COL774 ASSIGNMENT 3A

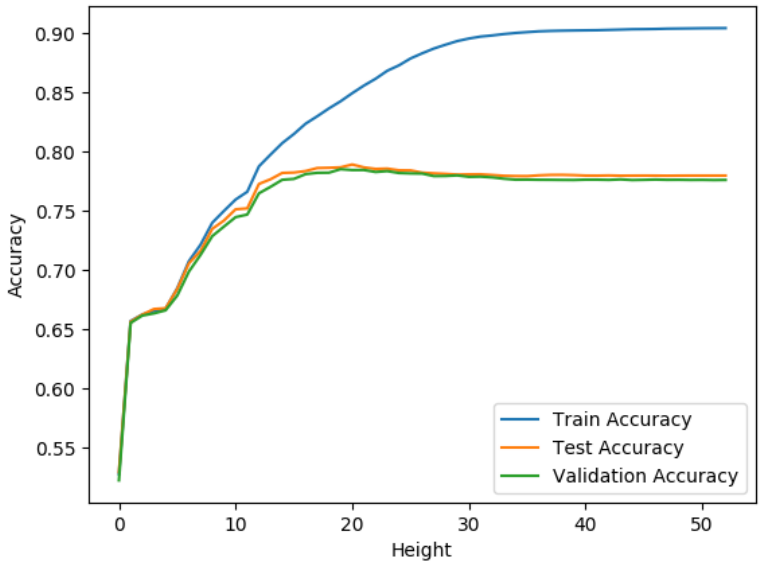
Raval Vedant Sanjay

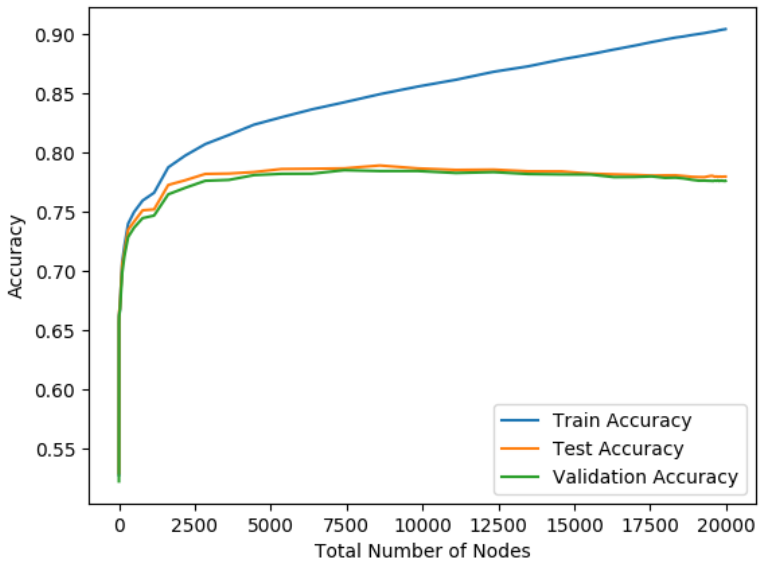
2017CS10366

1 Decision Tree construction and Pruning

Here, in order to construct the decision tree, I used the criteria of maximum mutual information amongst all the expandable nodes, for all the different attributes. I expanded the tree in such a way that it will go on increasing till a certain depth, which can be specified. Also, I am expanding the tree in a breadth-first style i.e going through all the open nodes at a level and then expanding them all to the best values to reach the other level and so on..

Here, in order to improve the speed, I would vectorize the functions and then use the numpy functions on them, which are very efficient as compared to using many for-loops! In the final tree constructed, the depth was around 52 and there were approx 20,000 nodes. The plot for the **accuracies v/s height of the tree** and **accuracies v/s number of nodes** are given below:



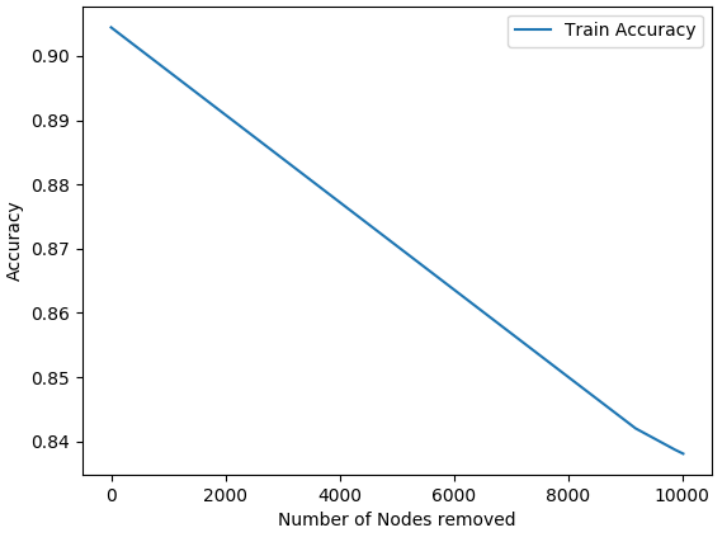


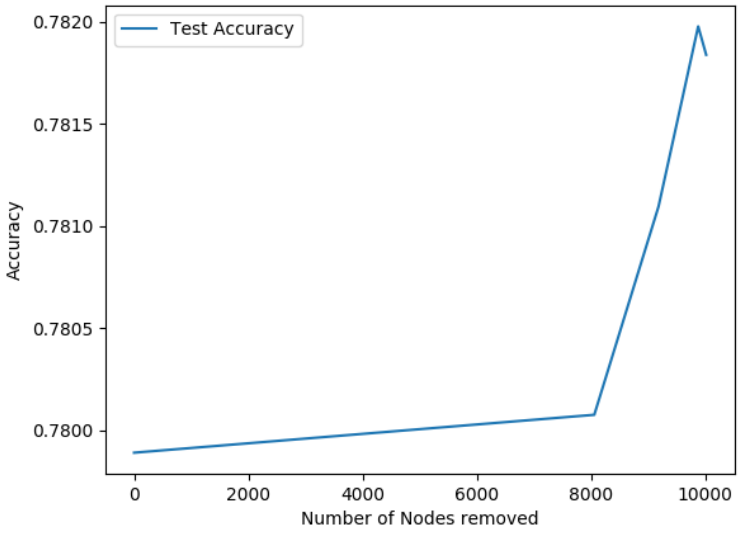
From these plots, we can see that the accuracies of training, test, and validation data is increasing monotonously in the initial region of the curve, but then the values for test and validation seems to get to a maximum and then their accuracies decrease slightly, while the accuracy for the training data still keeps on increasing and gets satured to a very high value of around 90%, while the other accuracies are like 77%. Thus, we can see that overfitting has happened while training the decision tree. Though, note that the overfitting is not too much here, as we can see that the slope for the test and validation data is very saturated after reaching their maxima.

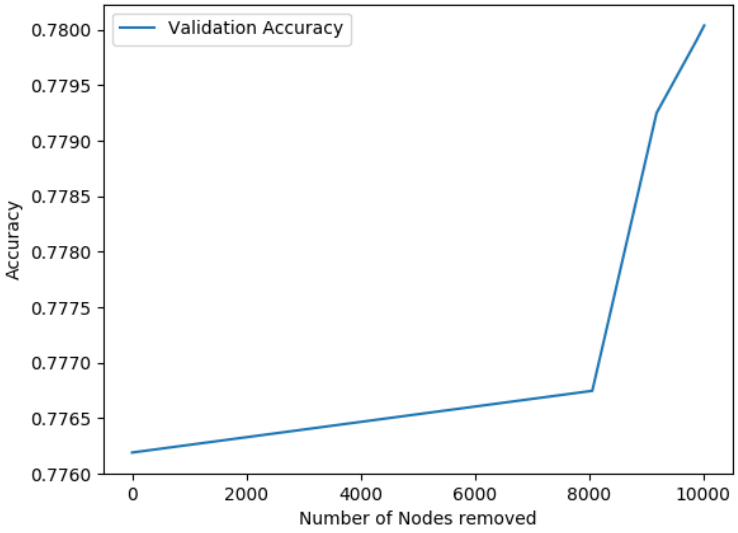
So, in order to prevent this, we would prune the tree iteratively in a greedy fashion in order to decrease the size of the tree and we would keep doing that till we can't prune any node (and correspondingly the subtree below it!). Here, I made the algorithm which would go from bottom to up, then check for the local accuracies of the validation dataset at a node and at it's children and then check if the accuracy at the node increase as compared to the accuracies at it's children and if yes, then we would prune that node and would return the resulting subtree. Here, for the iterative pruning, I would also check if that resulting tree performs good as compared to the previous tree or not. If it does, then it would be added to the list of the successive prunes and the previous tree would be updated, if not then it will be considered as a candidate for further pruning while the previous tree would be unchanged.

Note that since this algorithm goes from bottom to up and only deals with local accuracies, it works really fast, as compared to having run the entire validation set during each prune resulting in a lot of time and also having many prunes to check if the algorithm wasn't greedy!

The plots obtained after pruning for the training, test, and validation data:







Here, we can see that while pruning the nodes, the accuracy for the training dataset decreases linearly, from a high value of 90% to a relatively lower value of about 84%. The accuracies for the test and validation data increases slightly, and reach a value of 78%. Note that the entire procedure of constructing the tree and then pruning it, takes only about 10 mins!

2 Random Forest

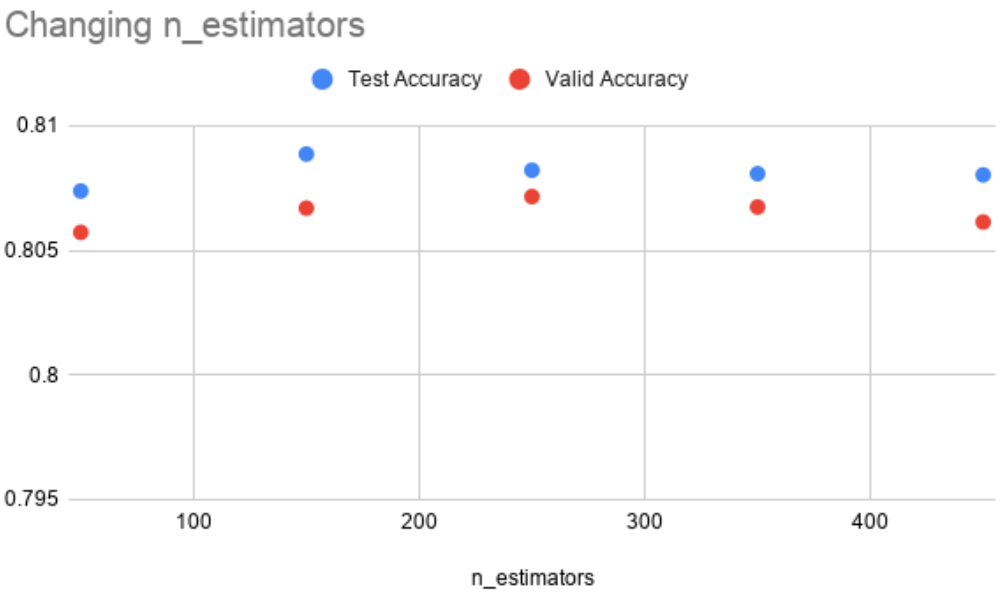
I used a self-made version of grid-search rather than using the sklearn implementation of it, because of the issues of the scoring function and the cross-validation involved. I parallelized the random forest to take upto 3 cores for their computations and my overall program took about 5 hours to complete. The best parameters obtained were **n_estimators = 450, max_features = 0.1, min_sample_split = 10**

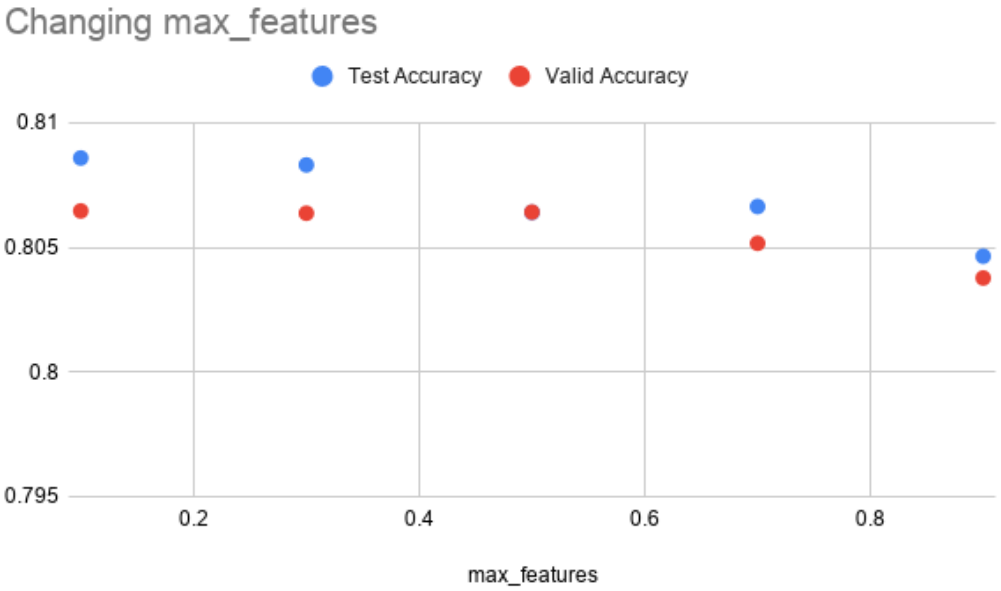
The accuracy values obtained are as below:

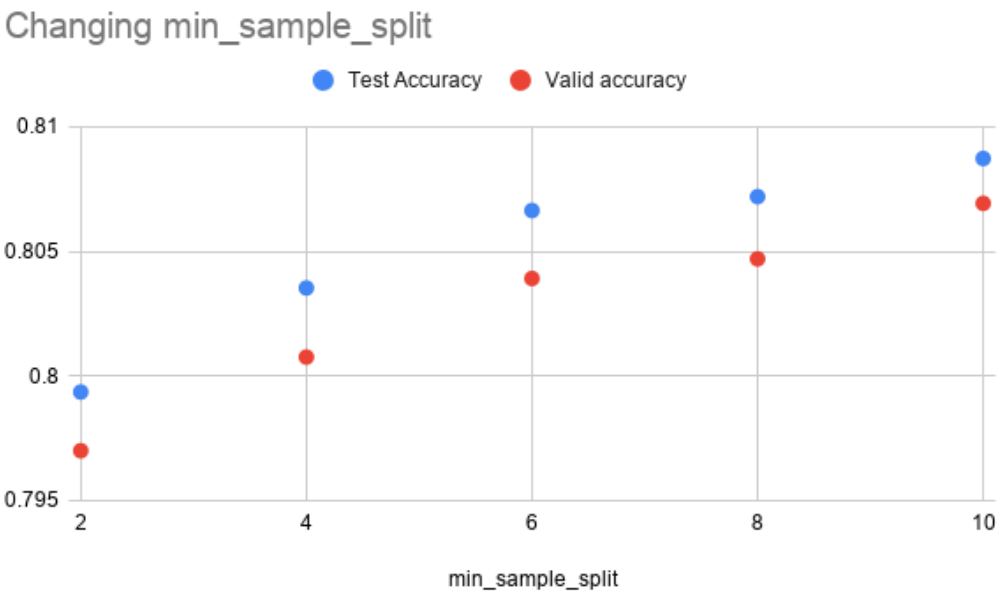
- Training accuracy: 0.8734566470415527
- Test accuracy: 0.8084928839645821
- Validation accuracy: 0.8068329315779714
- OOB accuracy: 0.810934433575943

So, here we can see that all these values are better than the ones which were obtained in the case of the pruned decision tree. So random forests work better!

Now, having obtained the best parameters, we now vary the parameter values one at a time and obtain the corresponding test and validation accuracies. The different scatter plots obtained corresponding to them are as follows:







All the above plots were made in the same scale for the y-axis in order to be easily able to compare between the sensitivities of the different parameters. As we can see from the plots, the values change the most corresponding to the changes in the values of **min_sample_split**, while the sensitivity is the least in the case of **n_estimators**