Decision Trees and Neural Networks

Prof. Parag Singla

Hardeep Kaur 2019CS10354

Assignment 3
COL774, Machine Learning
Indian Institute of Technology, Delhi

October 30, 2021

Contents

List of Figures

Decision Trees and Random Forests

Problem Statement

For this part of the assignment we will work on Bank Marketing data set.In this data set, the first row represents the labels of each column such that the last column represents whether the client subscribed (y/n) and following that, each row represents an example.We have to implement the decision tree algorithm for predicting whether client subscribed a term deposit.

1.1 Decision Tree Construction

Construction:
The decision tree is constructed using BFS. This keeps the tree balanced during construction. Each branch is grown just deeply enough to perfectly classify training examples. Since this strategy may lead to over fitting the data, pruning is performed later to make the model robust to random errors or noise in the training data. To construct the tree using DFS, one would have to provide a limit to the maximum depth to which it can be expanded, since otherwise it would keep growing the left branches.

Choosing the attribute to split:
Mutual information is used as the criteria for selecting the attribute to split on. In other words, split is performed on the attribute which leads to maximum reduction in entropy of class variable. Accuracy is computed as the percentage of correctly classified examples.

Handling Numerical Attributes:
To split at a numerical attribute, median of the attribute values is calculated from the training examples. A two way split is made based on whether the value of attribute is greater than the median or not.

Handling Categorical Attributes:
These are handled in following two ways:
(a) Performing a multi-way split by creating a branch for each possible value of the attribute
(b) Performing a two way split by first converting the attribute to a one-hot encoding, and then, treating each one-hot value as a separate Boolean valued attribute.

The following plot shows the training, test and validation accuracies vs number of nodes in the decision tree. We have used multi-way split for categorical attributes in this plot. Using one hot encoding for two-way splits also gave similar plot.

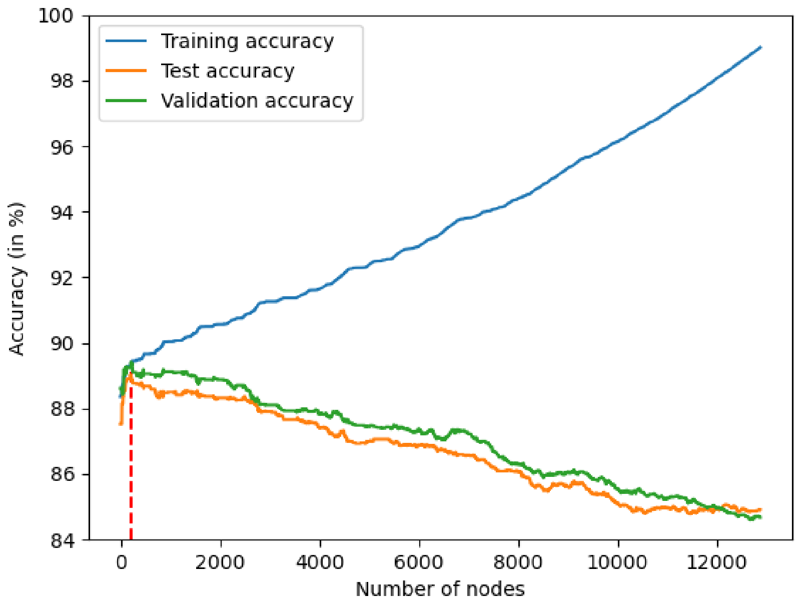

Figure 1.1: Accuracy vs number of nodes

Results:

1. Training Accuracy: 99%
2. Test Set Accuracy: 89.41%
3. Validation Accuracy: 84.67%

Observations:

- The accuracy of the tree over the training examples increases monotonically as the tree is grown. However the accuracy measured over independent test and validation data first increases and then decreases. Once the tree size exceeds approximately 200 nodes, further elaboration of the tree decreases its accuracy over the test and validation examples despite increasing its accuracy on the training examples.

- There may be two reasons for the above observation:

  1. Random errors or noise in data.
  2. Small number of examples associated with leaf nodes.

- Since we have grown the branches deeply enough to perfectly classify training examples, both the above reasons are valid and hence we need to make our model more robust to noise in the data.

  We have tried the following two approaches:

1. Stop growing the tree earlier , before it perfectly classifies the training data.
2. Allow the tree to over fit the data and then, post prune the tree.

- Although the first approach seems more direct, we found the the second approach of post pruning over fit trees to be more practical due to difficulty in estimating precisely when to stop growing the tree. We would use post pruning in the next section to avoid overfitting using the validation set.

1.2 Decision Tree Post Pruning

Reduced Error Pruning:

- Pruning a node involves deleting the subtree rooted at that node, making it a leaf node and assigning it the most common classification of training examples affiliated with that node.
- We first allow the tree to over fit the data. Nodes are then pruned greedily by iteratively picking a node to prune so that resultant tree gives maximum accuracy on the validation set.
- This is repeated until any further pruning leads to decrease in accuracy over the validation set.
- This method helps avoid over fitting because any leaf that was added due to noise in data is likely to be pruned.

Since pruning a node involves deleting the entire subtree rooted at that node, we have pruned the nodes in the lower half of the tree only, as we don't want all of the nodes (or most of the nodes) to be deleted due to an upper level node being pruned.

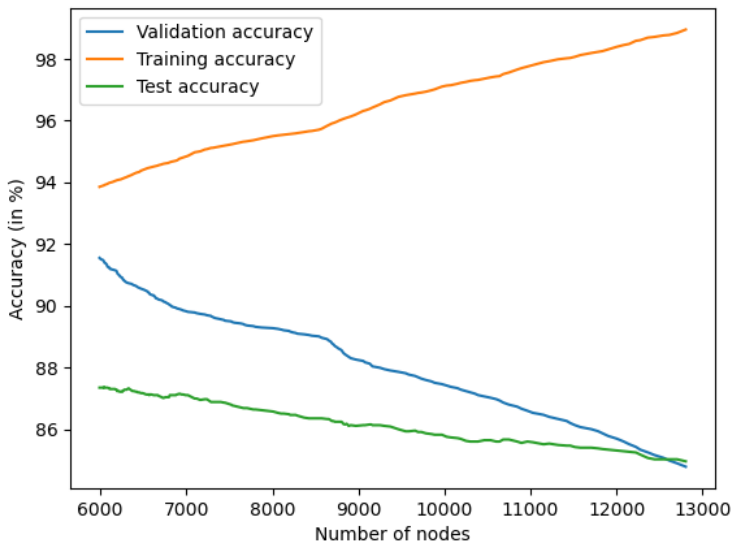

Figure 1.2: Post Pruning accuracy vs number of nodes

Results:

1. Training Accuracy: 93.85%
2. Test Set Accuracy: 87.34%
3. Validation Accuracy: 91.55%

Observations:

- The plot above shows the impact of reduced error pruning of the decision tree generated in section 1.1. Number of nodes in the tree decreases as the pruning proceeds, hence the accuracy curves could be seen from right to left.
- We observe that the accuracy of the tree over the training examples is decreasing with number of nodes. However the accuracy measured over independent test and validation data has increased significantly and has reached close to training set accuracy.
- This clearly indicates that the model has become robust to noise in training data.

1.3 Random Forests

Random Forests are extensions of decision trees, where we grow multiple decision trees in parallel on bootstrapped samples constructed from the original training data.In this part, we used the scikit-learn library of Python to grow a Random Forest. A grid search is performed over the following space of parameters using GridSearchCV.

- *n_estimators*: {50, 150, 250, 350, 450}
- *max_features*: {0.1, 0.3, 0.5, 0.7, 0.9}
- *min_samples_split*: {2, 4, 6, 8, 10}

Out-of-bag accuracy is used to tune to the optimal values for these parameters.

Results

- Optimal set of parameters:
  - *n_estimators*: 250
  - *max_features*: 0.5
  - *min_samples_split*: 4
- Out of Bag accuracy: 90.79%
- Training Set Accuracy: 99.89%
- Validation Set Accuracy: 90.40%
- Test Set Accuracy: 89.80%

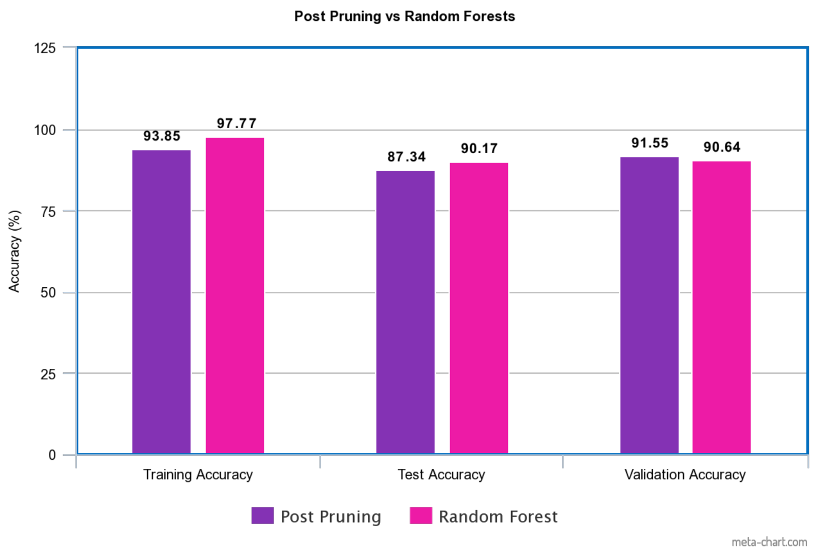

Figure 1.3: Comparison of Random Forests and Post Pruning

Observations:

- We observe that both the training and test accuracy have increased using the Random Forests.The validation set accuracy, however, is similar in both the cases.
- Random Forests use a larger number of classifiers or an 'ensemble of models' which makes the model more robust and increases the accuracy on test data set.
- We observe that optimal value for max number of features is half of the total number of features. This makes sense as choosing a fraction of features for each classifier makes the model resistant to over fitting.
- We also observe that the optimal value of minimum number of samples required to split an internal node is 10. This makes intuitive sense as splitting the node on basis of very few data samples would just lead to over fitting of training data.

1.4 Random Forests- Parameter Sensitivity Analysis

The accuracies on varying one parameter at a time is shown below:
The plots are as follows:

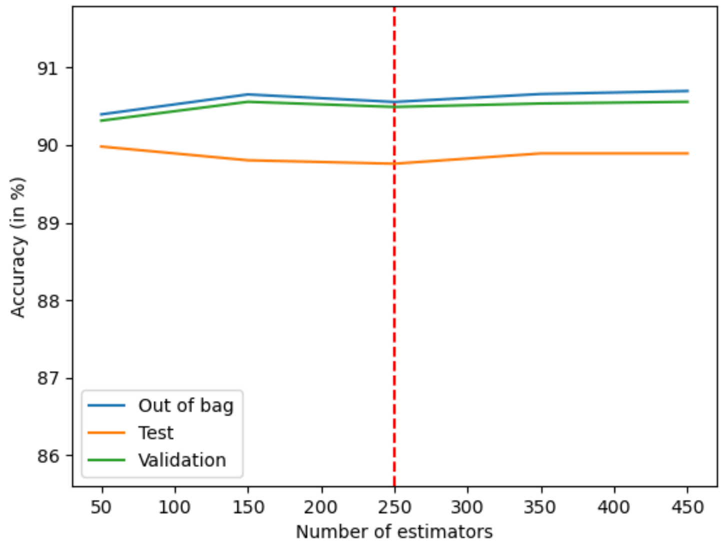

Figure 1.4: Accuracies on varying n_estimators

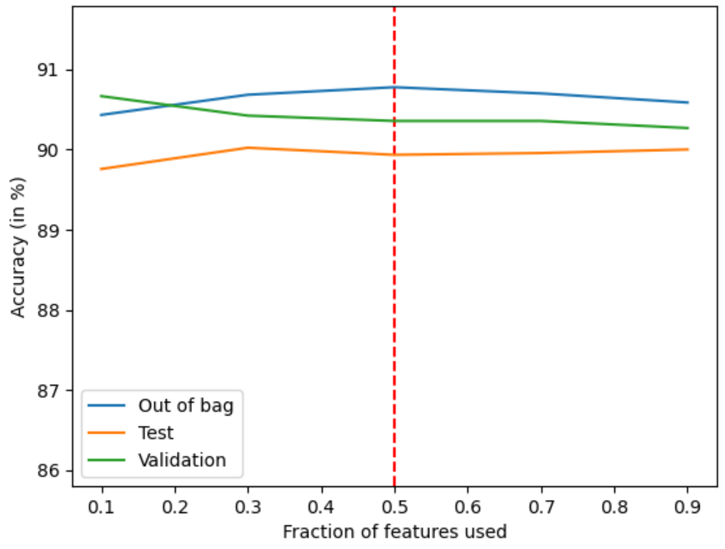

Figure 1.5: Accuracies on varying max_features

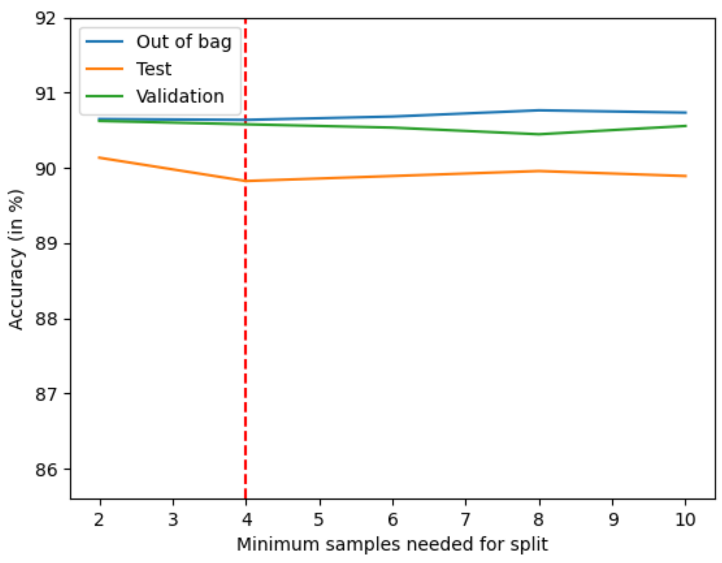

Figure 1.6: Accuracies on varying min_samples_split

Neural Networks

Problem Statement

We will work with the Poker Hand data set available on the UCI repository.The data set consists of 10 categorical attributes. The last entry in each row denotes the class label.

2.1 One Hot Encoding

We mentioned two approaches to handle categorical attributes in the first section. We used the first approach of multi way splitting in part 1 of this assignment. Now, we would use the second approach of one hot encoding for performing binary splits in case of the categorical attributes. The original data set had 16 features and the new data set (after encoding) has 85 features.

2.2 Implementation

We have implemented a generic neural network architecture to learn a model for multi-class classification. We have used Back propagation and Mini Stochastic Gradient Descent to train our model. Mean Squared Error (MSE) over each mini-batch is used as loss function.

$$J_b = \frac{1}{2M} \sum_{i=(b-1)M}^{bM} \sum_{l=1}^{r} (y_l^{(i)} - o_l^{(i)})^2$$

This part uses the sigmoid function as activation function for the units in output layer as well as in the hidden layer.
Parameter Initialization: This implementation uses *He initialization* for parameters. This involves multiplying the random initialized values with

$$\sqrt{\frac{2}{size_{l-1}}}$$

where $size_{l-1}$ is the size of the previous hidden layer. The biases are initialised as 0. The implementation is generic to generate architectures based on Mini Batch Size, Number of features, Hidden Layer Architecture, Number of target classes

2.3 Varying units in the hidden layer

Performance is analyzed in terms of accuracy and time taken by varying the number of hidden units in the single hidden layer from the set {5,10,15,20,25}. Learning rate is 0.1 and mini batch size, M is 100.

Stopping Criterion:

First, the average error is calculated over the mini batches in a single epoch. When the difference between average errors over two consecutive epochs goes below $\varepsilon = 0.00001$, convergence is declared. Plots are show below: **Results:**

- No. of hidden layer units: 5
  - Training Accuracy: 49.96 %
  - Test Accuracy: 50.05 %
  - Time taken: 5.45 s
  - Number of epochs: 27
  - Confusion Matrix:

$$\begin{vmatrix} 493254 & 7955 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 415219 & 7279 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 46750 & 872 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 20710 & 411 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3780 & 105 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1938 & 58 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1395 & 29 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 226 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

- No. of hidden layer units: 10
  - Training Accuracy: 49.93 %
  - Test Accuracy: 50.10 %
  - Time taken: 2.58 s
  - Number of epochs: 20
  - Confusion Matrix:

$$\begin{vmatrix} 500700 & 509 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 422161 & 337 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 47592 & 30 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 21113 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3885 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1976 & 20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1424 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 230 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

- No. of hidden layer units: 15
  - Training Accuracy: 49.67 %
  - Test Accuracy: 49.79 %
  - Time taken: 6.32 s
  - Number of epochs: 41
  - Confusion Matrix:

$$\begin{vmatrix}
484200 & 17009 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
408788 & 13710 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
46189 & 1433 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
20523 & 598 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3702 & 183 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1929 & 67 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1386 & 38 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
223 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
11 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{vmatrix}$$

- No. of hidden layer units: 20
  - Training Accuracy: 49.99 %
  - Test Accuracy: 50.06 %
  - Time taken: 5.93 s
  - Number of epochs: 41
  - Confusion Matrix:

$$\begin{vmatrix}
497712 & 3497 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
419544 & 2954 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
47277 & 345 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
20999 & 122 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3849 & 36 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1987 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1416 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
230 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{vmatrix}$$

- No. of hidden layer units: 25
  - Training Accuracy: 49.81 %
  - Test Accuracy: 49.78 %
  - Time taken: 11.75 s
  - Number of epochs: 43

– Confusion Matrix:

$$\begin{vmatrix} 481153 & 20056 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 405764 & 16734 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 45790 & 1832 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 20274 & 847 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3723 & 162 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1853 & 143 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1378 & 46 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 222 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

Observations

- We observe that the model performs more like a majority prediction model by only predicting two classes out of 10. This may be due to the bias in training data which contains 12493/25010 examples with label 0 and 10599/25010 examples with label 2.
- Also, the time taken to train the model increases with increasing the number of units in the hidden layer which is also expected as more units would require. more computations.
- From the accuracy plot, we observe that there is only a slight difference in accuracies obtained by varying the number of units in hidden layer. This was, however, expected to increase with increasing number of units. One reason for the observation may be the bias in the data.

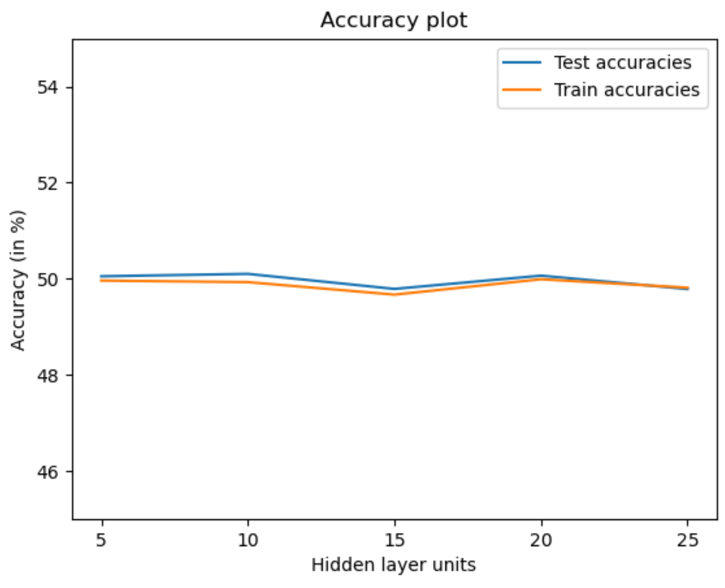

Figure 2.1: Accuracies on varying hidden layer units

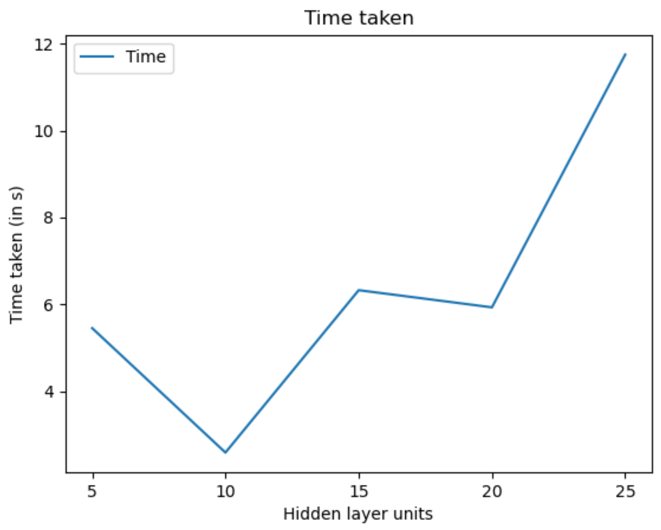

Figure 2.2: Time taken on varying hidden layer units

2.4 Adaptive Learning

In this section, we used adaptive learning rate inversely proportional to number of epochs i.e. $\eta = \frac{\eta}{\sqrt{2}}$ where $\eta_0 = 0.1$ is the seed value and e is the current epoch number.

Stopping Criterion:

First, the average error is calculated over the mini batches in a single epoch. When the difference between average errors over two consecutive epochs goes below $\varepsilon = 0.00001$, convergence is declared. Results and plots are shown below:
Results:

- No. of hidden layer units: 5
  - Training Accuracy: 49.64 %
  - Test Accuracy: 49.84 %
  - Time taken: 4.95 s
  - Number of epochs: 52
  - Confusion Matrix:

$$\begin{vmatrix}
478483 & 22726 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
402526 & 19972 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
45208 & 2414 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
20033 & 1088 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3636 & 249 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1821 & 175 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1347 & 77 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
221 & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
11 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{vmatrix}$$

- No. of hidden layer units: 10
  - Training Accuracy: 49.86%
  - Test Accuracy: 49.88 %
  - Time taken: 2.74 s
  - Number of epochs: 42
  - Confusion Matrix:

$$\begin{vmatrix}
485084 & 16125 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
408698 & 13800 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
46024 & 1598 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
20378 & 743 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3779 & 106 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1958 & 38 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1374 & 50 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
225 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{vmatrix}$$

- No. of hidden layer units: 15
  - Training Accuracy: 49.47 %
  - Test Accuracy: 49.54 %
  - Time taken: 4.98 s
  - Number of epochs: 45
  - Confusion Matrix:

$$\begin{vmatrix}
462958 & 38251 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
390013 & 32485 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
44094 & 3528 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
19529 & 1592 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3536 & 349 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1873 & 123 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1313 & 111 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
209 & 21 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
11 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{vmatrix}$$

- No. of hidden layer units: 20
  - Training Accuracy: 49.27 %
  - Test Accuracy: 49.47 %
  - Time taken: 5.84 s
  - Number of epochs: 75

  - Confusion Matrix:

$$\begin{vmatrix}
459346 & 41863 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
387064 & 35434 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
43613 & 4009 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
19326 & 1795 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3546 & 339 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1775 & 221 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1310 & 114 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
203 & 27 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
8 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{vmatrix}$$

- No. of hidden layer units: 25
  - Training Accuracy: 48.92 %
  - Test Accuracy: 48.48 %
  - Time taken: 5.57 s
  - Number of epochs: 71
  - Confusion Matrix:

$$\begin{vmatrix}
399210 & 101999 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
336858 & 85640 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
38046 & 9576 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
17048 & 4073 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2958 & 927 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1525 & 471 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1150 & 274 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
178 & 52 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
10 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{vmatrix}$$

Observations

- We observe that the accuracies obtained using adaptive learning rate are almost same as non adaptive model but the number of epochs taken to converge (using same stopping criterion) has increased.
- Also, the time taken to train the model increases with increasing the number of units in the hidden layer which is also expected as more units would require. more computations.
- From the accuracy plot, we observe that there is only a slight difference in accuracies obtained by varying the number of units in hidden layer. This was, however, expected to increase with increasing number of units. One reason for the observation may be the bias in the data.

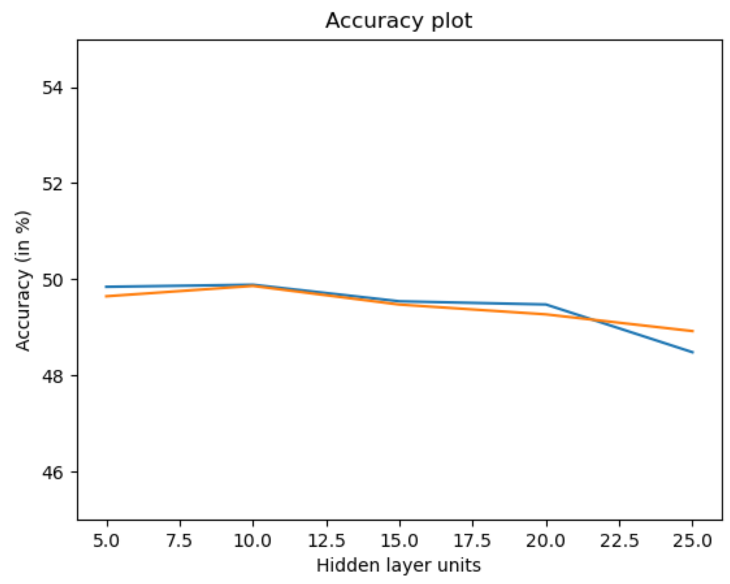

Figure 2.3: Accuracies on varying hidden layer units in Adaptive model

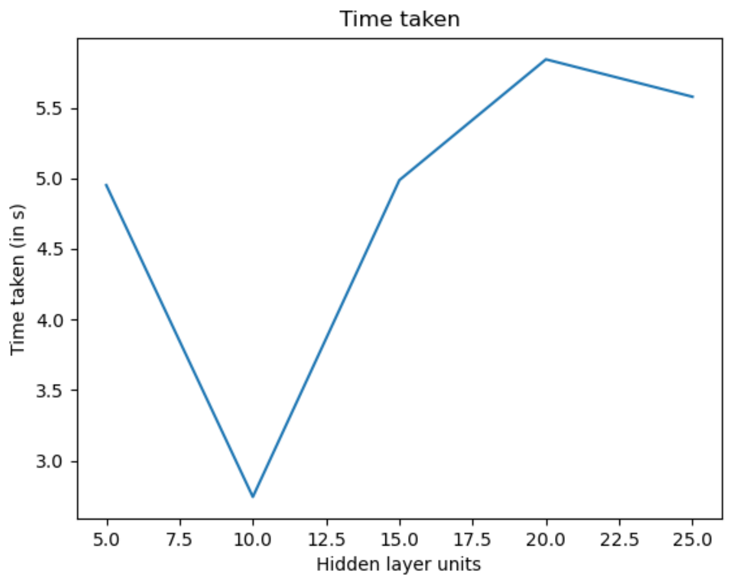

Figure 2.4: Time taken on varying hidden layer units in Adaptive model

2.5 Using Rectified Linear Activation Function

The rectified linear activation function or ReLU is a piece wise linear function that will output the input directly if it is positive, otherwise, it will output zero. We implemented a network with 2 hidden layers with 100 units each and used ReLU as the activation function for hidden layers except for the last layer. We ahve also used adaptive learning rate. This section compares the models obtained by using sigmoid and ReLU as activation functions for the perceptrons.

Results using Sigmoid Function, adaptive learning rate

- Training Accuracy: 49.95%
- Test Accuracy: 50.12 %
- Confusion Matrix:

$$\begin{vmatrix}
501209 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
422498 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
47622 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
21121 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3885 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1996 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1424 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
230 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{vmatrix}$$

Results using ReLU Function, adaptive learning rate

- Training Accuracy: 50.06%
- Test Accuracy: 50.14%
- Confusion Matrix:

$$\begin{vmatrix}
498226 & 2983 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
419320 & 3178 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
47189 & 433 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
20889 & 232 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3850 & 35 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1989 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1407 & 17 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
223 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
12 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{vmatrix}$$

Results using ReLU Function

- Training Accuracy: 69.34%
- Test Accuracy: 67.36%

Observations

- We observe that the accuracy increased by a small amount when we used ReLU function with adaptive learning rate. But when used without adaptive learning rate, the accuracy was increased significantly. Sigmoid function, on the other hand, gave almost same accuracy using both adaptive and non adaptive learning rates.

2.6 Using MLPClassifier from scikit-learn library

We implemented the same architecture as the previous function using MLPClassifier for both sigmoid ('logistic') as well as ReLU activation functions.
Results

Results using sigmoid Function

- Training Accuracy: 99.92%
- Test Accuracy: 97.76%

Results using ReLU Function

- Training Accuracy: 100%
- Test Accuracy: 99.26%

Observations

We observe that this implementation has significantly better results than ours.