**Decision Trees and Random Forests**

Dataset Used

The Dataset used, consisted of handwritten images of numbers ranging from 0 to 3. The images have the dimensions of 28 x 28, consisting of 3 channels each. Different folders store these images, where the name of the folder represents the label of the respective images. There are further two folders, testing and training, where the images have been divided for testing and training purposes. The images were created from scanning different handwritten numbers. They are monochrome in nature, but have been stored in 3 channels format. The task is to classify these images using decision tree and random forest.

Decision tree works on the principle of dividing the data into branches based on the attributes’ quantity to segregate the labels as much as possible. The objective of this division is to maximize the information gain, calculated as the difference between the parent and child entropy or gini coefficient, with each division. This division is usually done till will reach the leaf node, or a stage where the labels can no longer be further divided. In other words, a leaf node is a node where only one of the class labels are found, hence no more division is feasible. However, with no restrictions on parameters, decision trees are prone to overfitting. They are formed down to the last remaining data point, even if the last node contains no more than 2 data points. Not only does overfitting reduce the accuracy of predictions, it also increases the time complexity to form a decision tree, and reach a result. Hence, we require pruning to reduce overfitting by limiting the formation of nodes by moderating certain parameters like the maximum depth, minimum number of data points required before splitting etc.

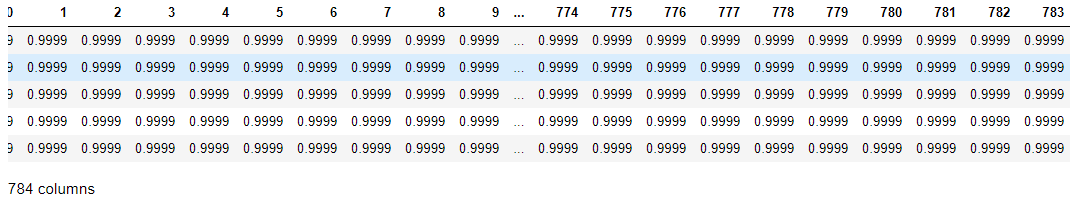
Random Forest, an example of ensemble learning, wherein decision is taken after considering many the results of multiple algorithms, uses many decision trees at its disposal. The only different parameter that it has is the number of estimators or the number of decision trees to be used.

Prepping the Dataset

C:\Users\Rahul Singh\Downloads\data\test-images\2\4.7.png

i Example of an Image in the Dataset

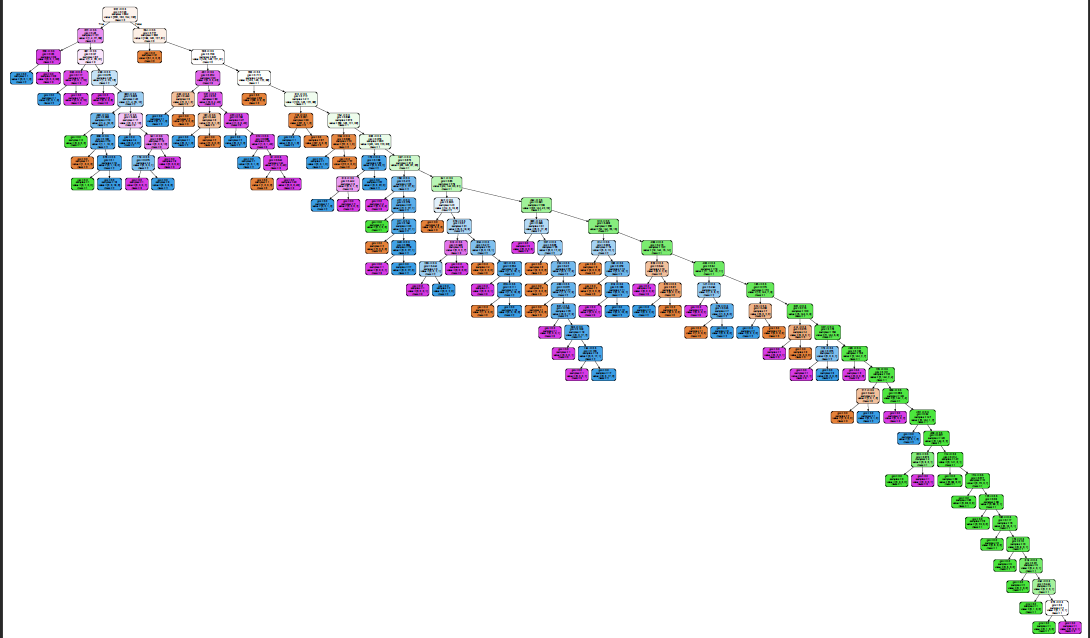
The dataset has to be in the form of flattened pixel values. To do so, the image is converted into a grayscale format from RGB format. Not much information is lost, since the images are primarily monochrome, but stored in RGB format. A pre-determined formula is used to convert the images, and store the results in one pixel. Then, the 28 x 28 image is flattened into 784 x 1 image. The same is done for all the images in both, testing as well as training folders. In our case, we have used GridSearchCV for the calculation of accuracy, and hence do not need to feed the training and testing data points separately. So, we mix up all the images to make one dataset. The labels have been saved separately while traversing through the images.



ii Example of the dataset formed

Decision Tree Implementation

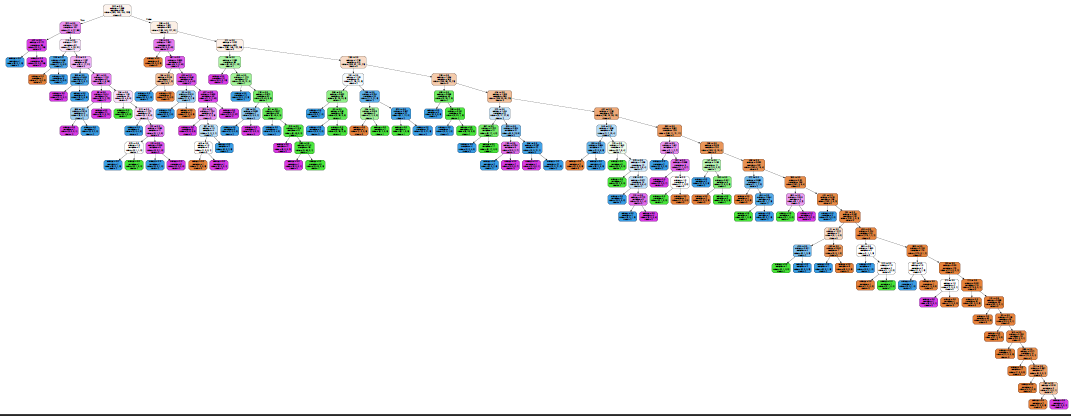
The decision tree formed with the default parameters looked as follows:



The above decision tree was visualized with the help of Graphviz. Here, we notice that the tree has 30 levels or depth. Each colour represents the dominant class found at that node. We can confirm 4 different colours for each class. The intensity of the colour is proportional to the difference between the dominant and the other classes’ numbers. Consequentially, nodes with a single class or label present have the most intense colours whereas the ones having equal amounts of each labels are depicted in white colour. A noticeable feature is how the tree gets divided in 2 major chunks, with the green nodes (or class ‘1’) dominating one of it. This happens due to overfitting, where the tree keeps on dividing until it reaches the last node to reveal only 3 instances of a class different from green i.e. purple (class ‘3’). If that were generalized, then the tree wouldn’t have grown this intricate and specific. Leaving that branch aside, almost all of the nodes end with a single instance, which again indicates overfitting. Hence, we hypertune certain parameters, and implement them simultaneously in an attempt to reduce overfitting.

Criterion

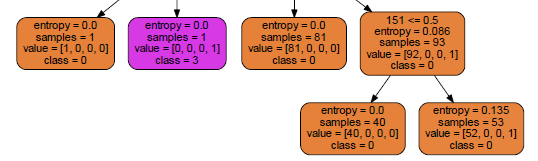
The criterion parameter decides the mathematical expression to calculate Information Gain. It can be either GINI or Entropy. Both of these consider the probability of a class appearing multiplied by another term, which is base 2 logarithm for entropy and complement of the original probability for GINI. Different trees are formed as a result, but deciding which one is better varies from dataset to dataset.

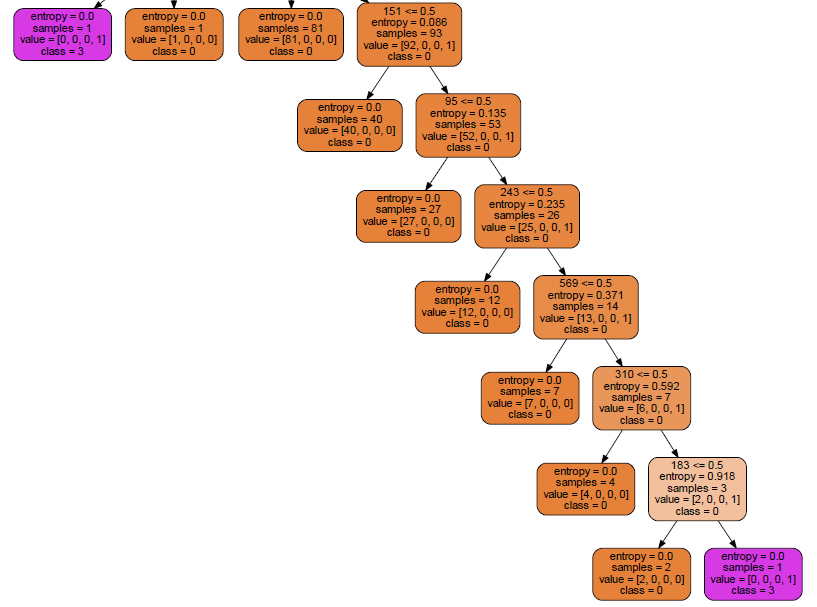


The above figure is the decision tree formed with the criterion as Entropy. As we can see, the length is reduced, and different classes have been classified as per the conditions. Another notable feature is that the width of the tree exceeds that of the GINI one. Instead of having a node split downwards, it continues to spread sideways. Meaning, in this case, the depth of each node is less when compared to the previous one. However, that does not change the fact that this suffers from overfitting too. We will be considering the effects of other parameters on both of these from here onwards.

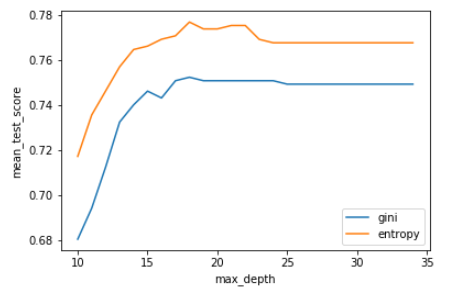
Maximum Depth

As the name suggests, maximum depth is the threshold beyond which no nodes are further split. It abruptly ends the splitting of nodes when a specific depth is reached. All the other underlying conditions, that would have been considered if not for the limitation, are completely negated. In other words, time and resources are saved.





The above 2 snippets are of the decision tree (with criterion set as ‘entropy’). The first one has been implemented with a Maximum Depth of 18, which gave the best results in terms of accuracy when fitted. The bottom few layers have been cut off, and the conditions will no longer be considered while predicting classes. The accuracy was tested for a range of values for max depth and the following result was obtained.

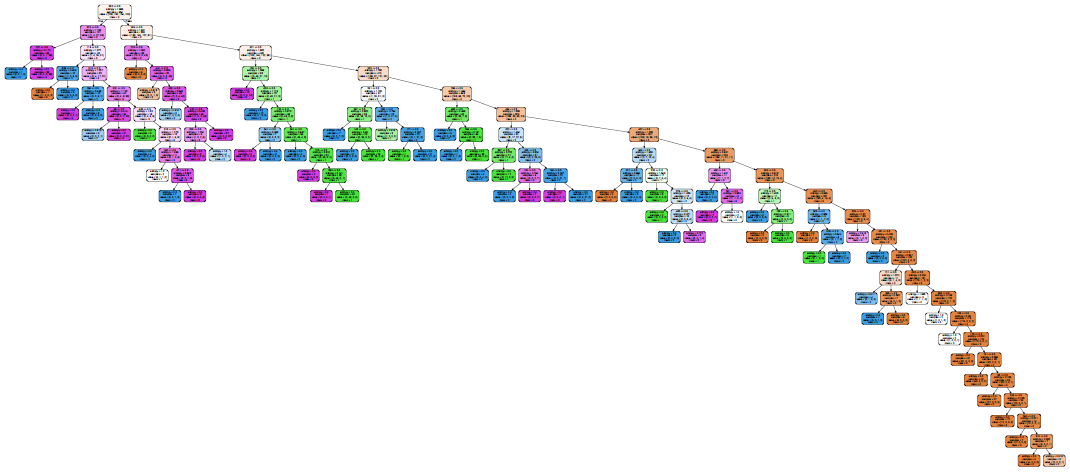


The line originating is that of the “GINI” criterion, and the one that begins after reaching the end is that of the “Entropy” (and will be the same for the remaining graphs in the report). As we can see, entropy yields a better result than GINI, probably because of lower depth that entropy had, to begin with. The results become stale after a certain value, which is supposedly the original depth of decision tree.

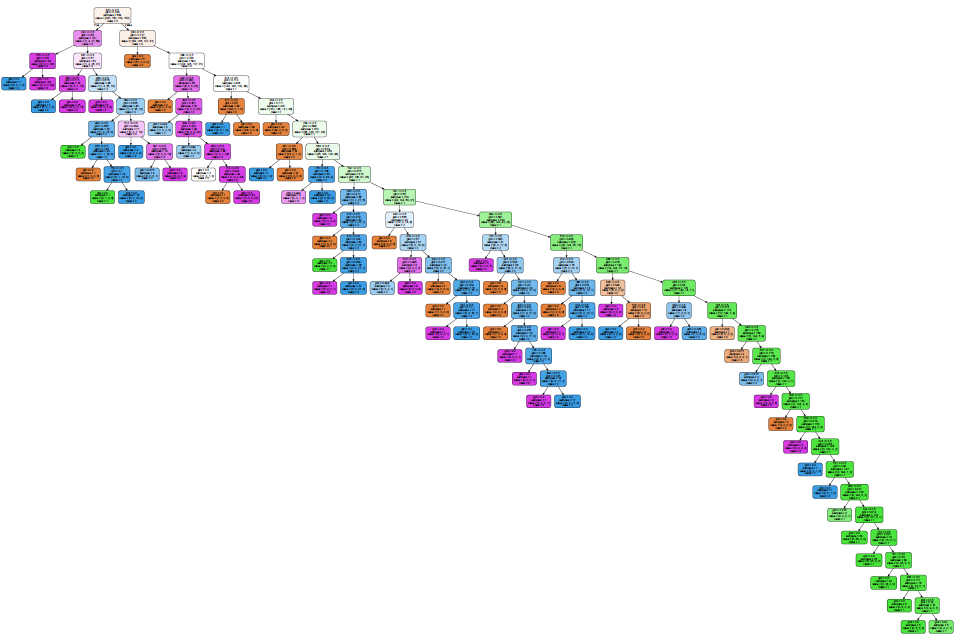
This helps improve accuracy because it eliminates the splitting of lower levels. Due to the process of selecting the splitting method that yields the highest Information Gain, the important, or rather the most valuable splits are done in the top few layers itself. The ones down below usually have miniscule information gain, and only lead to overfitting. Hence, maximum depth is a great tool to reduce overfitting.

Minimum Samples Split

The minimum samples split parameter defines a minimum threshold for the number of samples or data points to be present in a node for it to split. This avoids any unnecessary splits that would yield no significant result.

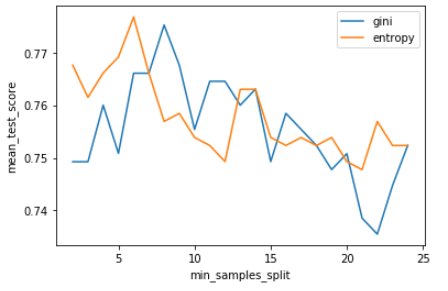


Min sample split 6, entropy



Min sample split 8, GINI

The above snippets are for the decision trees giving the best accuracies for their respective criterion in a specified range of Minimum Sample Split. We observe that the nodes are trimmed off at corners of few branches. Unlike the maximum depth, which indiscriminately cuts off the bottom nodes, this parameter takes a mathematical approach to decide which nodes to cut.

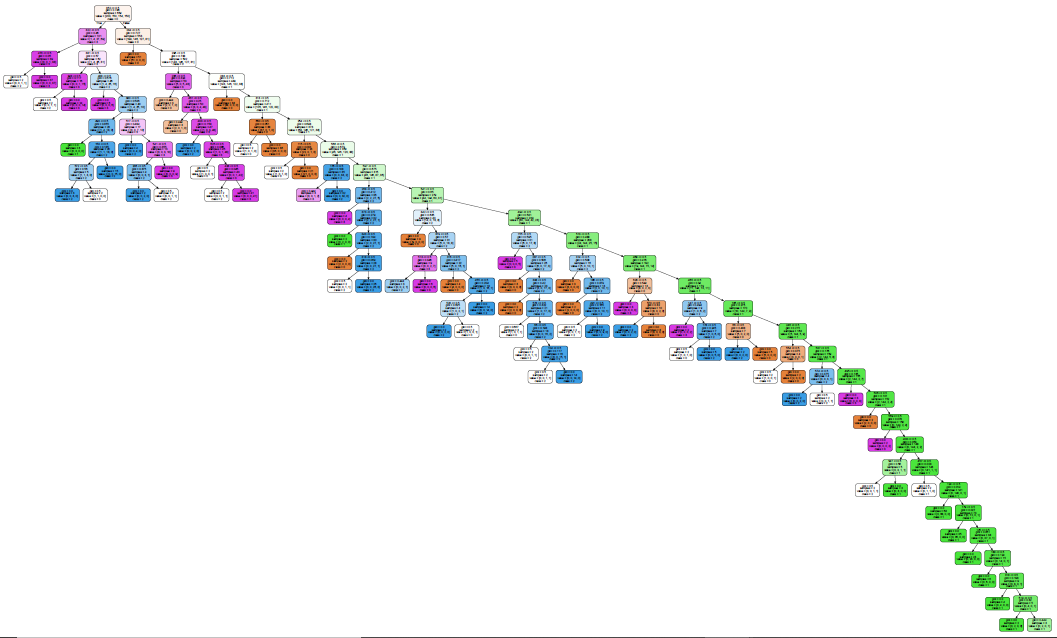


The above graph shows that minimum samples split gives a higher accuracy for the first few values. Maybe because of many important splits taking place at a fewer number of samples. It is clear is that the number of samples may or may not to be related to information gain.

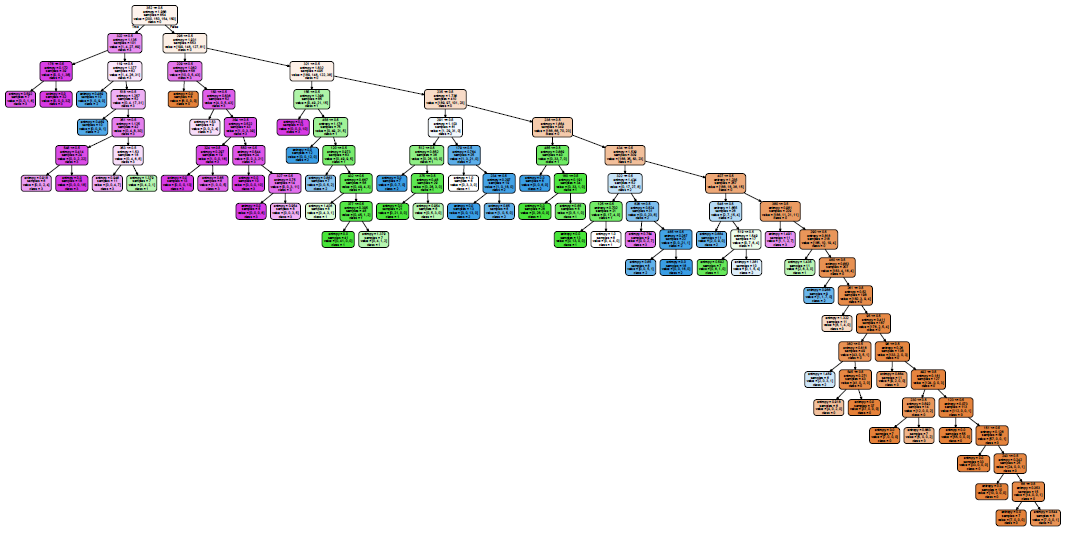
Overfitting usually happens because of unnecessary splits. One way to detect that is by the tree makes splits only to accommodate a few samples, which may be misleading, and could be generalized. By selecting a minimum samples split, we get rid of such sample splitting.

Minimum Sample Leaf

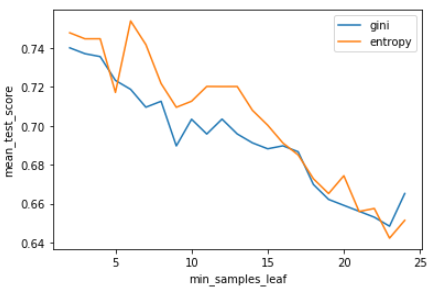
Minimum Sample Leaf defines the minimum number of samples to be found in the children nodes after splitting. It is similar to minimum samples split in the sense both depend on the number of samples in nodes. The only difference is that the minimum sample split limits the number of samples in the parent nodes whereas minimum samples leaf limits it in the children nodes.



Min samples leaf 2 gini



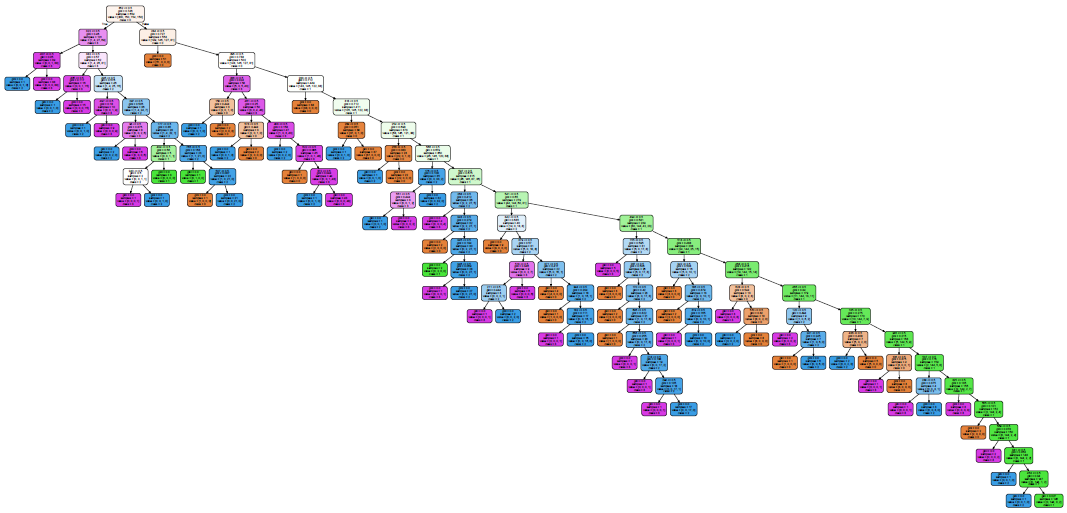
Min samples leaf 6 entropy



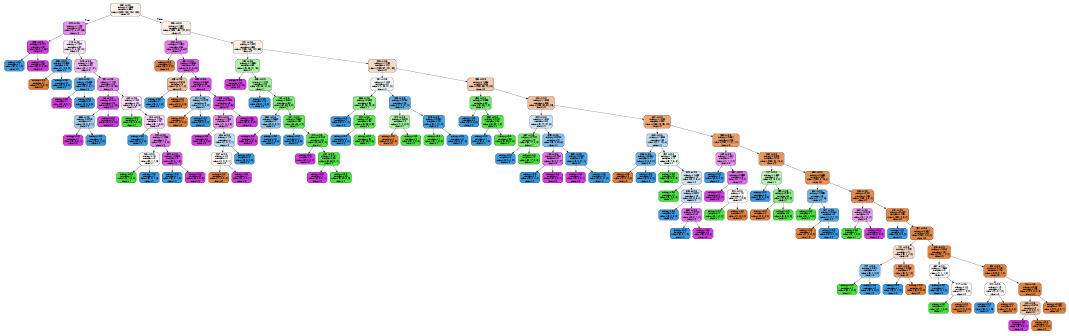
We observe that the patterns or trends in both the tree as well as the graph is similar to that of the Minimum Samples Split.

Minimum Impurity Decrease

Minimum impurity decrease defines the minimum information gain before splitting a node.

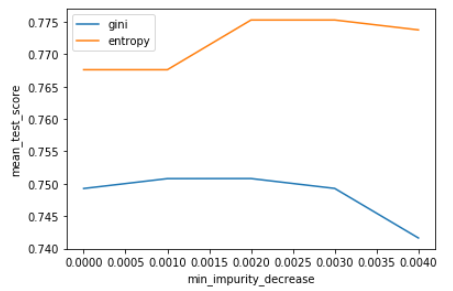


Min imp decrease 0.001 gini



Min imp decrease 0.002 entropy

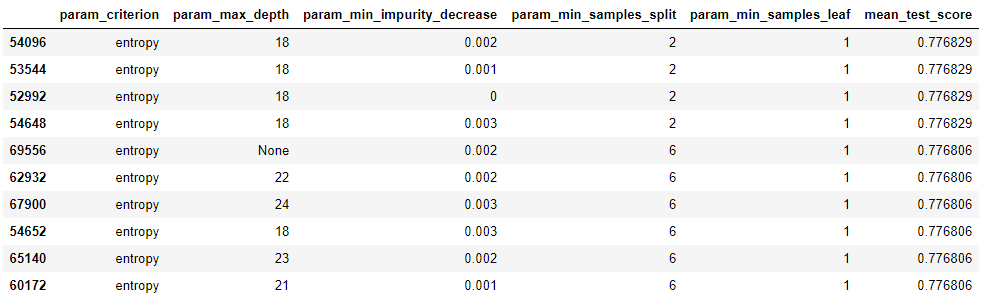
As we can see, the minimum impurity decrease factor seems effective at pruning the tree. The last few nodes have all been trimmed, and generalized in a few nodes.



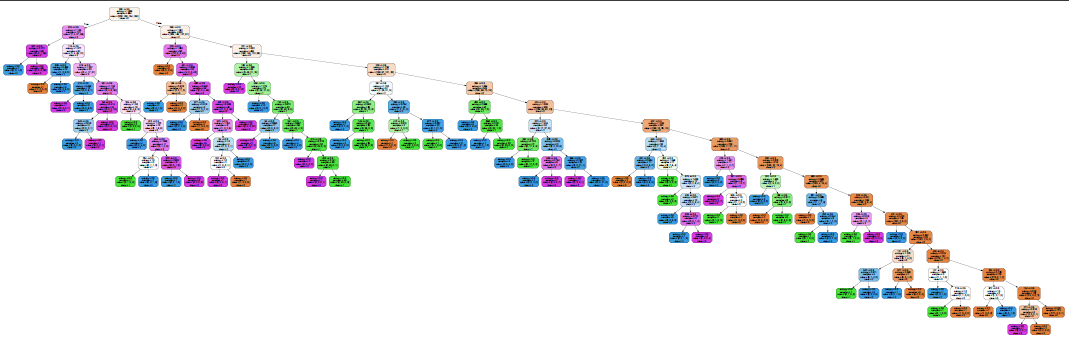
As mentioned before, the splits at the end of the tree usually have very less information gain, so this parameter takes care of that. The difference that these have is in the approach of cutting the tree short. Another difference between these parameters and Maximum depth is that they are capable of clipping the tree at any possible node unlike Maximum depth, which cuts off nodes after reaching a specific depth.

Best Fit

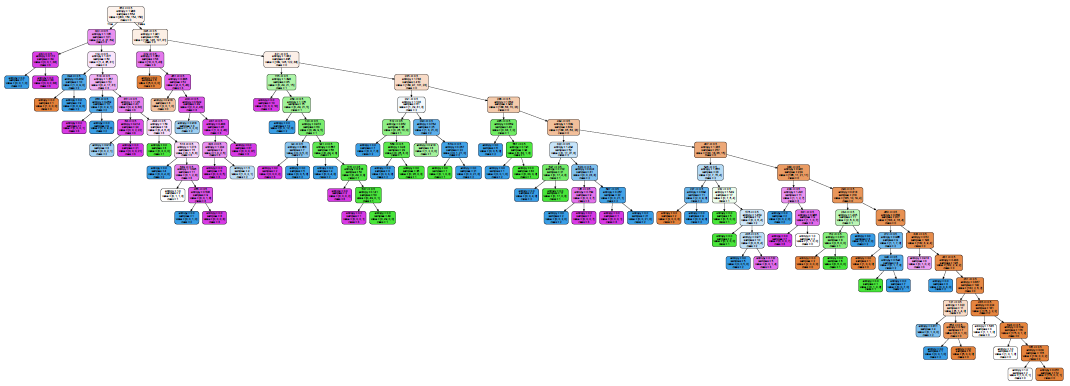
Decision Tree



Max\_Depth = 18, Min\_Impurity\_Decrease = 0.002

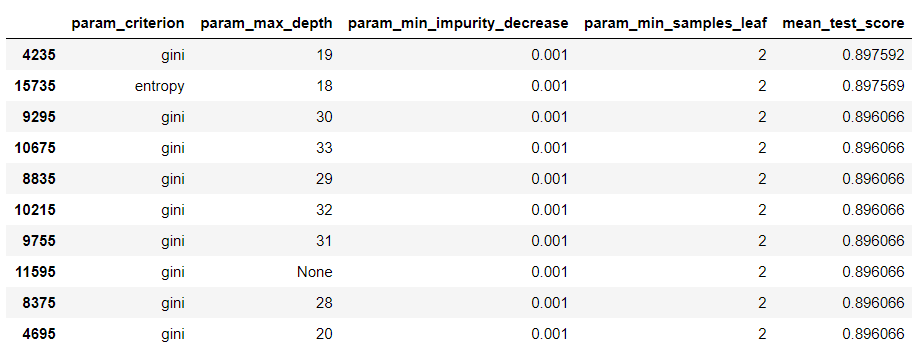


Min\_Impurity\_Decrease = 0.002, Min\_Saples\_Split = 6



Above are the depictions of two of the best fitting decision trees, considering accuracy as the metrics. It is observed that these two are almost similar and yield an accuracy of approximately 77%.

Random Forest



Random Forest yields the best accuracy of 89% for the same dataset.