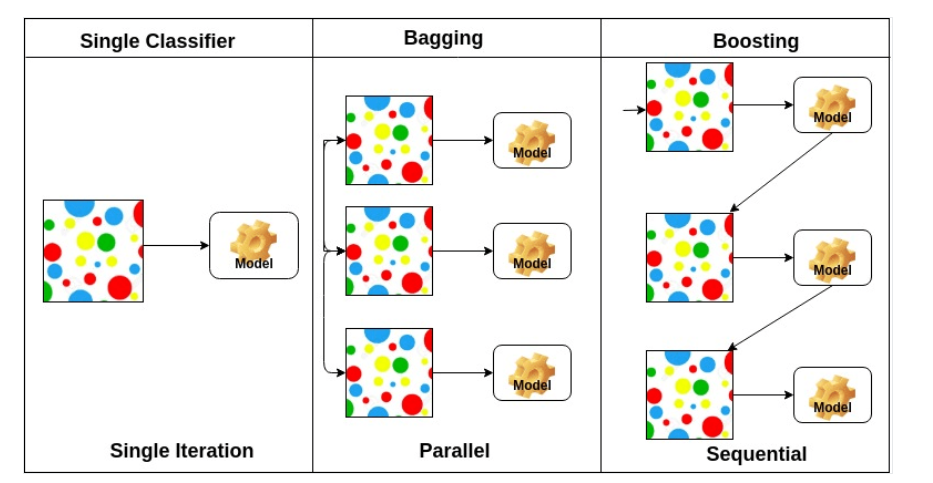
**Adaboost:**

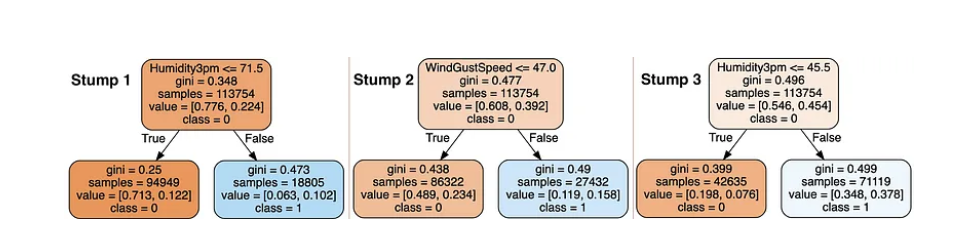
Like Random Forest, we use CART as a base estimator inside the Adaptive Boosting algorithm.

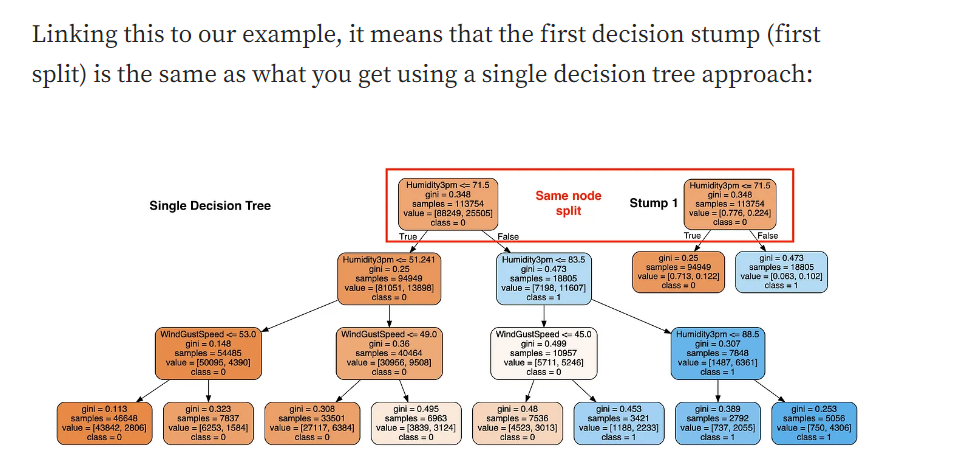


Above:

* 1. decision tree single
  2. Random forest
  3. Gradient boosting.

The core principle of AdaBoost is to fit a sequence of **weak learners**, such as **decision stumps**, on repeatedly modified versions of data. A decision stump is a decision tree that is only one level deep, i.e., it consists of only a root node and two (or more) leaves. Here is the example of 3 separate decision stumps from our AdaBoost model:

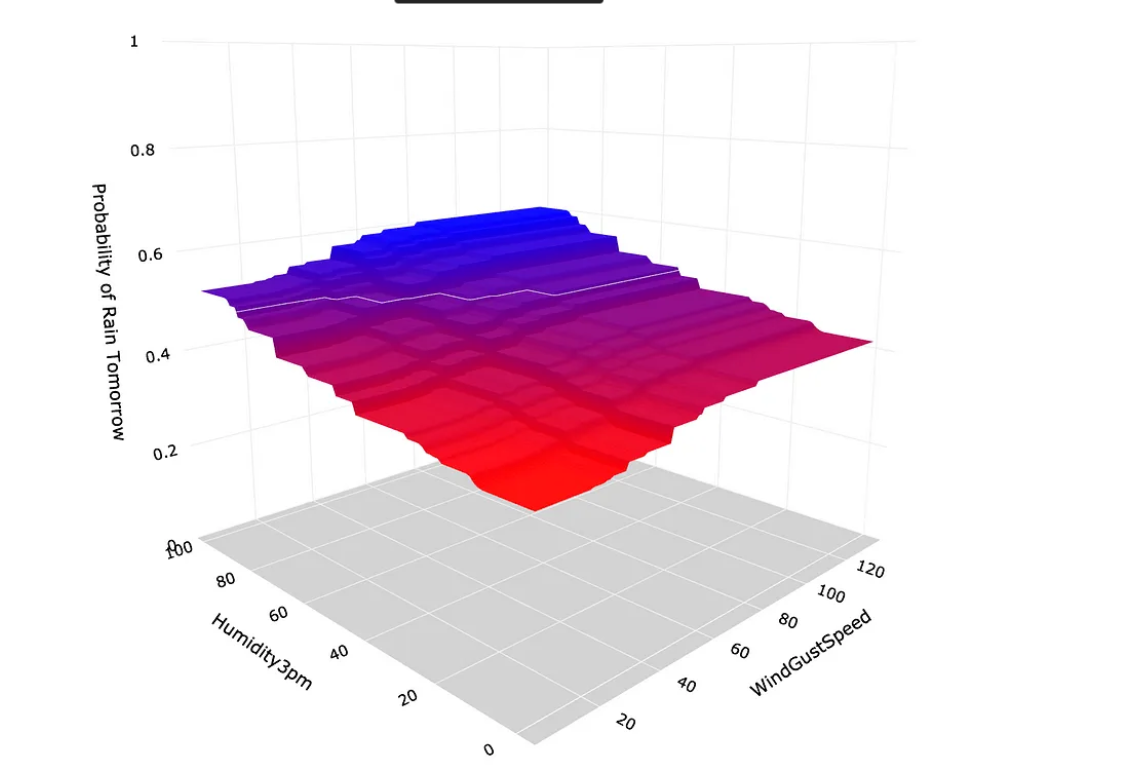
****

****

**Boosting iterations** consist of applying weights to each of the training samples (observations). Initially, those weights are equal across all observations, so the first step trains a weak learner on the original data.

For each successive iteration, the sample weights are individually modified, and the learning algorithm is reapplied to the reweighted data. Those training examples that were incorrectly predicted at the previous step have their weights increased. Meanwhile, the ones that were correctly predicted have their weights decreased. Hence, each subsequent weak learner is thereby forced to concentrate on the examples missed by the previous ones.

In the end, combining all of the weak learners into a final prediction generates a much “flatter” distribution of predictions. This is because the algorithm deliberately reduced the weights on the most confident examples and shifted focus towards harder to classify ones. As a result, we have the model predictions displayed in the graph below.

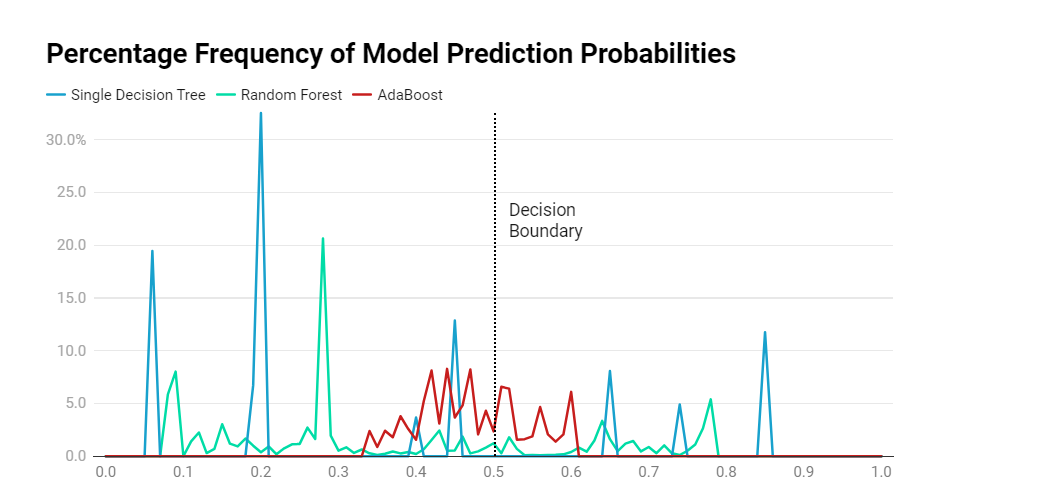
****

Note, the more weak learners (stumps) you add, the “flatter” the prediction distribution becomes.

**Prediction distribution**

Another way to visualize the prediction distribution is with a simple frequency line chart. As expected,

* The single decision tree has few spaced-out predictions.
* Random forest shows a much more even distribution of predictions.
* AdaBoost has all of its predictions located close to the decision boundary (at 0.5).

****

## Performance

While the three approaches produced very different probability distributions, the final classification results were very similar. The performance can be evaluated in many ways, but for the sake of simplicity, I am only showing accuracy (on test sample) here:

* Single Decision Tree: 82.843%
* Random Forest: 83.202%
* **AdaBoost: 83.033%, best performance.**

While the performance can be slightly improved with some additional hyperparameter optimization, the similarity of the results above tells us that we are fairly close to extracting the maximum information contained within the features used.

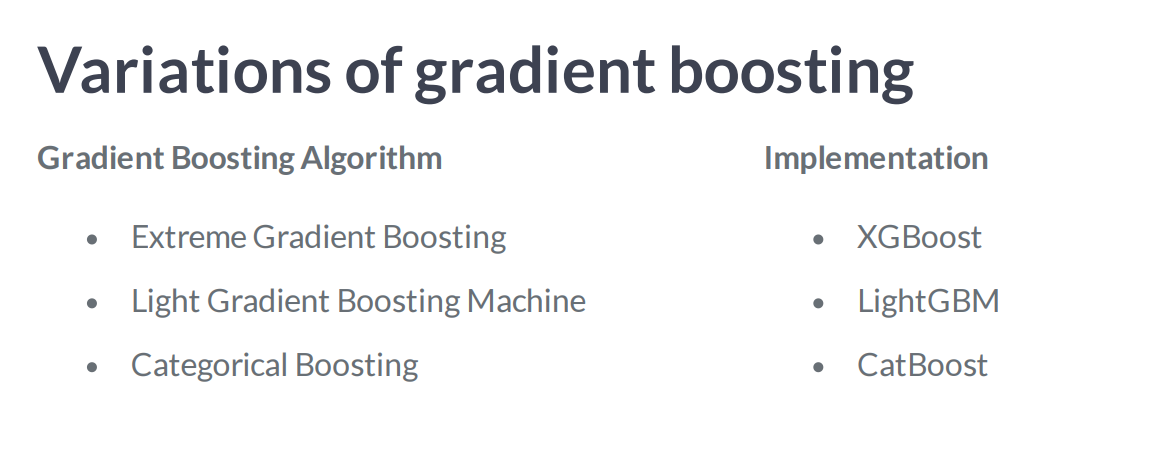
## ****AdaBoost Limitation****

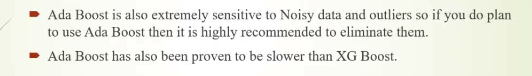
The resulting “flat” probability distribution of AdaBoost is its main limitation. Depending on your use case, it may not be an issue for you. Say if you only care about assigning the correct class, then the prediction probability is less relevant.

However, if you care more about the probability itself, you may want to use Random Forest, which provides you with probability predictions such as 9% or 78%, as shown in the rain prediction modeling above. This is in contrast to AdaBoost, where all predictions are much closer to 50%.

The core principle of AdaBoost is to fit a sequence of weak learners, such as decision stumps, on repeatedly modified versions of data. **A decision stump is a decision tree that is only one level deep, i.e., it consists of only a root node and two (or more) leaves.**

It was the first ensemble method algorithm due to its simplicity of only trees with only one deep level there more advanced ensemble methods such as Gradient Boosting and its variations:





Another disadvantage is **that through the sequential method each decision tree is only one level deep, i.e., it consists of only a root node and two (or more) leaves.**

**Gradient boosting and its variations accept trees with deeper levels, having models more robust for large dataset.**

**In Adaboost we use decision trees as estimators.**

**Adaboost sklearn parameters optimization Classification:**

[**https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html**](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html)

**base\_estimator= DecisionTreeClassifier(min\_samples\_leaf=1000, max\_depth=1)**

**n\_estimators: the number of trees.**

**learning\_rate:**  A traditional default value for the learning rate is 0.1 or 0.01, and this may represent a good starting point on your problem.

**Algorithm *{‘SAMME’, ‘SAMME.R’}, default=’SAMME.R’*:** If ‘SAMME.R’ then use the SAMME.R real boosting algorithm. estimator must support calculation of class probabilities. If ‘SAMME’ then use the SAMME discrete boosting algorithm. The SAMME.R algorithm typically converges faster than SAMME, achieving a lower test error with fewer boosting iterations.

**Adaboost sklearn paramters optimization Regression:**

[**https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html#sklearn.ensemble.AdaBoostRegressor**](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html#sklearn.ensemble.AdaBoostRegressor)

Same parameters than classification excluding the loss function:

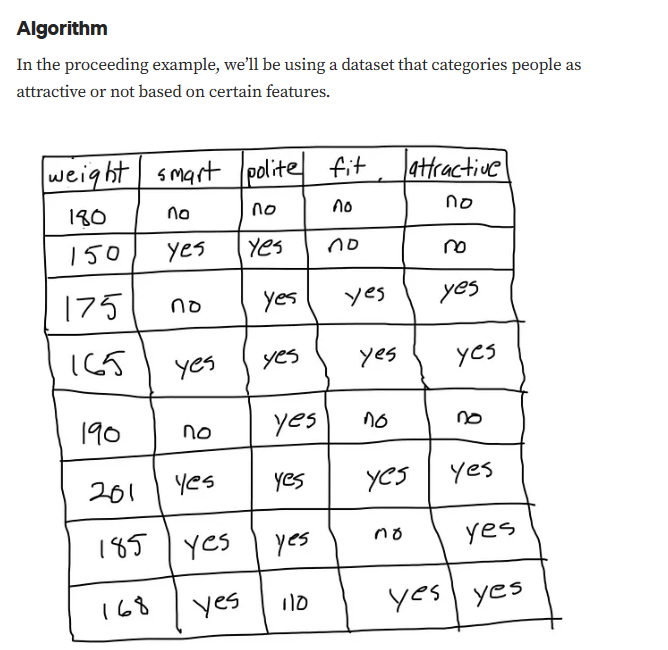
**Loss *{‘linear’, ‘square’, ‘exponential’}, default=’linear’***

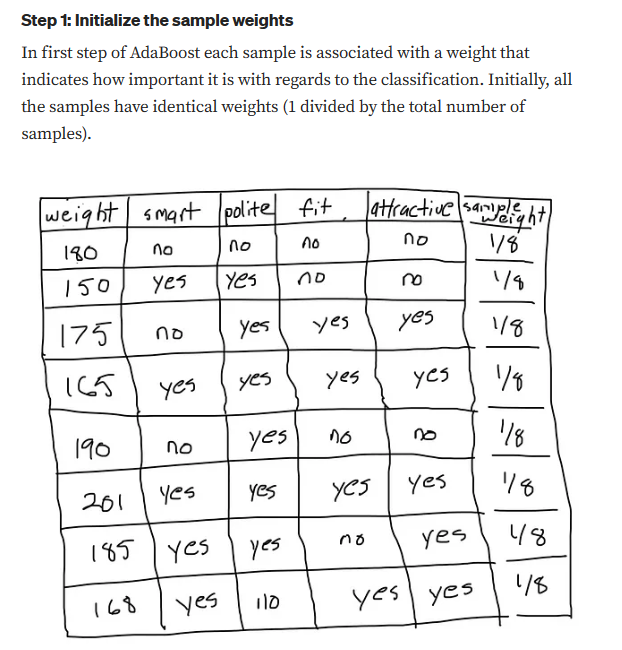
**The loss function to use when updating the weights after each boosting iteration.**

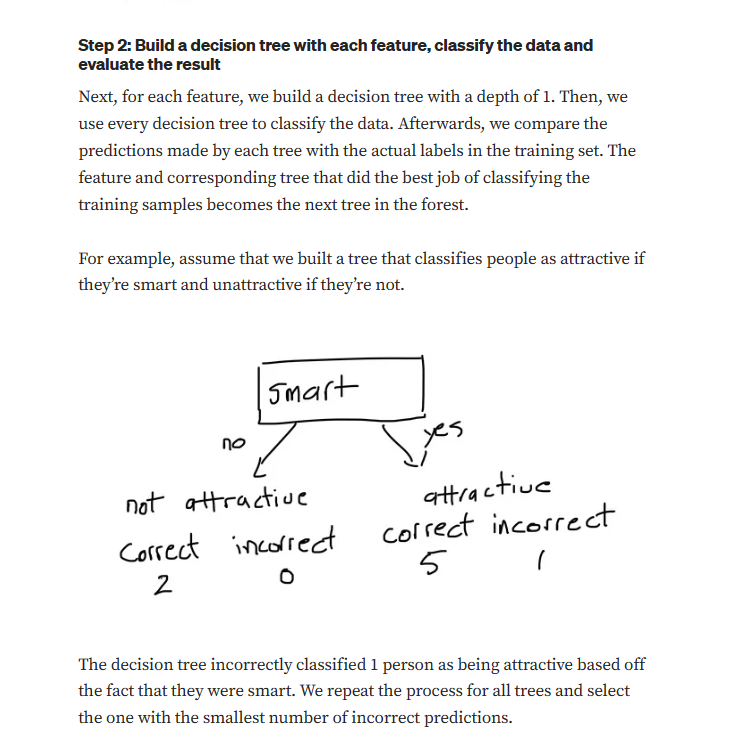
**Loss is exclusive to AdaBoostRegressor and sets the loss function to use when updating weights. This defaults to a linear loss function however can be changed to square or exponential.**

# In the case of AdaBoost, the shifting is done by up-weighting observations that were misclassified before, while Gradient Boosting identifies the difficult observations by large residuals computed in the previous iterations.

Example:

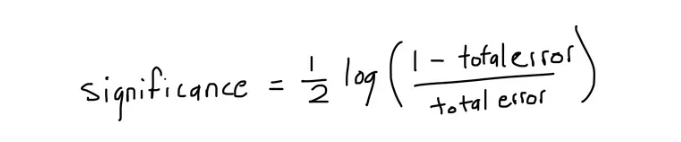


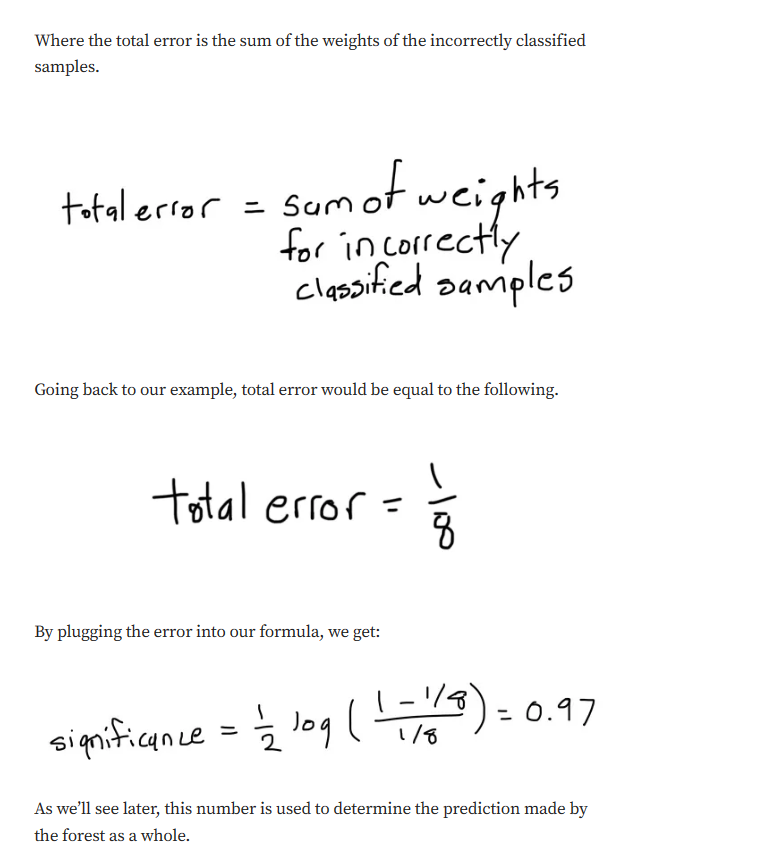


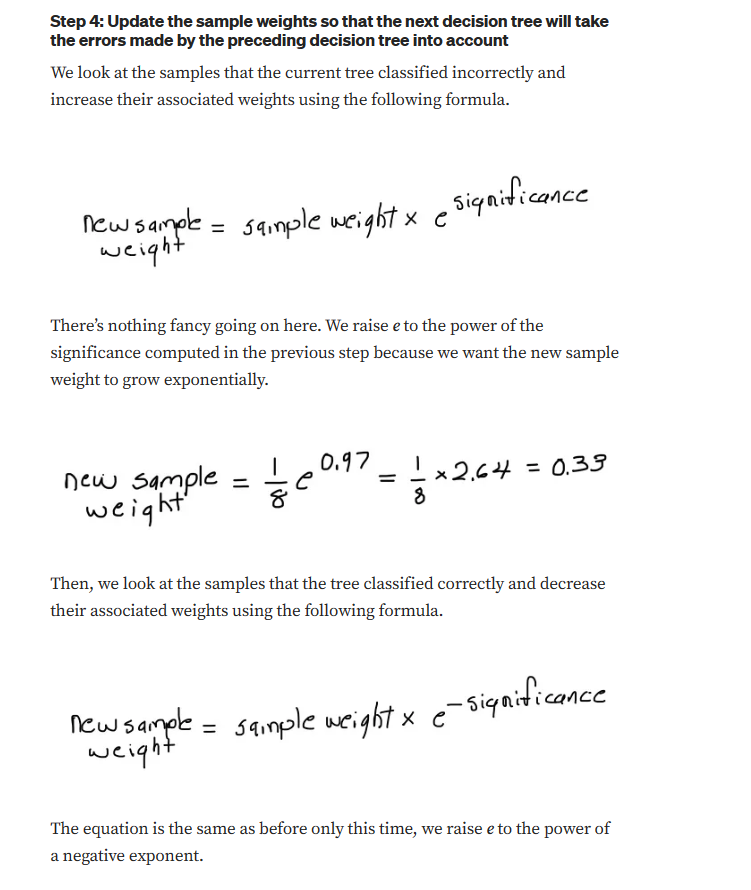


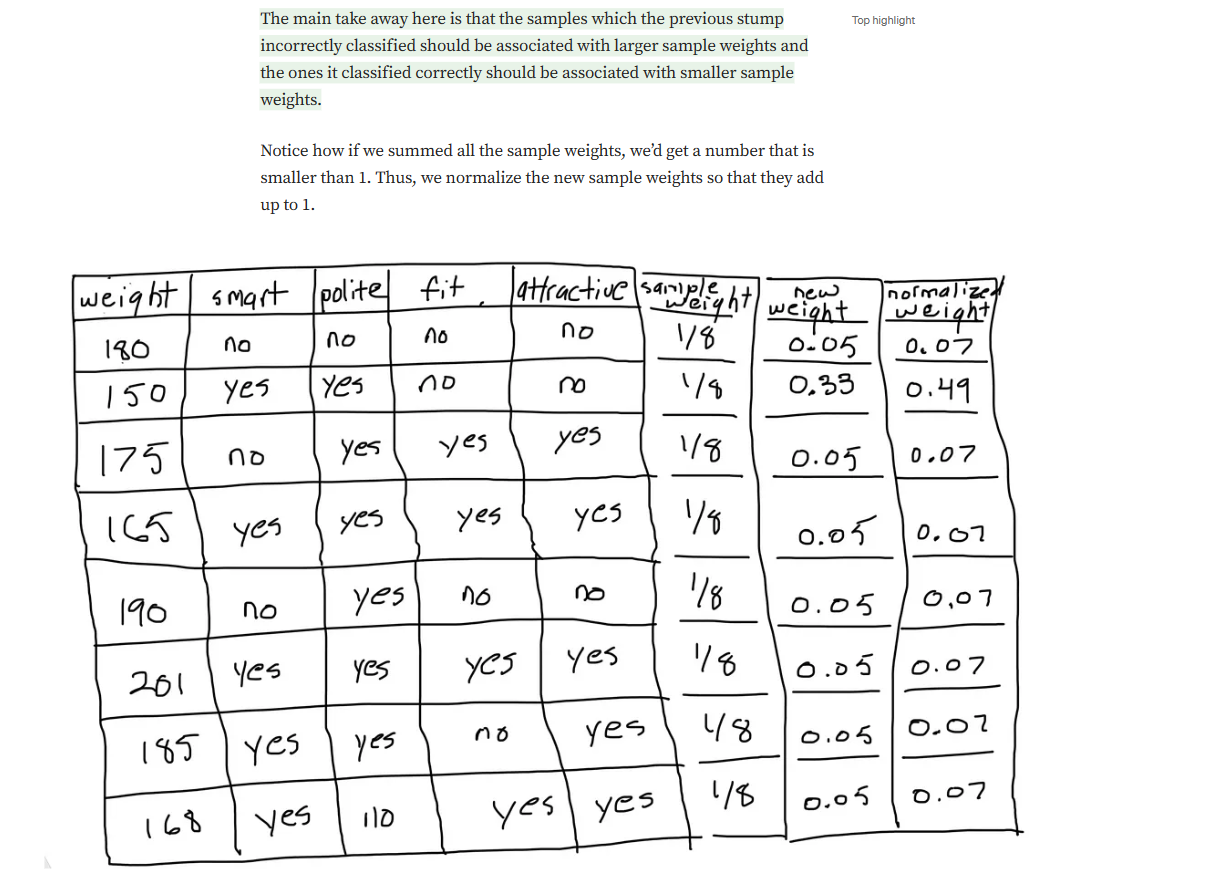
## Step 3: Calculate the significance of the tree in the final classification

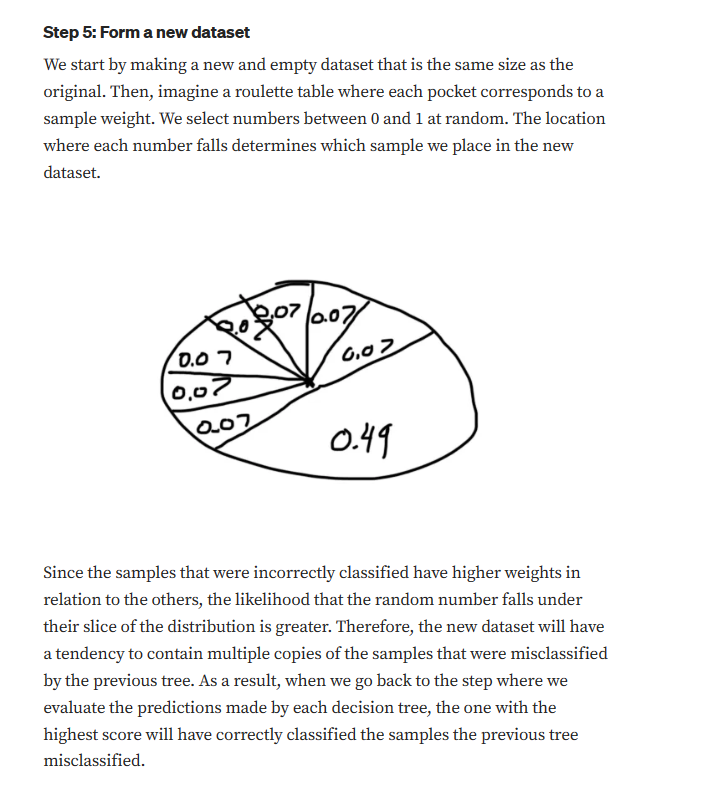
Once we have decided on a decision tree. We use the proceeding formula to calculate the amount of significance that it has in the final classification.

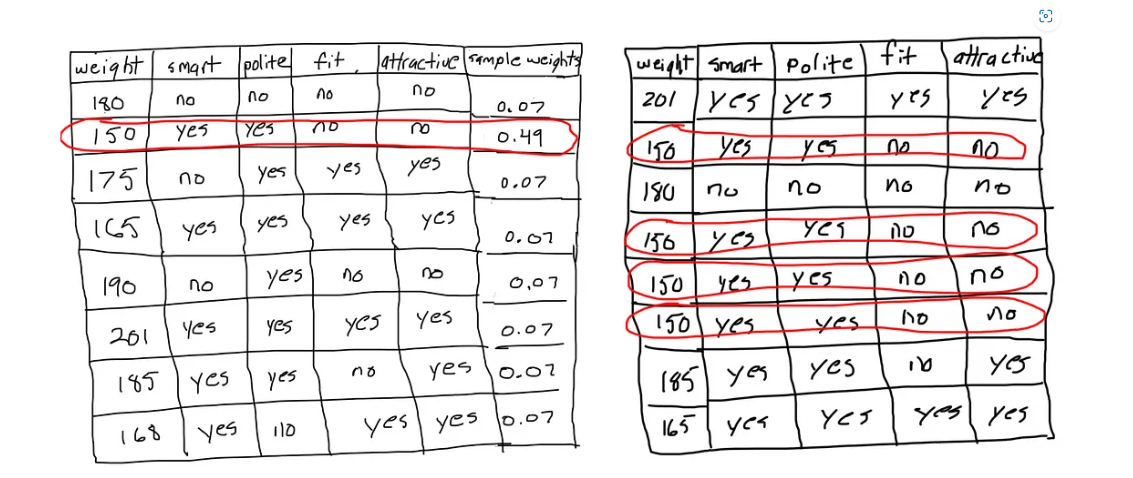












## Step 6: Repeat steps 2 through 5 until the number of iterations equals the number specified by the hyperparameter (i.e. number of estimators)

