COL774 Assignment 3

Navneel Singhal

December 19, 2020

Contents

1 Decision Trees and Random Forests

1. In this part, I implemented a decision tree as mentioned in the specifications. There are the following points regarding the implementation:

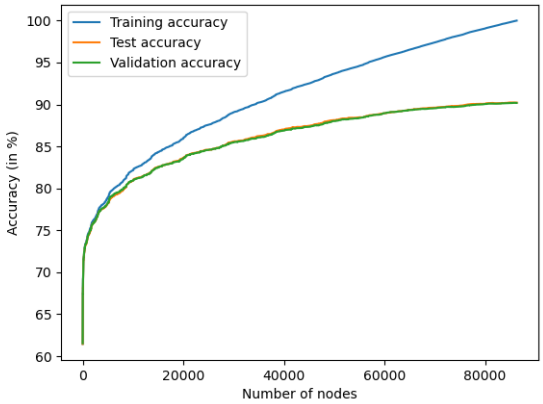
   (a) The tree is grown in BFS order, to keep it somewhat balanced so that there is no inherent bias towards one half of the range of values of an attribute (entropy measure and median splitting keep the graph quite balanced too, but in DFS, there is effectively no limit to the depth the left node will expand first, and hence might be slightly more imbalanced than BFS - while implementing DFS, I noticed that I needed to artificially bound the maximum depth of the tree, while BFS keeps the depth logarithmic in the number of nodes.)

   (b) Nodes also keep track of data that is pushed to them (all train, test and validation) if they are leaves.

   (c) Accuracy computation happens by keeping track of the total number of datapoints that are correctly classified; whenever a node is grown, we see how much the total number of correctly classified datapoints change, and this changes the accuracy accordingly.

   (d) Whenever we construct a node, it is automatically set to a leaf node. When expanding the node in the BFS order, we set it to an internal node and both of its children are made leaf nodes, so at any point, the decision tree has correctly marked leaf and internal nodes.

   (e) To keep track of pure nodes, we keep track of a class frequency for the training data using a dictionary. If the size of that dictionary is 1, that indicates that the node is pure.



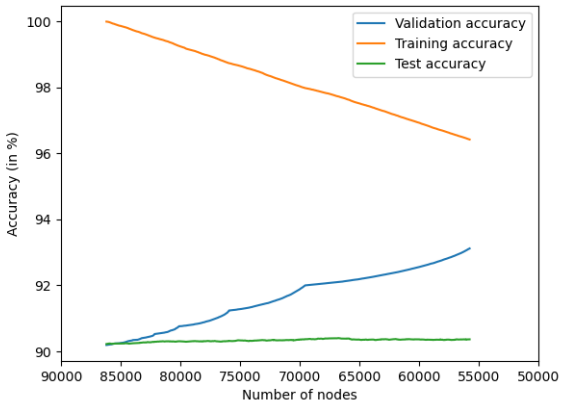
We see that the accuracy keeps on increasing as we increase the number of nodes for all three datasets, and 100% accuracy is achieved for the training set at about 86000 nodes. The training and the validation accuracy are more or less equal. This indicates that even though there is overfitting to the model, the validation/test accuracy still keeps on increasing by very small amounts (and not in proportion to the training accuracy) since the "noise" distribution (in training samples) that led to the overfitting in the first place is not totally independent of the noise distribution of the given validation and test samples. This is still noise since it doesn't generalize the data very well.

Data	Accuracy (without pruning)
Training	100.00%
Test	90.23%
Validation	90.18%

2. In this part, I implemented post-pruning as follows:

   (a) To keep track of accuracy (train, test and validation), we keep track of two values - number of currently correctly classified datapoints, and number of correctly classified datapoints if this node was a leaf node.

   (b) We maintain a heap of pairs (–increase in correctly classified datapoints, node) of non-leaf nodes. Then at each step, we remove the top node from the heap. If this node is already deleted, then we ignore this node. If the increase in frequency value is not the same as the value which can be found using variables set in the node, then this is an old instance and has been updated in the heap, and can be ignored. If this node has a non-negative increase, then we stop, since the pruning would no longer increase the validation accuracy. Now to actually prune this node, we would need to mark all the nodes in its subtree as deleted, mark this as a leaf, and update the correctly classified datapoints at this node, as well as all its ancestors. We would need to push its ancestors to the heap as well. It is simple to keep track of number of nodes as well as accuracies here.



Note that this graph has a reversed x-axis (so when we go from left to right, we go in the forward direction of pruning). We see that while the validation accuracy increases upon pruning, the training accuracy decreases. There is no significant change in the test accuracy, though. I believe that in case of a data set which has some noise and is not as clean as this dataset, pruning would have worked well, but in this data, since the test accuracy is not increasing, there seems to be some sort of "validation-overfitting" in the post-pruning, i.e., we remove nodes that are favourable to the validation set but don't matter to the test set, and hence instead of increasing the generalization power of the model, it seems to favourably tune the decision tree to the validation dataset.

Data	Accuracy (after pruning)
Training	96.42%
Test	90.37%
Validation	93.12%

3. In this part, I performed a grid search on the parameter space $(n\_estimators, max\_features, min\_samples\_split) \in \{50, 150, 250, 350, 450\} \times \{0.1, 0.3, 0.5, 0.5, 0.7\} \times \{2, 4, 6, 8, 10\}$. The optimal set of parameters is $(450, 0.7, 2)$.

Data	Accuracy
Training	100%
Test	96.40%
Validation	96.34%
Out of bag	96.35%

The results obtained using a random forest classifier are much better than those obtained in part b above. This is because of the following reasons:

(a) A larger number of classifiers makes the model more robust and reduces the variance in unstable models.

(b) Choosing a fraction of features (instead of all of them) for each classifier makes the model resistant to overfitting.

As for the minimum number of data points for a node to be split, a lower value makes the model less prone to underfitting, since we can fine-tune the leaf nodes by having a less amount of data at each leaf.

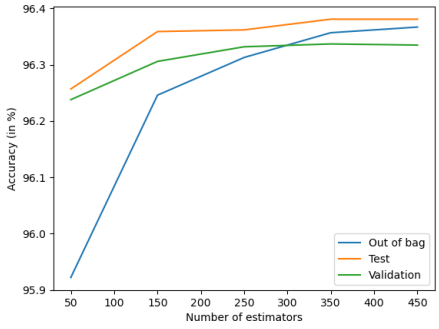
4. The accuracy variation is as follows:

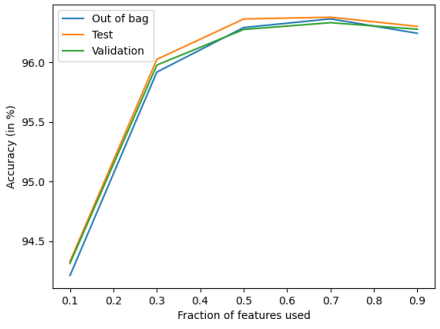
Estimators	Out of bag accuracy	Test accuracy	Validation accuracy
50	95.922%	96.257%	96.238%
150	96.246%	96.359%	96.306%
250	96.313%	96.363%	96.306%
350	96.357%	96.381%	96.337%
450	96.367%	96.381%	96.335%

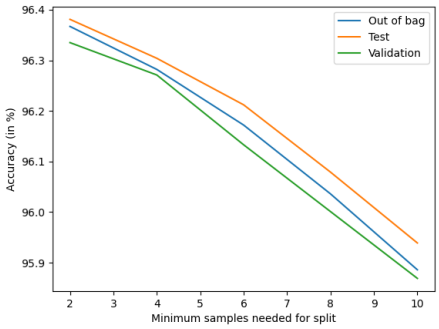
Fraction of features used	Out of bag accuracy	Test accuracy	Validation accuracy
0.1	94.212%	94.330%	94.314%
0.3	95.920%	96.027%	95.979%
0.5	96.294%	96.366%	96.278%
0.7	96.367%	96.381%	96.335%
0.9	96.246%	96.304%	96.280%

Minimum samples needed for split	Out of bag accuracy	Test accuracy	Validation accuracy
2	96.367%	96.381%	96.335%
4	96.282%	96.304%	96.271%
6	96.172%	96.212%	96.133%
8	96.036%	96.079%	96.001%
10	95.886%	95.939%	95.869%

The plots are as follows:







Firstly, we can observe that the more the number of estimators, the better the model (reason as above) as a general rule of thumb (and from the plot, after a certain point, the accuracy seems to saturate). As for the fraction of features used, if we use too less features or most of the features, the accuracy is not as

high, since in the former case, we need more estimators to correctly account for all possible sets of features (a fixed fraction of the original), and for the latter, having too many features brings overfitting into the picture. The minimum number of samples needed for split is mentioned

As far as sensitivity is concerned,

(a) The out of bag accuracy is quite sensitive to the number of estimators, and as the number of estimators increases, it seems to be a better representative of the test and validation accuracy. However, the test and validation accuracies are not as sensitive to the number of estimators.

(b) For the fraction of features used, the sensitivity is a bit more pronounced compared to the previous feature, and it is the most significant at lower fractions.

(c) For the minimum samples needed to split, the accuracy is almost linearly decreasing, and it is quite sensitive to this feature as compared to the previous features (towards the maxima).

2 Neural Networks

1. The fundamental equations are as mentioned in class. The neural network is fully characterized by the following variables:

   (a) θ matrices

   (b) Features and target classes

   (c) Architecture and activation function at each node

   (d) Stochastic gradient descent parameters

   Note that the formulae derived in class were for a single training example, and for each example, there is a δ and a unit output for each sample in mini-batch; as it turns out, this can be vectorized as well.
   One very non-trivial task was to initialize θ, and this was done using He initialization, where the θ for a layer is sampled from a distribution (chosen to be uniform in my implementation) whose variance is $\frac{2}{n}$ where n is the number of units in this layer. This works for both sigmoid as well as ReLU.

2. In this part, my chosen stopping criterion is the difference of the average error over the current epoch and the average error of the previous epoch is less than a given ε. The chosen value of ε is 10^{-4} for the sake of good convergence (which is also not that slow).

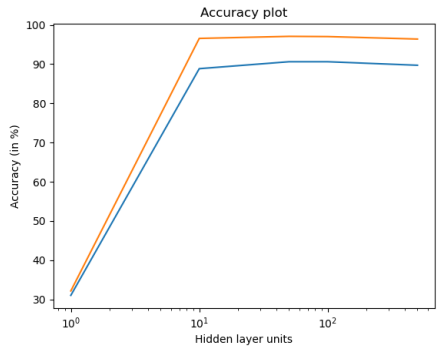
   The first table is for learning rate 0.1.

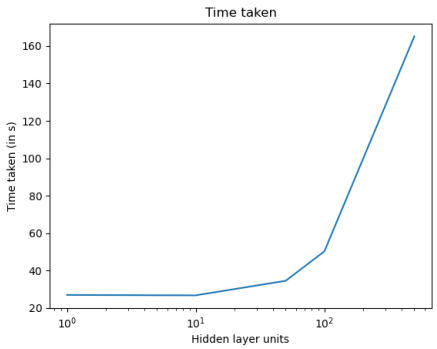
Hidden layer units	Test accuracy	Training accuracy	Time taken	Epochs
1	31.04%	32.16%	27.04s	127
10	88.84%	96.56%	26.80s	108
50	90.61%	97.08%	34.59s	103
100	90.61%	97.03%	50.31s	104
500	89.71%	96.39%	165.14s	79

   The second table is for learning rate 0.001.

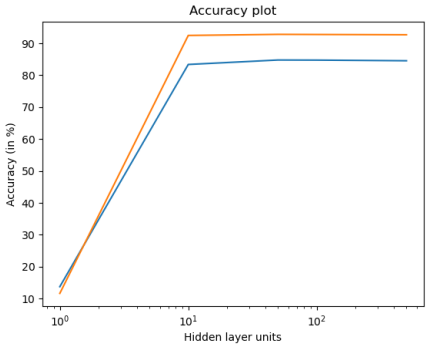
Hidden layer units	Test accuracy	Training accuracy	Time taken	Epochs
1	13.75%	11.66%	17.44s	102
10	83.38%	92.48%	205.93s	966
50	84.78%	92.82%	212.65s	618
100	84.75%	92.78%	311.73s	580
500	84.54%	92.69%	1216.10s	513

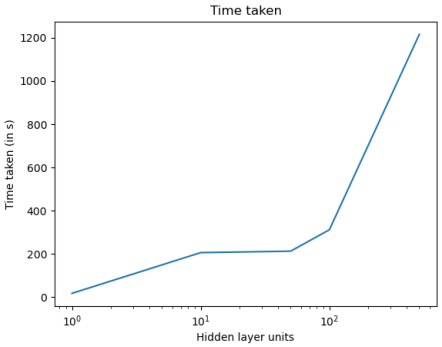
   The plots are as follows for learning rate 0.1:





The plots are as follows for learning rate 0.01:



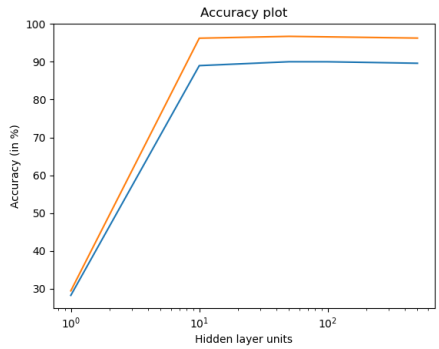


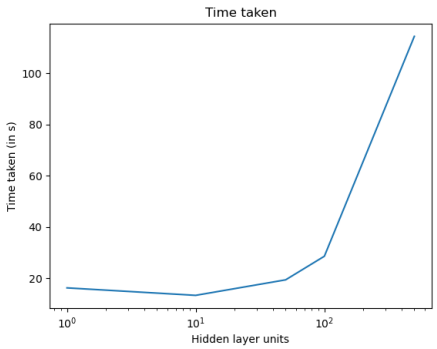
As can be seen from the plots, the training time increases quite quickly with the number of hidden layer units, however both the test and the training accuracies saturate after one point (in this case it is very slightly decreasing).

3. The stopping criterion didn't need to be changed and worked for both of these problems, and it is the same as the previous part.

Hidden layer units	Test accuracy	Training accuracy	Time taken	Epochs
1	28.26%	29.46%	16.25s	100
10	88.97%	96.22%	13.33s	69
50	90.00%	96.71%	19.39s	64
100	89.99%	96.57%	28.61s	63
500	89.60%	96.25%	114.46s	54

The plots are as follows:





The results are quite similar to the previous part, albeit with a very slight decrease in accuracy as compared to learning rate 0.1, but quite a significant increase as compared to the model with learning rate 0.001. The training time has indeed seen quite a good bit of reduction as well.

4. In this part, I experimented with a 2-hidden-layer neural network with 100 units each, with the hidden layers having sigmoid and ReLU activation.

Activation function	Test accuracy	Training accuracy	Time taken	Epochs
ReLU	93.21%	99.25%	25.85s	46
Sigmoid	90.4%	96.79%	41.17s	71

The test set accuracy when using ReLU activation is substantially better than the accuracy when using sigmoid activation, as can be seen in the table. As can be seen, ReLU performs better than sigmoid in both accuracy as well as time aspects. The lower time can be attributed to the fact that ReLU activation is faster to compute than sigmoid, as well as the lesser number of epochs needed for convergence.

For comparison with results using a single hidden layer with sigmoid, it can be seen that ReLU outperforms that, and sigmoid has almost the same accuracy for learning rate 0.1, and for learning rate 0.001, the results are much more significantly better.

5. In this part, I used the `sklearn` implementation of the neural network with a similar structure. The statistics are as follows.

Activation function	Test accuracy	Training accuracy	Time taken
ReLU	90.70%	97.95%	145.49s
Sigmoid	90.87%	96.79%	54.32s

As can be seen, my implementation performs better than the `sklearn` implementation when it comes to ReLU activation, and almost the same accuracy in the case of sigmoid. The time taken is also larger in the case of `sklearn` implementation, and I suspect that this is because of the initialization scheme (which I experimented with in part b as well).