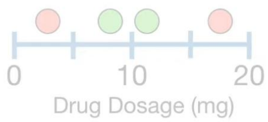
Predicted Drug Effectiveness

0.5

The very first step in fitting **XGBoost** to the **Training Data** is to make an initial prediction.



Predicted Drug Effectiveness

0.5

This prediction can be anything, for example, the **probability** of observing an effective dosage in the **Training Data**, but by default it is **0.5**, regardless of whether you are using **XGBoost** for **Regression** or **Classification**.

Predicted Drug Effectiveness

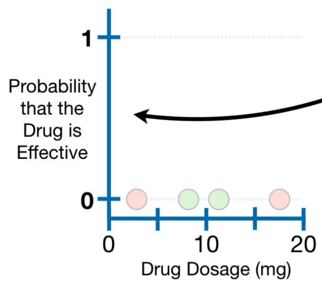
0.5

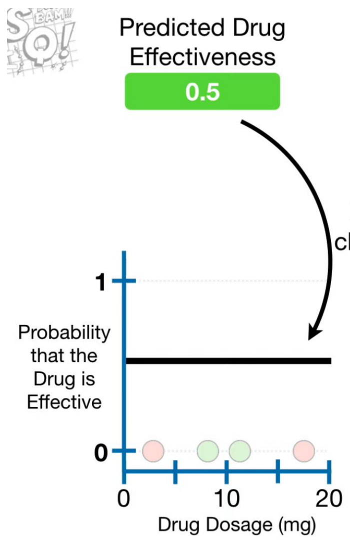
In other words, regardless of the **Dosage**, the default prediction is that there is a **50%** chance the drug is **Effective**.

Predicted Drug Effectiveness

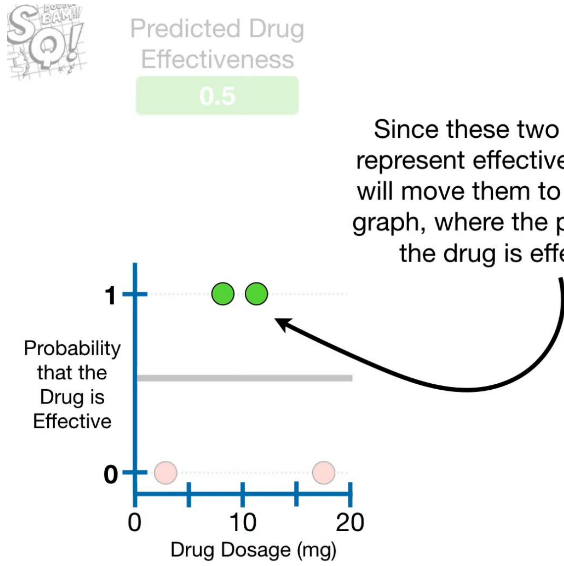
0.5

We can illustrate the initial prediction by adding a ***y*-axis** to our graph to represent the **Probability that the Drug is Effective**…

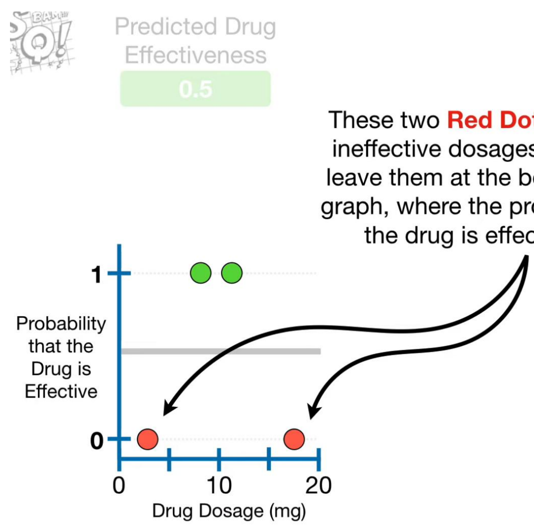




…and drawing a **thick black line** at **0.5** to represent a **50%** chance that the drug is effective.

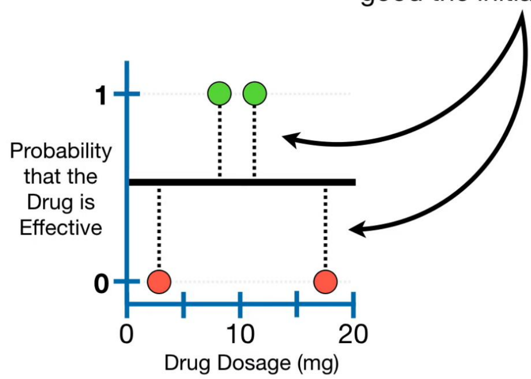


Since these two **Green Dots** represent effective dosages, we will move them to the top of the graph, where the probability that the drug is effective is **1**.

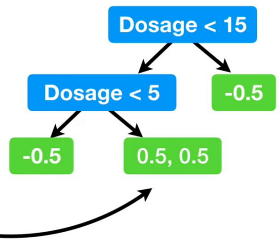


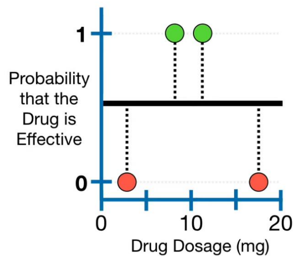
These two **Red Dots** represent ineffective dosages, so we will leave them at the bottom of the graph, where the probability that the drug is effective is **0**.

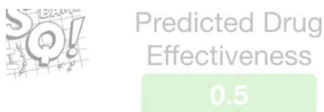
The **Residuals**, the differences between the **Observed** and **Predicted** values, show us how good the initial prediction is.



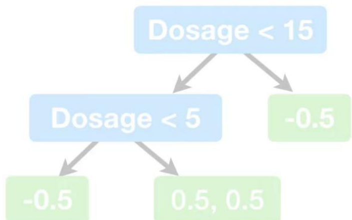
Now, just like we did for **Regression**, we fit an **XGBoost Tree** to the **Residuals**...



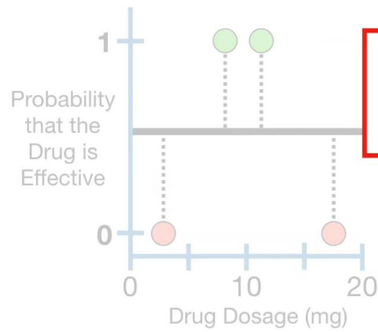




...however, since we are using **XGBoost** for **Classification**, we have a new formula for the **Similarity Scores**.

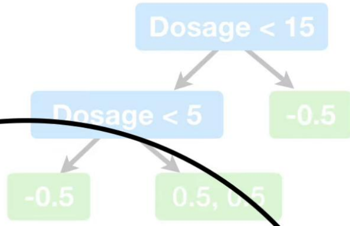


$$\boxed{\frac{(\sum \text{Residual}_i)^2}{\sum [\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)] + \lambda}}$$

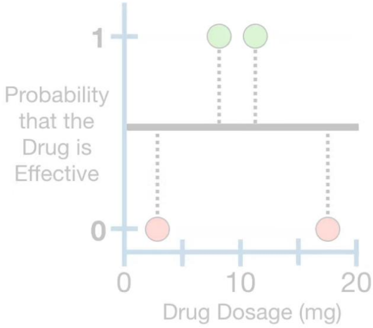


Predicted Drug Effectiveness

0.5

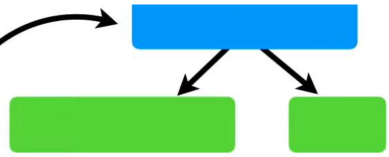


And just like for **Regression**, the denominator contains **λ (lambda)**, the **Regularization Parameter**…



$$\frac{\left(\sum \text{Residual}_i\right)^2}{\sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right] \boxed{+ \lambda}}$$

Now let's build a tree!!!



Just like for **Regression**, each tree starts out as a single leaf…

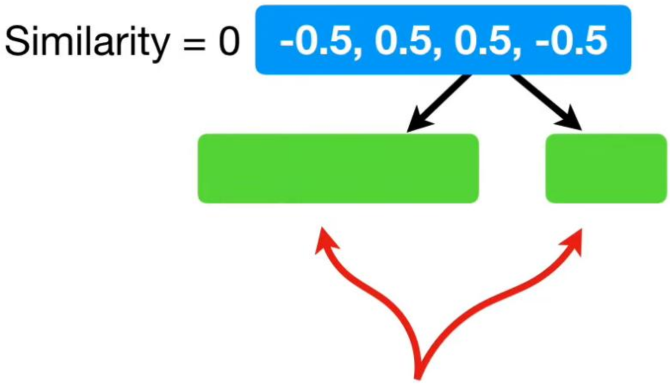
-0.5, 0.5, 0.5, -0.5

…and all of the **Residuals** go to the leaf.

-0.5, 0.5, 0.5, -0.5

Now we need to calculate a **Similarity Score** for the leaf.

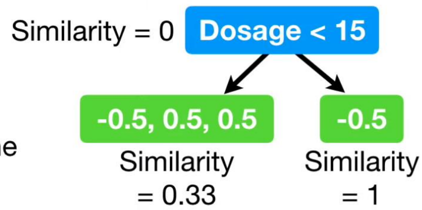
$$\frac{\left(\sum \text{Residual}_i\right)^2}{\sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right] + \lambda}$$



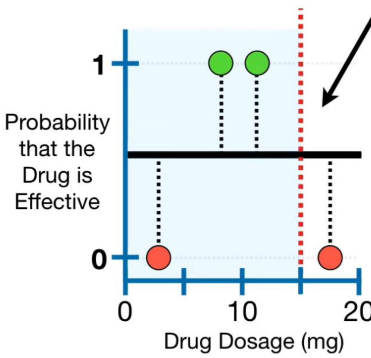
Now we need to decide if we can do a better job clustering similar **Residuals** if we split them into two groups.

Predicted Drug Effectiveness

0.5

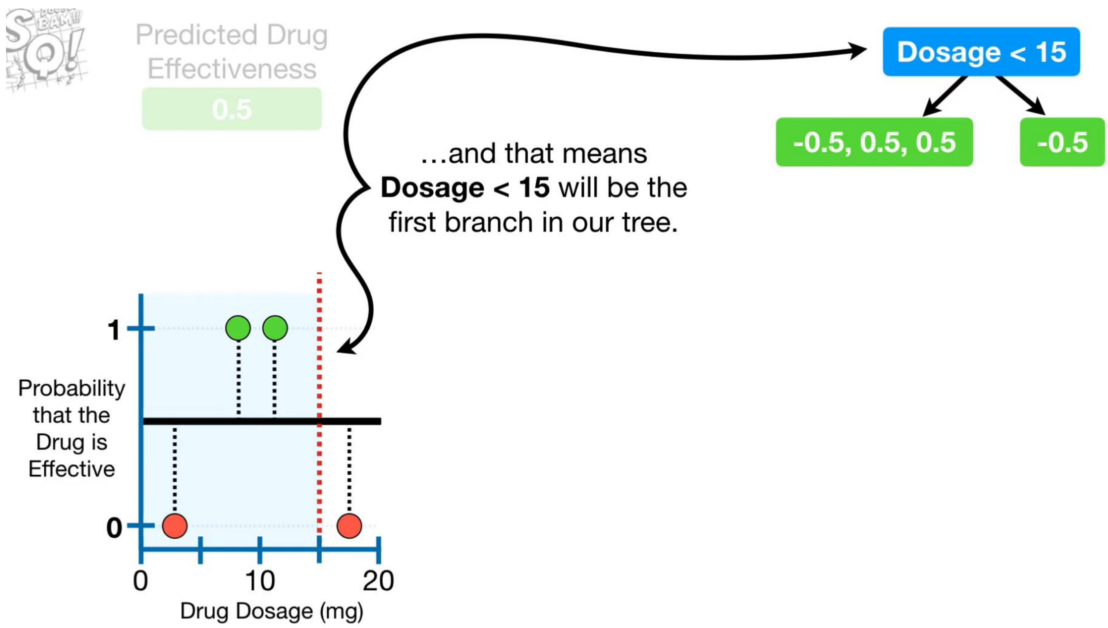


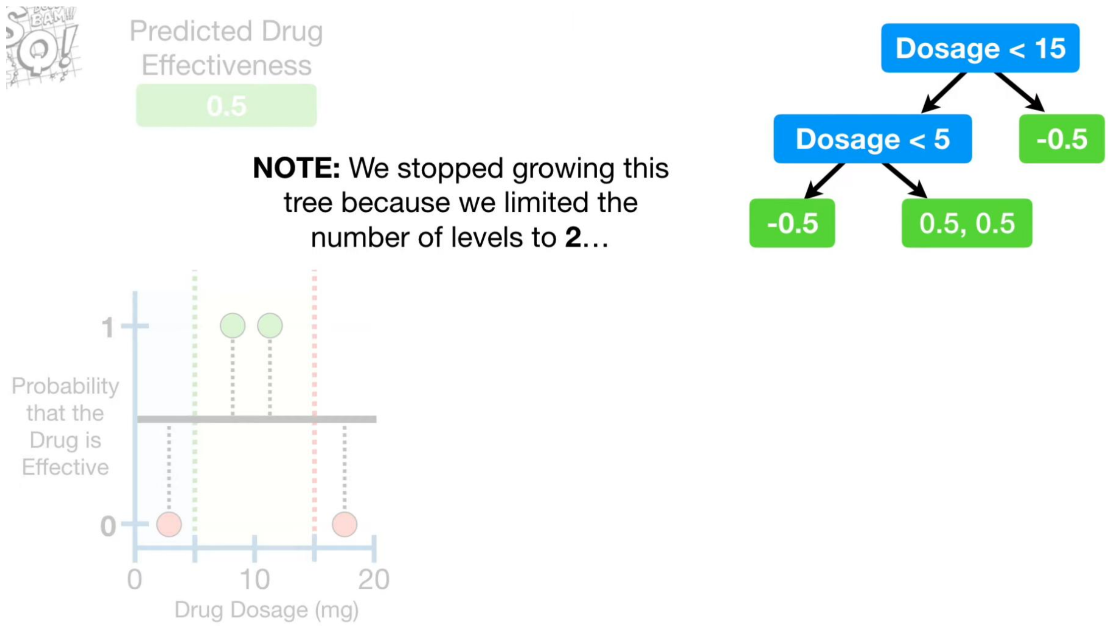
So when we split the **Observations** based on the threshold **Dosage < 15**, **Gain = 1.33**.



Gain = 0.33 + 1 - 0 = 1.33

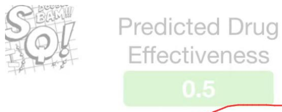
Since I'm such a nice guy, I'm going to tell you that no other threshold gives us a larger **Gain** value…

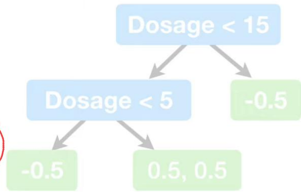




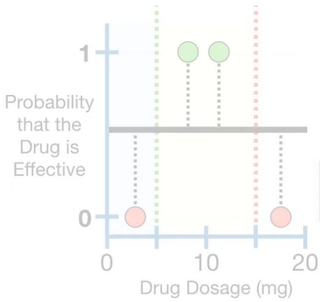
…however, **XGBoost** also has a threshold for the minimum number of **Residuals** in each leaf.

The minimum number of **Residuals** in each leaf is determined by calculating something called **Cover**.



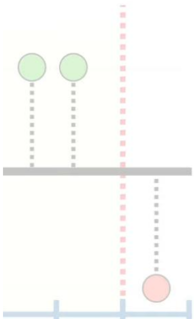


Cover is defined as the denominator of the **Similarity Score** minus **λ (lambda)**.



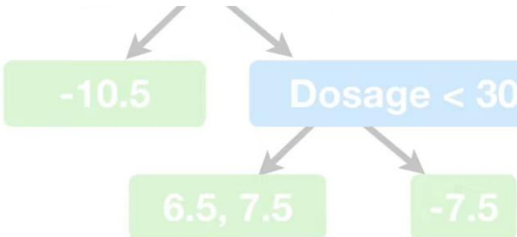
$$\text{Similarity} = \frac{\left(\sum \text{Residual}_i\right)^2}{\boxed{\sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right]} + \lambda}$$

In other words, when we are using **XGBoost** for **Classification**, **Cover** is equal to…



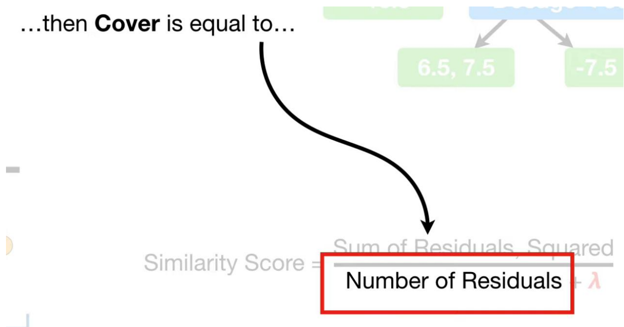
$$\text{Similarity} = \frac{\left(\sum \text{Residual}_i\right)^2}{\boxed{\sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right]} + \lambda}$$

In contrast, when **XGBoost** is used for **Regression** and we are using this formula for the **Similarity Score**…



$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + \lambda}$$

…then **Cover** is equal to…



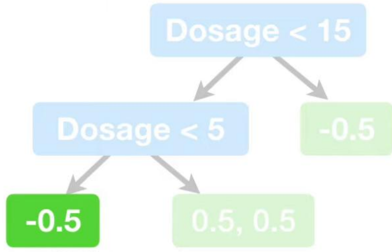
By default, the minimum value for **Cover** is **1**.

Thus, by default, when we use **XGBoost** for **Regression**, we can have as few as **1 Residual** per leaf.

In other words, when we use **XGBoost** for **Regression** and use the default minimum value for **Cover**, **Cover** has no effect on how we grow the tree.

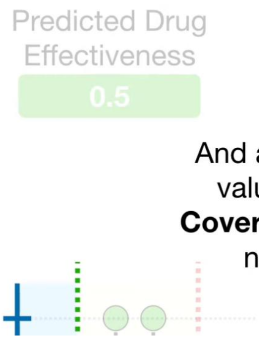
In contrast, things are way more complicated when we use **XGBoost** for **Classification** because **Cover** depends on the previously predicted probability of each **Residual** in a leaf.

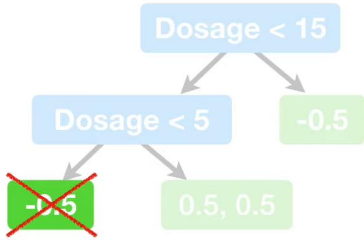
For example, the **Cover** for this leaf is…



…which is **0.25**.

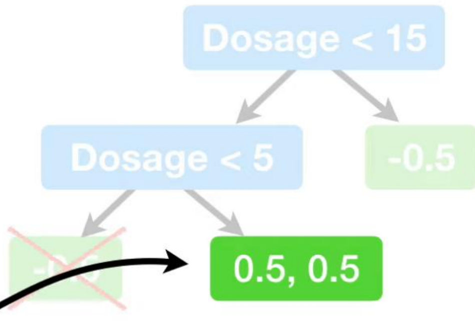
$$\text{Cover} = 0.5 \times (1 - 0.5) \boxed{= 0.25}$$



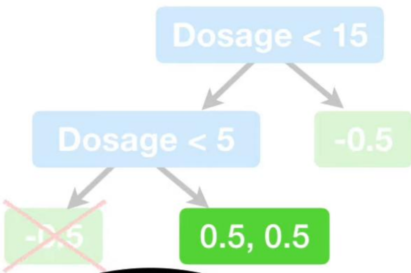


And and since the default value for the minimum **Cover** is **1**, **XGBoost** would not allow this leaf.

$$\text{Cover} = 0.5 \times (1 - 0.5) = 0.25$$

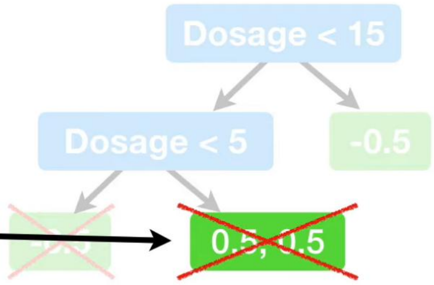


Likewise, the **Cover** for this leaf…



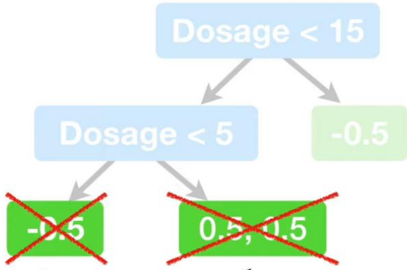
…is equal to **0.5**.

$$\text{Cover} = 0.5 \times (1 - 0.5) + 0.5 \times (1 - 0.5) \boxed{= 0.5}$$

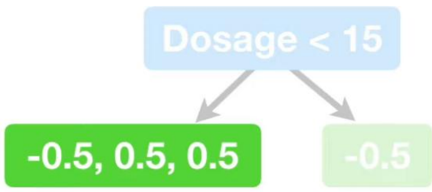


So, by default, **XGBoost** would not allow this leaf either.

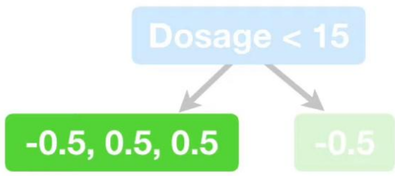
Cover = 0.5 × (1 - 0.5) + 0.5 × (1 - 0.5) = 0.5



Since these leaves are not allowed, let's removed them…



…and go back to this leaf.



…and that means **Cover = 0.75**…

Cover = 3 × [0.5 × (1 - 0.5)] $\boxed{= 0.75}$

…so **XGBoost** would not allow this leaf either.

Cover = 3 × [0.5 × (1 - 0.5)] = 0.75

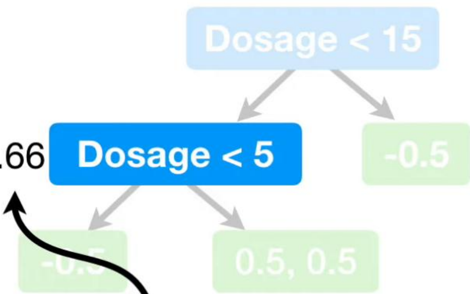
Ultimately, if we used the default minimum value for **Cover**, **1**, then we would be left with the **Root**, and **XGBoost** requires trees to be larger than just the **Root**.

So, in order to prevent this from being the worst example ever, let's set the minimum value for **Cover = 0**.

That means setting the **min_child_weight** parameter equal to **0**.

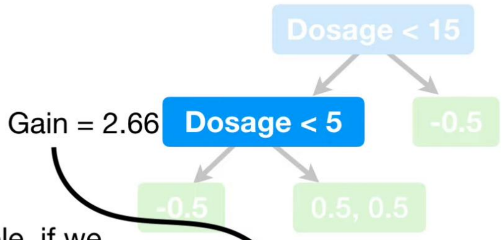
Tree Pruning

Gain = 2.66



Just like we did in **Part 1**, we prune by calculating the difference between the **Gain** associated with the lowest branch and a number we pick for ***γ* (gamma)**.

Gain - *γ* =

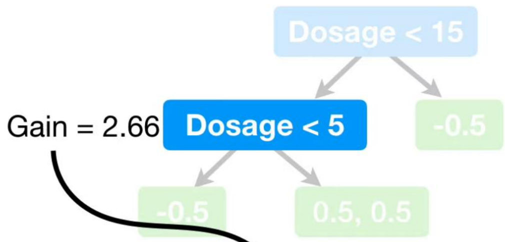


For example, if we plugged in the **Gain**…

$2.66 - 2 = 0.66$

…and set ***γ* = 2**…

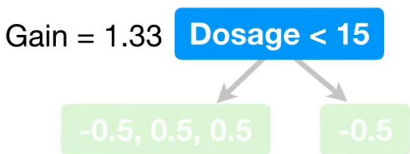
…then we would not prune because the difference is a ***positive*** number.



$2.66 - 3 = -0.34$

In contrast, if we set ***γ* = 3**…

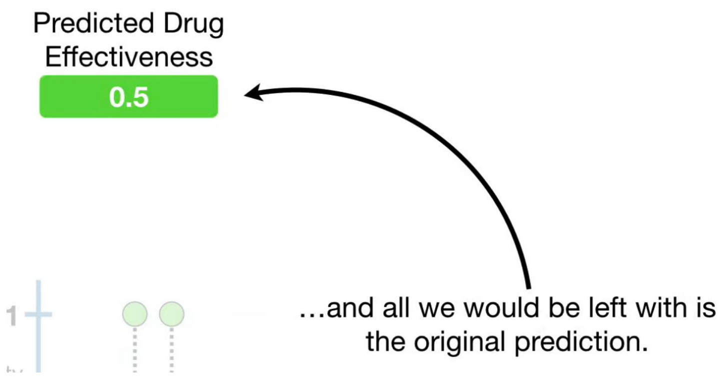
…then we would prune because the difference is a ***negative*** number.



And we would also prune this branch, because… **1.33** - **3** = **a negative number**…

Gain

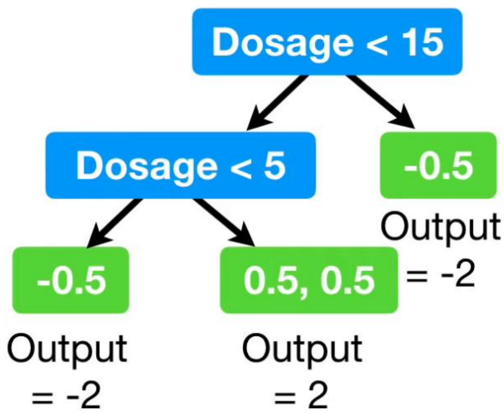
***γ* (gamma)**



Output of XGBoost Tree for Classification

For **Classification**, the the **Output Value** for a leaf is…

$$\frac{(\sum \text{Residual}_i)}{\sum [\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)] + \boxed{\lambda}}$$

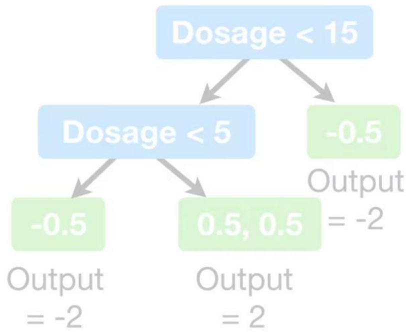


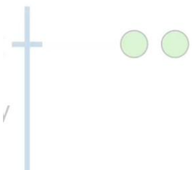
Hooray!!!
The first tree is complete!

Now that we have built the first tree, we can make new **Predictions**.

Predicted Drug Effectiveness

0.5

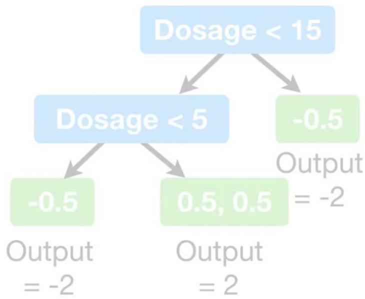




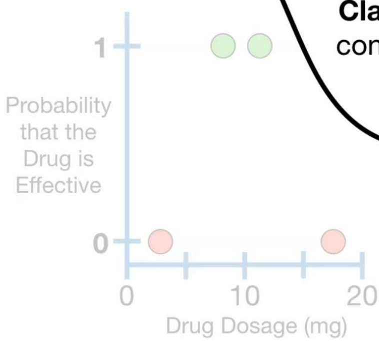
Just like other boosting methods, **XGBoost** for **Classification** makes new predictions by starting with the initial prediction.

Predicted Drug Effectiveness

0.5



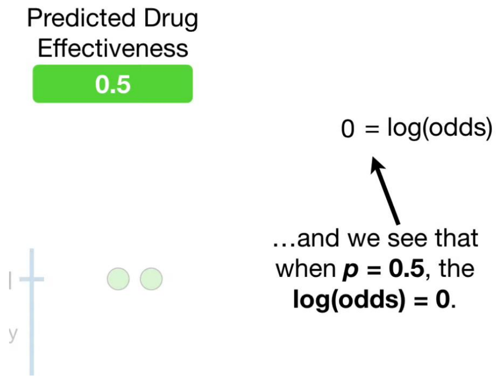
However, just like with unextreme **Gradient Boost** for **Classification**, we need to convert this probability to a **log(odds)** value.

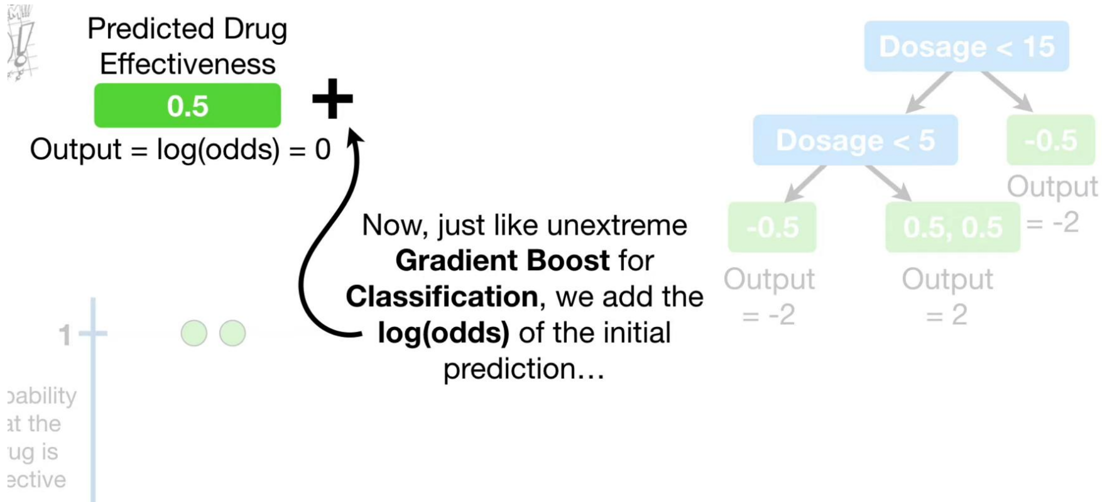


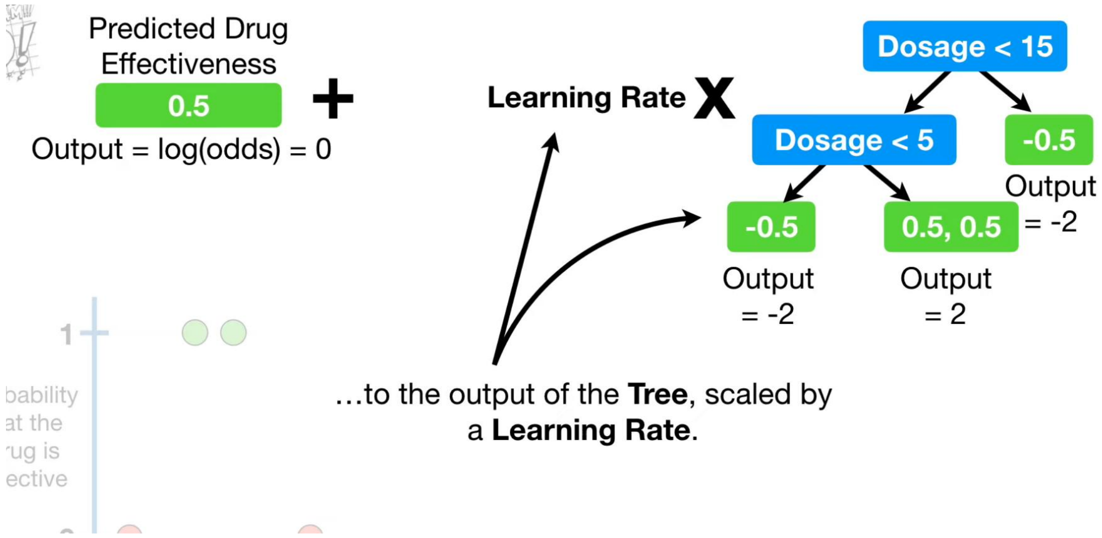
$$\frac{p}{1-p} = \text{odds}$$

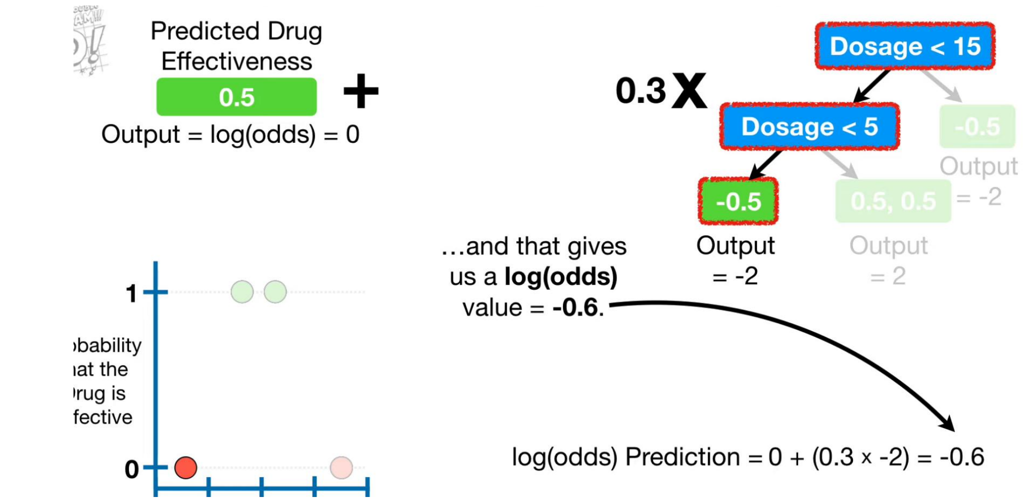
$$\log\left(\frac{p}{1-p}\right) = \log(\text{odds})$$

…we can get a formula that converts probabilities to the **log(odds)** by taking the log of both sides.


Predicted Drug Effectiveness
0.5
0 = log(odds)
...and we see that when ***p* = 0.5**, the **log(odds) = 0**.


Predicted Drug Effectiveness
0.5
Output = log(odds) = 0
Now, just like unextreme **Gradient Boost** for **Classification**, we add the **log(odds)** of the initial prediction...
Dosage < 15
Dosage < 5
-0.5
Output = -2
-0.5
Output = -2
0.5, 0.5
Output = 2


Predicted Drug Effectiveness
0.5
Output = log(odds) = 0
Learning Rate X
Dosage < 15
Dosage < 5
-0.5
Output = -2
-0.5
Output = -2
0.5, 0.5
Output = 2
...to the output of the **Tree**, scaled by a **Learning Rate**.

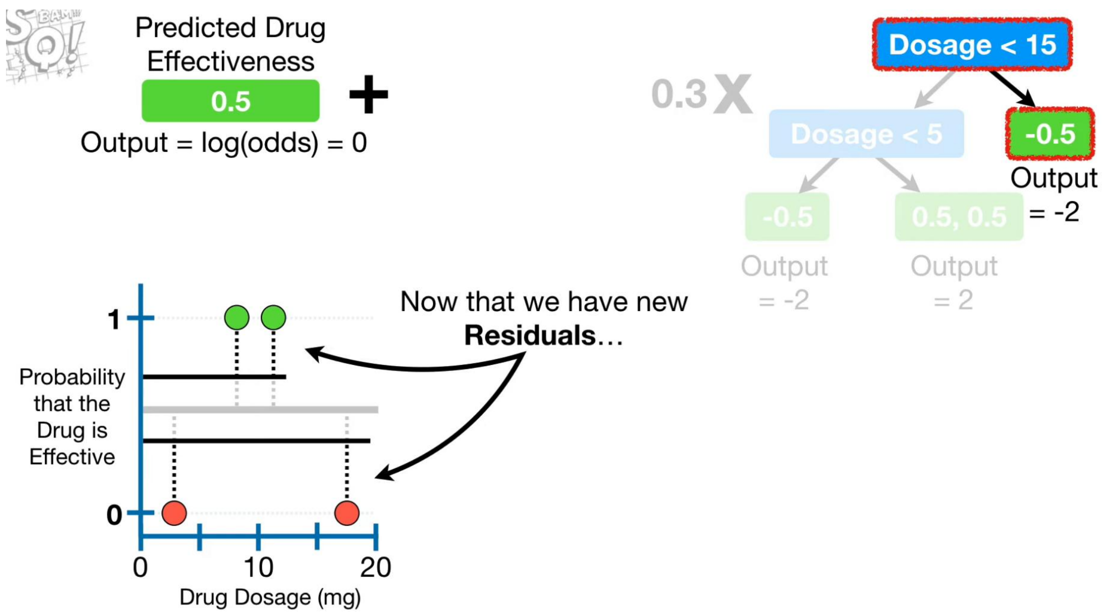


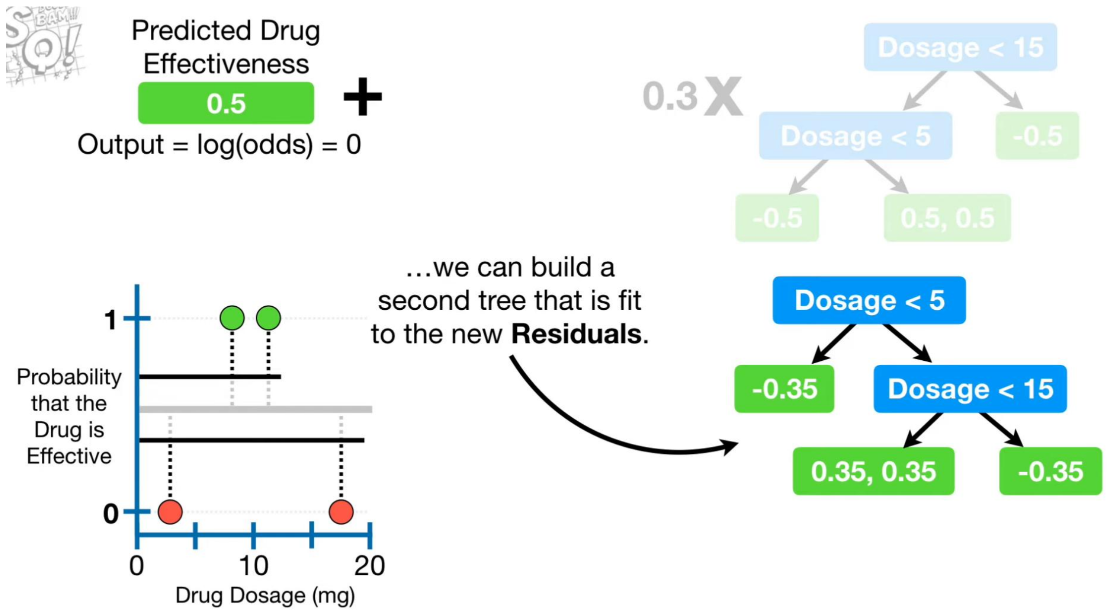
To convert a **log(odds)** value into a probability, we plug it into the **Logistic Function**.

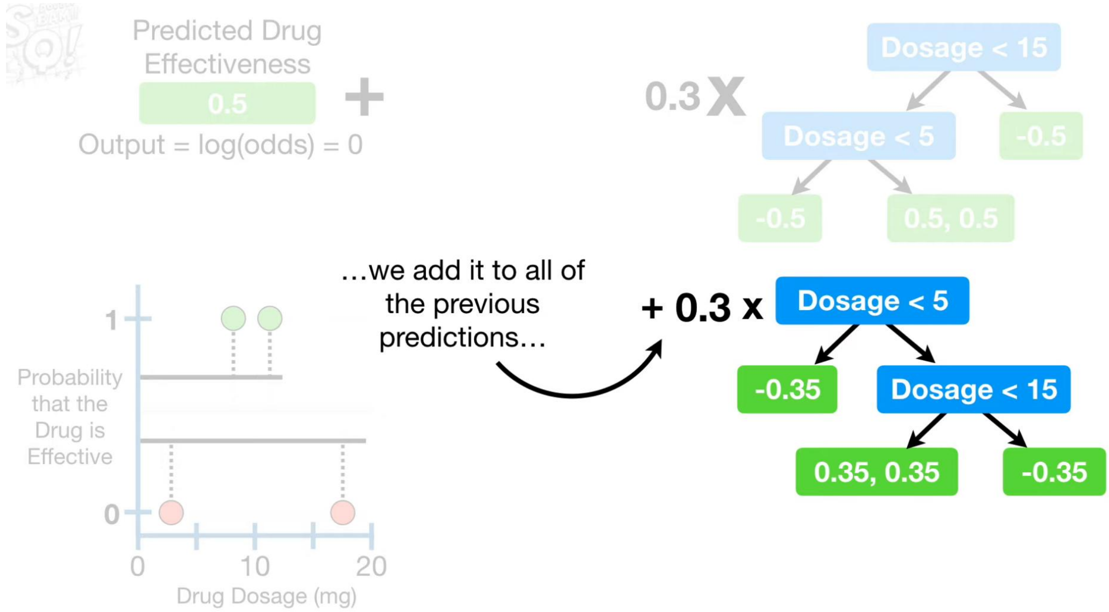
$$\text{Probability} = \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}$$

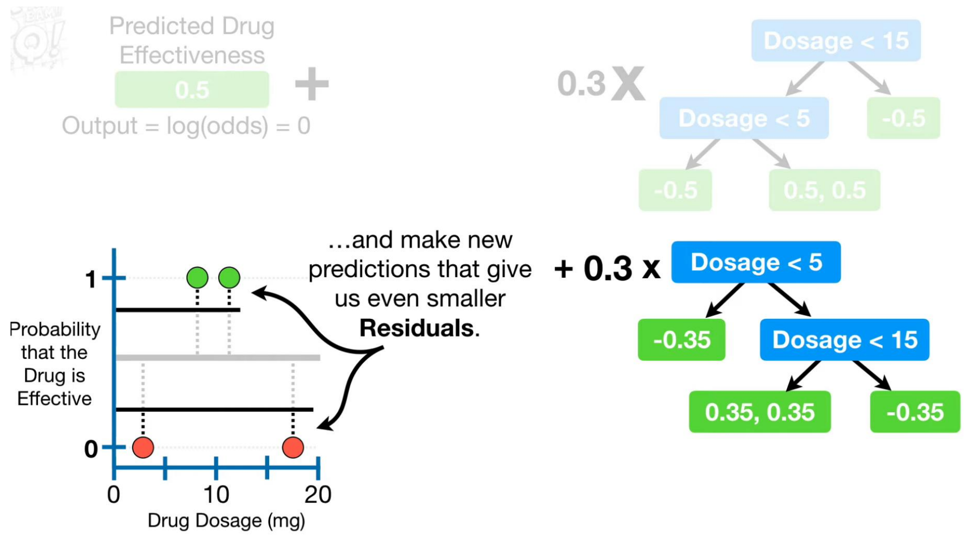
...and the the new predicted probability is **0.35**.

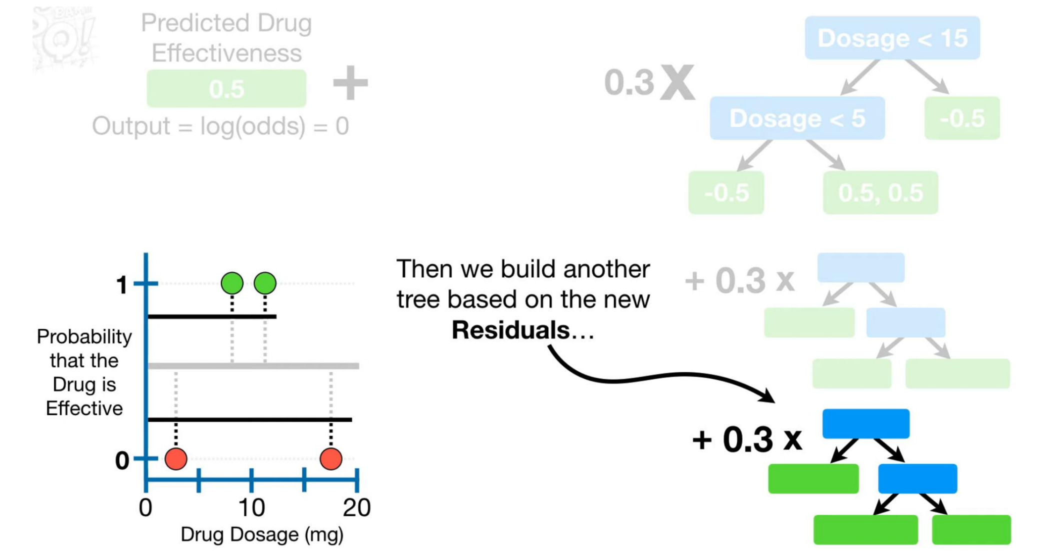
$$\text{Probability} = \frac{e^{-0.6}}{1 + e^{-0.6}} \boxed{= 0.35}$$

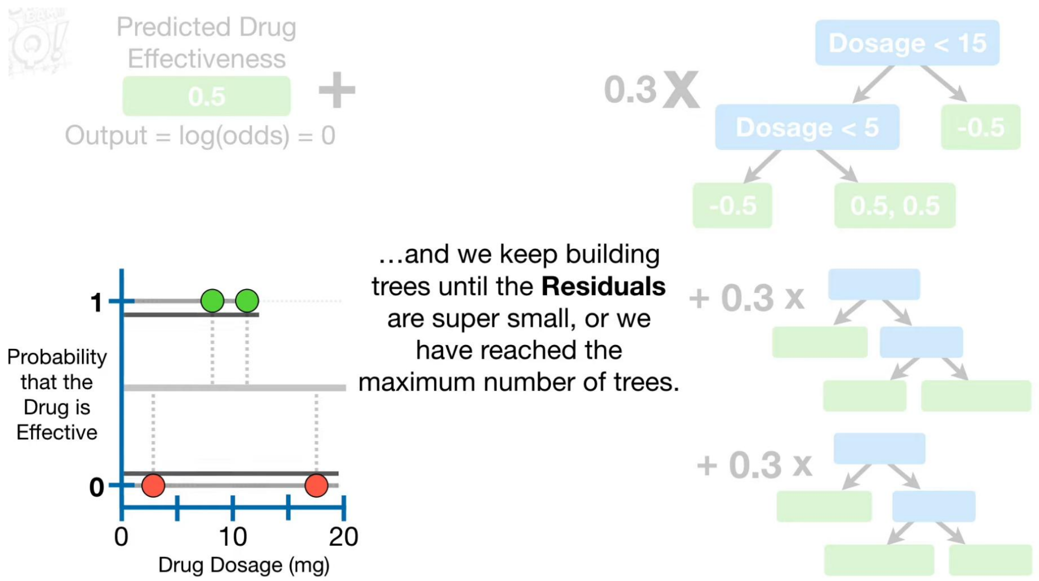



Predicted Drug Effectiveness
0.5
Output = log(odds) = 0
0.3 X
Dosage < 15
Dosage < 5
-0.5
-0.5
0.5, 0.5
...we can build a second tree that is fit to the new **Residuals**.
Probability that the Drug is Effective
1
0
0
10
20
Drug Dosage (mg)
Dosage < 5
-0.35
Dosage < 15
0.35, 0.35
-0.35


Predicted Drug Effectiveness
0.5
Output = log(odds) = 0
0.3 X
Dosage < 15
Dosage < 5
-0.5
-0.5
0.5, 0.5
...we add it to all of the previous predictions...
+ 0.3 x
Dosage < 5
-0.35
Dosage < 15
0.35, 0.35
-0.35
Probability that the Drug is Effective
1
0
0
10
20
Drug Dosage (mg)


Predicted Drug Effectiveness
0.5
Output = log(odds) = 0
0.3 X
Dosage < 15
Dosage < 5
-0.5
-0.5
0.5, 0.5
...and make new predictions that give us even smaller **Residuals**.
+ 0.3 x
Dosage < 5
-0.35
Dosage < 15
0.35, 0.35
-0.35
Probability that the Drug is Effective
1
0
0
10
20
Drug Dosage (mg)


Predicted Drug Effectiveness
0.5
Output = log(odds) = 0
0.3 x
Dosage < 15
Dosage < 5
-0.5
-0.5
0.5, 0.5
1
0
Probability that the Drug is Effective
0
10
20
Drug Dosage (mg)
Then we build another tree based on the new **Residuals**…
+ 0.3 x
+ 0.3 x


Predicted Drug Effectiveness
0.5
Output = log(odds) = 0
0.3 x
Dosage < 15
Dosage < 5
-0.5
-0.5
0.5, 0.5
1
0
Probability that the Drug is Effective
0
10
20
Drug Dosage (mg)
…and we keep building trees until the **Residuals** are super small, or we have reached the maximum number of trees.
+ 0.3 x
+ 0.3 x