COL-774 Machine Learning
Assignment -3 (Semester I, 2021-2022)

1) Decision Tree -

Part a)

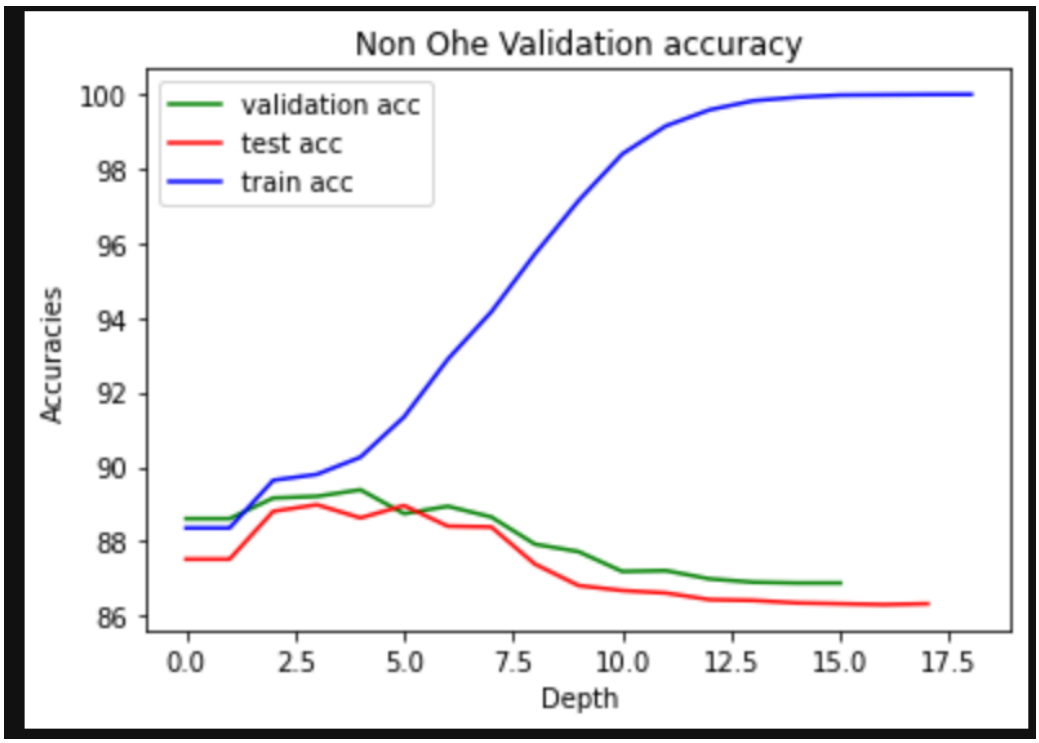
Decision Tree Construction --
The data is read and selected categorical(having multiple values) columns are converted into Ohe Hot Vector Representation.

- List of columns transformed to get OHEs (only for OHE case) categorical_cols = ['job', 'marital', 'education', 'contact', 'month', 'campaign', 'poutcome']
- All the other columns are either numerical or categorical(with only 2 values)
- All the cols having ‘yes’, ‘no’ are converted to True(1) and False(0)

Using multi way split for categorical data-

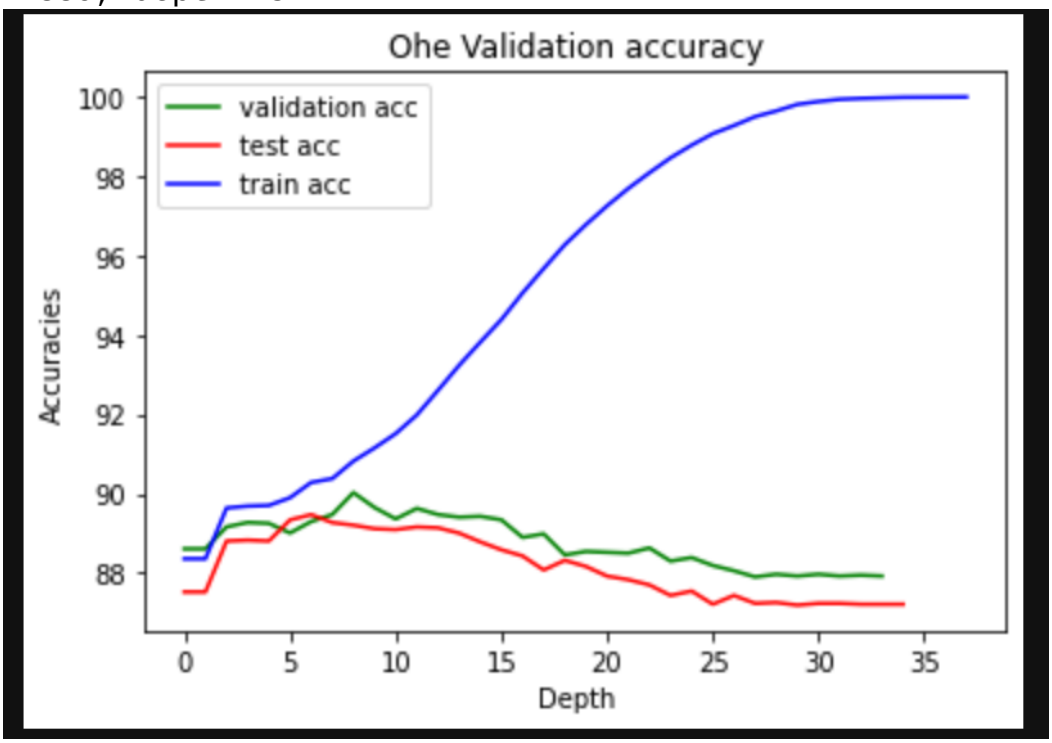
Without pruning --
Nodes: 10574, depth: 18

Variation of the accuracies depth wise for train, test, validation--



Using OHE for categorical data-

Ohe Tree Not pruned-
Nodes: 7889, depth: 37



Part b)Decision tree Pruning -

Using Reduced Error Pruning on the Trees obtained with Validation set data.

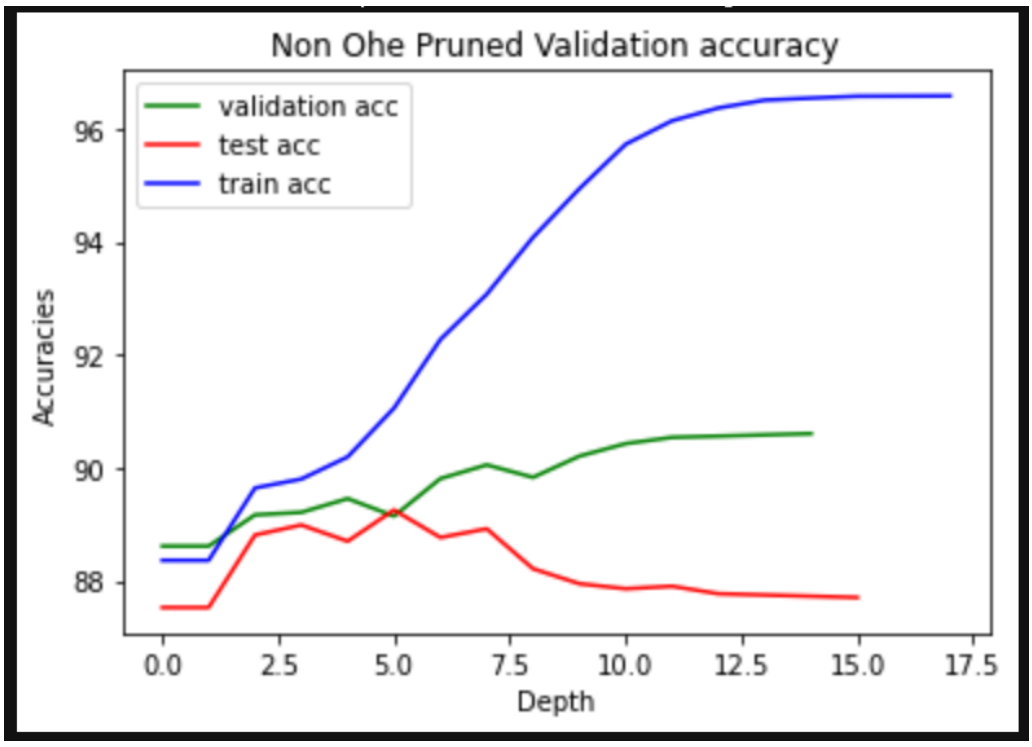
Here I am using Top Down way to move and prune the tree, but in code there is just a parameter prune_below , which tells whether the below sub tree is pruned or not.

There is significant decrease in the number of nodes after pruning tree as we can see below

Tree After Pruning(non OHE) --
Nodes: 5478, depth: 17

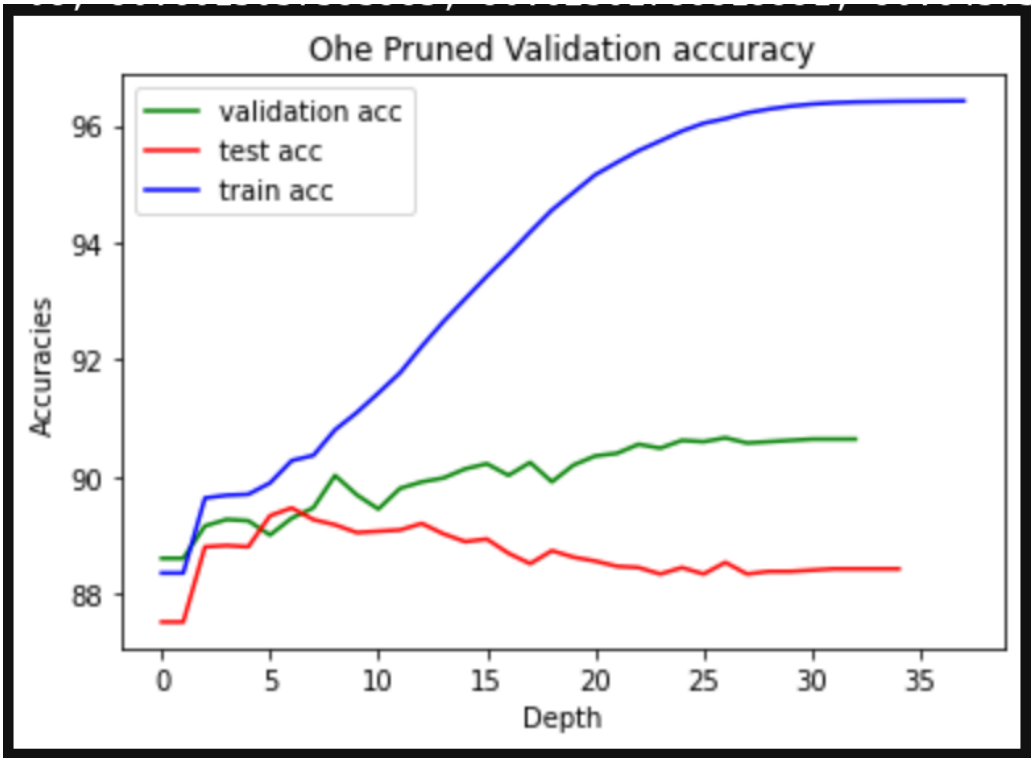
At last depth -
Validation Accuracy = 91.1764705882352
Test Accuracy = 88.07785888077859
Train Accuracy = 96.2010617120106

Best Test acc at depth = 6
Test Accuracy = 89.27228489272




```
Ohe Tree After pruning --
Nodes: 4519, depth: 37
At last depth -
Validation Accuracy = 90.64573197700133
Test Accuracy = 88.43176288431764
Train Accuracy = 96.41672196416722

Best Test acc at depth = 6
Test Accuracy = 89.471355894
```



As we can see if we stop at depth 6-7 that will be the best model we get as it will not over fit the train data.

Part c)

Here I am using the **OHE dataset** generated in part 1 as dataset as there will be **no problem of maping categorical data** to integers as they will only be boolean value.

Using ,seeed(0) To keep the data consistent for every run for Random Forest Algorithm

Best Values got --

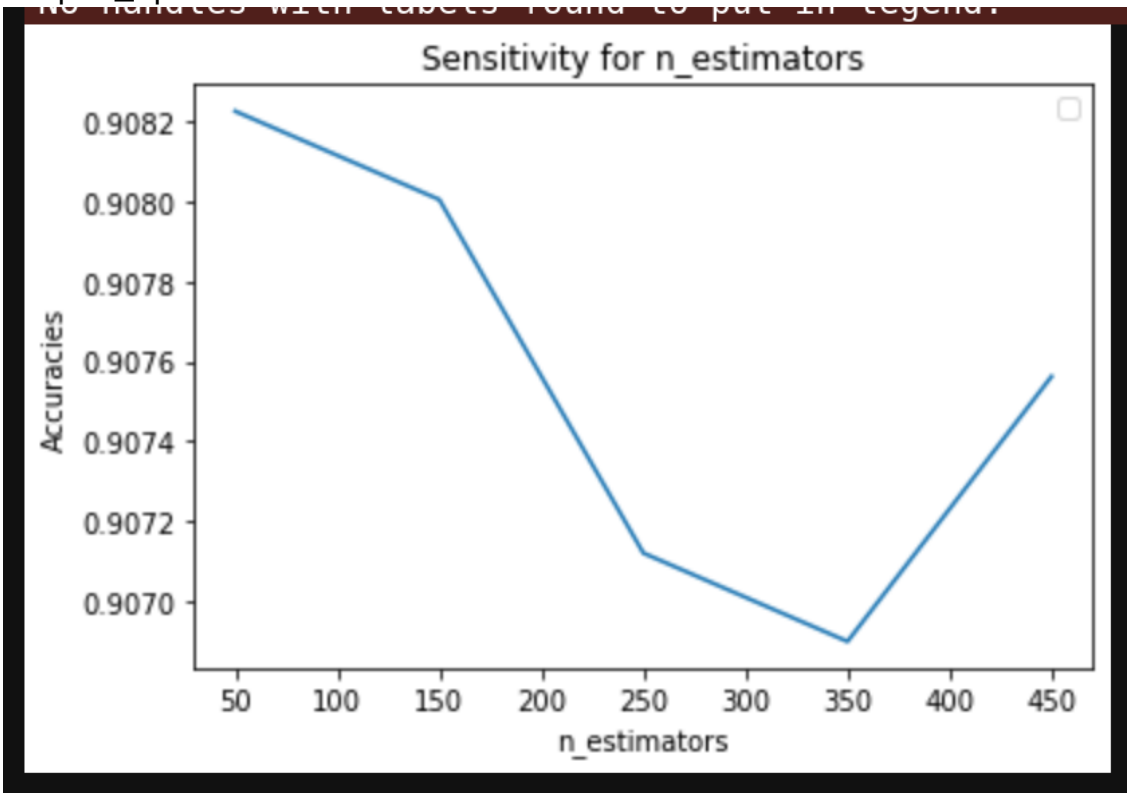
```
Best n_estimator     : 50
Best max_feature     : 0.3
Best min_sample_split: 4

Train Accuracy     : 0.9949955761999557
Test Accuracy      : 0.9015704490157045
Validation Accuracy: 0.9082264484741265
CPU times: user 1h 6min 35s, sys: 1.16 s, total: 1h 6min 36s
Wall time: 1h 6min 37s
```

Part d)Parameter Sensitivity Analysis-

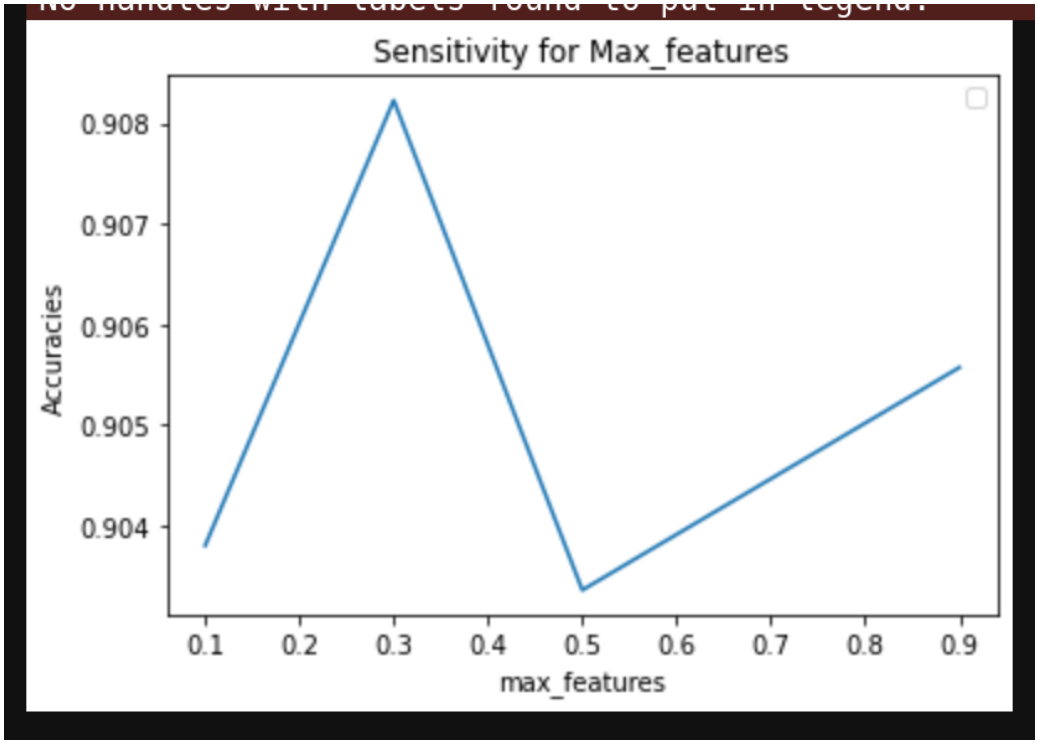
For n_estimator = [50, 150, 250, 350, 450]

```
max_feature     : 0.3
min_sample_split: 4
```



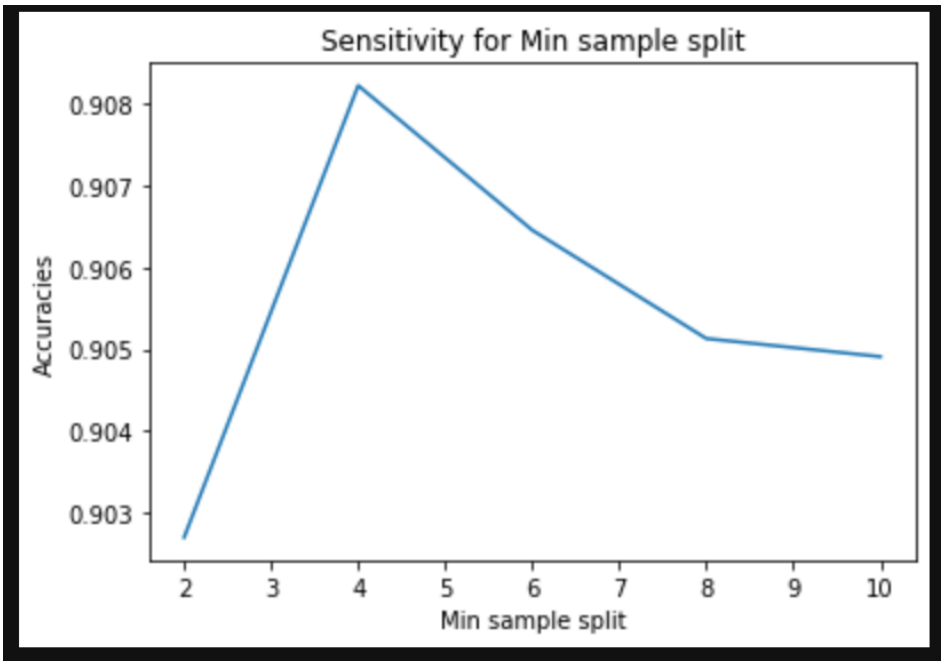
So we can see that affect of n_estimatro is roughly by 0.001 on the accuracy, i.e for acc in 100% it’s 0.1

For Max-Feature = [0.1, 0.3, 0.5, 0.7, 0.9]
n_estimator : 50
min_sample_split: 4



So we can see that affect of n_estimatro is roughly by 0.004 on the accuracy, i.e for acc in 100% it’s 0.4

For Min Sample Split -[2 4 6 8 10]
n_estimator : 50
max_feature : 0.3



so we can see that affect of n_estimatro is roughly by 0.005 on the accuracy, i.e for acc in 100% it's 0.5

2) Neural Networks -

Part a)

The input data is read and converted into one hot vector representaion.

For eg -

a	b
1	4
1	0
3	0

To --> One Hot Vector Representation

a.1	a.3	b.4	b.0
1	0	1	0
1	0	0	1
0	1	0	1

Part b)

The Neural network is implemented the class NN, all the methods are implemented in it, from back propagation to forward pass, etc.

Back Propagation is implemented in function – *back_propagation()*
The main idea is to keep on going upstream in the model and calculate the gradients needed for every layers.

Since we are using Smini batch Stochastic Gradient Descent so at every epoch the **data is shuffeled randomly** and then a batch is taken from that shuffeled data.

The **cost is calculated for every batch** but when the code runs it only shows for few intervals rather than dumping everything for ease of readibility

The **weights are assigned randomly** and also scaled (divided by sqrt(feature size) so that they don't become very large.
(*This proved very effective especially to suprass the 50 accuracy at which many a times the model get stuck*)

The NN class is capeble of handling these input parameters --

• Mini-Batch Size (M)
• Number of features/attributes (n)
• Hidden layer architecture: List of numbers denoting the number of perceptrons in the correspond-
ing hidden layer. Eg. a list [100 50] specifies two hidden layers; first one with 100 units and second
one with 50 units.
• Number of target classes (r)

There are also other parameters which are used to play with the model
• Learning Rate type -- adaptive or normal(i.e constant)
• Max epochs
• Leraning rate
• Stopping Criteria , etc

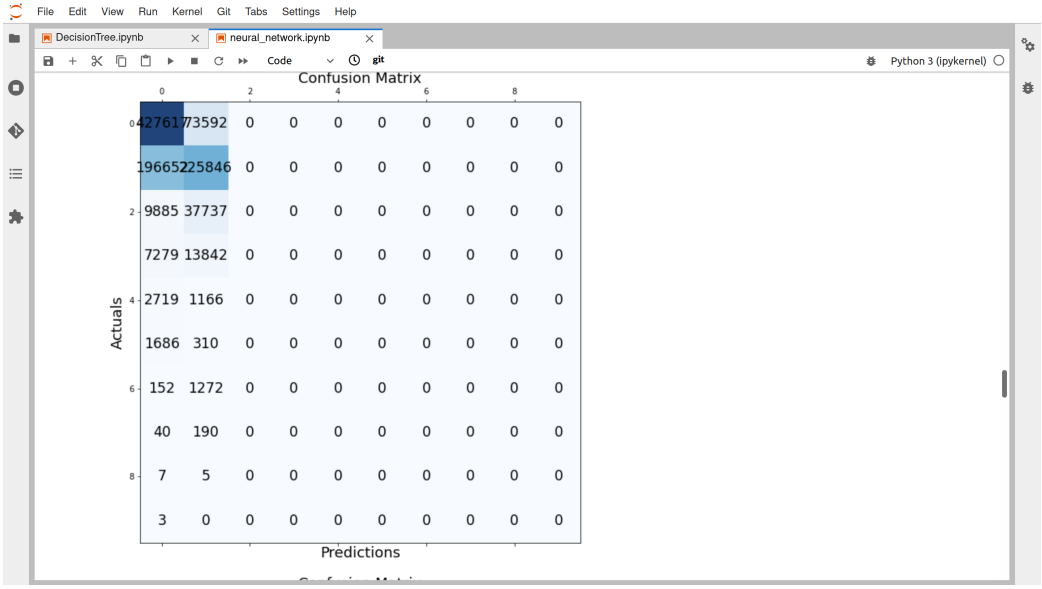
Part c)

Here I used the stopping criteria as 0.0001 but the difference is not between two consecutive epochs rather it's between k epochs (here k = 100)

Learning Rate Used – 0.1
Stopping Criteria – 0.0001 (difference between errors 100 epochs away)
Hidden layers units - {5, 10, 15, 20, 25}

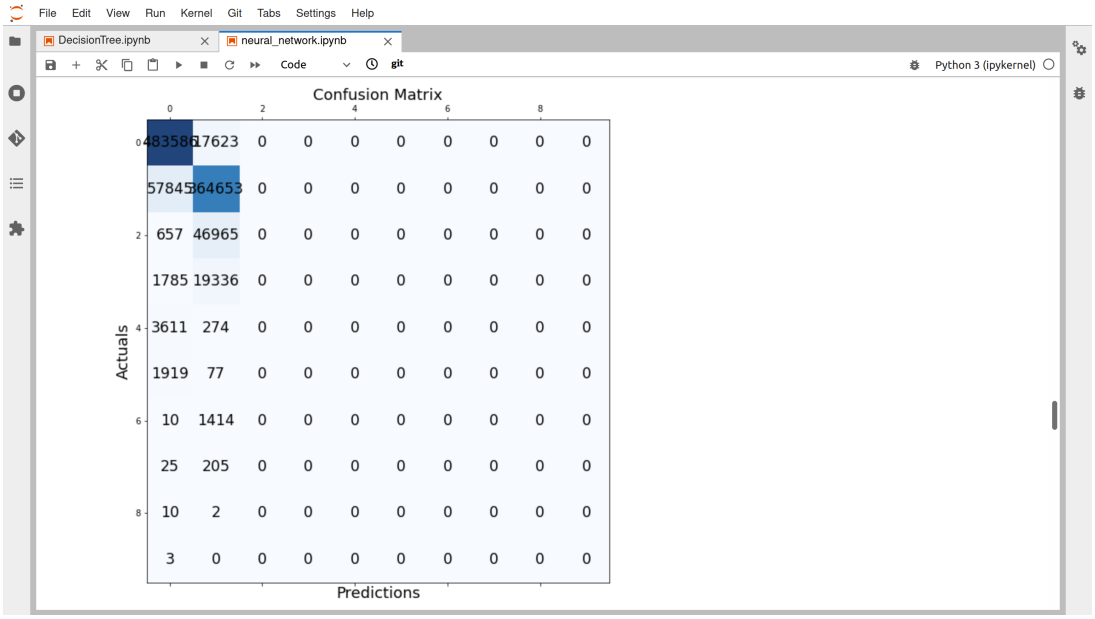
For Hidden Layer = [5]
Train Accuracy : 0.66613354
Test Accuracy : 0.653463

Final Error : 1.129717
epochs taken : 7900
Stopping Value(for dff in 100 epochs err) : 0.0001
Time Taken : 0:06:09.99679
Confusion Matrix -



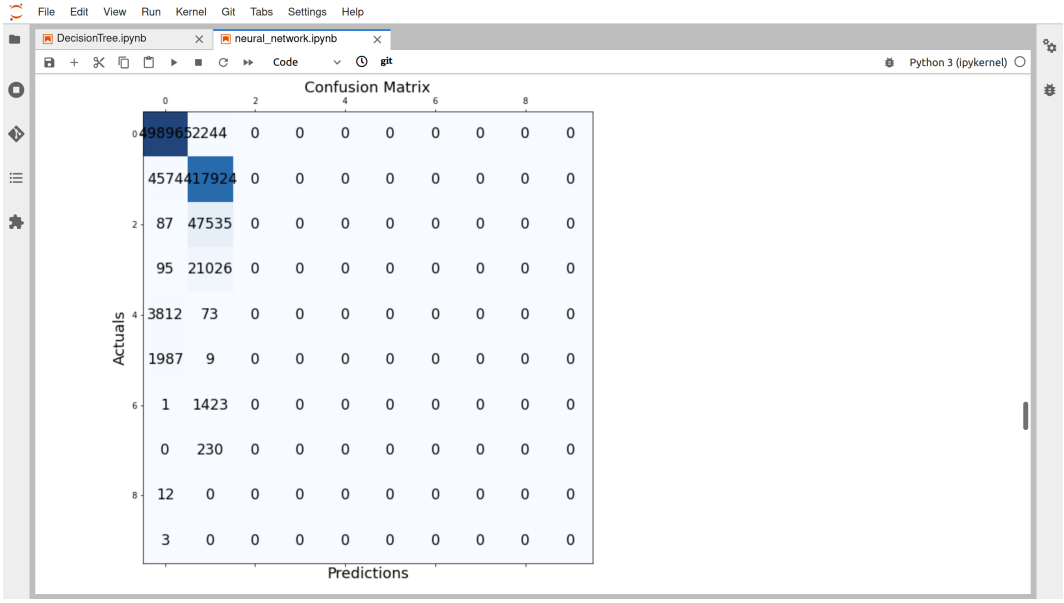
For Hidden Layer = [10]
Train Accuracy : 0.862455
Test Acuracy : 0.848239
Final Error : 0.5939979745997764
epochs taken : 10000
Stopping Value(for dff in 100 epochs err) : 0.0001
Time Taken to train 0:08:27.288799

Confusion Matrix -



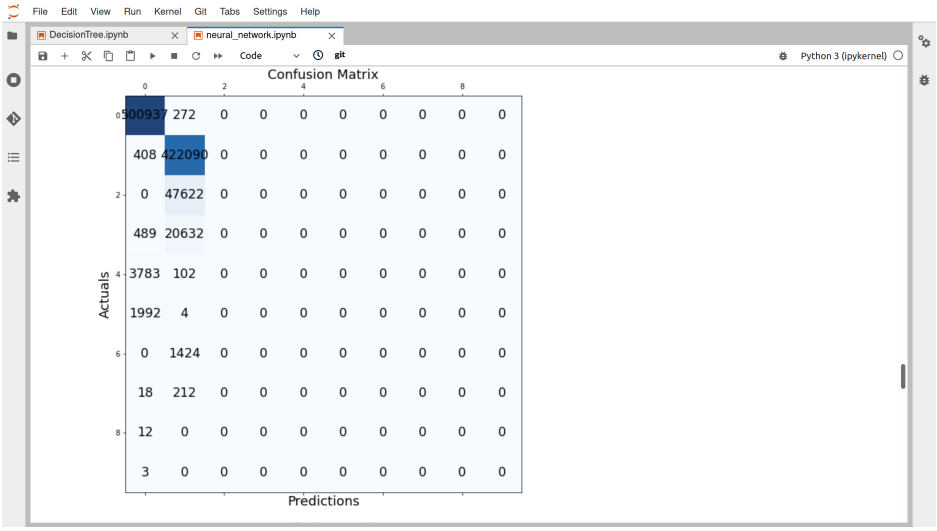
For Hidden Layer = [15]
Train Accuracy : 0.920631
Test Accuracy : 0.916889
Final Error   : 0.30707065
epochs taken   : 7900
Stopping Value(for dff in 100 epochs err) : 0.0001
Time Taken to train 0:07:24.484306

Confusion Matrix -



For Hidden Layer = [20]
Train Accuracy : 0.923310
Test Accuracy : 0.923027
Final Error   : 0.274203
epochs taken   : 9300
Stopping Value(for dff in 100 epochs err) : 0.0001
Time Taken to train 0:08:31.368144

Confusion Matrix -



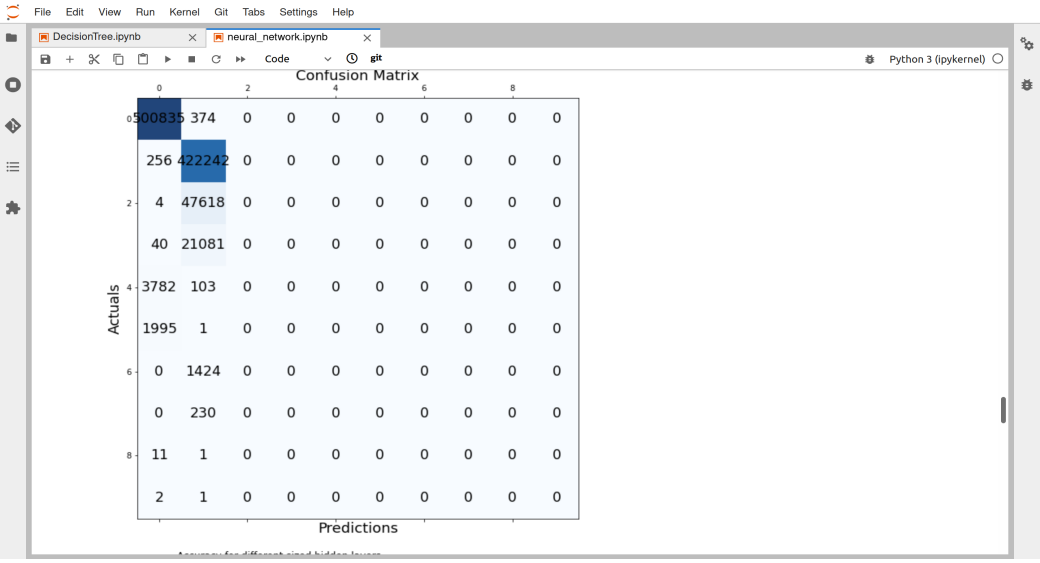
For Hidden Layer = [25]

Here it looks stopping criteria of 0.001 had been sufficient as for 0.0001 model keeps on training and there is very less improvement in the error.

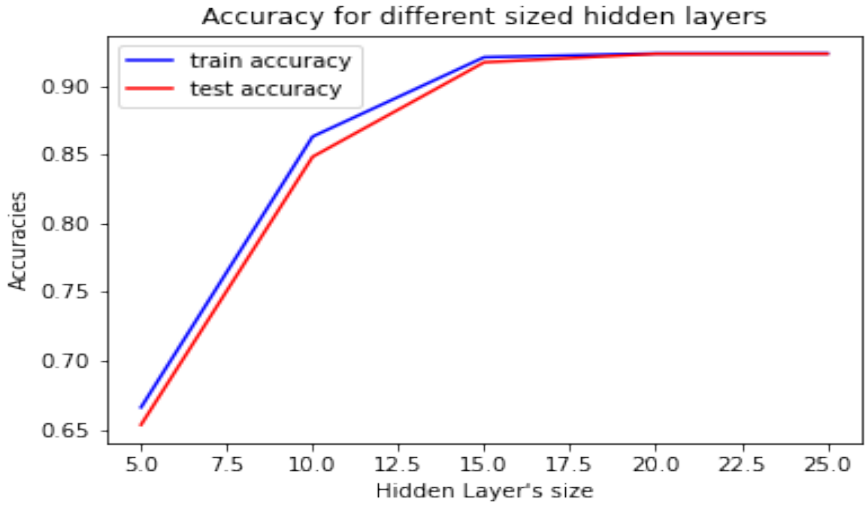
```
Train Accuracy : 0.923310
Test Accuracy  : 0.923077
Final Error    : 0.272134
epochs taken   : 10000
Stopping Value(for dff in 100 epochs err) : 0.0001
Time Taken to train 0:09:50.370113
```

Althouogh it took lots of epochs this accuracy was first achieved at -
epoch 4700 | error: 0.304762 | acc: 92.331068

Confusion Matrix -



Plot for accuracy --



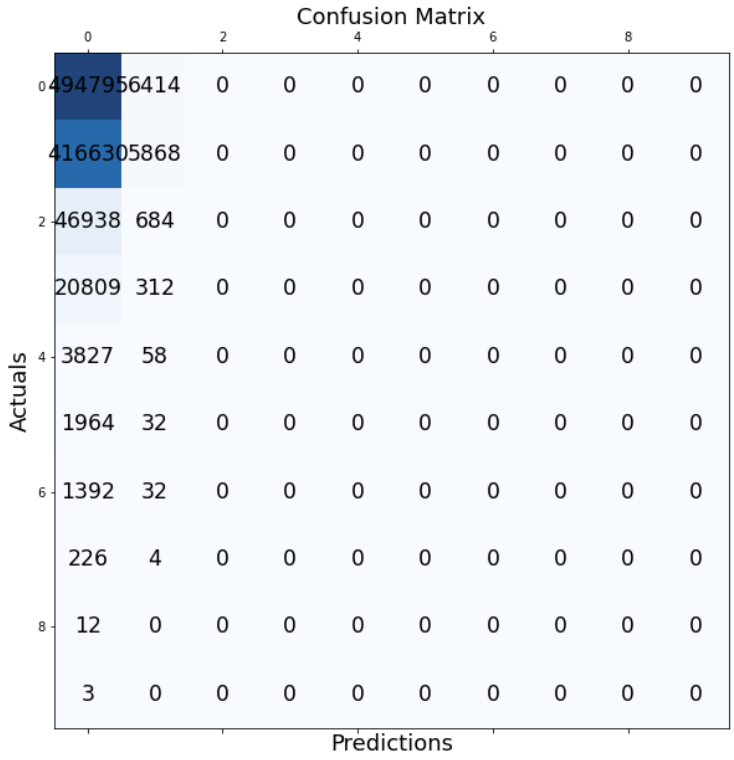
Part d) Using Adaptive Learning Rate

For Adasptive learning rate we need to decrease the stopping criteria as without it, the model is only learning to predic 1 class and therefore stopping at about ~49-50 accuracy.

So the **Stopping criteria** is reduced to= 0.000001
But still the change in error is very small and the learning stops without giving better accuracy

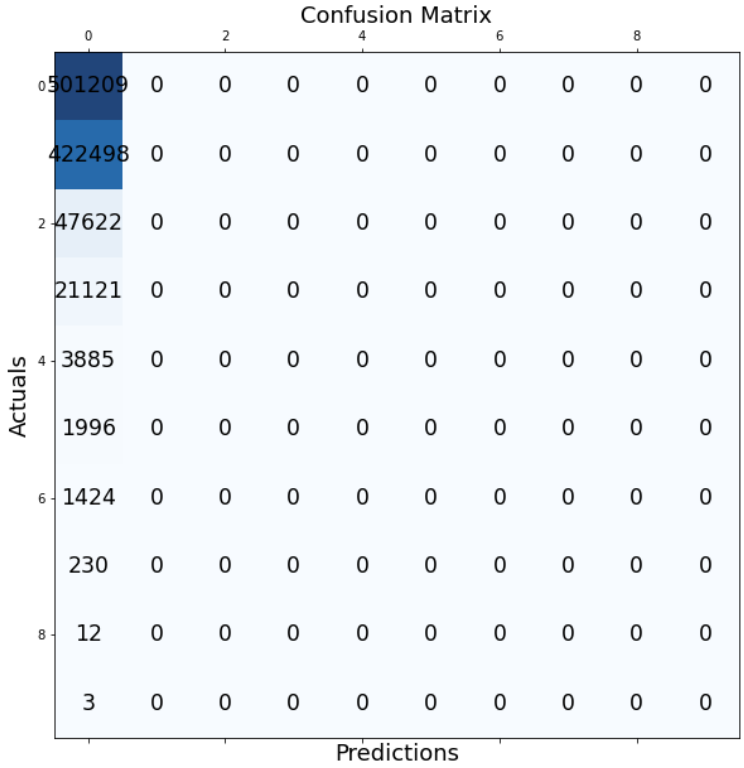
Hidden Layer = [5]
Final Accuracy : 0.499440
Test Accuracy : 0.500663
Final Error   : 1.4214362
epochs taken   : 1000
Stopping Value(for dff in 100 epochs err) : 1e-06
Time Taken to train 0:00:46.011529

Confusion Matrix -



Hidden Layer = [10]
Final Accuracy : 0.499520
Test Accuracy : 0.501209
Final Error   : 1.4236859
epochs taken   : 1000
Stopping Value(for dff in 100 epochs err) : 1e-06
Time Taken to train 0:00:48.192984

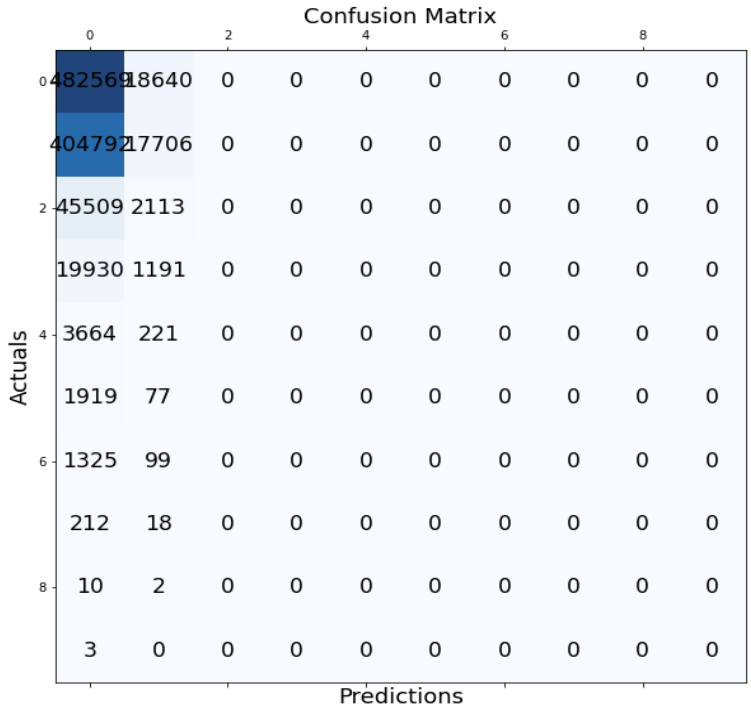
Confusion Matrix -



Hidden Layer = [15]

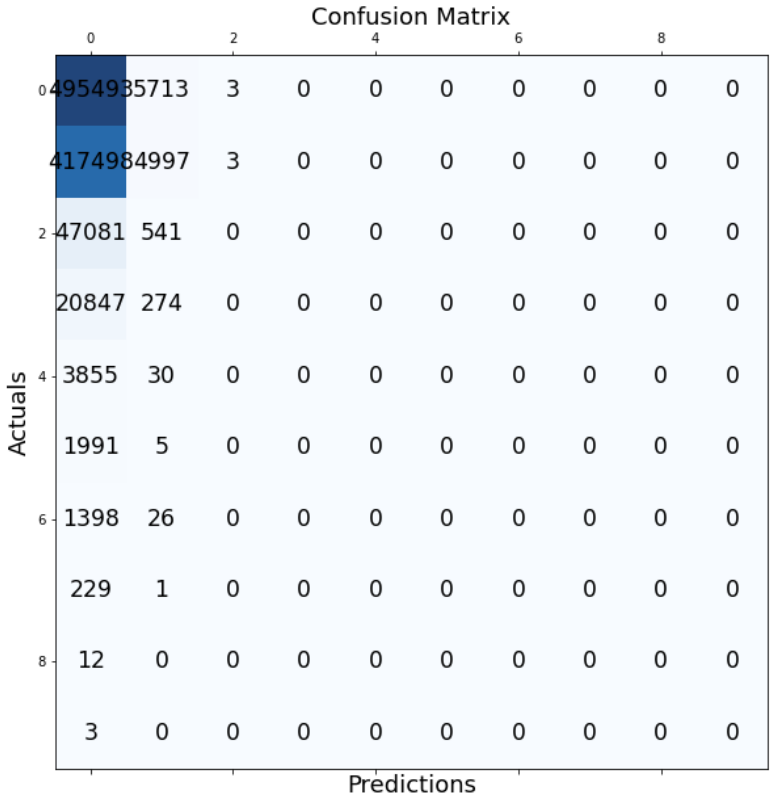
```
Final Accuracy : 0.500319
Test Accuracy  : 0.500275
Final Error    : 1.4211306
epochs taken   : 1000
Stopping Value(for dff in 100 epochs err) : 1e-06
Time Taken to train 0:00:53.505227
```

Confusion Matrix -

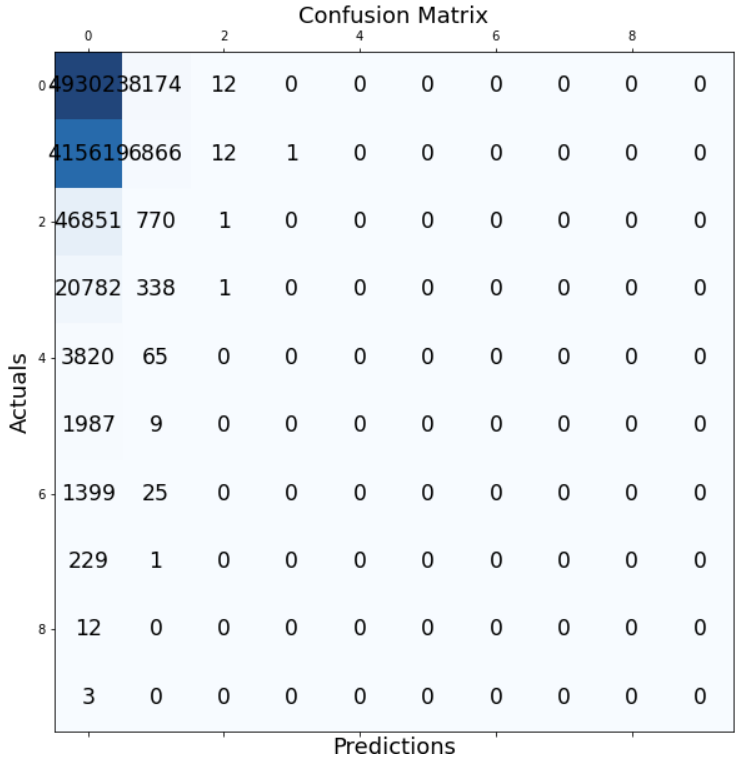


Hidden Layer = [20]
Train Accuracy : 0.50039
Test Accuracy : 0.50049
Final Error   : 1.424194628026504
epochs taken   : 1000
Stopping Value(for dff in 100 epochs err) : 1e-06
Time Taken to train 0:00:55.339872

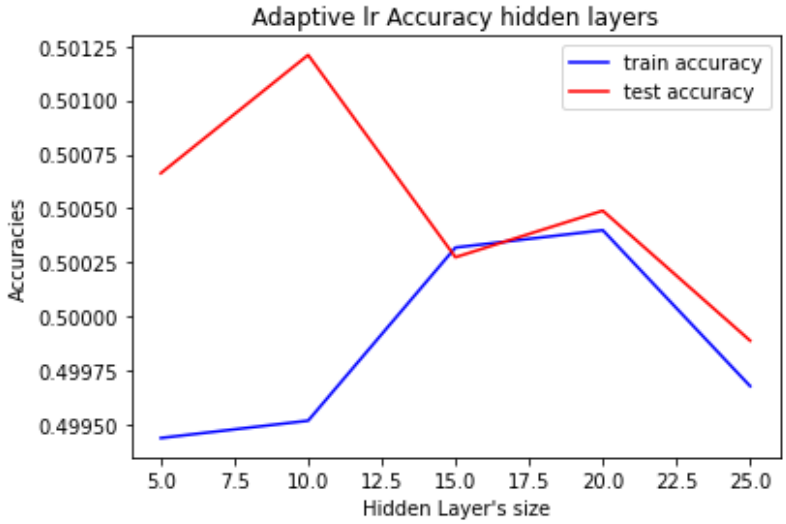
Confusion Matrix -



Hidden Layer = [25]
Train Accuracy : 0.49968
Test ACCURACY : 0.49989
Final Error   : 1.4252139046508765
epochs taken   : 1000
Stopping Value(for dff in 100 epochs err) : 1e-06
Time Taken to train 0:00:59.521151
Confusion Matrix-



Plot for accuracies based on
Hidden layers units - {5, 10, 15, 20, 25}



It is clear that the model can't train well with adaptive learning rate for learning rate = 0.1, it's the the model is stuck at a local minima and can't reach the global maximua

Part e) Using ReLu

Using ReLu with constant learning rate--

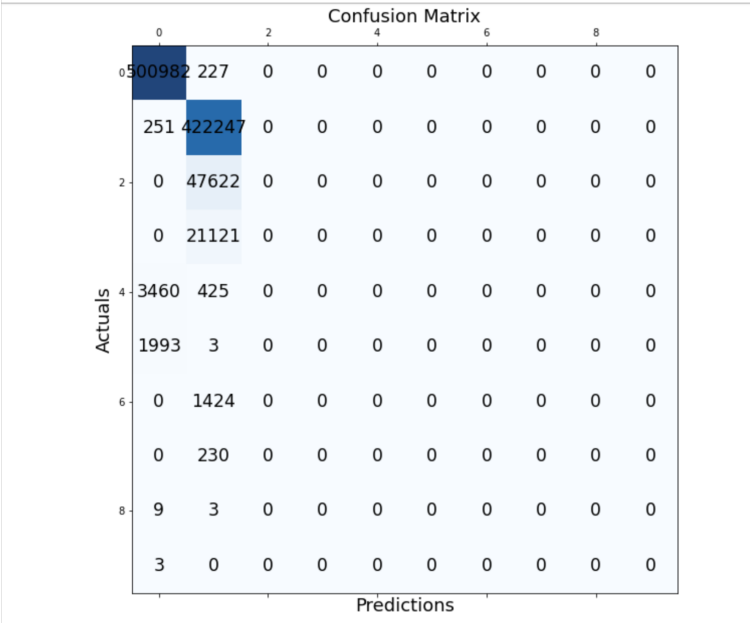
lr = 0.1

Normal learning rate (**6.36 iterations/sec**)

```
epoch 0     | error: 2.598367 | acc: 46.761295
epoch 40    | error: 1.727880 | acc: 71.031587
epoch 80    | error: 0.481530 | acc: 92.331068
epoch 120   | error: 0.427272 | acc: 92.331068
epoch 160   | error: 0.404499 | acc: 92.331068
No significant updates, EXITTING ----------------------
```
Final Accuracy : 0.9233106757297082
Final Error   : 0.4003308427625953
epochs taken   : 200
Stopping Value(for dff in 100 epochs err) : 0.003

For Test --
Accuracy = 92.3373%
Time taken = 31 sec

<u>Using ReLu along with adaptive learning rate</u>



time Taken : 06:24
Epochs taken = 1200
```
epoch 0     | error: 2.598367 | acc: 46.761295
epoch 200   | error: 2.048145 | acc: 56.653339
epoch 400   | error: 1.732724 | acc: 69.332267
epoch 600   | error: 1.162381 | acc: 84.598161
epoch 800   | error: 0.659460 | acc: 92.163135
epoch 1000  | error: 0.555209 | acc: 92.327069
epoch 1100  | error: 0.529087 | acc: 92.331068
epoch 1200  | error: 0.512360 | acc: 92.331068
```

For Test --

Accuracy = 92.1474%