1 Decision Trees

For a do `python3 Q1a.py`, For b do `python3 Q1b.py`, For c, d do `python3 Q1c.py`, For autograding do `./run_dt.sh <train_data> <validation_data> <test_data> <output_file>`

(a) Implemented a Decision Tree Classifier using Mutual Information as the criteria for selecting the attribute to split on. Splitting is done using the median of the data instance coming to that node.

$$\text{MI(Y, X)} = \text{H(Y)} - \text{H(Y | X)}$$

$$\text{H(Y)} = -\sum_{k=1}^{r} p_k log(p_k)$$

$$\text{H(Y | X)} = \sum_x P(X = x) H(Y|X = x)$$

GrowTree Algorithm:

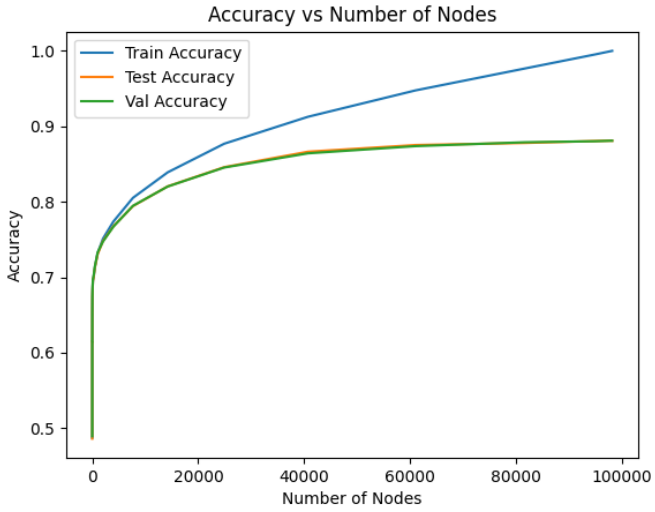
```
GrowTree(S, node):
 if y = y[0] for all < x, y > ∈ S:
  return new Leaf(y[0])
 else:
  j = BestAttribute(S)
  S0 = all < x, y > such that x < median(x[:, j])
  S1 = all < x, y > such that x ≥ median(x[:, j])
  node = Node(j)
  node.left = GrowTree(S0, node.left)
  node.right = GrowTree(S1, node.right)
```

Best Attribute Algorithm:

```
ChooseBestAttribute(x, y):
 MI = []
 H(Y) = −∑_1^r p_k log(p_k)
 for j in x.columns
  calculate P_x
  calculate H(Y | X)
  MI_col = H(Y) - H(Y | X)
  MI.append(MI_col)
 j = argmax(MI)
 return j
```

Observations:

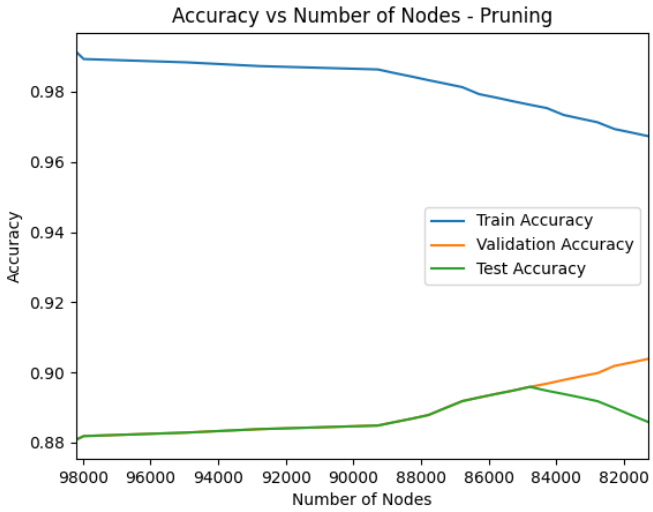
1. `Time To Grow Tree`: 174.002 seconds
2. `Number of Nodes`: 92108
3. `Depth of Tree`: 21
4. `Training Accuracy`: 100%
5. `Validation Accuracy`: 88.09%
6. `Test Accuracy`: 88.08%



Observations: We can observe that the training accuracy increases with the number of nodes. Also from the validation accuracy and test accuracy we can infer that the decision tree is overfitting the data.

(b) Post Pruning

1. `Number of Nodes in Pruned Tree`: 81275
2. `Training Accuracy`: 96.73%
3. `Validation Accuracy`: 90.18%
4. `Test Accuracy`: 87.58%



Conclusion: Post pruning reduces overfitting. The training accuracy decreases as nodes are pruned from the tree. We can observe that the as the validation accuracy increases the test accuracy also increases.

(c) Random Forest Classifier: `Best Set of Parameters` are n_estimators = 250, max_features = 0.7, min_samples_split = 2

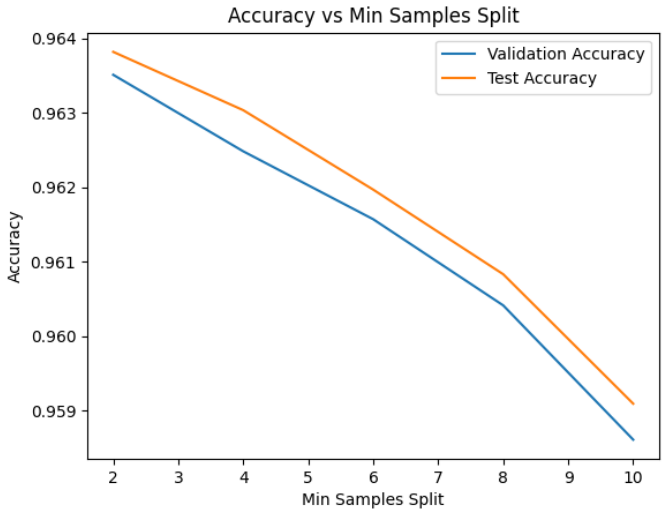
Observations:

1. Training Time: 998.67 seconds
2. Training Accuracy: 100%
3. Validation Accuracy: 96.349%
4. Test Accuracy: 96.345%
5. Out of Bag Accuracy: 96.3239%

(d) Parameter Sensitivity Analysis.

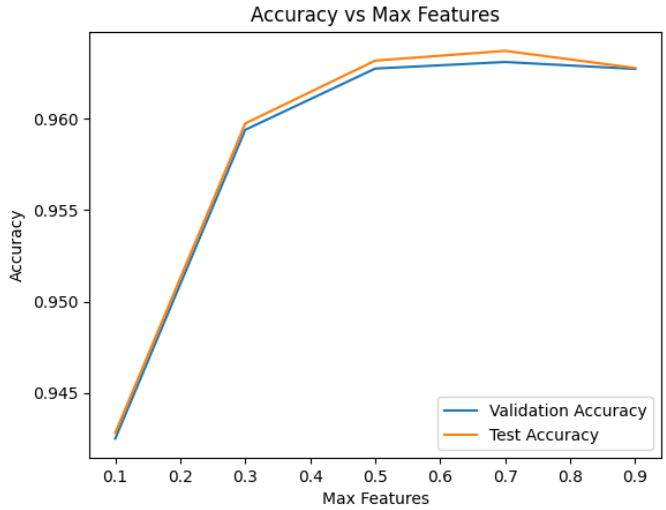
(a) n_estimators = 250, max_features = 0.7, min_samples_split = [2, 4, 6, 8, 10]

min_samples_split	Validation Accuracy	Test Accuracy
2	96.35	96.38
4	96.24%	96.30%
6	96.15%	96.19%
8	96.04%	96.08%
10	95.8%	95.90%



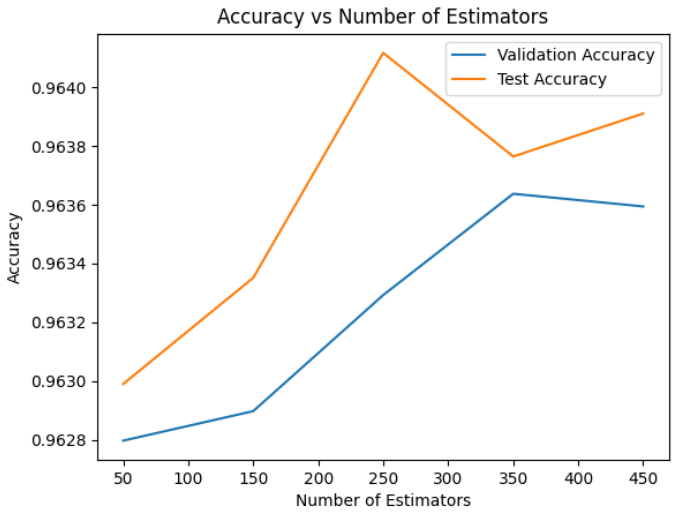
(b) n_estimators = 250, max_features = [0.1, 0.3, 0.5, 0.7, 0.9], min_samples_split = 2

max_features	Validation Accuracy	Test Accuracy
0.1	94.25	94.28
0.3	95.94%	95.97%
0.5	96.27%	96.31%
0.7	96.31%	96.37%
0.9	96.27%	96.28%



(c) n_estimators = [50, 150, 250, 350, 450], max_features = 0.7, min_samples_split = 2

n_estimators	**Validation Accuracy**	**Test Accuracy**
50	96.279	96.298
150	96.289%	96.335%
250	96.329%	96.412%
350	96.363%	96.376%
450	96.359%	96.391%



Conclusion: n_estimators specifies the number of decision trees in Random Forest, max_features specifies the number of fetures searched over while choosing the split, min_samples_split specifies the number of samples required to split an internal node. Accuracy decreases as min_samples_split increases. Accuracy increases and then decreases slightly with increase in max_features. Best set of Parameters are 250, 0.7, 2. The accuracy obtained by Random Forest is slightly more than the Decision Tree.

2 Neural Network

For b do `python3 Q2b.py`, For c do `python3 Q2c.py`, For d do `python3 Q2d.py`, For e do `python3 Q2e.py`, For autograding `./run_nn.sh <train_data> <test_data> <output_file> <M> <HLL> <A>`

Normalized the features dividing by 255 and converted the class labels into One Hot Encoding

(a) Implemented a generic neural network with input parameters as mini batch size(M), number of features, hidden layer architecture, number of target classes.

`Forward Propogation`:

$$h(\theta) = g(\theta^T \cdot X)$$

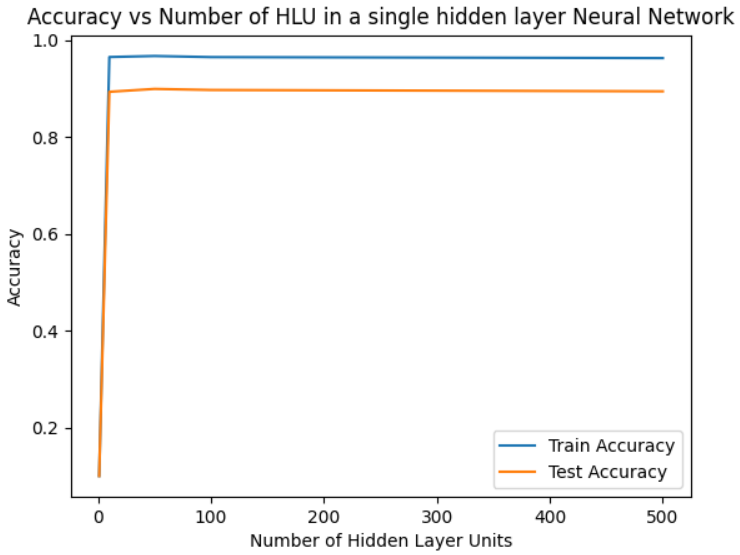
`Backward Propogation`:

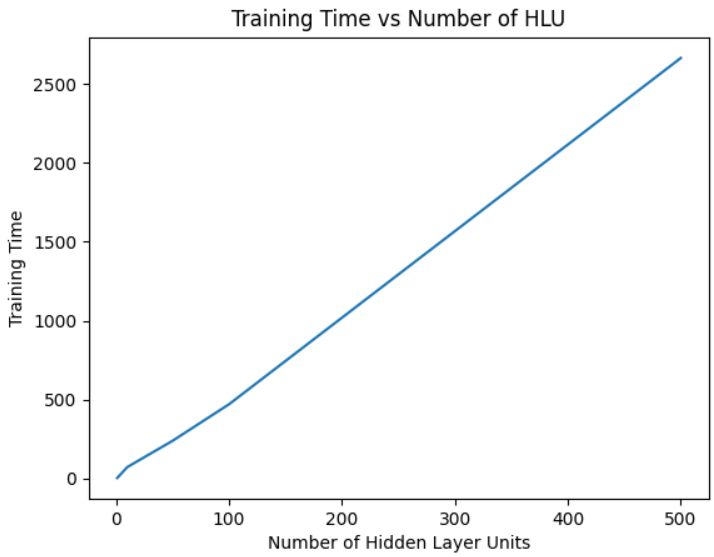
$$\text{Output Layer: } \delta_j = (y_j - o_j) \cdot o_j \cdot (1 - o_j)$$
$$\text{Hidden Layer: } \delta_j = \sum_{l \in downNumber(j)} \delta_l \cdot \theta_{lj} \cdot o_j \cdot (1 - o_j)$$

(b) Accuracy vs Number of Units in the Hidden Layer for a single Layer for Static Learning $\eta = 0.1$

Number of Units in Hidden Layer	**Training Time(s)**	**Final Cost**	**Training Accuracy**	**Test Accuracy**
[1]	1.92	epochs = 11, cost = 0.45	1%	1%
[10]	81.196	epochs = 99, cost = 0.015	96.53%	89.35%
[50]	296.06	epcohs = 77, cost = 0.013	96.74%	89.95%
[100]	692.99	epochs = 71, cost = 0.014	96.51%	89.73%
[500]	2666.32	epochs = 68, cost = 0.014	96.31%	89.45%

Observation: Time to coverge increases as the number of units in the hidden layer increases.

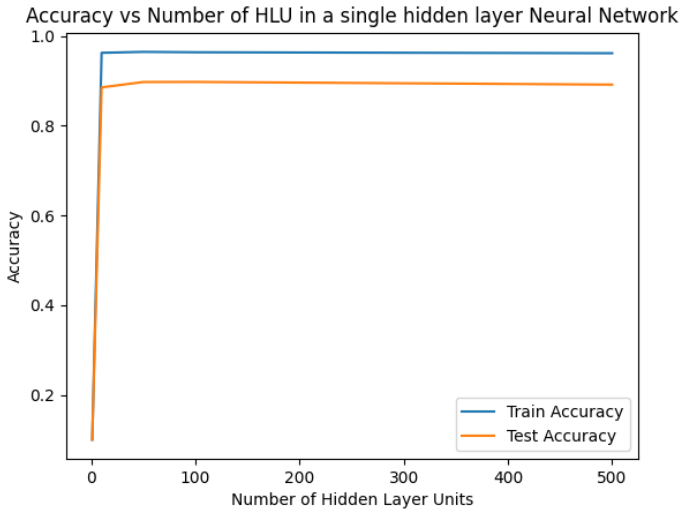




(c) Accuracy vs Number of Units in the Hidden Layer for a single Layer for Adaptive Learning $\eta = 0.5$

Number of Units in Hidden Layer	**Training Time(s)**	**Final Cost**	**Training Accuracy**	**Test Accuracy**
[1]	1.88	epochs = 11, cost = 0.45	1%	1%
[10]	49.28	epochs = 77, cost = 0.029	96.22%	88.51%
[50]	155.64	epochs = 51, cost = 0.015	96.43%	89.71%
[100]	310.55	epochs = 46, cost = 0.015	96.34%	89.74%
[500]	1189.89	epochs = 36, cost = 0.017	96.16%	89.24%

Observation: Adaptive Learning converges much faster than static learning. Time to converge increases with number of units in the hidden layer.



(d) Hidden Layer Architecture - [100, 100]. Adaptive Learning - $\eta = 0.5$

Observations	**Sigmoid**	**Relu**
Training Time(s)	596.05	263.895
Final Cost	epochs = 59, cost = 0.017	epochs = 25, cost = 0.011
Training Accuracy	96.33%	97.95%
Test Accuracy	89.6%	90.25%

Observaton: Relu converges much faster than sigmoid. The accuracy obtained by using relu is also higher than that of sigmoid.

(e) MLP Classifier with following parameters

1. `Hidden Layer Architecture`: (100, 100)
2. `activation`: relu
3. `solver`: sgd
4. `batch size`: 100
5. `learning_rate`: invscaling
6. `learning_rate_init`: 0.5
7. `power_t`: 0.5
8. `max_iter`: 100

Observations:

1. `Training Time`: 759.17 seconds
2. `Final Loss`: 0.06727
3. `Training Accuracy`: 98%
4. `Test Accuracy`: 91.21%

Conclusion: We obtain almost same accuracy with MLP Classifier and part 2d.