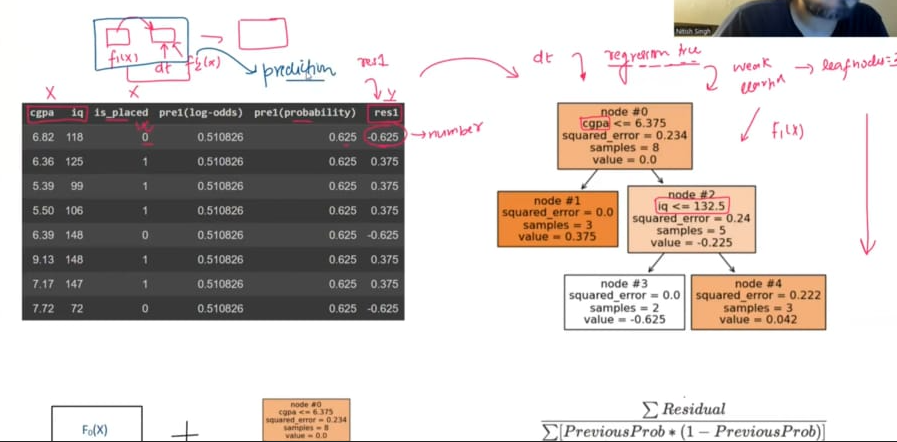
**Gradient Boost has a range between 8 leaves to 32 leaves.**

Because of the limit on leaves, one leaf can have multiple values. Predictions are in terms of log(odds), but these leave are derived from probability, which causes disparity. So, we can’t just add the single leaf we got earlier and this tree to get new predictions because they’re derived from different sources. We have to use some kind of transformation. The most common form of transformation used in Gradient Boost for Classification is :



The numerator in this equation is the sum of residuals in that particular leaf.

The denominator is sum of (previous prediction probability for each residual ) \* (1 - same previous prediction probability).

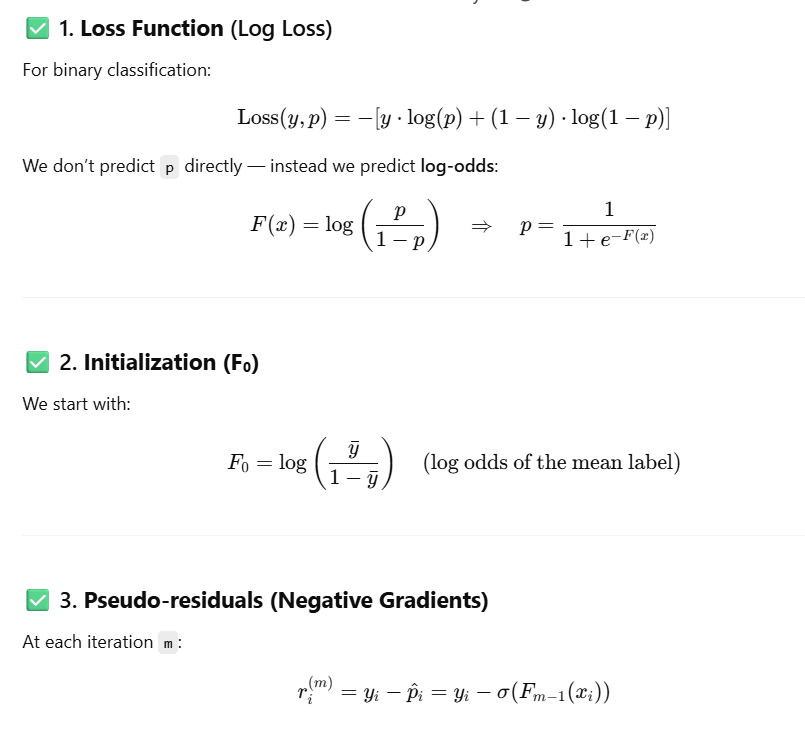
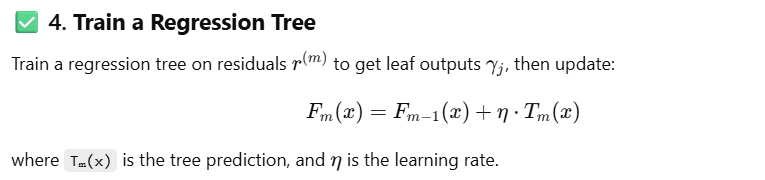


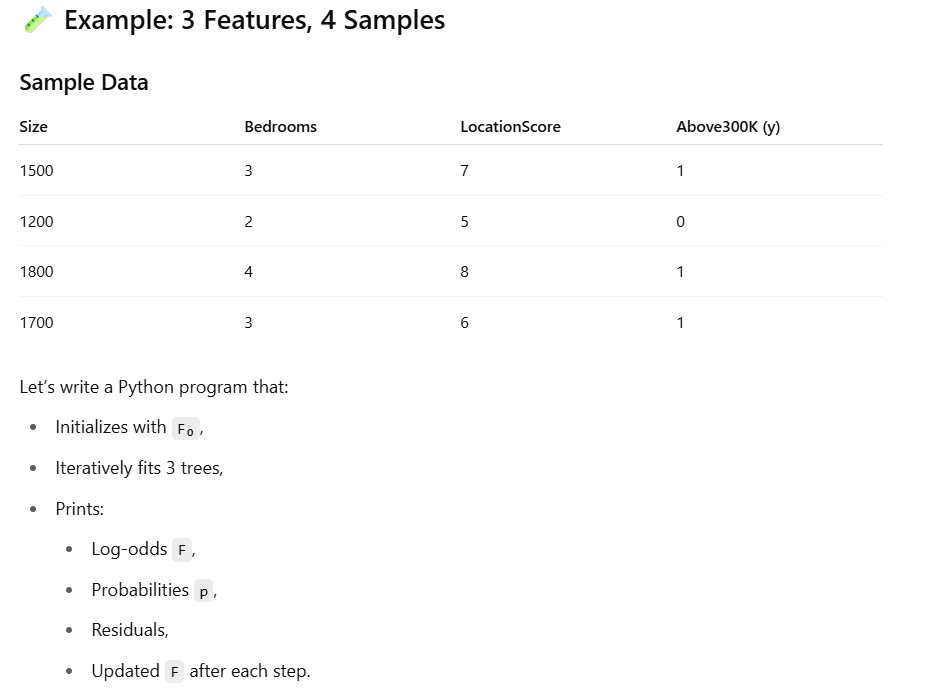
**intuition behind Gradient Boosting for Classification**, step-by-step with all formulas, and then walk through an example with **realistic sample data** and all outputs for a **3-tree binary classification model**.

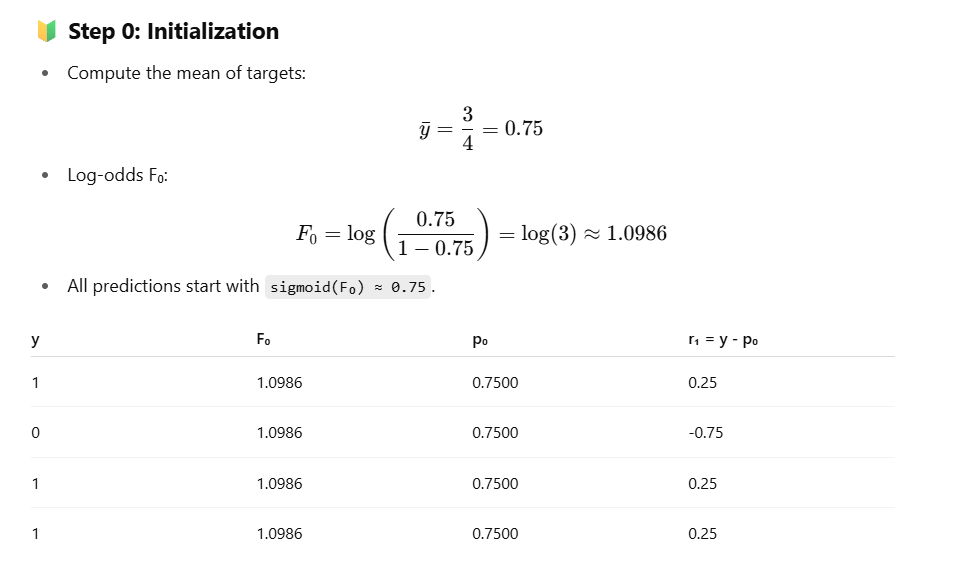
**🎯 Goal**

Train a binary classifier (e.g., predict if a house sells above 300K: Yes or No) using **Gradient Boosting Classification**, which works by:

* **Minimizing log-loss**,
* **Fitting decision trees to pseudo-residuals (gradients of the loss)**,
* And **updating the model in function space**.

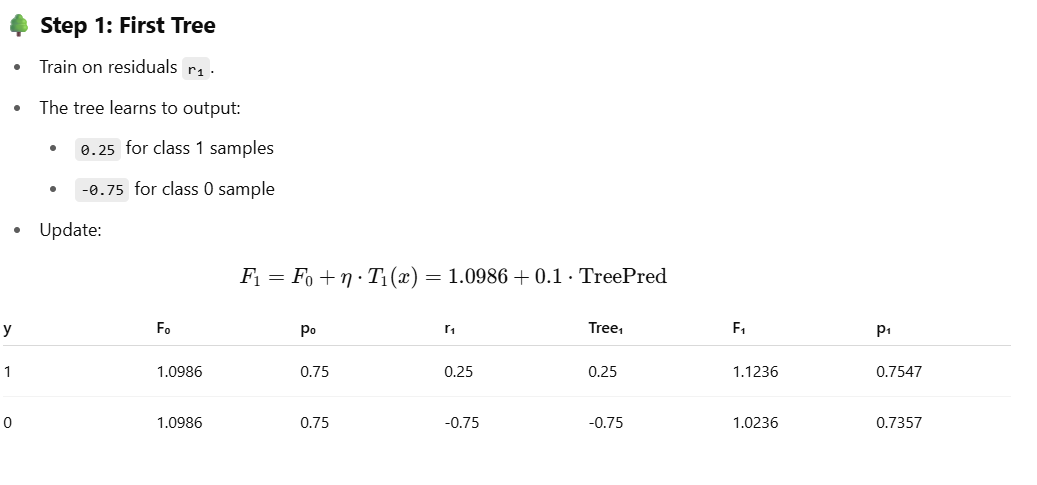
  




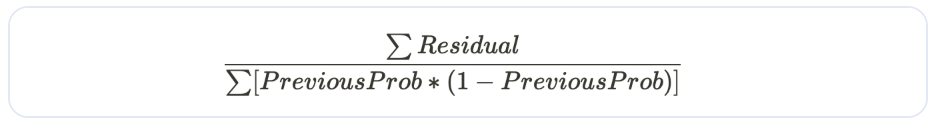


Here Po is calculated using sigmoid function as below

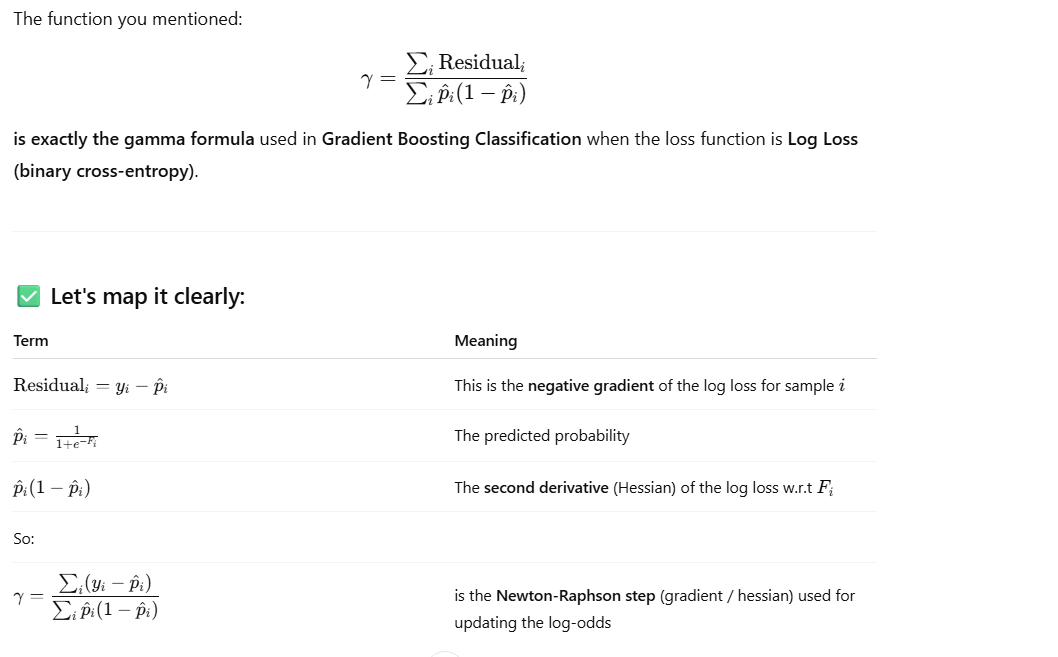
Sigmoid = 1/ (1+ e(-F(x)))

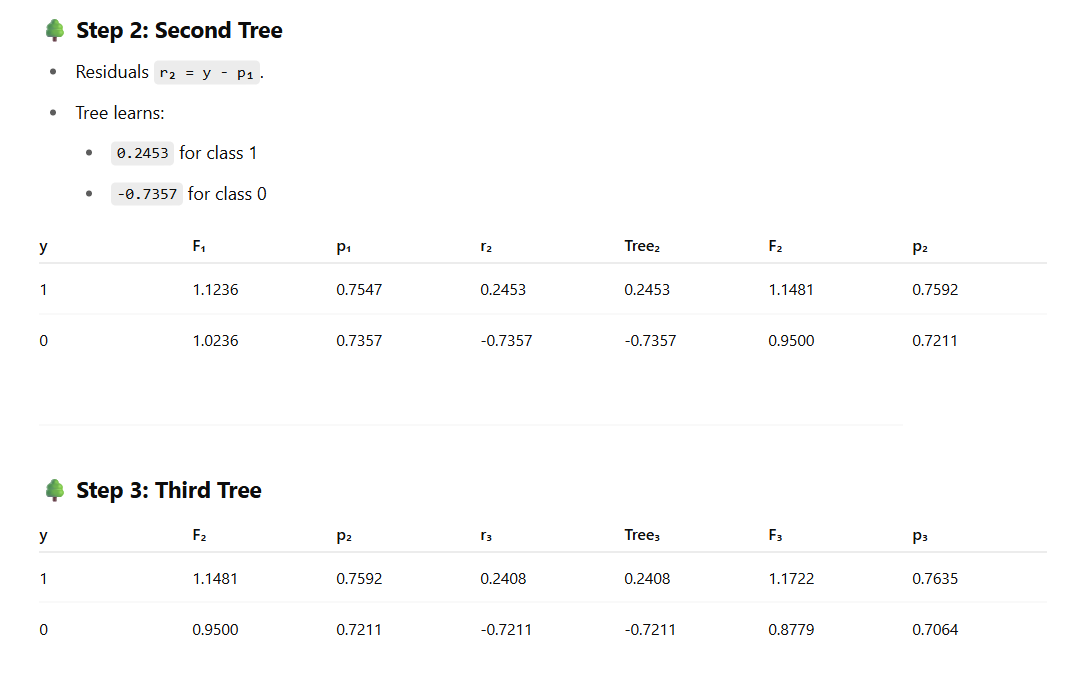


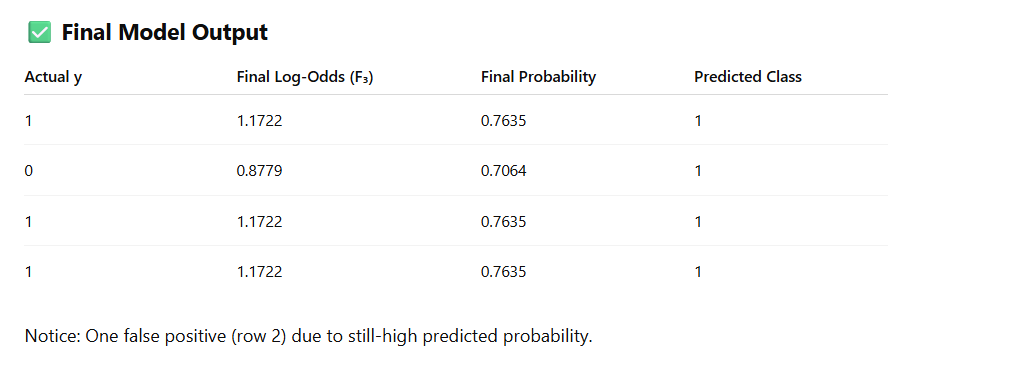
Here TreePred is the Prediction made by **Tree 1** for input sample x as the node may have multiple row values in one due to tree height limit so it can’t nbe the same tree value as were in regression so need to be calculated using below formula-

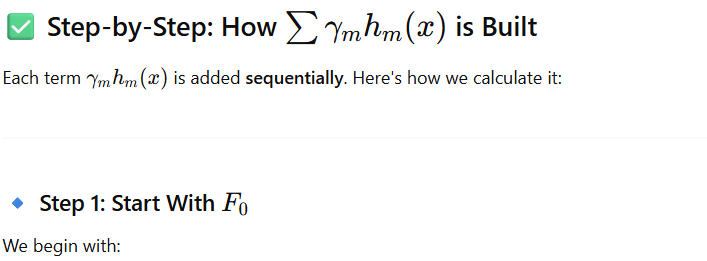


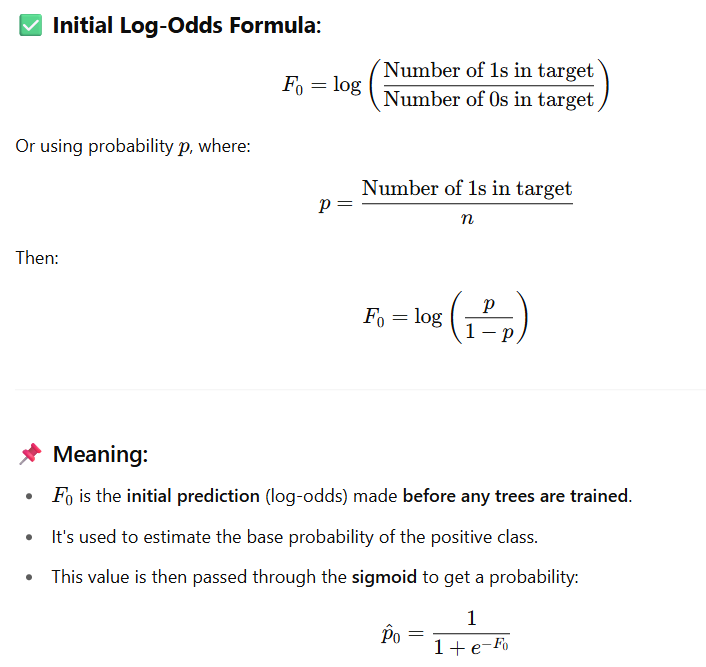
This above is same as below mathematical formula

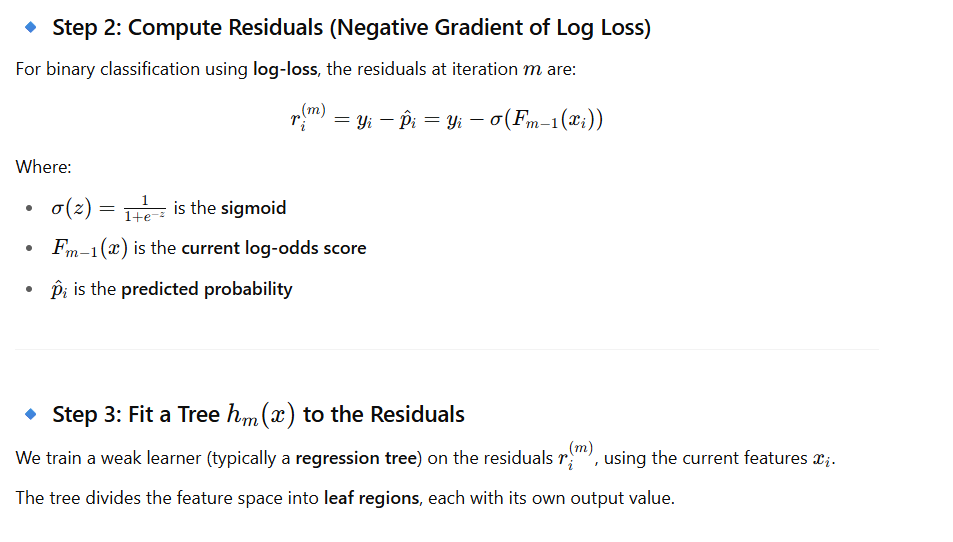


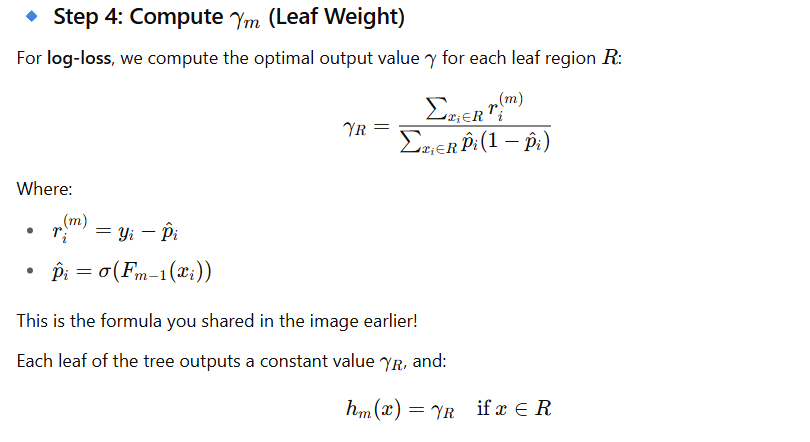


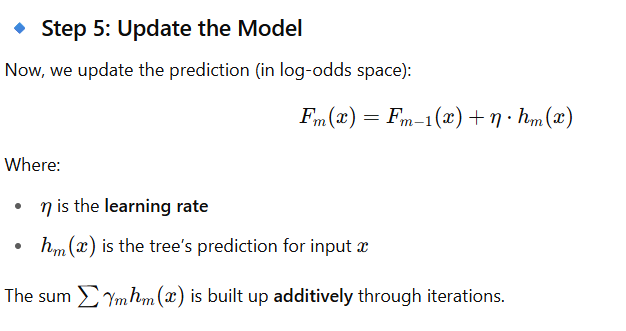


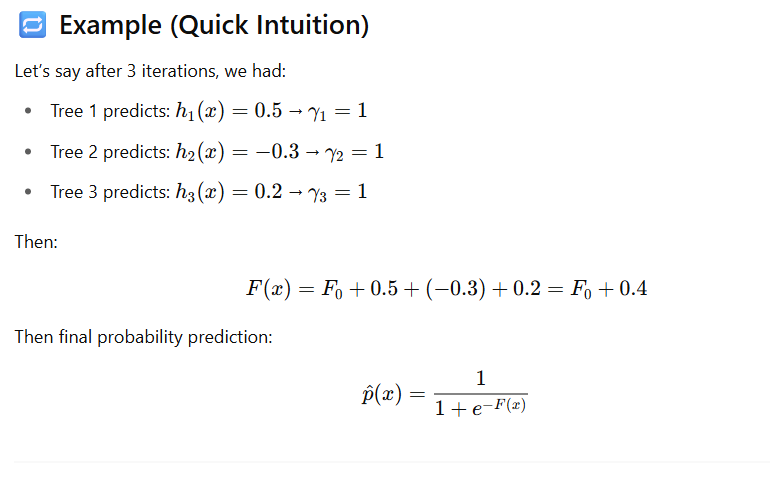


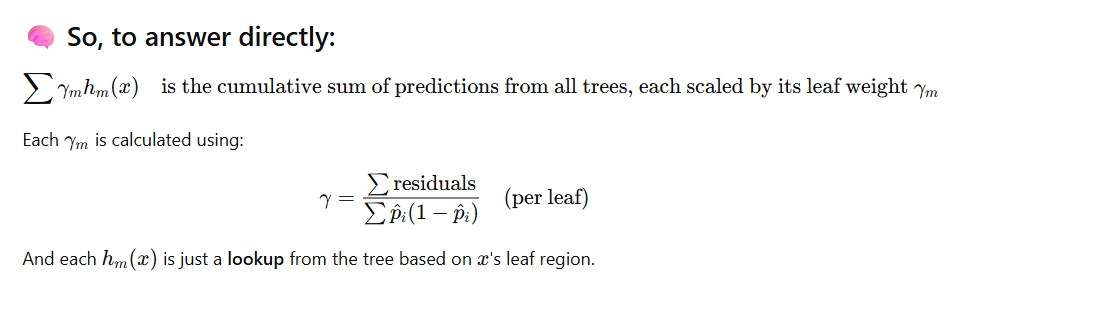


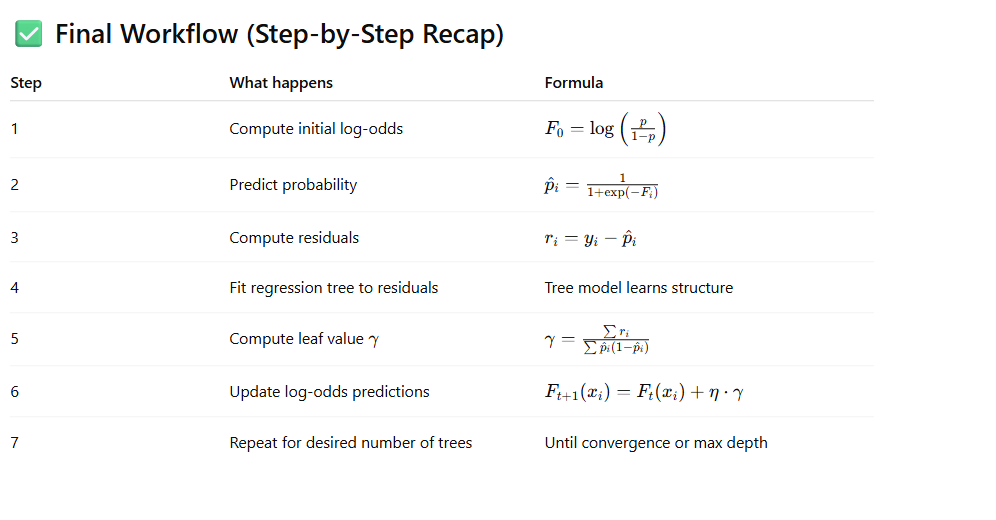












**🔢 Dataset Used**

plaintext

CopyEdit

Index | Feature1 | Feature2 | Feature3 | Feature4 | y

------------------------------------------------------

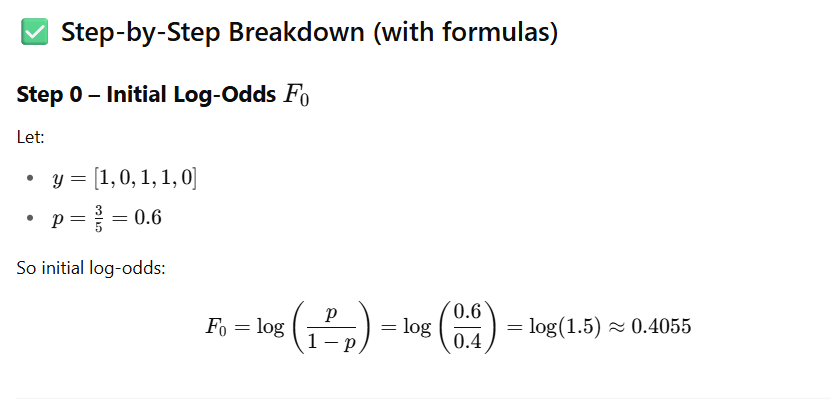
0 | 2.5 | 1.2 | 3.1 | 0.5 | 1

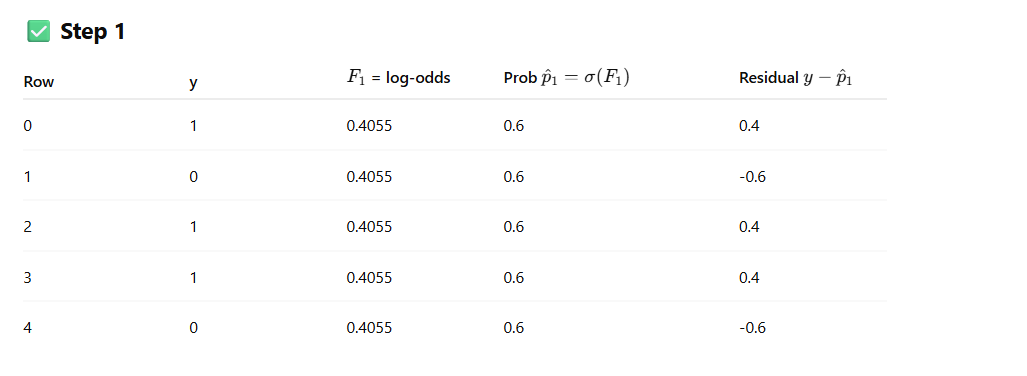
1 | 1.0 | 0.7 | 2.2 | 1.0 | 0

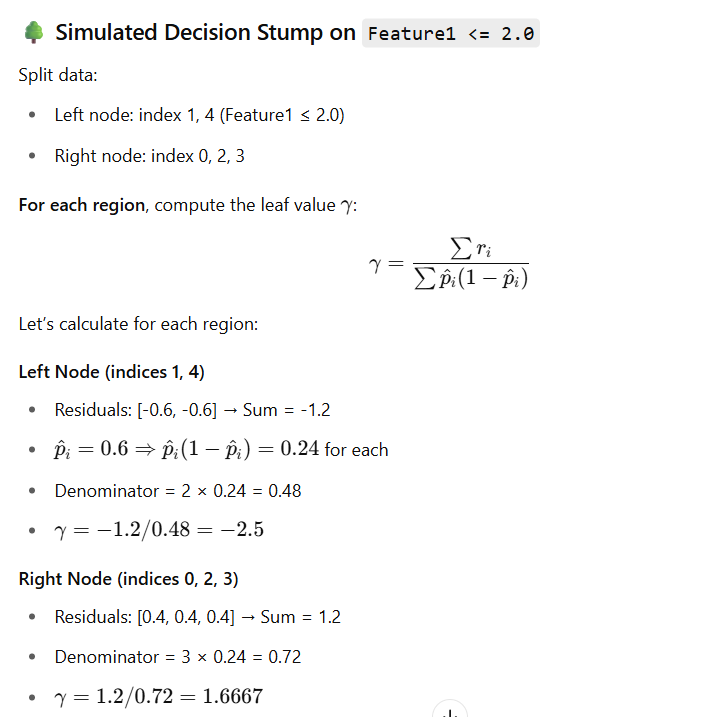
2 | 3.6 | 2.5 | 4.0 | 0.3 | 1

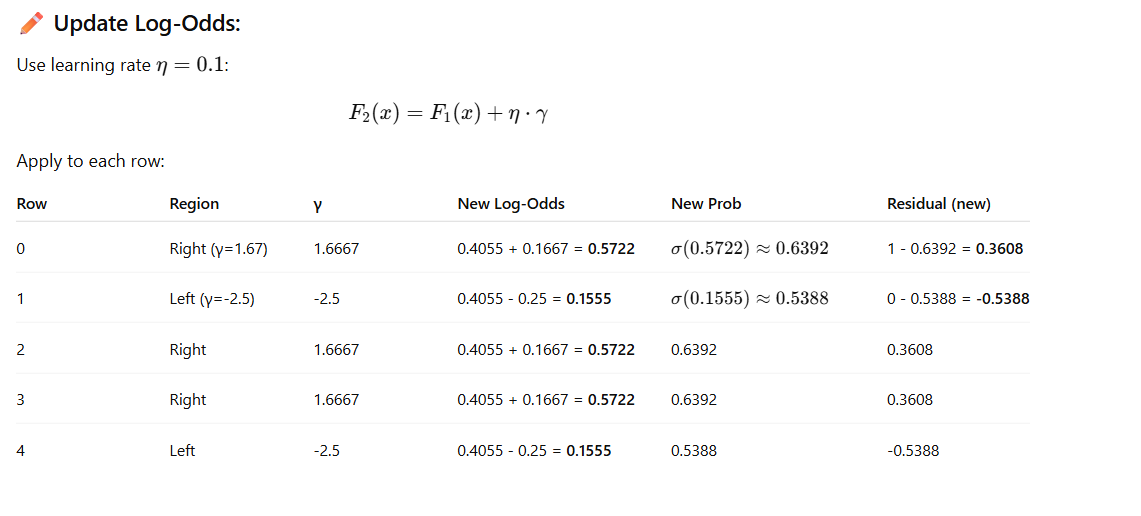
3 | 4.0 | 3.6 | 5.1 | 0.2 | 1

4 | 0.5 | 0.3 | 1.0 | 1.2 | 0











import numpy as np

import pandas as pd

from sklearn.tree import DecisionTreeRegressor

# Sample binary classification dataset

df = pd.DataFrame({

'Feature1': [2, 1, 3, 4, 2, 1],

'Feature2': [1, 2, 1, 3, 2, 1],

'Feature3': [0, 1, 0, 1, 0, 1],

'Target': [0, 1, 0, 1, 0, 1]

})

# Step 1: Initial log-odds F0

n\_pos = df['Target'].sum()

n\_neg = len(df) - n\_pos

F0 = np.log(n\_pos / n\_neg) # initial log odds

df['F'] = F0 # initial log-odds prediction

# Learning rate

lr = 0.1

# Initialize tracking

all\_rounds = []

# Step 2: Compute initial probabilities and residuals

df['P'] = 1 / (1 + np.exp(-df['F']))

df['Residual'] = df['Target'] - df['P']

# Perform 3 rounds of boosting

for round\_num in range(1, 4):

# Step 3: Train weak learner on residuals

features = ['Feature1', 'Feature2', 'Feature3']

tree = DecisionTreeRegressor(max\_depth=1)

tree.fit(df[features], df['Residual'])

# Step 4: Apply tree to assign leaf nodes

leaf\_ids = tree.apply(df[features])

df[f'Leaf{round\_num}'] = leaf\_ids

# Step 5: Calculate gamma for each leaf

gammas = {}

for leaf in np.unique(leaf\_ids):

indices = df[f'Leaf{round\_num}'] == leaf

residuals = df.loc[indices, 'Residual']

probs = df.loc[indices, 'P']

numerator = residuals.sum()

denominator = (probs \* (1 - probs)).sum()

gamma = numerator / denominator if denominator != 0 else 0

gammas[leaf] = gamma

# Step 6: Apply gamma and update predictions

df[f'Gamma{round\_num}'] = df[f'Leaf{round\_num}'].map(gammas)

df[f'F{round\_num}'] = df['F'] + lr \* df[f'Gamma{round\_num}']

df['F'] = df[f'F{round\_num}'] # update log-odds

df['P'] = 1 / (1 + np.exp(-df['F'])) # update probability

df['Residual'] = df['Target'] - df['P'] # update residuals

# Save results for this round

round\_df = df[[

'Feature1', 'Feature2', 'Feature3', 'Target',

f'F{round\_num}', 'P', 'Residual',

f'Leaf{round\_num}', f'Gamma{round\_num}'

]].copy().rename(columns={

f'F{round\_num}': 'Log-Odds (F)',

'P': 'Probability (P)',

'Residual': 'Residual',

f'Leaf{round\_num}': 'Leaf ID',

f'Gamma{round\_num}': 'Leaf Output (γ)'

})

round\_df.insert(0, 'Round', round\_num)

all\_rounds.append(round\_df)

# Combine results

results\_df = pd.concat(all\_rounds, ignore\_index=True)

results\_df = results\_df.round(4)

results\_df.head(18) # Display full results for all rounds