Report

By: Anmol Agarwal

2018CS10327

Directory Structure:

Main directory contains Report, Q1 sub-directory and Q2 sub-directory.

Q1 directory contains 1a.py, 1c.py, and 1d.py.

Q2 directory contains 2a.py and 2f.py.

Some python programs are generic and hence used across multiple parts by passing corresponding arguments.

How to Run:

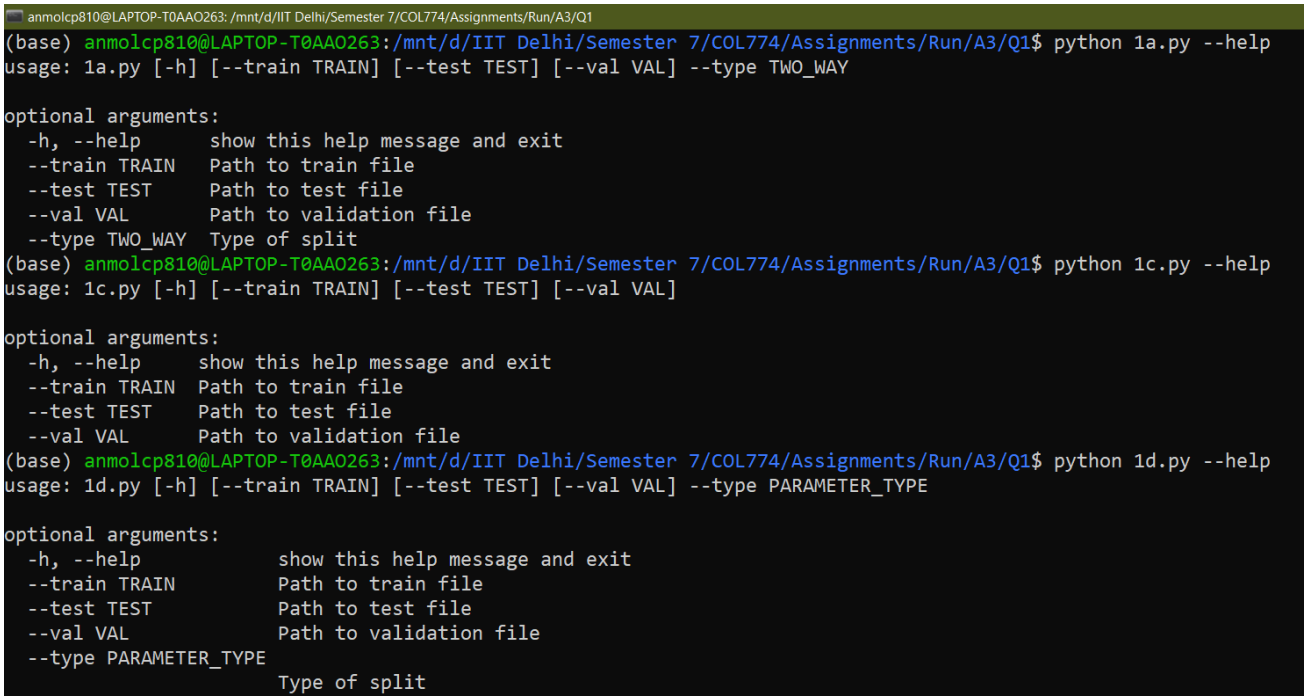
Please run “chmod +x run.sh” to make it executable.

Q1.

./run.sh 1 train_path test_path validation_path part

This run two programs in part a,b of type 0 (multi-way split) and type 1 (two-way split). Here “type” is another argument passed internally in run.sh.

More details of how to run: (shown in figure on next page)

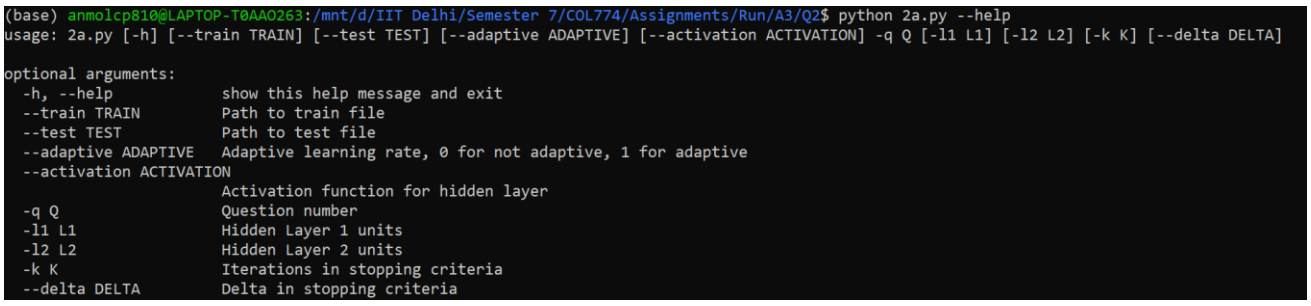


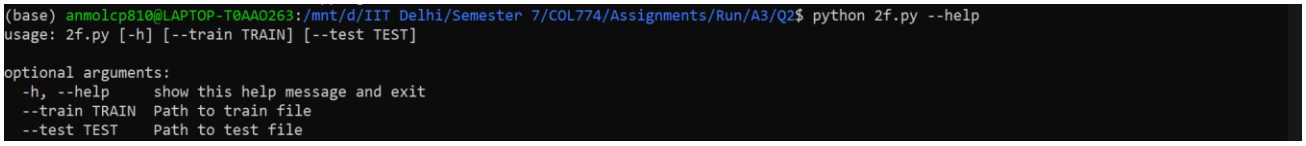
Q2.

./run.sh 2 train_path test_path

Part two has more arguments for each python file deciding adaptive learning rate, activation function, hidden layers, stopping criteria, etc.

More details of how to run :

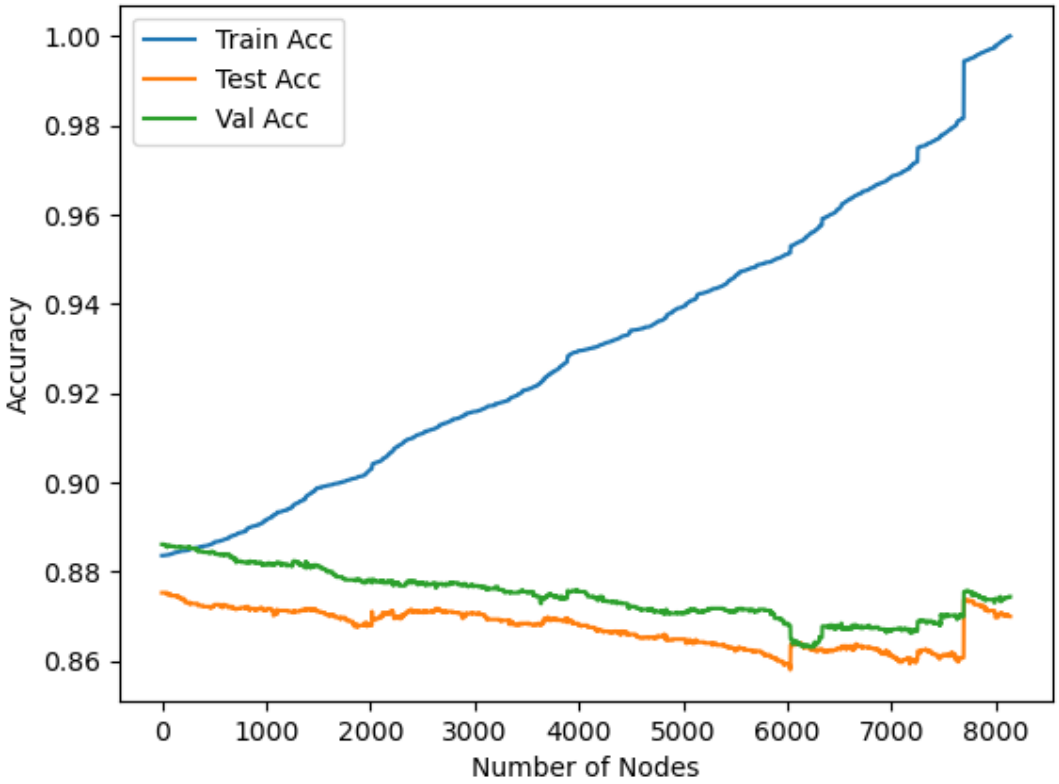




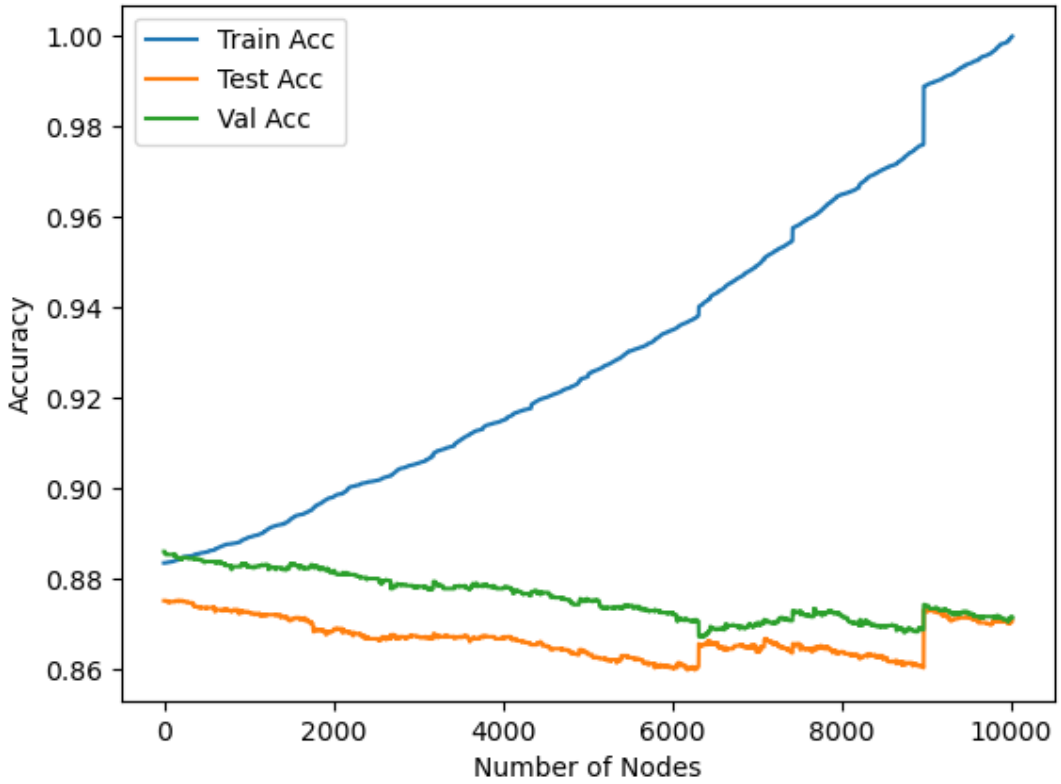
Description:

Q1

a. In part a of Q1(a), I used pd.get_dummies() to convert data to one-hot encoding format. Rest the approach to construct decision tree was same for both parts two-way and multi-way splits, i.e. choosing best attribute having highest mutual information.



Accuracy for two-way split



Accuracy for Multi-way Split

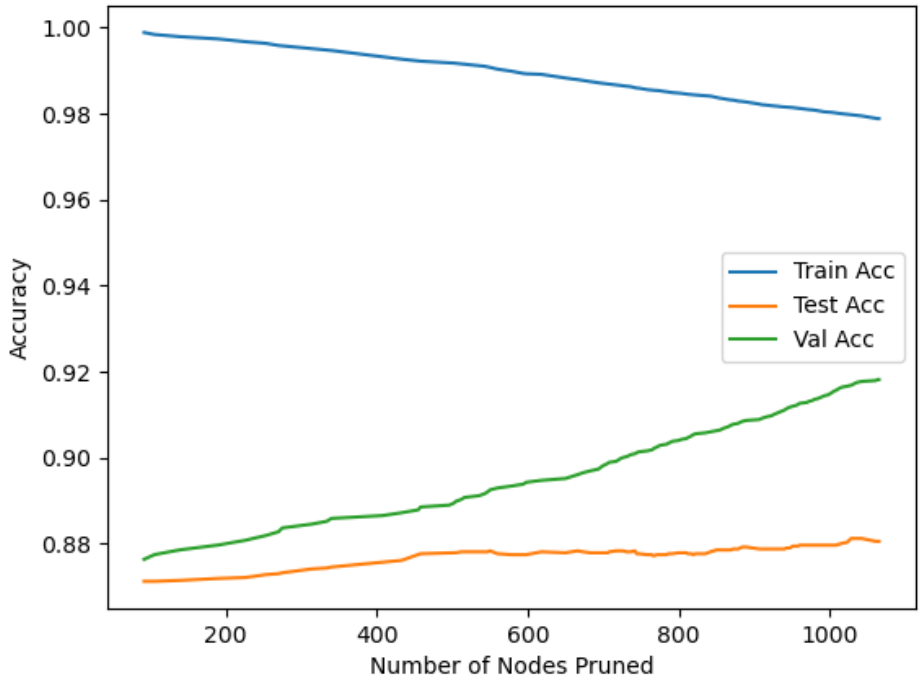
Observations:

1. Multi-way split expands more nodes (10015) compared to two-way split (which expanded around 8200 nodes).
2. Test/val accuracy was higher for two-way split. One reason might be that, model has become more fine-grained because we considering even categories as attributes, so we have more attributes. Hence the model is able to learn better.

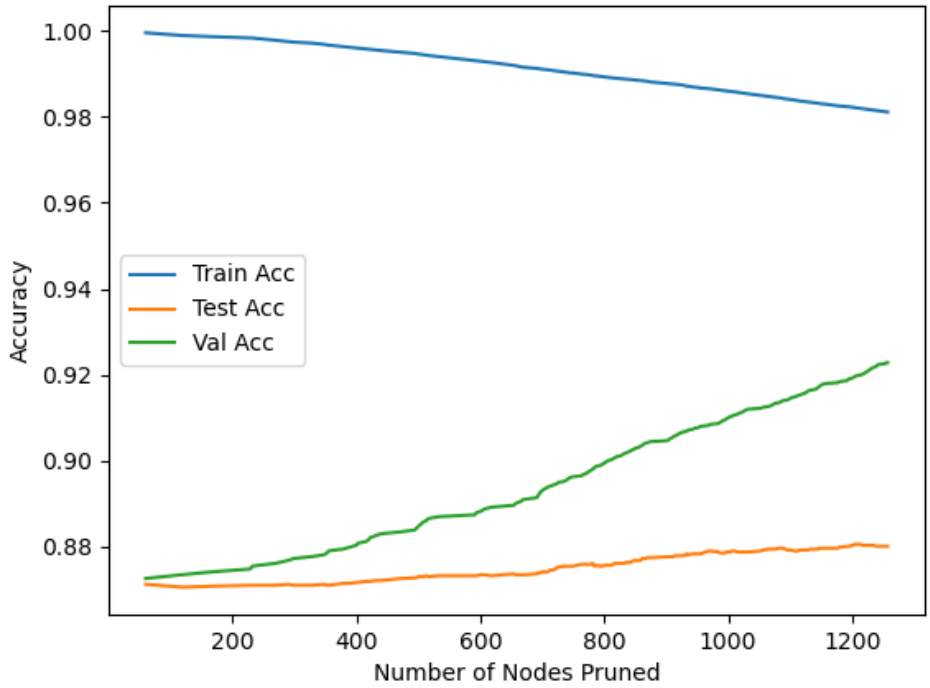
Note: When running part a, code also does part b by pruning the nodes as well.

b. I followed the bottom up approach to prune the nodes.
Approach: I maintained a dataframe of predictions on validation set, and whenever predicting on a node, I only changed those entries of dataframe which corresponded to part of dataset present at that node. This led to efficient

prediction in accuracy. And after prediction, I changed the entries to old values.



Two way split



Multi Way Split

Observations:

1. Train Accuracy decreases as more nodes are pruned while both validation and test accuracies increase.
2. The extent of increase in validation accuracy is higher than increase test accuracy.
3. Pruning does lead to improvement in general performance as seen in test set.

Note: When making programs for submission, I have used variable two_way = 0 or 1, if two_way is 0 then multi-way approach is used else two_way.

This is also reflected in png file names, i.e. accuracy_{two_way}.png.

c. I used sklearn.ensemble.RandomForestClassifier to train, and sklearn.model_selection.ParameterGrid to list out the parameters. And selected the best parameters with highest OOB score.
Best Parameters:
1. n_estimators: 350
2. max_features: 0.3
3. min_samples_split: 8

OOB Score: 0.9076255253262553

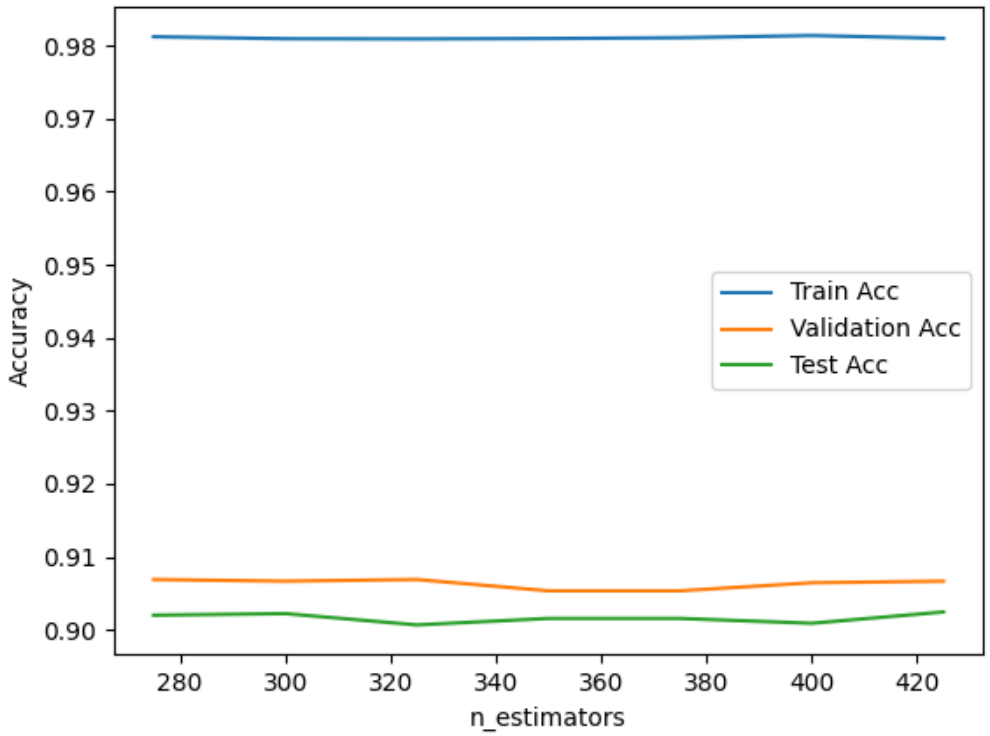
Train Accuracy: 0.9808947135589471

Validation Accuracy: 0.9062361786819991

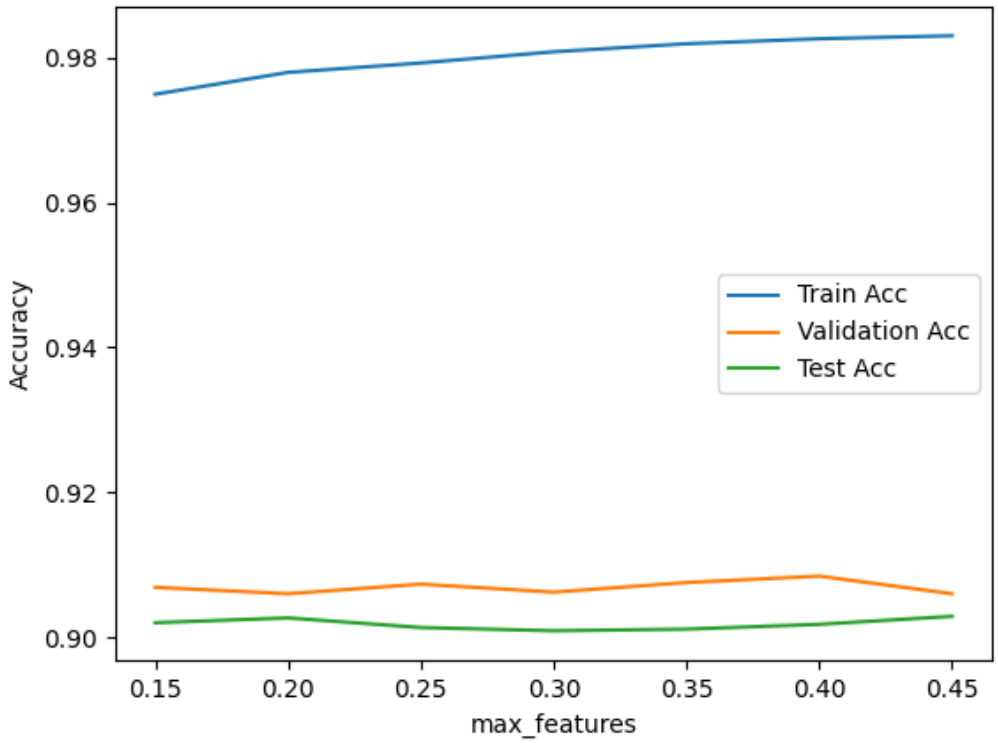
Test Accuracy: 0.9006856890068569

To handle categorical attributes, I simply converted to one-hot encoding.

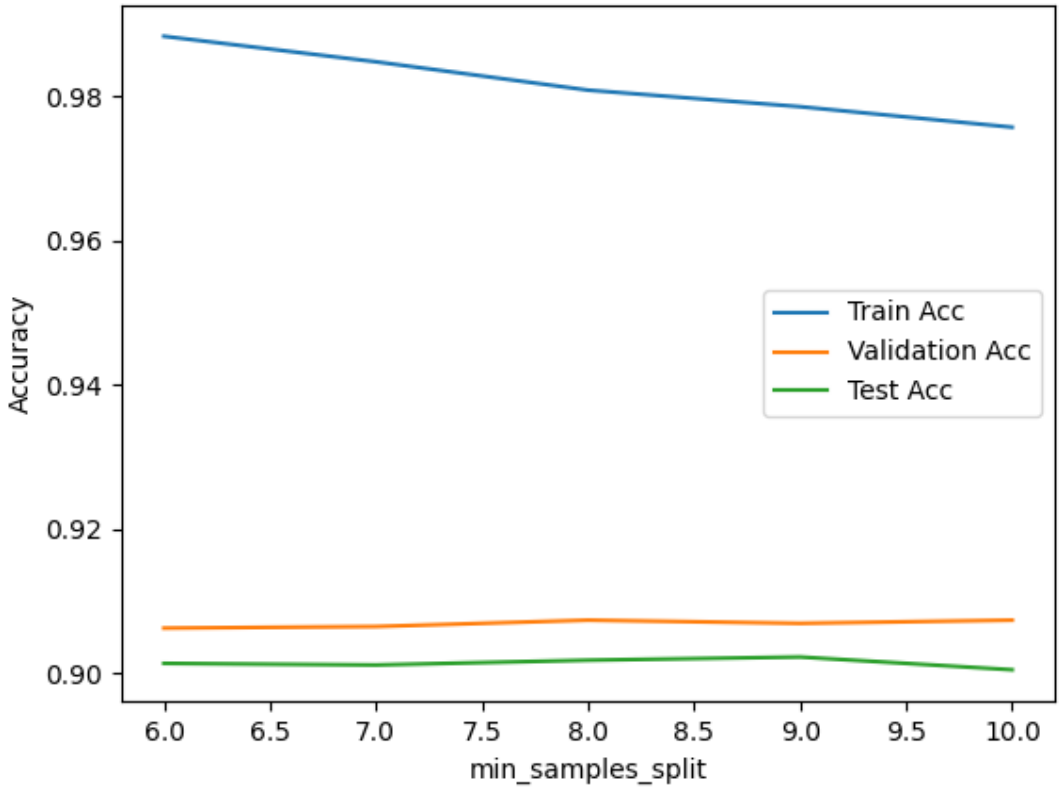
d. Variation in accuracies with varying values on one dimension are given below:



n_estimators



max_features



min_samples_split

Observations:

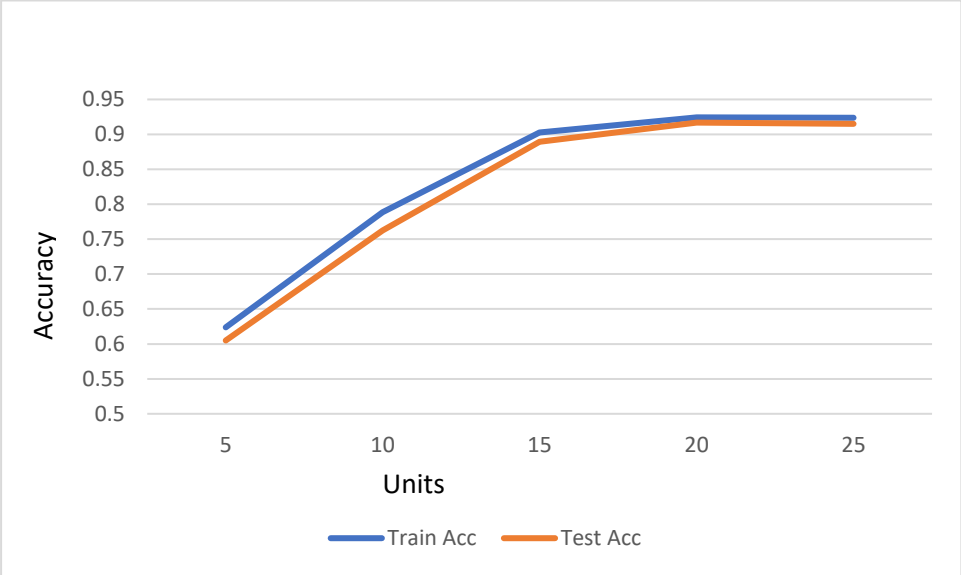
1. The models are not sensitive to varying values of parameters in terms of validation/test accuracies.
2. But training accuracy does get affect to some extent.
3. This is probably because, in random forest setting the model becomes robust to variations in values of the parameters and it also does not get affected by random noise in training data.

Q2.

a. I just ran pd.get_dummies() to convert training data to one-hot encoding. Moreover, for y-label, since the values were numeric, I converted it to string column and then ran pd.get_dummies()

b. I implemented the forward and backward propagation as was given in lecture notes. I stored netj and outj values for node j at some layer l and these values were used during backpropagation.
Moreover, I converted elementwise operations given in slides to matrix operations for efficiency.
And during backpropagation, I also stored the delta_j = -(dJ/dnetj) values and updated the parameters theta in one go after computing all the deltas.
Note: In part a and b, there was nothing to run, but only to implement, hence when parts a and b are run using run.sh, the code will actually run part c.

c. Below figure shows the test/train accuracy with number of units in hidden layer.



Observations:

1. Accuracy increases with number of units.

Stopping Criteria:

Hidden Layers	Iterations	Delta	lr	batch	Training Time	Epochs	Train Acc	Test Acc
5	30	1.00E-07	0.1	100	20.99798751	822	0.62375	0.60495
10	30	1.00E-07	0.1	100	290.4864535	7337	0.7888	0.76253
15	40	5.00E-08	0.1	100	265.8787086	6047	0.90248	0.88959
20	40	5.00E-08	0.1	100	799.2288294	16937	0.92435	0.91687
25	50	5.00E-08	0.1	100	519.6255953	10098	0.92395	0.91487

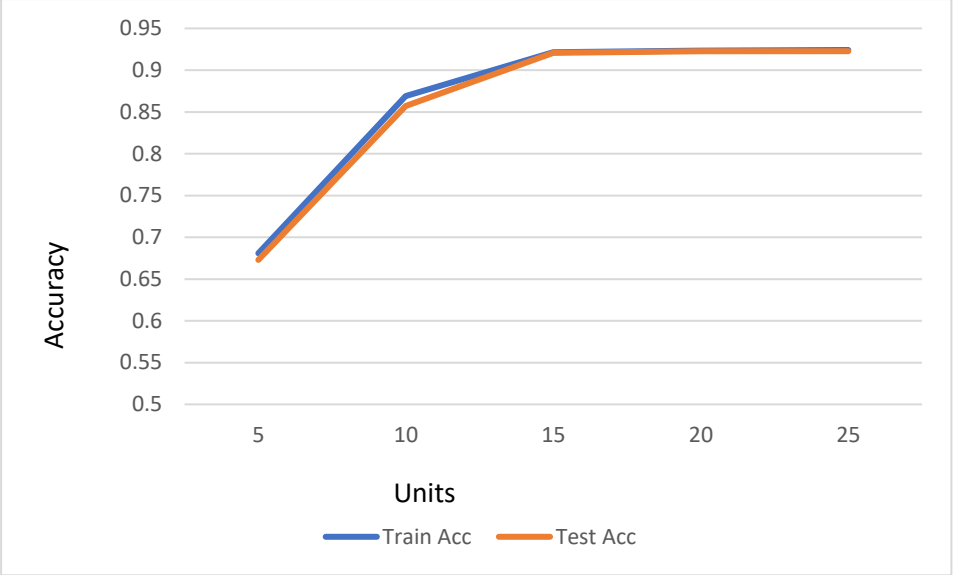
Confusion Matrix:

Please refer to files in Q2 directory named as confusion_matrix_test_{part}_{hidden_units}.txt

Example: confusion_matrix_test_c_5.txt

Note: When run.sh is called, the new files created will be named as confusion_matrix_test_{part}_{hidden_units}_{activation}.txt, for example confusion_matrix_test_e_100_relu.txt.
This has been done to prevent overwriting of earlier files.

d. Below figure shows the test/train accuracy with number of units in hidden layer with adaptive learning rate.



Observations:

1. Accuracy again increases with number of units.
2. The accuracy at given number of units is higher with adaptive learning compared to that in part c.

Stopping Criteria:

Hidden Layers	Iterations	Delta	lr	batch	Training Time	Epochs	Train Acc	Test Acc
5	30	1.00E-07	0.1	100	272.519	10629	0.68085	0.67293
10	30	1.00E-07	0.1	100	184.836	4778	0.86913	0.85718
15	40	5.00E-08	0.1	100	116.081	2737	0.92167	0.92084
20	40	5.00E-08	0.1	100	1066.26	22577	0.92359	0.92252
25	50	5.00E-08	0.1	100	1038.62	20216	0.92419	0.92272

Confusion Matrix:

Please refer to files in Q2 directory named as confusion_matrix_test_{part}_{hidden_units}.txt

Example: confusion_matrix_test_d_5.txt

Note: When run.sh is called, the new files created will be named as confusion_matrix_test_{part}_{hidden_units}_{activation}.txt, for example confusion_matrix_test_e_100_relu.txt.
This has been done to prevent overwriting of earlier files.

e. I had made a generic neural network which could accept any form of activation function and its derivation, so I just defined ReLU activation function and its derivative and used that inside train function.

Stopping Criteria:

Hidden Layers	Iterations	Delta	lr	batch	Training Time	Epochs	Train Acc	Test Acc	Activation
100, 100	50	4.00E-08	0.1	100	4603.97	23727	0.91879	0.91696	Sigmoid
100, 100	50	4.00E-08	0.1	100	0.04831	1	0.47401	0.47378	Relu

Confusion Matrix:

Please refer:

confusion_matrix_test_e_100_relu.txt

confusion_matrix_test_e_100_sigmoid.txt

Observations:

1. Large neural networks take longer to train and need stricter stopping criteria.
2. The accuracies could have been better if models were trained for longer and convergence criteria used was even stricter.

f. Training Accuracy: 1.0
Test Accuracy: 0.9894
Confusion Matrix:

[[501086 0 0 0 114 9 0 0 0 0]
[25 422472 1 0 0 0 0 0 0 0]
[616 470 46520 16 0 0 0 0 0 0]
[1250 2 346 19522 0 0 1 0 0 0]
[3717 0 0 0 168 0 0 0 0 0]
[1848 0 0 0 8 140 0 0 0 0]
[467 0 157 345 0 0 455 0 0 0]
[0 0 0 230 0 0 0 0 0 0]
[5 0 0 0 3 4 0 0 0 0]
[2 0 0 0 0 1 0 0 0 0]]

Using training library, the accuracies are much better than what I had implemented.
Reasons:

1. One reason might be using a different loss function (categorical cross entropy).
2. The implementation used beneath MLPclassifier must be more efficient because it took very less time compared to what I implemented.
3. They have used better adaptive learning rate, where they are also taking into account whether training loss is improving or not, and they then reduce the learning rate by a certain amount.
4. They only stop when learning rate is too small.