COL774 – Machine Learning

Assignment – 3

Report

Submitted By:
Siddharth Gupta
Entry No. - 2021EE10627

Submitted To:
Parag Singla

Indian Institute of Technology, Delhi

Hauz Khas, New Delhi

Question – 1: Decision Trees

1. **Overview:**

   This question deals with decision trees and random forests. The task is that of binary classification. Decision trees are suitable models when the dataset is small. This is because the model is not a converted to an optimisation problem. Therefore, the risk of underfitting is eliminated. On the other hand, overfitting is a major concern in this model and different techniques like pre-pruning and post-pruning are utilized to avoid overfitting.

2. **Assignment Questions:**

   a) I implement the algorithm for decision tree in this part. Decision tree is constructed using a recursive algorithm.

   The metric to choose the best attribute to split is taken as mutual information. The best attribute is the one with maximum mutual information.

$$MI(Y, X_j) = H(Y) - H(Y|X_j)$$

$$H(Y|X_j) = \sum_{x} P(X_j = x)\, H(Y|X_j = x)$$

   where,

   $Y - output\ values$

   $X_j - attribute$

   There are two hyperparameters in the model-

   1. Encoding of the input data – ordinal or one-hot encoding
   2. Maximum Depth - $\{5, 10, 15, 20, 25\}$

   We observe that a node can be split multiple times on a continuous-valued attribute but only once on a discrete-valued attribute.

   We pre-process the input data using ordinal encoding in this part.

   The accuracies for various maximum depth values are –

Depth	**Train Accuracy**	**Test Accuracy**
5	88.55 %	57.60 %
10	99.65 %	59.87 %
15	99.71 %	59.87 %
20	99.71 %	59.87 %
25	99.71 %	59.87 %

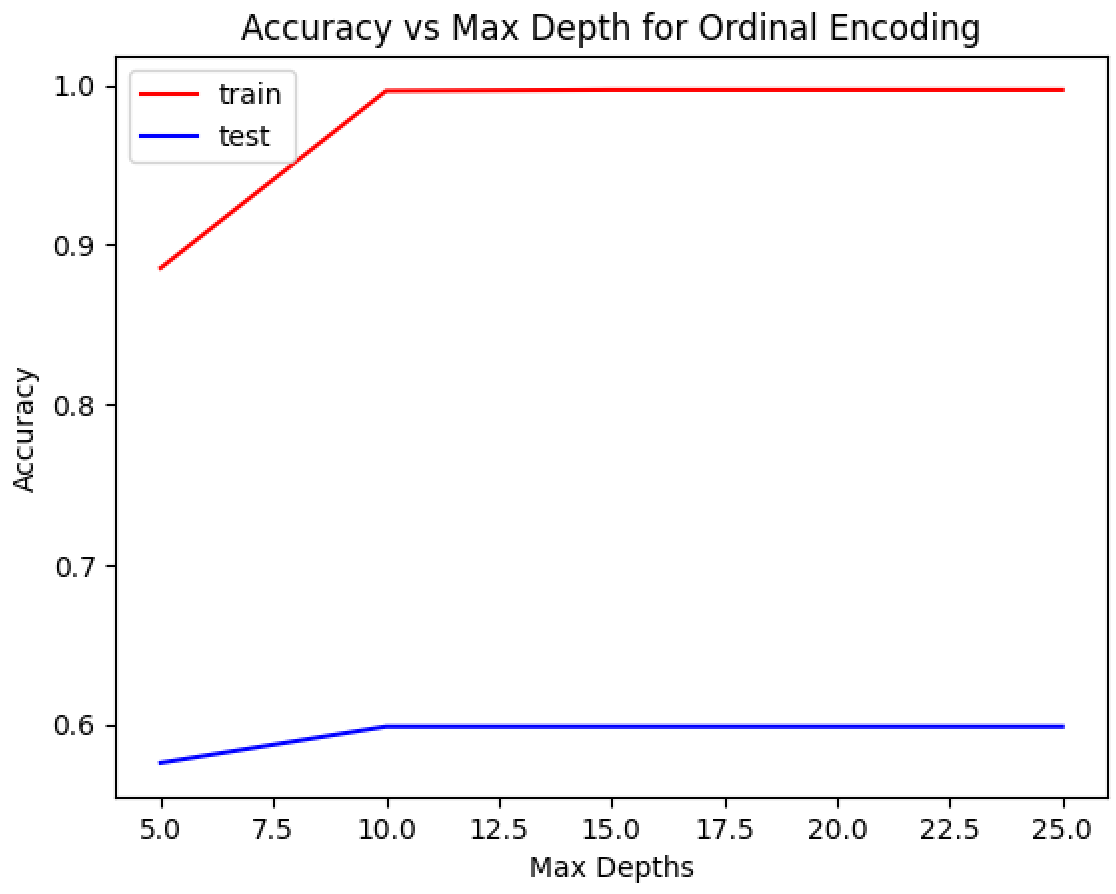
We consider two baseline models – only win predictions and only lose predictions. The test accuracies for these models are –

Model	**Test Accuracy**
Only Win Predictions	$49.63\ \%$
Only Lose Predictions	$50.36\ \%$

We observe that –

1. At $depth = 5$, both train and test accuracies are minimum. This is because at such small depth, the decision tree underfits the data.
2. On increasing $depth$, the accuracies on train and test set increase. Therefore, we can conclude that at these depths, decision tree no longer underfits the data.
3. We also see that at $depth = 15, 20, 25$, the training and test accuracies do not change. This is because the tree saturates at a lower depth and the rest of the tree is redundant for the predictions.
4. We verify that decision tree is a much better model than only win and only loss models. This is to be expected since both the baseline models are extremely naïve and are implemented to check the distribution of positives and negatives in the data.

We also plot the accuracy against the maximum depth for better visualization –



b) We now experiment with one-hot encoding. One-hot encoding splits each categorical attribute into k new features, where k is the number of unique values the attribute can take.
This way each categorical attribute of the encoded data becomes binary valued.
We observe that the models take longer to train with one-hot encoding. This is to be expected since the number of features has increased drastically with one-hot encoding and therefore the tree has the capacity to grow to much more depth.
The accuracies with varying maximum depths are –

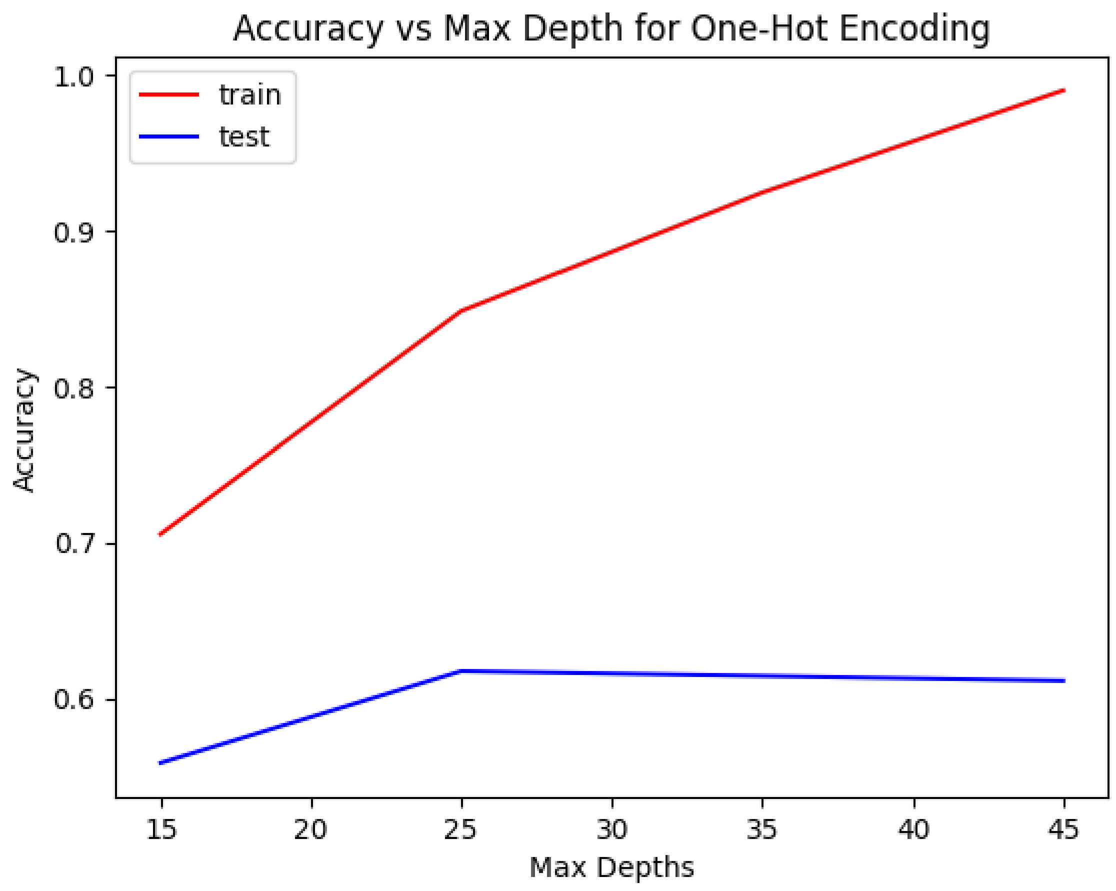
Depth	**Train Accuracy**	**Test Accuracy**
15	70.53 %	55.84 %
25	84.83 %	61.73 %
35	92.44 %	61.42 %
45	99.00 %	61.11 %

We observe that –

1. On increasing the maximum depth of the decision trees, the train accuracy increases. This is because a bigger tree allows the training data to be separated into a greater number of pure data subsets.
2. From $depth = 15$ to $depth = 25$, the train and test accuracies increase. This implies that at $depth = 15$, the model underfits the data.
3. However, on increasing the depth further, the test accuracy starts to fall. This implies that on increasing $depth$ beyond 25, the model overfits the training data.

Therefore, we conclude that for the given datasets, $depth = 25$ is the optimal value of max_depth for the decision tree.

We also plot the accuracy against the maximum depth for better visualization –



Comparing with the accuracies obtained with ordinal-encoded data (for $depths = 15, 25$), we observe that –

Depths	**Ordinal Encoding**		**One-Hot Encoding**	
	Train	**Test**	**Train**	**Test**
15	99.71 %	59.87 %	70.53 %	55.84 %
25	99.71 %	59.87 %	84.83 %	61.73 %

1. For $depth = 15$, both train and test accuracies are lower in the case of one-hot encoding. This means that at $depth = 15$, ordinal encoding better fits the data.
2. For $depth = 25$, the train accuracy for one hot encoding is lower but the test accuracy is higher. This implies that at this depth, one-hot encoding is a better choice because when compared, ordinal encoding leads to overfitting of the model to the train data.

c) In the above two parts, we saw that the decision trees construct grossly over-fit to the training data. This is a major concern in decision tree models. One technique to avoid this is post-pruning. In this, we iteratively look for the node in the tree, which on converting to a leaf node, results in maximum increase of validation accuracy.

Therefore, in each iteration, we prune one node of the tree. We do this till the validation accuracy can be increased by pruning the nodes.

The initial and final number of nodes in the decision tree are –

Initial Max Depth	**Initial Nodes**	**Final Nodes**
15	1573	601
25	3469	1075
35	4597	1469
45	5763	2075

We can see that the number of nodes decrease drastically after post-pruning. This helps in saving memory and prediction time.

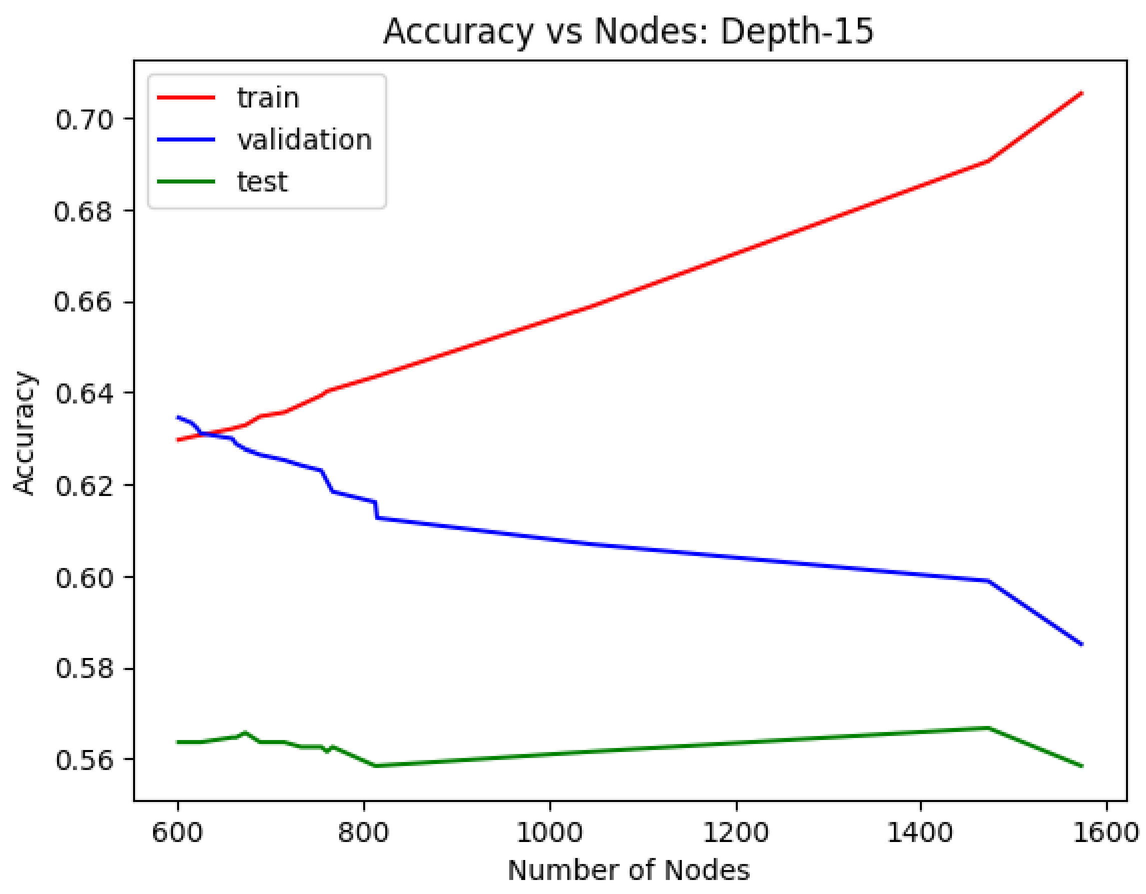
The accuracies of the final pruned tree on train, validation and test sets are –

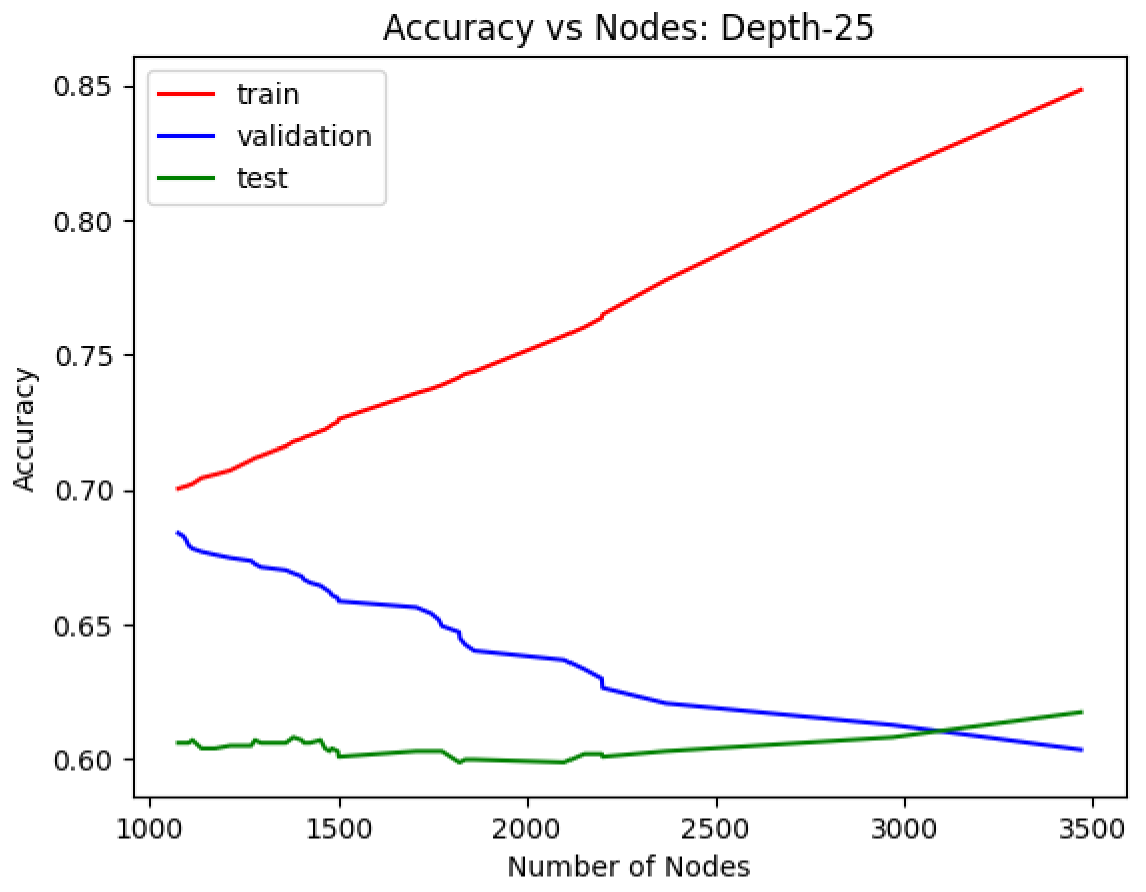
Depth	**Train Accuracy**	**Validation Accuracy**	**Test Accuracy**
15	62.96 %	63.44 %	56.35 %
25	70.03 %	68.39 %	60.59 %
35	72.69 %	70.68 %	60.59 %
45	76.92 %	73.44 %	61.11 %

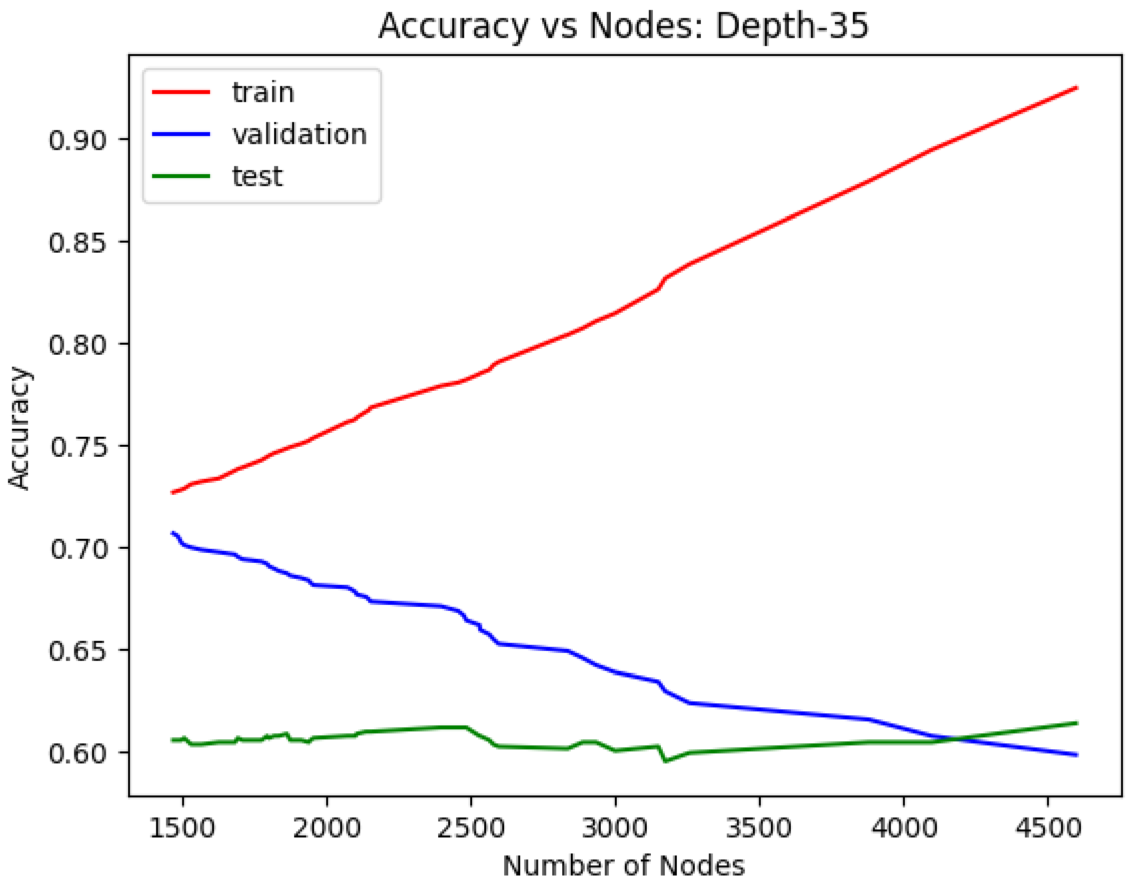
We observe that –

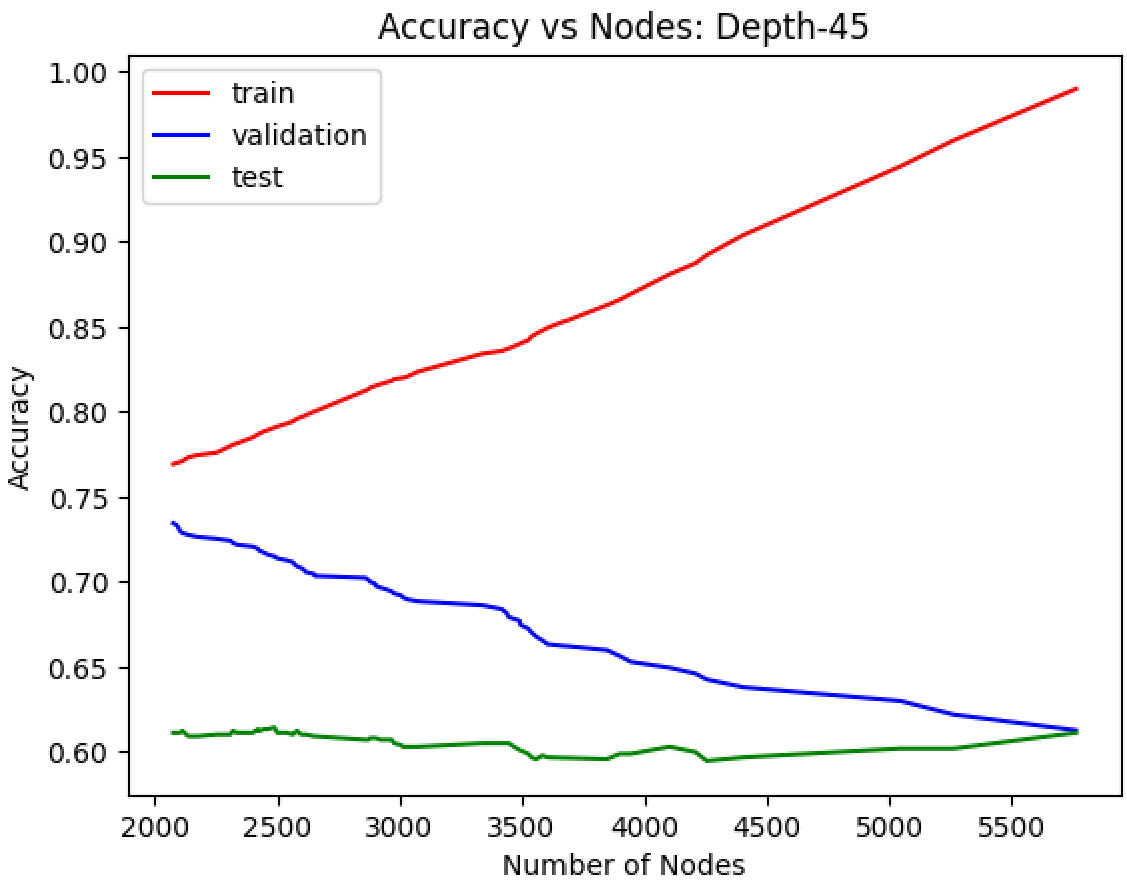
1. We observe that the test accuracies for the final pruned tree are slightly less than the test accuracies without pruning. This is because the tree is now slightly overfitting to the validation set. To avoid this, we can try to set a limit on the number of nodes that can be pruned.
2. In the last part, we saw that the model corresponding to $max\_depth = 25$ was the best model. However, now the decision tree corresponding to $original\_max\_depth = 45$ is now the best model. This is because with this depth the train was allowed to grow the deepest and the issue of overfitting was taken care by pruning.

We also plot the accuracies against the number of nodes in the pruning process –






Accuracy vs Nodes: Depth-35
train
validation
test
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
1500
2000
2500
3000
3500
4000
4500
Number of Nodes


Accuracy vs Nodes: Depth-45
train
validation
test
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
2000
2500
3000
3500
4000
4500
5000
5500
Number of Nodes

We observe that –

1. Validation accuracy increases continuously with decreasing number of nodes. This was the condition to continue pruning.
2. Training accuracy decreases continuously with decreasing number of nodes. This is because we are reducing the number of leaf nodes (nodes with pure data).
3. Test accuracy does not show a continuous trend. Overall, it is decreasing in every case (except the first). This is because we are overfitting on validation data now.

d) We now implement the decision tree model using $scikit - learn$ library. The aim is to compare our implementation with the $scikit - learn$ implementation.
We also experiment with different values of hyperparameters, namely, max_depth and ccp_alpha. ccp_alpha parameter is used to decide the extent of post-pruning of the decision tree.

The accuracies for train, validation, and test data for different max_depth and default ccp_alpha are –

max_depth	Train Accuracy	Validation Accuracy	Test Accuracy
15	71.31 %	58.50 %	60.59 %
25	85.46 %	60.57 %	62.97 %
35	94.44 %	61.49 %	66.08 %
45	99.54 %	62.18 %	63.80 %

We observe that –

1. On increasing max_depth, train and validation accuracies increase. Test accuracy also increases initially but decreases when $max\_depth = 45$.

Therefore, optimal value of max_depth (all other parameters are default) is 45.

The accuracies for train, validation, and test data for different ccp_alpha and default max_depth are –

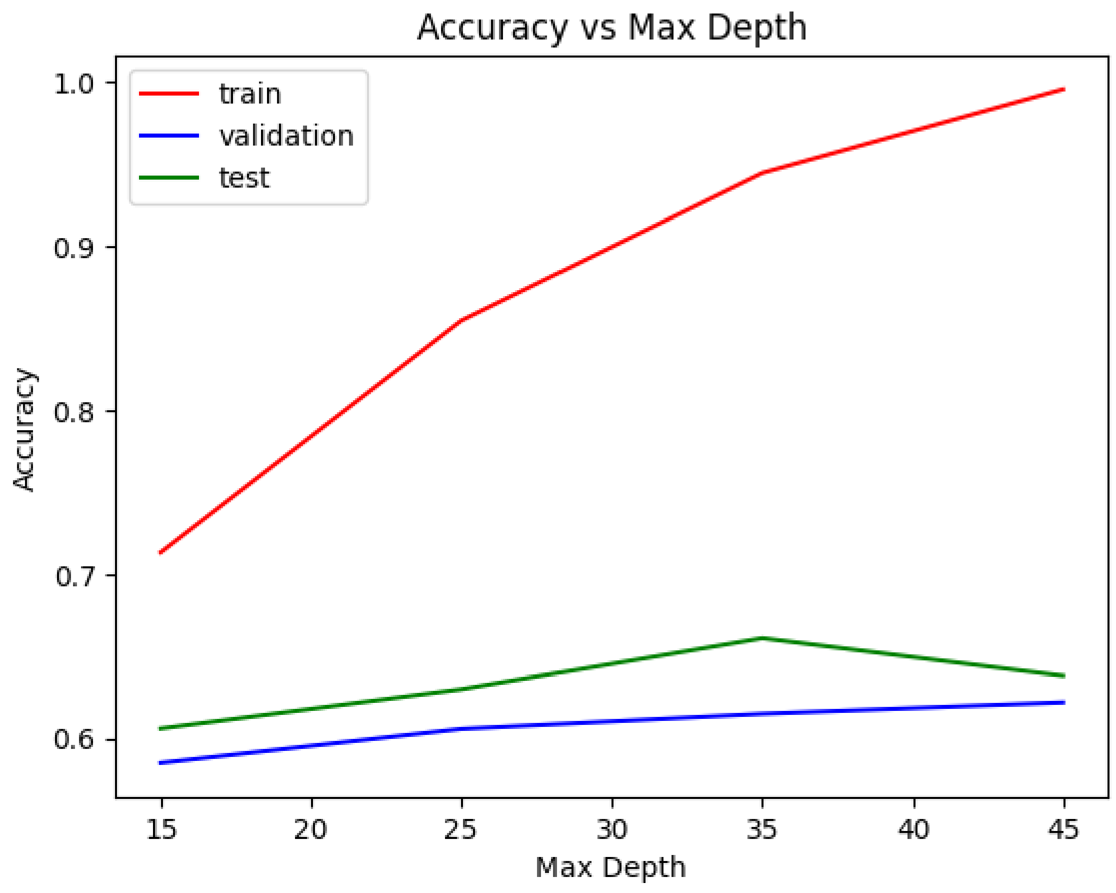
ccp_alpha	Train Accuracy	Validation Accuracy	Test Accuracy
0.001	68.94 %	63.21 %	66.28 %
0.01	53.44 %	50.00 %	51.80 %
0.1	50.33 %	47.35 %	49.63 %
0.2	50.33 %	47.35 %	49.63 %

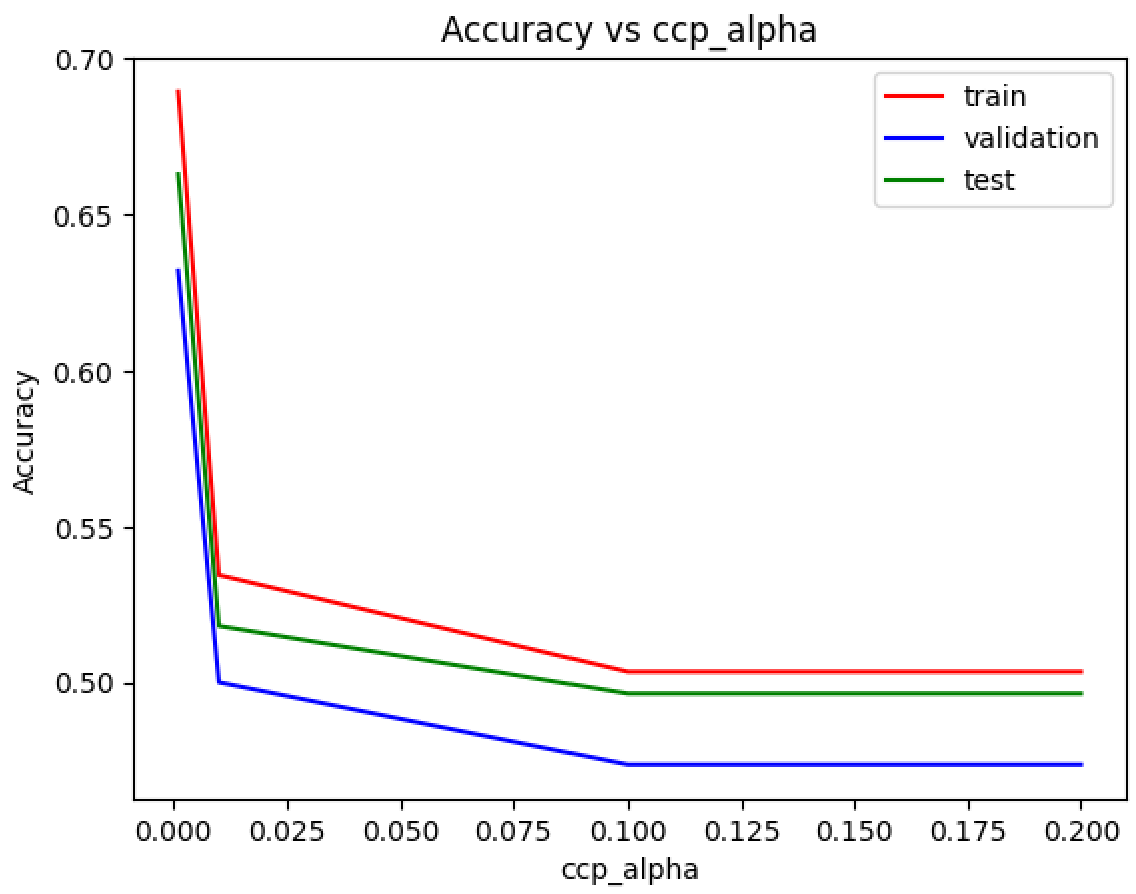
We observe that –

1. On increasing ccp_alpha, train, validation and test accuracies decrease. This is because at too large values, the decision tree is pruned to the extent that it underfits the data.
2. At all the values except the first, the test accuracy is almost equal to the accuracy obtained with only lose and only win models. This means that at such high values, the tree is almost pruned to the root.

Therefore, optimal value of ccp_alpha (all other parameters are default) is 0.001.

We plot the above data for better visualisation –





With the optimal parameters found above, we implement another model. The accuracies of this model on train, validation and test sets are –

Model	**Train Accuracy**	**Test Accuracy**
Sci-Kit Learn	68.94 %	66.28 %
Custom *(With Pruning)*	76.92 %	61.11 %
Custom *(Without Pruning)*	84.83 %	61.73 %

Comparing the accuracies in the above table, we observe that –

1. $scikit - learn$'s implementation gives the best test accuracy. The training accuracy is slightly low because this implementation does not overfit the train data.
2. Custom implementation (without pruning) overfits the training data the most.
3. Custom implementation (with pruning) is slightly worse than the one without pruning because the former overfits the validation set. This problem is not encountered in $scikit - learn$'s implementation because it does not use a separate validation data for pruning.

e) We now implement random forests. Random forests involve training multiple decision trees on bootstrapped samples of the original training data. We use $scikit - learn$ library to implement this.

We experiment with different values of different hyperparameters. In particular, we vary three hyperparameters –

1. $n\_estimators$ – It specifies the number of decision trees to be trained in the random forest.
2. $max\_features$ – It specifies the number of attributes (chosen randomly) to compare for the best split at any node.
3. $min\_samples\_split$ – It specifies the minimum number of data points required to consider splitting a node.

We use the out-of-bag accuracy as the metric to compare the model's performance.

We manually perform grid search to compare all the possible models. We use the training set to tune these three hyperparameters. We do not need a validation set here because the OOB accuracy is equivalent to validation accuracy in its purpose.

After comparing the values, we get that the optimal parameters are –

$$n\_estimators = 350$$

$$max\_features = 0.9$$

$$min\_samples\_split = 10$$

The train, validation and test accuracies for the optimal parameters are –

Train Accuracy	**Validation Accuracy**	**Test Accuracy**	**OOB Accuracy**
98.03 %	70.00 %	80.91 %	72.64 %

We compare the above best accuracies with our custom implementation of decision tree and $scikit - learn$'s implementation of decision tree –

	Train Accuracy	**Validation Accuracy**	**Test Accuracy**
Random Forest $scikit - learn$	98.03 %	70.00 %	80.91 %
Decision Tree $scikit - learn$	68.94 %	63.21 %	66.28 %
Decision Tree $custom$	76.92 %	73.44 %	61.11 %

We observe that –

1. Random forest is the best model for the given data. The accuracy of this model far exceeds rest of the two models. This is because random forest greatly reduce overfitting and improves generalisation of the model.

Question – 2: Neural Networks

1. **Overview:**

   This question is about neural networks. Neural networks the one of the most popular machine learning models used today. The task is of multi-class classification. We classify images based on the number of distinct objects in the image. The number of objects can vary between 1 and 5.

   We use cross-entropy loss as the error function and $softmax$ activation function for the output layer.

   We experiment with different layer architectures and activation function for hidden layers.

2. **Assignment Questions:**

   a) We implement a class - $NeuralNetwork$. This is the class to instantiate if we want to build a neural network model.

      We use mini-batch stochastic gradient descent to optimize the loss function.

      There are a number of hyperparameters in the neural network model –

      1. Layer Architecture – The number of layers and the number of perceptrons in each layer can be varied according to the need.
      2. Learning Rate – The learning rate used in gradient descent to update each of the weights and biases.
      3. Batch Size – The size of the mini-batch.
      4. Activation Function – We experiment with $sigmoid$ and $ReLU$ as activation functions for the hidden layers.
      5. Feature Size – The size of the input feature.
      6. Number of Classes – The number of unique target classes in the data.

      My implementation of $NeuralNetwork$ is generic to the above hyperparameters.

I initialise the weight matrix using Xavier Initialisation. This initialises an $(m \times n)$ matrix by sampling numbers from a Normal Distribution with mean 0 and standard deviation given by –

$$mean = 0$$
$$std\_dev = \sqrt{\frac{2}{m+n}}$$

This initialisation is chosen after experimenting with multiple initialisation schemes – sampling from standard normal distribution, zero initialisation, Xavier Initialisation.

Sampling from standard normal and zero initialisations result in the problem of vanishing gradients after a very few numbers of epochs.

Therefore, we chose Xavier Initialisation.

For the bias terms, I experimented with all-ones, all-zeros and Xavier Initialisation. The best results were given by all-zeros initialisation.

Therefore, the all the bias terms are initialised to 0.

b) We fix the number of hidden layers to 1. We experiment with different number of perceptrons in the hidden layer.
We fix all the hyperparameters (except the number of perceptrons) as –

$$batch\_size = 32$$
$$\#\, classes = 5$$
$$\#\, hidden\_layers = 1$$
$$feature\_size = 1024$$

We needed to choose a suitable stopping criterion to terminate the gradient descent algorithm. The stopping criterion chosen is as follows –

- I compute the change in loss value after every iteration. I store the last 100 such values in a queue.
- At each iteration, I check if the difference between the average of last 50 and the 50 before that is less than a threshold ($\in$).

My stopping criteria is therefore –

$$difference\ in\ averages \leq (\in = 10^{-8})$$
$$or$$
$$\#\, epochs \geq 500$$

With this hyperparameters, we compute the precision, recall and F1 score on the test and train data.
The score come out to be –

Metrics for 1 Perceptron

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	84.45 %	87.80 %	94.52 %	94.32 %	89.20 %	90.94 %
Class-2	76.31 %	72.57 %	64.50 %	64.14 %	69.91 %	68.09 %
Class-3	62.88 %	61.74 %	51.63 %	46.23 %	56.70 %	52.57 %
Class-4	48.46 %	46.15 %	31.47 %	35.29 %	38.16 %	40.00 %
Class-5	60.34 %	60.27 %	92.77 %	92.51 %	73.12 %	72.99 %

Metrics for 5 Perceptrons

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	87.10 %	89.58 %	94.92 %	93.88 %	90.84 %	91.68 %
Class-2	76.41 %	74.30 %	71.08 %	67.17 %	73.65 %	70.55 %
Class-3	62.85 %	62.16 %	59.47 %	57.78 %	61.12 %	59.89 %
Class-4	56.29 %	51.36 %	46.06 %	50.26 %	50.67 %	50.81 %
Class-5	70.22 %	67.13 %	84.69 %	76.47 %	76.78 %	71.50 %

Metrics for 10 Perceptrons

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	89.57 %	92.57 %	95.48 %	92.57 %	92.43 %	92.57 %
Class-2	77.23 %	74.86 %	73.40 %	72.22 %	75.27 %	73.52 %
Class-3	63.08 %	64.02 %	63.21 %	60.80 %	63.15 %	62.37 %
Class-4	59.56 %	55.42 %	45.11 %	51.87 %	51.34 %	53.59 %
Class-5	70.96 %	68.05 %	86.27 %	78.60 %	77.87 %	72.95 %

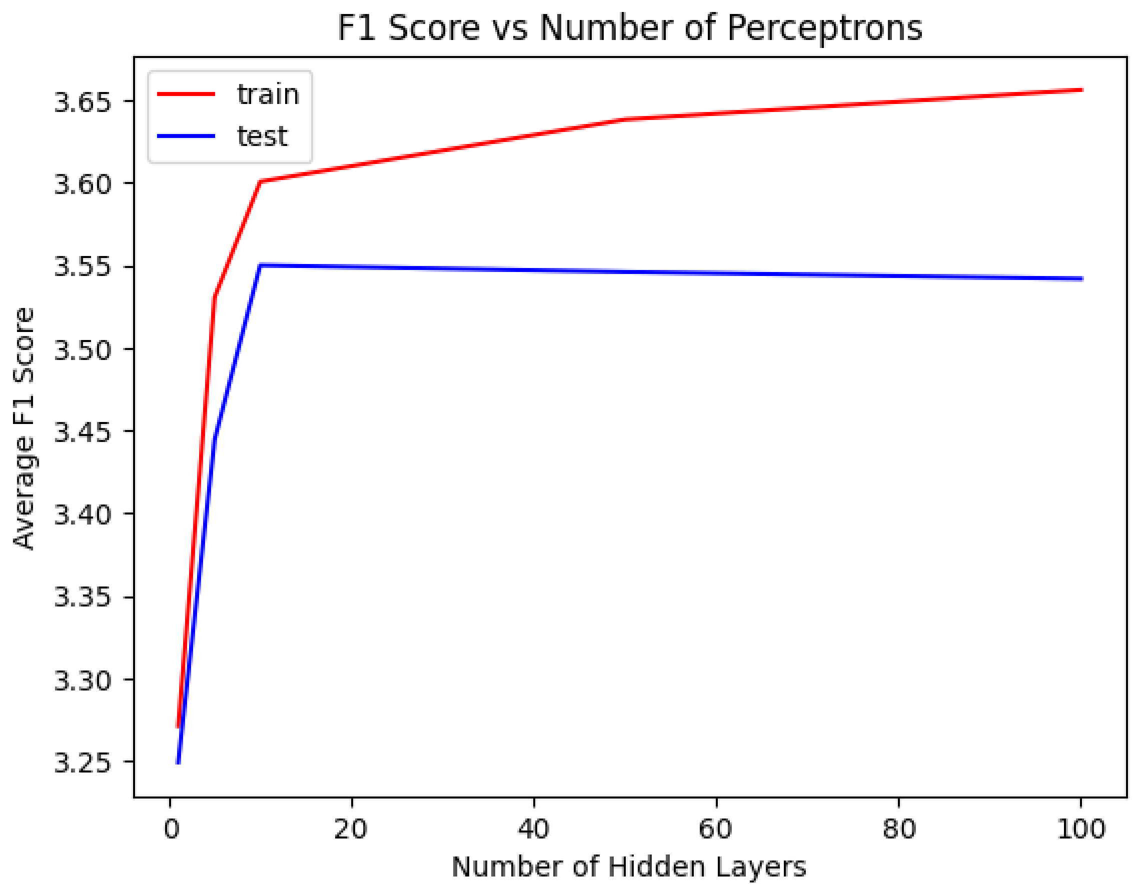
Metrics for 50 Perceptrons

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	90.10 %	93.47 %	93.75 %	93.88 %	91.89 %	93.68 %
Class-2	75.87 %	74.87 %	75.37 %	73.73 %	75.62 %	74.30 %
Class-3	64.64 %	62.28 %	61.16 %	54.77 %	62.85 %	58.28 %
Class-4	60.16 %	53.88 %	50.54 %	55.61 %	54.93 %	54.73 %
Class-5	72.77 %	70.04 %	85.27 %	77.54 %	78.52 %	73.60 %

Metrics for 100 Perceptrons

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	90.74 %	94.64 %	93.50 %	92.57 %	92.10 %	93.59 %
Class-2	76.10 %	74.87 %	76.33 %	75.25 %	76.22 %	75.06 %
Class-3	64.93 %	63.27 %	60.91 %	56.28 %	62.86 %	59.57 %
Class-4	59.77 %	52.79 %	52.39 %	55.61 %	55.83 %	54.16 %
Class-5	73.64 %	68.96 %	84.31 %	74.86 %	78.61 %	71.79 %

We also plot the average F1 score against the number of perceptrons in the hidden layers –



We observe that –

1. Train accuracy increases with increasing number of perceptrons. This is because more perceptrons results in a greater number of parameters to be optimized which means that the model will fit the training data more.
2. The best test accuracy is for the model with 10 perceptrons in the hidden layer. This can be explained as – since we have only one hidden layer, too many perceptrons in the layer can cause the neural network to overfit the training data. This is verified since the training accuracy keeps on increasing with increasing number of perceptrons in the layer.

c) We now vary the number of hidden layers and number of perceptrons in each layer. The batch size, learning rate and stopping criterion are kept the same as in previous part.
With this approach, the metrics are –

Metrics for {512}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	89.65 %	92.98 %	92.33 %	92.57 %	90.97 %	92.77 %
Class-2	73.15 %	71.64 %	75.48 %	72.72 %	74.29 %	72.18 %
Class-3	63.35 %	62.13 %	55.89 %	52.76 %	59.39 %	57.06 %
Class-4	57.36 %	52.68 %	52.53 %	57.75 %	54.84 %	55.10 %
Class-5	73.09 %	69.54 %	82.78 %	73.26 %	77.64 %	71.35 %

Metrics for {512, 256}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	94.89 %	96.44 %	95.28 %	94.75 %	95.08 %	95.59 %
Class-2	82.07 %	79.89 %	81.69 %	80.30 %	81.88 %	80.10 %
lass-3	69.02 %	68.10 %	65.06 %	63.31 %	66.98 %	65.62 %
Class-4	60.17 %	58.33 %	54.93 %	59.89 %	57.43 %	59.10 %
Class-5	73.68 %	72.36 %	83.83 %	77.00 %	78.43 %	74.61 %

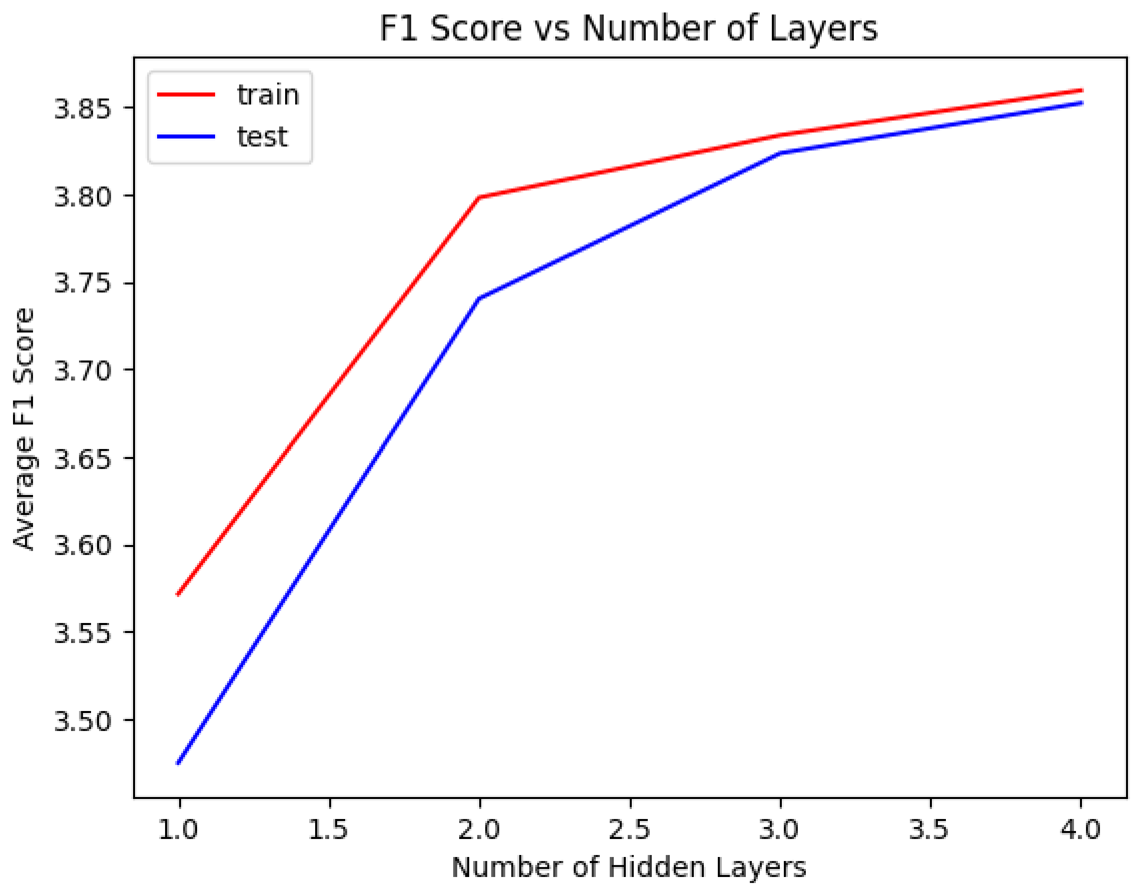
Metrics for {512, 256, 128}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	96.88 %	98.21 %	96.19 %	96.06 %	96.53 %	97.13 %
Class-2	84.53 %	83.16 %	84.02 %	82.32 %	84.27 %	82.74 %
Class-3	69.73 %	69.63 %	67.52 %	66.83 %	68.61 %	68.20 %
Class-4	59.37 %	60.10 %	53.13 %	58.82 %	56.08 %	59.45 %
Class-5	73.01 %	72.33 %	83.45 %	79.67 %	77.88 %	75.82 %

Metrics for {512, 256, 128, 64}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	96.34 %	97.83 %	97.61 %	98.68 %	96.97 %	98.26 %
Class-2	85.62 %	86.24 %	83.72 %	82.32 %	84.66 %	84.23 %
Class-3	69.20 %	69.15 %	69.41 %	69.84 %	69.30 %	69.50 %
Class-4	60.64 %	60.91 %	53.48 %	56.68 %	56.84 %	58.72 %
Class-5	74.01 %	72.19 %	82.83 %	79.14 %	78.17 %	75.51 %

We also plot the average F1 score against the number of perceptrons in the hidden layers –



We observe that –

1. With increasing number of hidden layers, both train and test scores increase. This is because a greater number of parameters are optimised to identify the trends in the data.

d) We now experiment with an adaptive learning rate. The learning rate of the model is varied as –

$$\eta = \frac{0.1}{\sqrt{epoch}}$$

<u>NOTE</u>: Initial learning rate given in the assignment is 0.01. However, with this value, the model ran into the problem of vanishing gradients for $\{512, 256, 128, 64\}$ architecture. Hence, $\eta_o = 0.1$ was chosen.

The above formula implies that the learning rate decreases with increasing number of epochs. This helps in converging to the minima faster because there is less risk of overshooting the minima or oscillating about it.

The time taken to train these models is slightly less than the time taken with constant learning rate. However, we do not see much improvement.

With this approach, the metrics for various architectures are –

Metrics for {512}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	90.32 %	93.08 %	87.11 %	88.20 %	88.68 %	90.58 %
Class-2	71.79 %	70.96 %	68.60 %	66.66 %	70.16 %	68.75 %
Class-3	57.85 %	58.09 %	64.34 %	61.30 %	60.92 %	59.65 %
Class-4	56.17 %	50.60 %	42.13 %	44.91 %	48.15 %	47.59 %
Class-5	69.51 %	66.06 %	84.17 %	78.07 %	76.14 %	71.56 %

Metrics for {512, 256}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	91.80 %	94.73 %	93.25 %	94.32 %	92.52 %	94.52 %
Class-2	77.13 %	77.95 %	75.73 %	73.23 %	76.42 %	75.52 %
Class-3	63.26 %	62.68 %	64.39 %	63.31 %	63.82 %	63.00 %
Class-4	58.07 %	55.78 %	50.29 %	56.68 %	53.90 %	56.23 %
Class-5	73.09 %	71.28 %	81.44 %	74.3 %	77.04 %	72.77 %

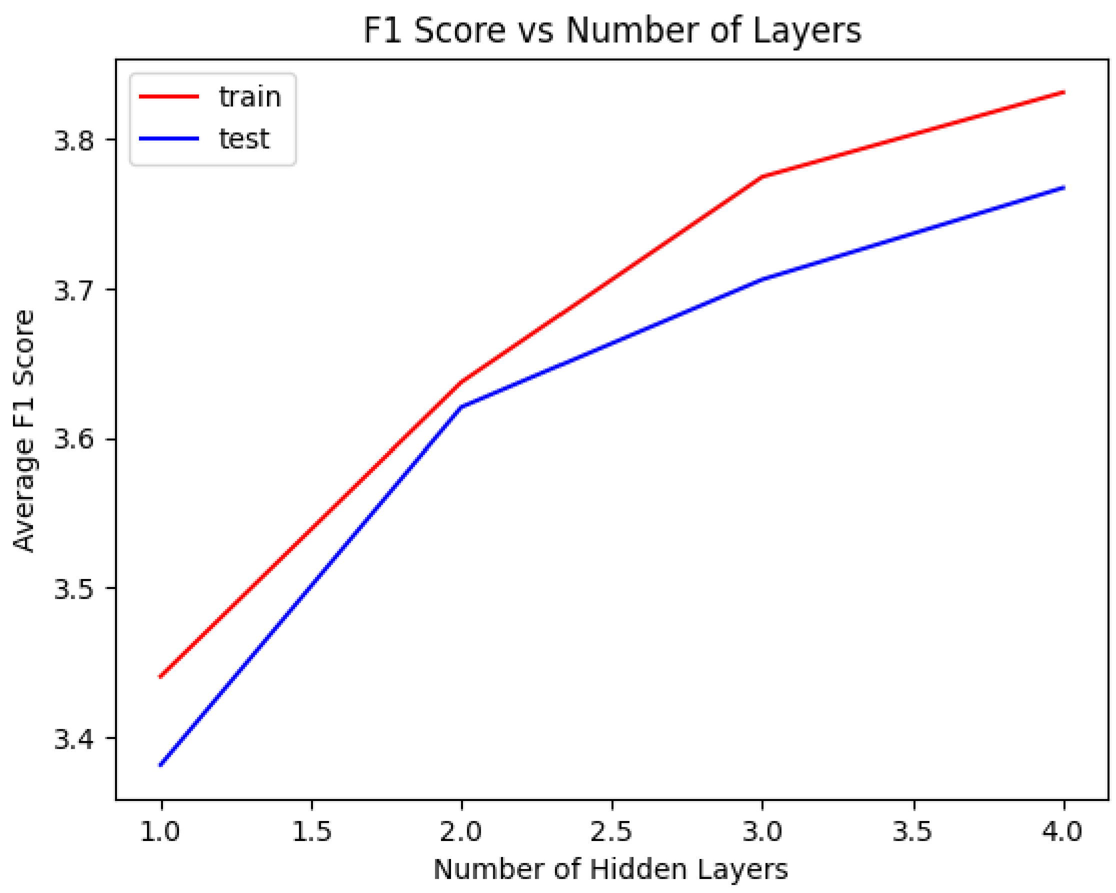
Metrics for {512, 256, 128}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	94.70 %	96.47 %	95.28 %	95.63 %	94.99 %	96.05 %
Class-2	82.56 %	81.18 %	79.97 %	76.26 %	81.25 %	78.64 %
Class-3	67.53 %	65.02 %	67.57 %	66.33 %	67.55 %	65.67 %
Class-4	59.48 %	57.86 %	52.49 %	55.08 %	55.76 %	56.43 %
Class-5	73.23 %	70.38 %	83.21 %	77.54 %	77.90 %	73.79 %

Metrics for {512, 256, 128, 64}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	95.82 %	97.35 %	96.65 %	96.50 %	96.23 %	96.92 %
Class-2	84.65 %	82.63 %	82.30 %	79.29 %	83.46 %	80.92 %
Class-3	68.97 %	66.83 %	68.34 %	65.82 %	68.65 %	66.32 %
Class-4	59.71 %	59.01 %	54.03 %	57.75 %	56.73 %	58.37 %
Class-5	73.90 %	71.07 %	82.63 %	77.54 %	78.03 %	74.16 %

We also plot the average F1 score against the number of perceptrons in the hidden layer –



We observe that –

1. Both train and test scores increase with increasing number of hidden layers.

Adaptive learning rate is a good approach in many scenarios. The reason it does not work so well here is that it needs a greater number of epochs to converge. However, since the code was taking too long to run, I did not increase it too much.

e) Till now we have been using $sigmoid$ as the activation function for the hidden layers. In this part, we experiment with another popular activation function, namely, ReLU. ReLU function is defined as –

$$ReLU(x) = \max(0, x)$$

The derivative of ReLU is computed using sub-gradients as –

$$\frac{d}{dx}\big(ReLU(x)\big) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

We again use an adaptive learning rate with the same variation as in the previous part.

With this approach, the metrics for various architectures are –

Metrics for {512}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	88.01 %	91.59 %	90.51 %	90.39 %	89.24 %	90.98 %
Class-2	70.56 %	69.19 %	72.24 %	69.19 %	71.39 %	69.19 %
Class-3	59.44 %	58.94 %	59.98 %	56.28 %	59.71 %	57.58 %
Class-4	57.07 %	54.08 %	48.20 %	56.68 %	52.26 %	55.35 %
Class-5	73.13 %	72.10 %	79.81 %	73.26 %	76.33 %	72.67 %

Metrics for {512, 256}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	94.25 %	96.92 %	96.54 %	96.50 %	95.38 %	96.71 %
Class-2	83.06 %	81.21 %	81.85 %	80.80 %	82.45 %	81.01 %
Class-3	69.01 %	66.66 %	70.18 %	67.33 %	69.59 %	67.00 %
Class-4	63.99 %	58.79 %	55.67 %	57.21 %	59.54 %	57.99 %
Class-5	76.43 %	74.47 %	84.07 %	76.47 %	80.07 %	75.46 %

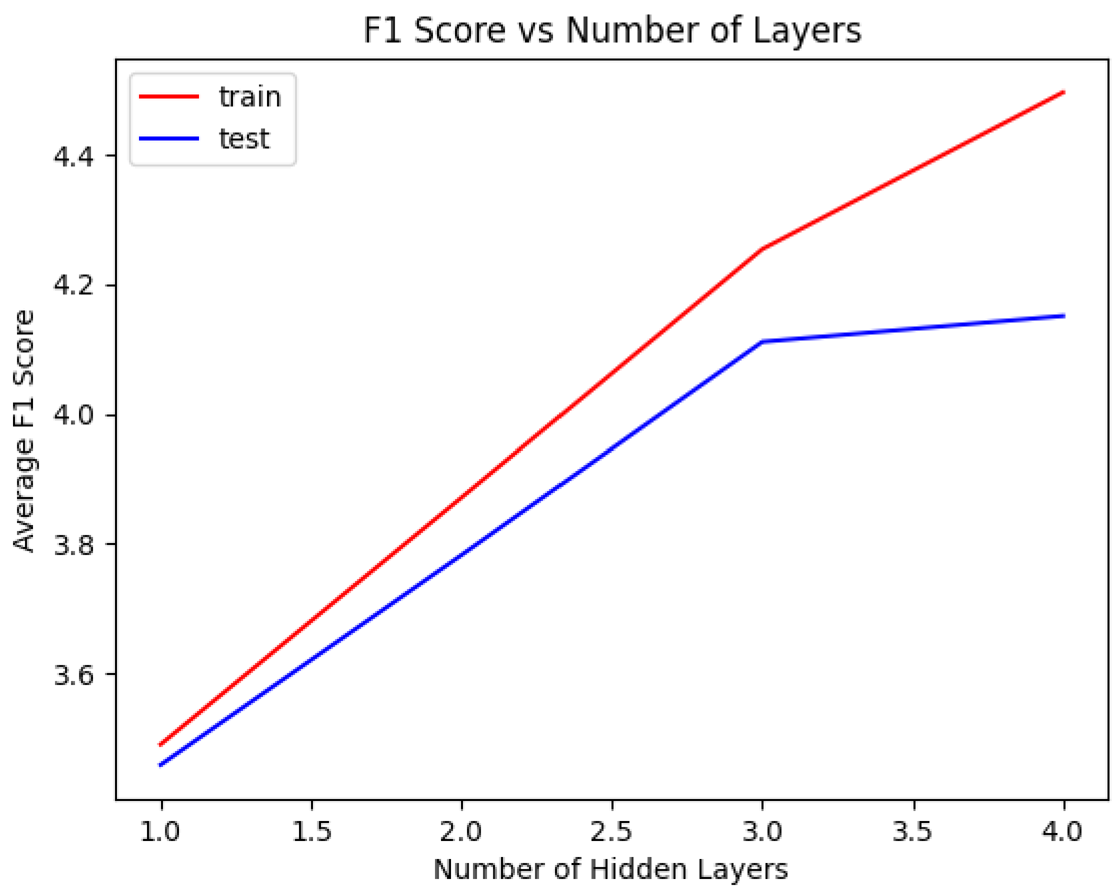
Metrics for {512, 256, 128}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	99.12 %	99.55 %	97.41 %	97.81 %	98.25 %	98.67 %
Class-2	93.54 %	88.60 %	89.38 %	86.36 %	91.41 %	87.46 %
Class-3	81.57 %	78.57 %	81.86 %	71.85 %	81.71 %	75.06 %
Class-4	73.69 %	66.00 %	66.13 %	70.58 %	69.71 %	68.21 %
Class-5	78.68 %	79.00 %	90.76 %	84.49 %	84.29 %	81.65 %

Metrics for {512, 256, 128, 64}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	99.13 %	99.56 %	99.39 %	99.12 %	99.26 %	99.34 %
Class-2	96.14 %	90.50 %	94.53 %	91.41 %	95.35 %	9.95 %
Class-3	87.54 %	81.11 %	88.57 %	73.36 %	88.05 %	77.04 %
Class-4	81.14 %	64.15 %	76.74 %	72.72 %	78.88 %	68.17 %
Class-5	85.80 %	81.11 %	90.48 %	78.07 %	88.08 %	79.56 %

We also plot the average F1 score against the number of perceptrons in the hidden layer –



We observe that –

1. This model performs much better than the earlier models. This confirms that ReLU is a better activation function than sigmoid function in most of the cases. This is because using sigmoid function leads to a greater risk of vanishing gradients problem since the derivative is always between 0 and 1.
2. While using ReLU function, we observe that both test and train accuracies increase with increasing number of hidden layers.

f) In this part, we compare our custom implementation of neural networks with $scikit - learn$'s implementation. We use $MLPClassifier$ for this purpose. We still use cross-entropy loss as the loss function and stochastic gradient descent to optimise the loss function.

The hyperparameters are –

$$batch\_size \ = \ 32$$
$$leanring\ rate : adaptive\ ('invscaling')$$
$$activation\ function : ReLU$$

With these parameters the metrics are –

Metrics for {512}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	69.05 %	72.43 %	91.93 %	89.51 %	78.86 %	80.07 %
Class-2	50.92 %	51.26 %	41.75 %	40.90 %	45.88 %	45.50 %
Class-3	44.34 %	43.47 %	31.91 %	30.15 %	37.11 %	35.60 %
Class-4	44.52 %	38.92 %	33.01 %	31.01 %	37.91 %	34.52 %
Class-5	59.96 %	51.51 %	82.01 %	80.74 %	69.27 %	65.79 %

Metrics for {512, 256}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	76.19 %	78.51 %	90.01 %	87.77 %	82.56 %	82.88 %
Class-2	57.01 %	54.14 %	51.97 %	49.49 %	54.37 %	51.71 %
Class-3	46.48 %	47.01 %	38.98 %	35.67 %	42.40 %	40.57 %
Class-4	45.68 %	41.71 %	37.94 %	39.03 %	41.45 %	40.33 %
Class-5	62.82 %	58.22 %	76.94 %	73.79 %	69.17 %	65.09 %

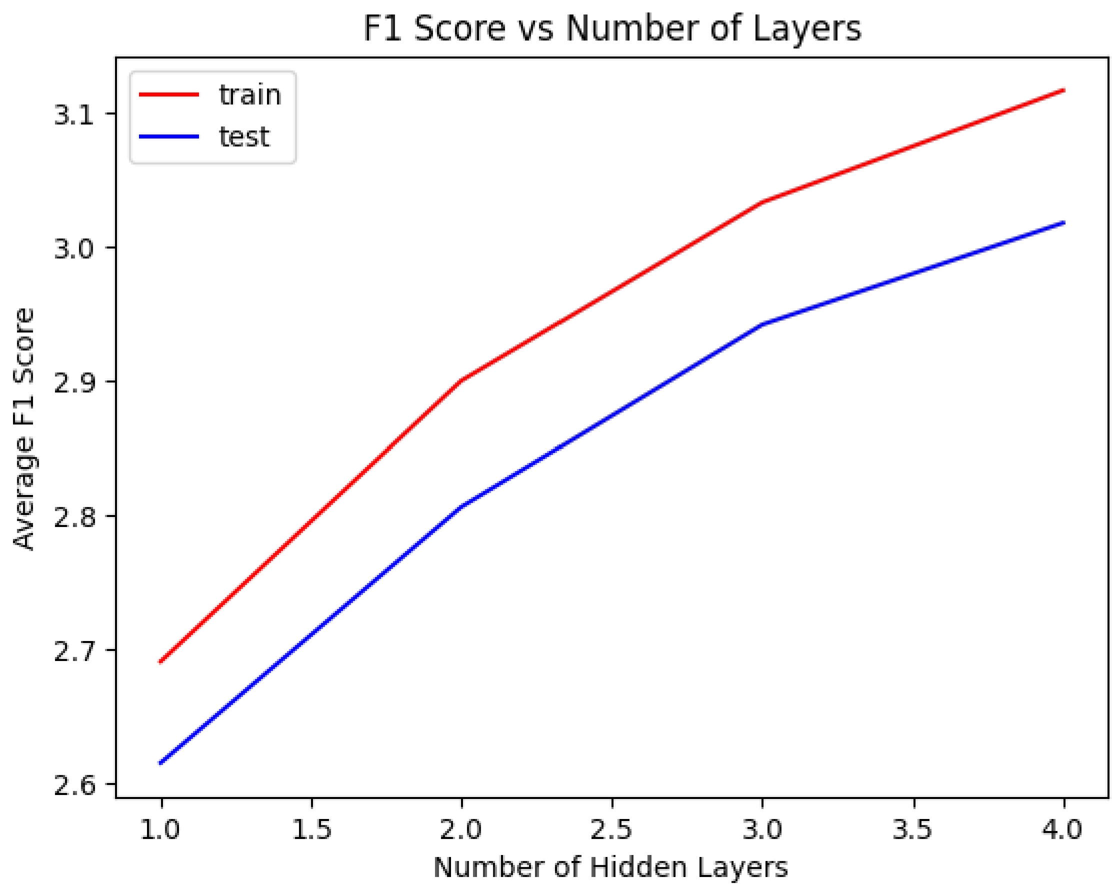
Metrics for {512, 256, 128}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	81.44 %	83.05 %	89.29 %	87.77 %	85.18 %	85.35 %
Class-2	60.88 %	58.42 %	59.95 %	56.06 %	60.41 %	57.21 %
Class-3	48.95 %	50.93 %	43.03 %	41.20 %	45.80 %	45.55 %
Class-4	45.24 %	40.68 %	41.43 %	44.38 %	43.25 %	42.45 %
Class-5	65.02 %	61.08 %	72.64 %	66.31 %	68.62 %	63.58 %

Metrics for {512, 256, 128, 64}

	Precision		**Recall**		**F1 Score**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Class-1	83.30 %	84.08 %	89.14 %	89.95 %	86.12 %	86.91 %
Class-2	63.12 %	60.86 %	63.95 %	56.56 %	63.53 %	58.63 %
Class-3	50.61 %	50.56 %	46.31 %	45.22 %	48.36 %	47.74 %
Class-4	45.89 %	42.23 %	43.47 %	46.52 %	44.65 %	44.27 %
Class-5	67.25 %	64.17 %	70.73 %	64.17 %	68.95 %	64.17 %

We also plot the average F1 score against the number of perceptrons in the hidden layer –



We observe that –

1. Both test and train scores increase with number of hidden layers.
2. $MLPClassifier$ does not perform as well as our custom implementation using ReLU function. However, it takes significantly less training time.

3. <u>Conclusion:</u>

In all the approaches, we see that model with a greater number of hidden layers perform better. This is because a greater number of hidden layers allows the model to better learn the complex trends in the data.

With this training data and architectures, we observe that vanishing gradients was a significant problem in training the models. We resolved this problem by changing the initialisation method of the weights and playing around with different learning rates. Though the implementation of the model was moderately difficult, tuning the hyperparameters was a much more demanding task.