COL774
Machine Learning
Assignment 3

Rishi Jain
2020CS10373

October 31, 2022

Contents

Libraries Used

- numpy
- scipy
- matplotlib
- nltk
- sklearn
- sys
- os
- time
- xgboost
- lightgbm

1 Decision Trees, Random Forests and Gradient Boosted Trees

1.1 Mammographic mass lesion severity prediction

Dataset Description

Age, Shape, Margin, and Density of mammographic mass lesions made up the data, which a machine learning model was to be trained to categorise as benign or malignant based on. Some properties in the data had missing values, which were handled appropriately.

1.1.1 Decision tree classifier

The following outcomes were attained after training a scikit-learn decision tree on the training split.

- *Training Accuracy* : 92.32%
- *Validation Accuracy* : 75.3%
- *Test Accuracy* : 69.6%

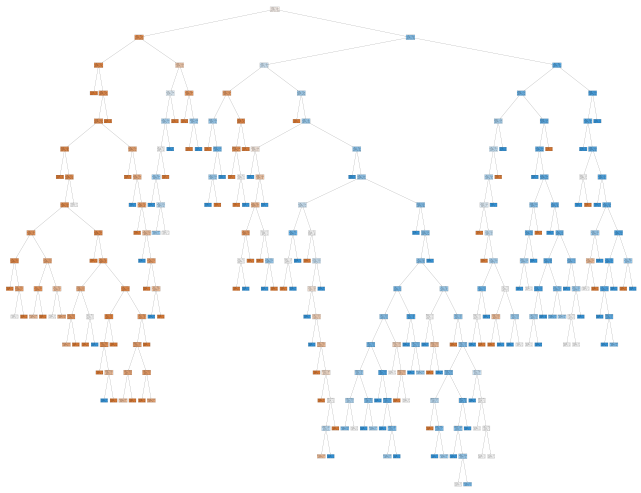


Figure 1: Decision Tree Classifier

The model clearly depicts an overfit on the training set, resulting in a high training accuracy but poor performance on the test set.

1.1.2 Decision tree parameters grid search

Grid search was performed using scikit-learn's *GridSearchCV* for the decision tree classifier on the hyperparameters `max_depth`, `min_samples_split` and `min_samples_leaf`.

Best estimator results

- `max_depth` : 3
- `min_samples_split` : 2
- `min_samples_leaf` : 12
- *Training Accuracy* : 80.92%
- *Validation Accuracy* : 89.26%
- *Test Accuracy* : 75.89%

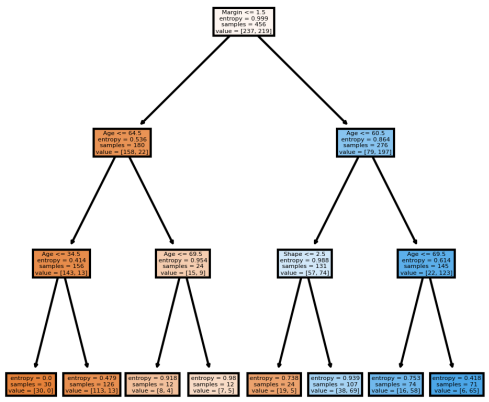


Figure 2: Decision Tree Classifier Grid Search

The above grid search has prevented the tree's depth from overfitting the training set, which has improved the tree's test performance.

1.1.3 Cost complexity pruning of decision tree

Scikit-learn pruning was used to extract a variety of ccp alphas parameters and corresponding impurities from the default decision tree. It was observed and plotted how ccp alphas affected the parameter variation.

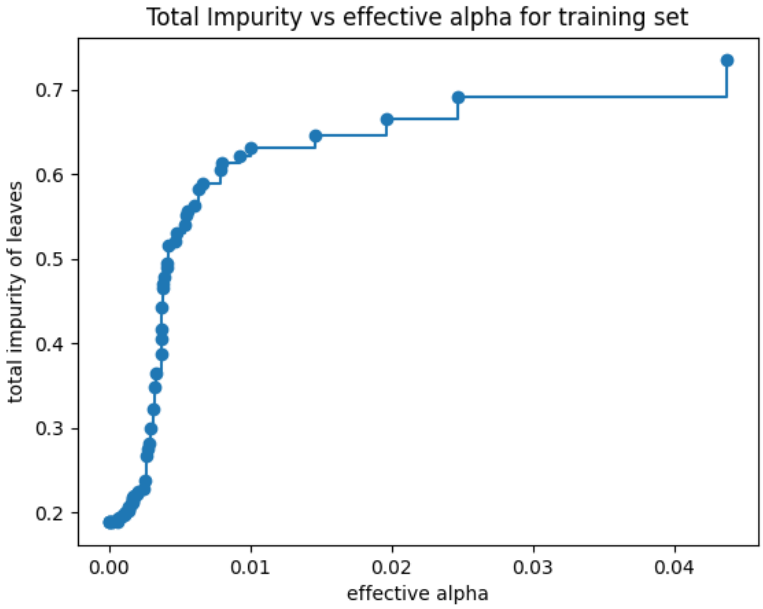


Figure 3: `ccp_alpha` vs. Total impurity on training data

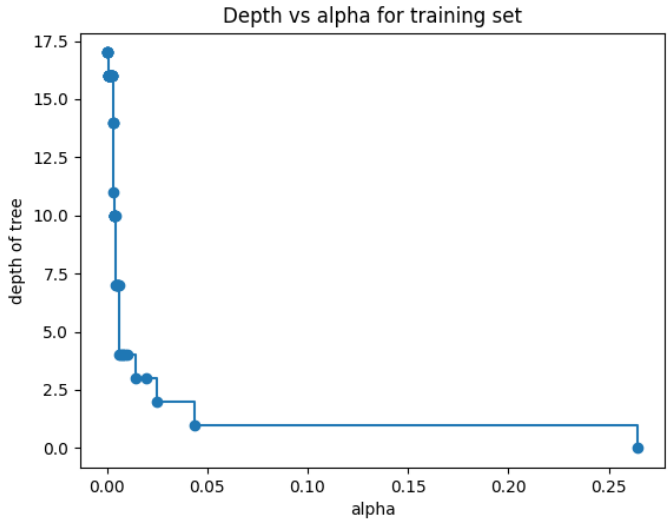


Figure 4: Variation of tree node count and depth with `ccp_alpha`

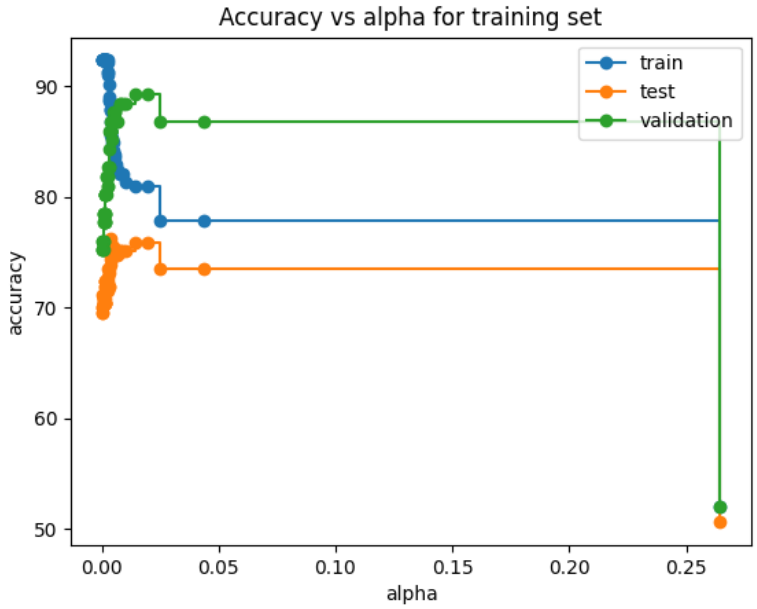


Figure 5: Variation of accuracy with `ccp_alpha`

The best classifier was selected by performance on validation set and the following results were obtained.

- `ccp_alpha` : 0.014555438724583387
- *Depth* : 3
- *Training Accuracy* : 80.92%
- *Validation Accuracy* : 89.26%
- *Test Accuracy* : 75.89%

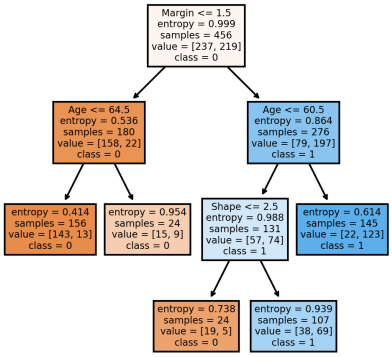


Figure 6: Optimally pruned decision tree

The model is prevented from overfitting by impurity-based pruning of the tree, which lowers training accuracy while raising test set performance. The figure above illustrates the impact of pruning on the validation/test set performance. The model starts to underfit at larger alpha values, which lowers all three accuracies. The model has the same metrics as a grid search-constructed tree and is undoubtedly superior to a simple tree.

1.1.4 Random forest

For the optimal random forest model, a grid search was performed on the hyperparameters `n_estimators`, `min_samples_split` and `max_features`.

Best estimator results

- `n_estimators` : 50
- `max_features` : 0.3
- `min_samples_split` : 26
- *Training Accuracy* : 82.89%
- *Out-of-bag Accuracy* : 80.92%
- *Validation Accuracy* : 88.43%
- *Test Accuracy* : 77.07%

The model outperforms the decision trees mentioned before due to the robustness of random forest and decrease in variance.

1.1.5 Missing Data Imputation

Imputation was performed on the training data using scikit-learn's simple imputer to fill up the missing data based on median and mode.

Imputation by median

Default decision tree :

- *Training Accuracy* : 91.81%
- *Validation Accuracy* : 74.81%
- *Test Accuracy* : 72.57%

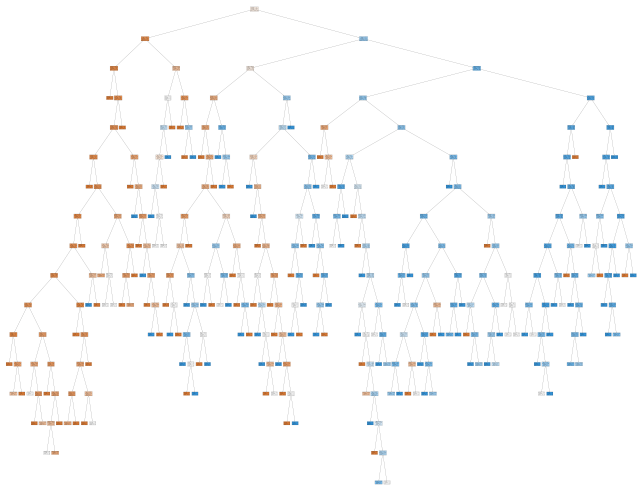


Figure 7: Decision Tree Classifier

Decision tree grid search :

Best estimator results

- `max_depth` : 4
- `min_samples_split` : 2
- `min_samples_leaf` : 12
- *Training Accuracy* : 80.266%
- *Validation Accuracy* : 87.41%
- *Test Accuracy* : 79.1%

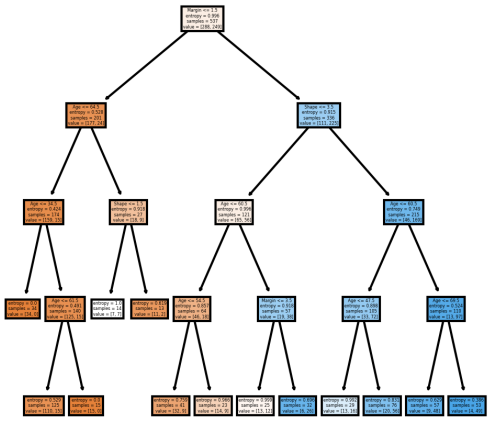


Figure 8: Decision Tree Classifier Grid Search

Cost complexity pruning of default decision tree :

Best estimator results

- `ccp_alpha` : 0.00637979111220045
- *Training Accuracy* : 82.12%
- *Validation Accuracy* : 87.4%
- *Test Accuracy* : 80.2%

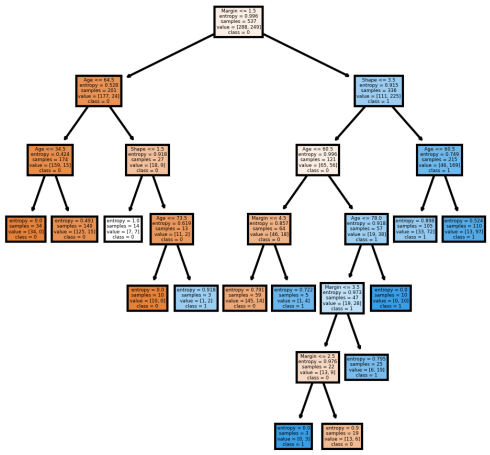


Figure 9: Optimally pruned decision tree

Random forest classifier grid search :

Best estimator results

- `n_estimators` : 50
- `max_features` : 0.9
- `min_samples_split` : 28
- *Training Accuracy* : 81.4%
- *Out-of-bag Accuracy* : 79.3%
- *Validation Accuracy* : 85.2%
- *Test Accuracy* : 80.55%

The aforementioned results demonstrate that imputation by median allows the model to learn more about the data's trend and, as a result, performs noticeably better than when these data points are ignored.

Imputation by mode

Default decision tree :

- *Training Accuracy* : 90.6890%
- *Validation Accuracy* : 75.55%
- *Test Accuracy* : 71.87%

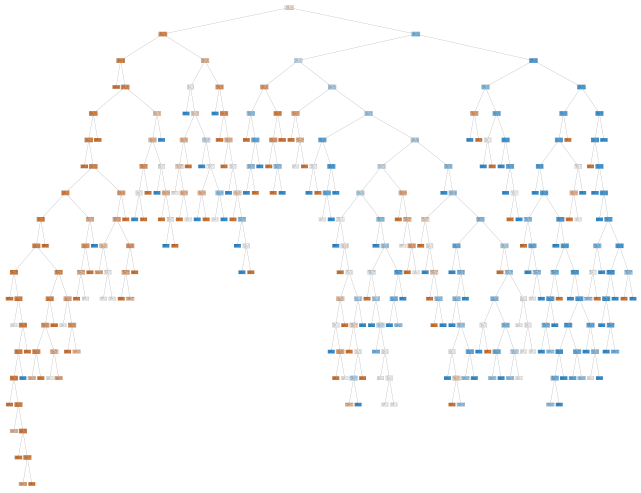


Figure 10: Decision Tree Classifier

Decision tree grid search :

Best estimator results

- `max_depth` : 6
- `min_samples_split` : 6
- `min_samples_leaf` : 1
- *Training Accuracy* : 83.05%
- *Validation Accuracy* : 87.40%
- *Test Accuracy* : 75.69%

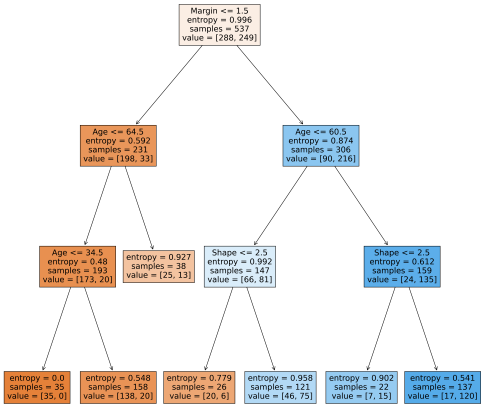


Figure 11: Decision Tree Classifier Grid Search

Cost complexity pruning of default decision tree :

Best estimator results

- `ccp_alpha` : 0.0032
- *Training Accuracy* : 82.87%
- *Validation Accuracy* : 86.67%
- *Test Accuracy* : 76.05%

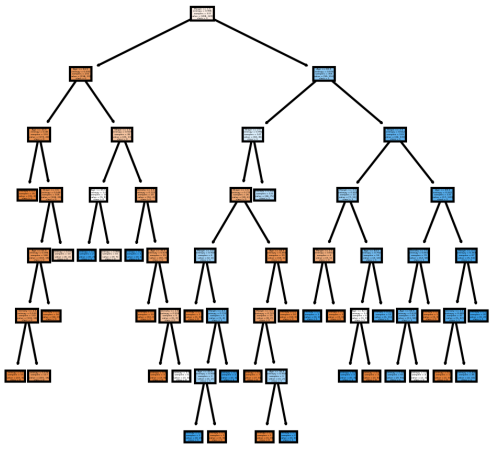


Figure 12: Optimally pruned decision tree

Random forest classifier grid search :
Best estimator results

- `n_estimators` : 150
- `max_features` : 0.3
- `min_samples_split` : 30
- *Training Accuracy* : 81.93%
- *Out-of-bag Accuracy* : 79.70%
- *Validation Accuracy* : 86.67%
- *Test Accuracy* : 79.51%

Once more, imputation allows the model to acquire additional insight into the data and outperforms ignoring the data points with missing attributes. With the exception of the random forest, when the mode imputation triumphs, all scenarios yield higher performance on the test set when the median imputation is used.

1.1.6 Gradient boosted trees : XGBoost

The xgboost package was used to build extreme gradient boosting-based estimators, and grid search was used to fine-tune their hyperparameters.
Best estimator results :

- `n_estimators` : 10
- `max_depth` : 4

- `subsample` : 0.3
- *Training Accuracy* : 82.46%
- *Validation Accuracy* : 88.43%
- *Test Accuracy* : 75.1%

1.2 Drug rating prediction

Dataset Description

The information was divided into two text attributes—a drug review and the patient's matching condition—and three numerical attributes—date and useful count. In order to forecast the user rating for the medicine, a machine learning model was to be trained. Scikit-numerical Learn's data conversion tool was used to turn the text properties into *CountVectorizer* after removal of stop-words.

1.2.1 Decision tree classifier

The following outcomes were attained after training a scikit-learn decision tree on the training split.

- *Training Accuracy* : 100%
- *Validation Accuracy* : 57.27%
- *Test Accuracy* : 57.04%
- *Training time* : 169.93s

There is no doubt that the training data were significantly overfit, which causes the test and validation accuracy to be low.

1.2.2 Decision tree parameters grid search

Grid search for the decision tree classifier on the hyperparameters was done manually by varying `max_depth`, `min_samples_split` and `min_samples_leaf`.
Best estimator results

- `max_depth` : 115
- `min_samples_split` : 2
- `min_samples_leaf` : 1
- *Training Accuracy* : 100%
- *Validation Accuracy* : 57.3%
- *Test Accuracy* : 57.08%
- *Train Time* : 180 s

Grid search on these hyperparameters had no impact on the decision tree that was produced. This implies that using just one decision tree might be the best option available.

1.2.3 Cost complexity pruning of decision tree

Scikit-learn pruning was used to extract a variety of ccp alphas parameters and corresponding impurities from the default decision tree. It was observed and plotted how ccp alphas affected the parameter variation. For analysis, the final ccp alphas value corresponding to a simple single-node tree was disregarded. For the selection of the best model, every 50^{th} value in `ccp_alpha` was used to train and validate considering the huge number of values. Also the proceess is very time consuming so only some values of alphas are taken from 32000 values provided by scikit learn.

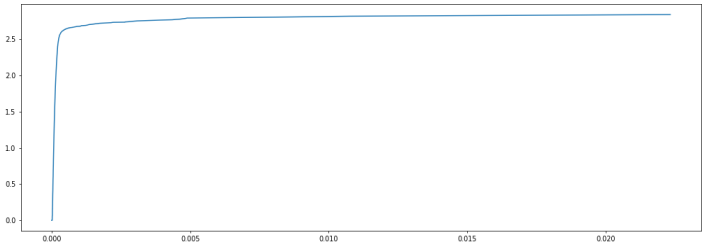


Figure 13: `ccp_alpha` vs. Total impurity on training data

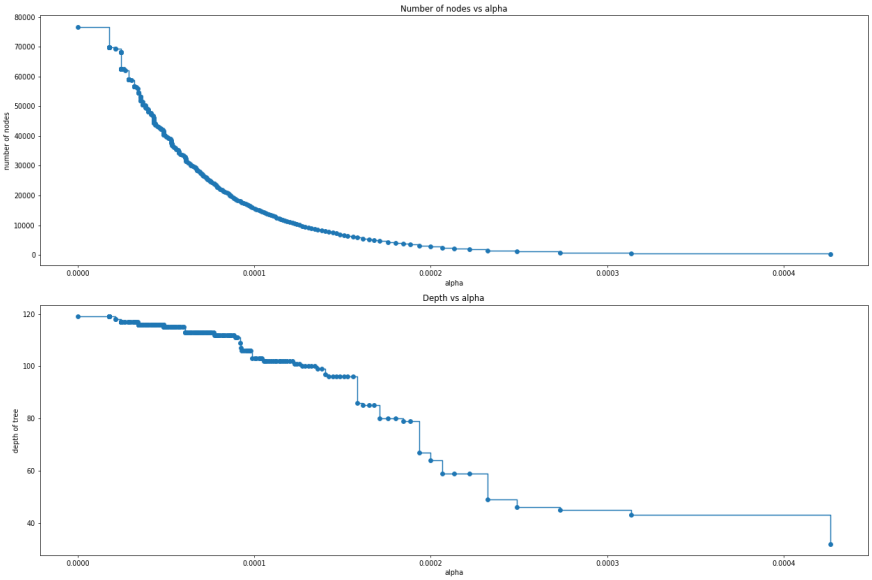


Figure 14: Variation of tree node count and depth with `ccp_alpha`

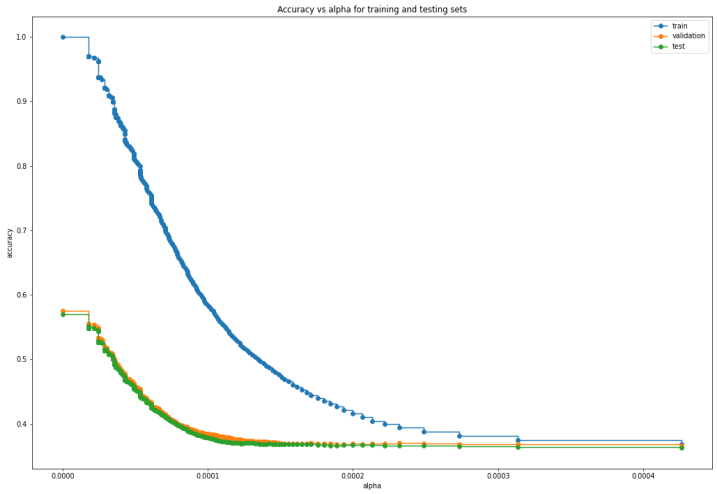


Figure 15: Variation of accuracy with `ccp_alpha`

The best classifier was selected by performance on validation set and the following results were obtained.

- `ccp_alpha` : 0.0
- *Training Accuracy* : 100.0%
- *Validation Accuracy* : 57.27%
- *Test Accuracy* : 57.04%
- *Training time* : 169.69s

The results above allow us to deduce that the decision tree's performance is very sensitive to depth, with training,validation and test sets all following the same trend in view of depth of the tree and a monotonic decrease is observed in all three accuracies with pruning.

1.2.4 Random forest

For the optimal random forest model, a grid search was performed on the hyperparameters `n_estimators`, `min_samples_split` and `max_features`.

Best estimator results

- `n_estimators` : 450
- `max_features` : 0.8
- `min_samples_split` : 2
- *Training Accuracy* : 100.0%

- *Out-of-bag Accuracy* : 64.61%
- *Validation Accuracy* : 64.28%
- *Test Accuracy* : 64.4%
- *Training time* : 928.61s

When using a random forest instead of a single decision tree, we see a considerable improvement in test accuracy since random forests have less volatility and are therefore more robust. The training accuracy still shows overfitting, which gradient boosting may help to reduce.

1.2.5 Gradient boosted trees : XGBoost

The xgboost package was used to build extreme gradient boosting-based estimators, and grid search was used to fine-tune their hyperparameters.

Best estimator results :

- `n_estimators` : 450
- `max_depth` : 40
- `subsample` : 0.4
- *Training Accuracy* : 76.4667%
- *Validation Accuracy* : 53.4502%
- *Test Accuracy* : 53.0521%
- *Training time* : 631.52s

The decrease in training accuracy in gradient-boosted trees in this instance strongly suggests that there aren't enough trees on the provided grid for grid search. The varied parameters are (`n_estimators`).

1.2.6 Gradient boosted machines : LightGBM

Gradient boosted machines based estimators were constructued using the *lightgbm* package and grid search was performed to tune it's hyperparameters.
Best estimator parameters :

Best estimator results :

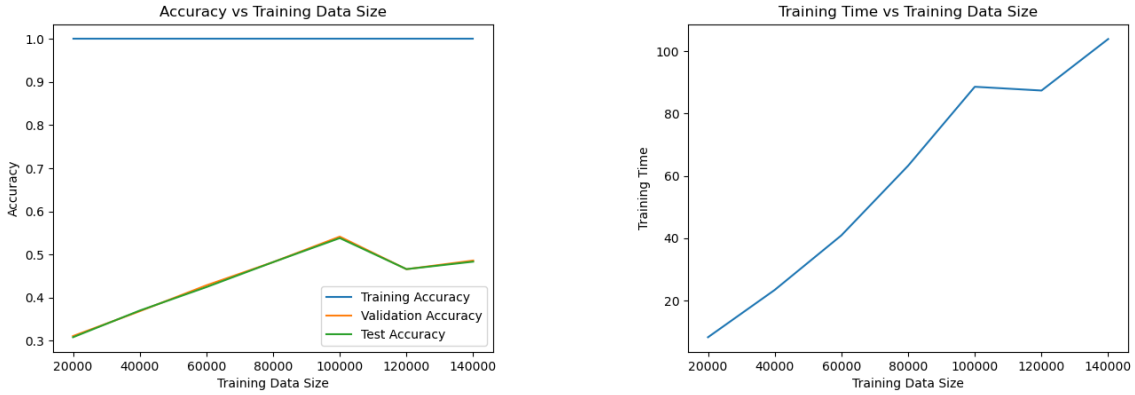
- `n_estimators` : 2000
- `max_depth` : 40
- `subsample` : 0.4
- *Training Accuracy* : 97.36%
- *Validation Accuracy* : 64.52%

- *Test Accuracy* : 64.54%
- *Training time* : 334.19s

Owing to the scalability to large datasets of gradient boosted machines, LightGBM perferms the best among all the previous models, with a visible reduction in overfit on training set when compared to the random forest model in section 1.2.4.
The decision tree models train obviously faster than the other models outlined above, with pruning and parameter fixing requiring almost the same amounts of time while taking somewhat longer than the default decision tree. The multi-estimator models that perform the slowest with parameter fixing are random forest and XGBoost, respectively. LightGBM performs better on the test data and achieves training times that are comparable to those of a decision tree.

1.2.7 Training with varying amount of data

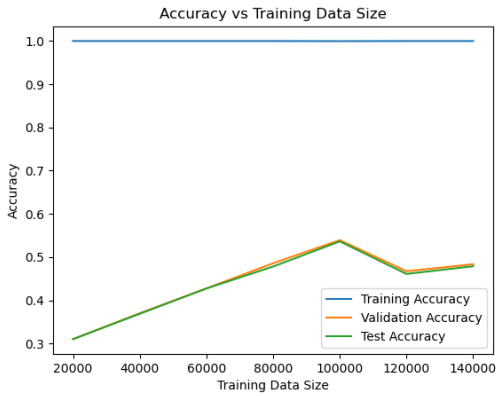
The models previously mentioned were trained using various sizes of training data, each of which was created by randomly selecting data points from the training set for the necessary size (with replacement for sizes exceeding the available training data). The following variations in training time and accuracy were noted.



(a) Accuracy variance

(b) Training time variance

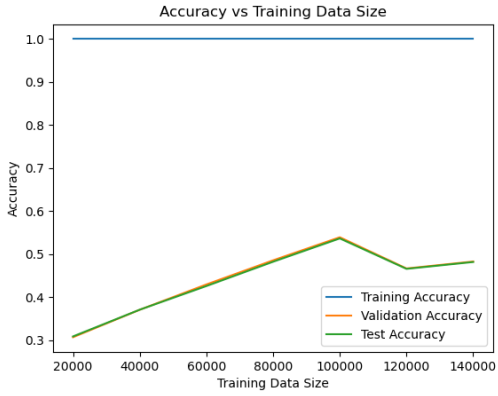
Figure 16: Decision tree classifier



(a) Accuracy variance

(b) Training time variance

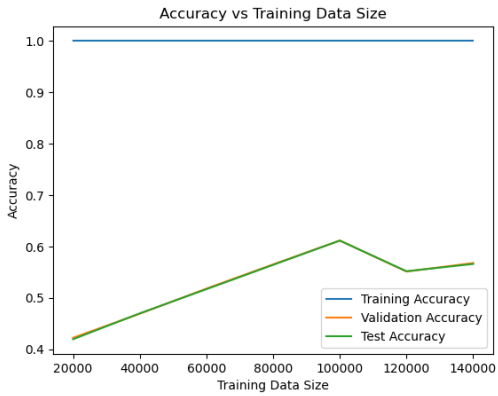
Figure 17: Decision tree classifier parameter grid search



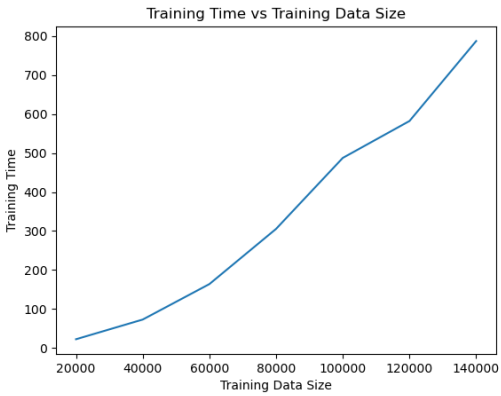
(a) Accuracy variance

(b) Training time variance

Figure 18: Decision tree classifier cost complexity pruning

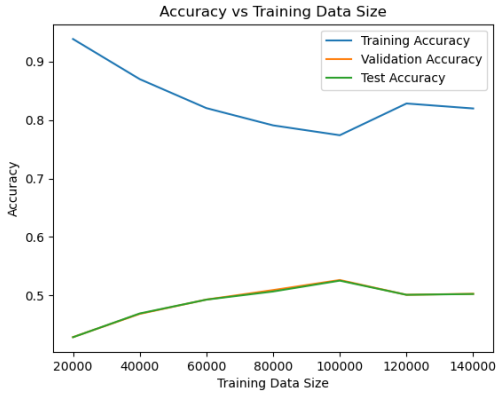


(a) Accuracy variance

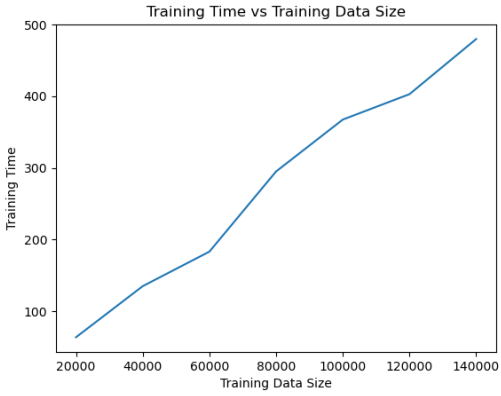


(b) Training time variance

Figure 19: Random forest classifier grid search

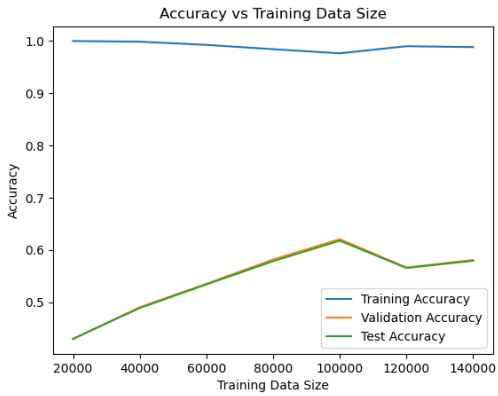


(a) Accuracy variance

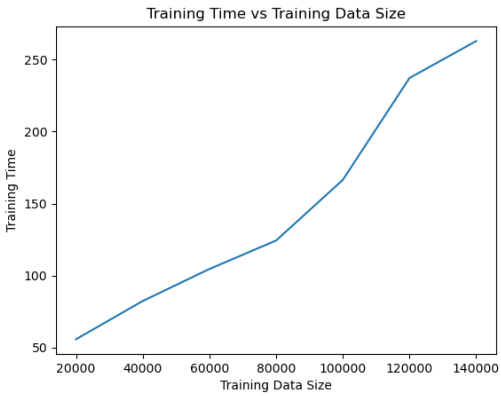


(b) Training time variance

Figure 20: XGBoost classifier grid search



(a) Accuracy variance



(b) Training time variance

Figure 21: LightGBM classifier grid search

With the aforementioned facts, it is obvious that the models with the mentioned sampling exhibit the same pattern. Generally speaking, every model performs better on the test data as it receives more information with the amount of data. The obvious trade-off is the length of time required to train on huge datasets, which reduces it to the processing power a person possesses.

2 Neural Networks

Dataset description

The Fasion MNIST data consisted of 70000 28×28 images flattened and split into training and test sets. I constructed a neural network for the classification as follows

2.1 Neural network backend

- To make it easier to train and forecast neural networks using the chosen architecture, activation function (sigmoid/ReLU), learning rate, batch size, and number of classes, a python function was developed.
- In the case of sigmoid activation, Xavier's initialization was used, and He's initialization was used in the case of ReLU activation.
- With the appropriate base/starting value, the learning rate could be either constant or adaptive.
- Stochastic gradient descent is used in the training phase to do backpropagation on the objective/loss function, which could either be mean square error or binary cross entropy error.
- A universal stopping criterion was kept to be the minimum change in loss(MSE/BCE) averaged over an epoch as 10^{-4} and a cap on number of epochs as 100, whichever happens first.

2.2 Single hidden layer network with sigmoid activation

Neural networks with single hidden layer were trained and tested for different number of hidden neurons.

5 hidden layer units

- *Training accuracy* : 84.3%
- *Test accuracy* : 81.7%
- *Training time* : 63.8s

- *Test data confusion matrix* :

$$\text{confusion matrix} = \begin{pmatrix} 946.0 & 1.0 & 17.0 & 0.0 & 1.0 & 2.0 & 0.0 & 1.0 & 0.0 & 5.0 \\ 6.0 & 710.0 & 18.0 & 136.0 & 1.0 & 167.0 & 0.0 & 2.0 & 0.0 & 11.0 \\ 27.0 & 16.0 & 811.0 & 27.0 & 0.0 & 43.0 & 0.0 & 3.0 & 0.0 & 68.0 \\ 4.0 & 128.0 & 44.0 & 743.0 & 0.0 & 99.0 & 0.0 & 15.0 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 908.0 & 0.0 & 29.0 & 3.0 & 10.0 & 1.0 \\ 10.0 & 115.0 & 35.0 & 54.0 & 2.0 & 469.0 & 0.0 & 28.0 & 2.0 & 94.0 \\ 0.0 & 0.0 & 1.0 & 1.0 & 55.0 & 1.0 & 941.0 & 8.0 & 56.0 & 1.0 \\ 1.0 & 9.0 & 7.0 & 36.0 & 6.0 & 56.0 & 1.0 & 926.0 & 0.0 & 18.0 \\ 4.0 & 1.0 & 1.0 & 0.0 & 26.0 & 0.0 & 29.0 & 0.0 & 931.0 & 1.0 \\ 2.0 & 20.0 & 66.0 & 2.0 & 1.0 & 163.0 & 0.0 & 14.0 & 0.0 & 800.0 \end{pmatrix}$$

10 hidden layer units

- *Training accuracy* : 87%
- *Test accuracy* : 84.24%
- *Training time* : 55.9s

- *Test data confusion matrix* :

$$\text{confusion matrix} = \begin{pmatrix} 950.0 & 1.0 & 13.0 & 2.0 & 0.0 & 2.0 & 0.0 & 1.0 & 0.0 & 3.0 \\ 2.0 & 758.0 & 15.0 & 95.0 & 0.0 & 108.0 & 0.0 & 5.0 & 0.0 & 18.0 \\ 30.0 & 11.0 & 857.0 & 40.0 & 1.0 & 44.0 & 0.0 & 4.0 & 0.0 & 42.0 \\ 7.0 & 126.0 & 35.0 & 771.0 & 0.0 & 99.0 & 0.0 & 5.0 & 0.0 & 8.0 \\ 0.0 & 0.0 & 3.0 & 1.0 & 908.0 & 1.0 & 31.0 & 7.0 & 12.0 & 1.0 \\ 5.0 & 73.0 & 39.0 & 83.0 & 0.0 & 551.0 & 0.0 & 13.0 & 0.0 & 76.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 53.0 & 0.0 & 929.0 & 5.0 & 37.0 & 0.0 \\ 1.0 & 9.0 & 6.0 & 7.0 & 7.0 & 22.0 & 1.0 & 956.0 & 1.0 & 11.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 31.0 & 0.0 & 39.0 & 0.0 & 950.0 & 0.0 \\ 5.0 & 22.0 & 32.0 & 1.0 & 0.0 & 173.0 & 0.0 & 4.0 & 0.0 & 841.0 \end{pmatrix}$$

15 hidden layer units

- *Training accuracy* : 88.6%
- *Test accuracy* : 86.1%
- *Training time* : 66.35s

- *Test data confusion matrix* :

$$\text{confusion matrix} = \begin{pmatrix} 963.0 & 3.0 & 11.0 & 3.0 & 0.0 & 1.0 & 0.0 & 1.0 & 0.0 & 1.0 \\ 2.0 & 766.0 & 11.0 & 90.0 & 0.0 & 99.0 & 0.0 & 2.0 & 1.0 & 13.0 \\ 22.0 & 15.0 & 874.0 & 37.0 & 3.0 & 38.0 & 0.0 & 10.0 & 0.0 & 41.0 \\ 6.0 & 117.0 & 39.0 & 790.0 & 0.0 & 98.0 & 0.0 & 3.0 & 0.0 & 4.0 \\ 0.0 & 2.0 & 2.0 & 0.0 & 927.0 & 0.0 & 26.0 & 4.0 & 8.0 & 4.0 \\ 2.0 & 74.0 & 33.0 & 71.0 & 0.0 & 609.0 & 0.0 & 10.0 & 0.0 & 97.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 41.0 & 1.0 & 939.0 & 4.0 & 35.0 & 1.0 \\ 2.0 & 9.0 & 4.0 & 9.0 & 5.0 & 21.0 & 1.0 & 963.0 & 1.0 & 11.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 24.0 & 0.0 & 34.0 & 0.0 & 955.0 & 0.0 \\ 3.0 & 13.0 & 26.0 & 0.0 & 0.0 & 133.0 & 0.0 & 3.0 & 0.0 & 828.0 \end{pmatrix}$$

20 hidden layer units

- *Training accuracy* : 88.6%
- *Test accuracy* : 86.4%
- *Training time* : 70.2s
- *Test data confusion matrix* :

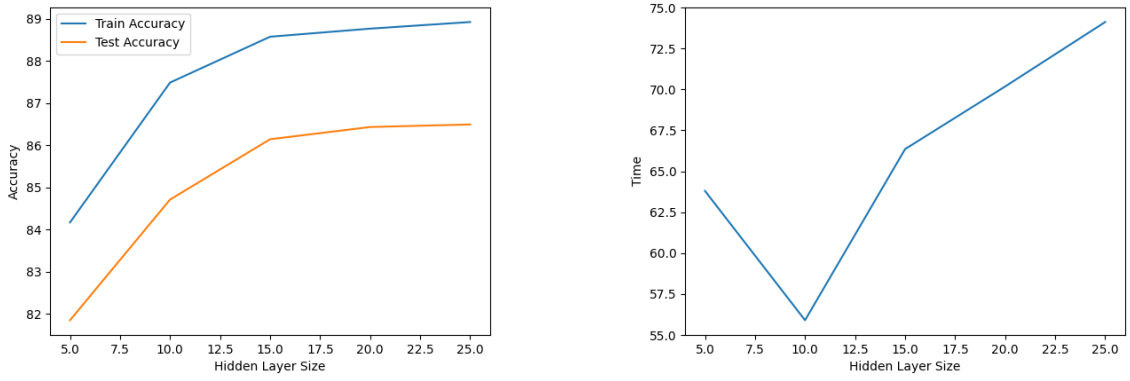
$$\text{confusion matrix} = \begin{pmatrix} 963.0 & 2.0 & 12.0 & 2.0 & 0.0 & 1.0 & 0.0 & 1.0 & 0.0 & 4.0 \\ 2.0 & 784.0 & 11.0 & 95.0 & 0.0 & 109.0 & 0.0 & 5.0 & 0.0 & 10.0 \\ 22.0 & 14.0 & 875.0 & 36.0 & 1.0 & 32.0 & 0.0 & 9.0 & 0.0 & 39.0 \\ 4.0 & 112.0 & 33.0 & 783.0 & 0.0 & 83.0 & 0.0 & 4.0 & 0.0 & 3.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 934.0 & 0.0 & 30.0 & 2.0 & 6.0 & 1.0 \\ 4.0 & 57.0 & 31.0 & 76.0 & 0.0 & 602.0 & 0.0 & 10.0 & 0.0 & 75.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 39.0 & 0.0 & 934.0 & 4.0 & 37.0 & 0.0 \\ 1.0 & 9.0 & 5.0 & 7.0 & 2.0 & 19.0 & 1.0 & 964.0 & 1.0 & 20.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 24.0 & 0.0 & 35.0 & 0.0 & 956.0 & 0.0 \\ 4.0 & 21.0 & 32.0 & 0.0 & 0.0 & 154.0 & 0.0 & 1.0 & 0.0 & 848.0 \end{pmatrix}$$

25 hidden layer units

- *Training accuracy* : 88.9%
- *Test accuracy* : 86.5%
- *Training time* : 74.5

- *Test data confusion matrix* :

$$\text{confusion matrix} = \begin{pmatrix} 960.0 & 3.0 & 12.0 & 1.0 & 0.0 & 1.0 & 0.0 & 1.0 & 0.0 & 2.0 \\ 1.0 & 780.0 & 9.0 & 91.0 & 0.0 & 102.0 & 0.0 & 11.0 & 0.0 & 18.0 \\ 23.0 & 10.0 & 888.0 & 33.0 & 2.0 & 40.0 & 0.0 & 6.0 & 0.0 & 41.0 \\ 6.0 & 114.0 & 34.0 & 795.0 & 0.0 & 88.0 & 0.0 & 4.0 & 0.0 & 5.0 \\ 0.0 & 1.0 & 0.0 & 1.0 & 933.0 & 0.0 & 32.0 & 2.0 & 8.0 & 1.0 \\ 6.0 & 68.0 & 28.0 & 72.0 & 0.0 & 610.0 & 0.0 & 11.0 & 0.0 & 87.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 39.0 & 0.0 & 934.0 & 5.0 & 34.0 & 0.0 \\ 1.0 & 5.0 & 4.0 & 7.0 & 2.0 & 21.0 & 0.0 & 959.0 & 1.0 & 13.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 24.0 & 0.0 & 34.0 & 0.0 & 957.0 & 0.0 \\ 3.0 & 18.0 & 25.0 & 0.0 & 0.0 & 138.0 & 0.0 & 1.0 & 0.0 & 833.0 \end{pmatrix}$$



(a) Accuracy variance

(b) Training time variance

Figure 22: Single hidden layer neural network

Observations

- The models trained with the above hyperparameters. Different networks converged at different number of epochs. Thus, variation can be seen in plot of time vs size
- The model overfits because, as would be predicted, training accuracy rises with unit count. Although at a slower rate, training accuracy also appears to stabilise at increasing numbers of units.

2.3 Adaptive Learning Rate

The neural network was trained on the same hyperparameters as above using the following adaptive learning rate setting :

$$\eta_t = \frac{\eta_0}{\sqrt{t}}, \eta_0 = 0.1$$

5 hidden layer units

- *Training accuracy* : 76.9067%
- *Test accuracy* : 76.02%
- *Training time* : 63.5s
- *Test data confusion matrix* :

$$\text{confusion matrix} = \begin{pmatrix} 949.0 & 2.0 & 213.0 & 3.0 & 2.0 & 2.0 & 0.0 & 1.0 & 0.0 & 8.0 \\ 6.0 & 665.0 & 15.0 & 150.0 & 0.0 & 155.0 & 0.0 & 15.0 & 0.0 & 14.0 \\ 9.0 & 5.0 & 225.0 & 9.0 & 0.0 & 20.0 & 0.0 & 0.0 & 0.0 & 26.0 \\ 20.0 & 235.0 & 252.0 & 803.0 & 0.0 & 405.0 & 0.0 & 3.0 & 0.0 & 26.0 \\ 0.0 & 1.0 & 1.0 & 3.0 & 884.0 & 2.0 & 37.0 & 7.0 & 8.0 & 3.0 \\ 0.0 & 20.0 & 14.0 & 8.0 & 0.0 & 43.0 & 0.0 & 1.0 & 0.0 & 8.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 61.0 & 0.0 & 898.0 & 4.0 & 46.0 & 0.0 \\ 2.0 & 37.0 & 10.0 & 7.0 & 9.0 & 57.0 & 0.0 & 941.0 & 1.0 & 36.0 \\ 4.0 & 0.0 & 19.0 & 0.0 & 44.0 & 2.0 & 65.0 & 5.0 & 945.0 & 6.0 \\ 10.0 & 35.0 & 251.0 & 17.0 & 0.0 & 314.0 & 0.0 & 23.0 & 0.0 & 873.0 \end{pmatrix}$$

10 hidden layer units

- *Training accuracy* : 85.2167%
- *Test accuracy* : 83.65%
- *Training time* : 96.09s

- *Test data confusion matrix* :

$$\text{confusion matrix} = \begin{pmatrix} 946.0 & 3.0 & 15.0 & 5.0 & 0.0 & 6.0 & 0.0 & 1.0 & 0.0 & 10.0 \\ 13.0 & 686.0 & 12.0 & 107.0 & 1.0 & 150.0 & 0.0 & 7.0 & 0.0 & 16.0 \\ 27.0 & 7.0 & 833.0 & 39.0 & 1.0 & 43.0 & 0.0 & 6.0 & 0.0 & 50.0 \\ 7.0 & 167.0 & 39.0 & 758.0 & 0.0 & 149.0 & 0.0 & 8.0 & 0.0 & 11.0 \\ 0.0 & 1.0 & 0.0 & 1.0 & 888.0 & 4.0 & 39.0 & 5.0 & 13.0 & 3.0 \\ 1.0 & 85.0 & 57.0 & 80.0 & 0.0 & 416.0 & 0.0 & 13.0 & 0.0 & 51.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 67.0 & 0.0 & 909.0 & 5.0 & 43.0 & 0.0 \\ 2.0 & 12.0 & 3.0 & 7.0 & 6.0 & 32.0 & 0.0 & 947.0 & 0.0 & 18.0 \\ 1.0 & 1.0 & 0.0 & 0.0 & 37.0 & 2.0 & 52.0 & 1.0 & 944.0 & 1.0 \\ 3.0 & 38.0 & 41.0 & 3.0 & 0.0 & 198.0 & 0.0 & 7.0 & 0.0 & 840.0 \end{pmatrix}$$

15 hidden layer units

- *Training accuracy* : 86.0817%
- *Test accuracy* : 84.23%
- *Training time* : 75.27s

- *Test data confusion matrix* :

$$\text{confusion matrix} = \begin{pmatrix} 939.0 & 0.0 & 15.0 & 3.0 & 0.0 & 4.0 & 0.0 & 1.0 & 0.0 & 3.0 \\ 14.0 & 726.0 & 16.0 & 84.0 & 0.0 & 143.0 & 0.0 & 8.0 & 0.0 & 15.0 \\ 31.0 & 14.0 & 847.0 & 38.0 & 2.0 & 49.0 & 0.0 & 7.0 & 0.0 & 59.0 \\ 8.0 & 163.0 & 43.0 & 792.0 & 0.0 & 125.0 & 0.0 & 4.0 & 0.0 & 5.0 \\ 0.0 & 2.0 & 1.0 & 1.0 & 885.0 & 4.0 & 37.0 & 3.0 & 11.0 & 4.0 \\ 0.0 & 61.0 & 34.0 & 76.0 & 0.0 & 443.0 & 0.0 & 12.0 & 0.0 & 80.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 63.0 & 1.0 & 914.0 & 5.0 & 46.0 & 0.0 \\ 2.0 & 12.0 & 3.0 & 6.0 & 10.0 & 36.0 & 0.0 & 958.0 & 1.0 & 18.0 \\ 1.0 & 0.0 & 0.0 & 0.0 & 39.0 & 0.0 & 49.0 & 0.0 & 942.0 & 1.0 \\ 5.0 & 22.0 & 41.0 & 0.0 & 1.0 & 195.0 & 0.0 & 2.0 & 0.0 & 815.0 \end{pmatrix}$$

20 hidden layer units

- *Training accuracy* : 86.3217%
- *Test accuracy* : 84.59%
- *Training time* : 93.49s

- *Test data confusion matrix* :

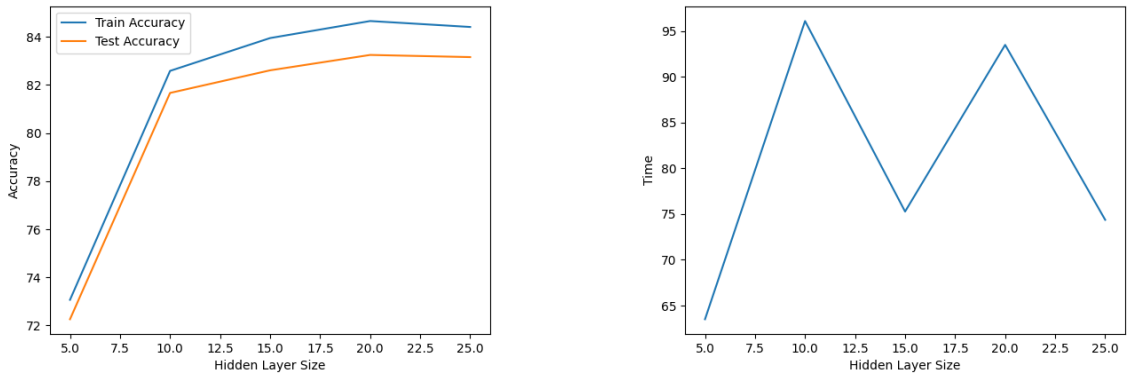
$$\text{confusion matrix} = \begin{pmatrix} 942.0 & 1.0 & 15.0 & 1.0 & 0.0 & 3.0 & 0.0 & 1.0 & 0.0 & 2.0 \\ 13.0 & 730.0 & 13.0 & 105.0 & 0.0 & 133.0 & 0.0 & 10.0 & 0.0 & 11.0 \\ 29.0 & 8.0 & 864.0 & 42.0 & 2.0 & 46.0 & 0.0 & 8.0 & 0.0 & 57.0 \\ 9.0 & 155.0 & 41.0 & 765.0 & 0.0 & 118.0 & 0.0 & 2.0 & 0.0 & 8.0 \\ 0.0 & 2.0 & 1.0 & 2.0 & 894.0 & 2.0 & 35.0 & 3.0 & 14.0 & 6.0 \\ 0.0 & 72.0 & 31.0 & 77.0 & 0.0 & 493.0 & 0.0 & 12.0 & 0.0 & 72.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 60.0 & 0.0 & 912.0 & 4.0 & 45.0 & 0.0 \\ 2.0 & 15.0 & 5.0 & 8.0 & 9.0 & 33.0 & 0.0 & 959.0 & 1.0 & 18.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 35.0 & 0.0 & 53.0 & 1.0 & 940.0 & 0.0 \\ 5.0 & 17.0 & 29.0 & 0.0 & 0.0 & 172.0 & 0.0 & 0.0 & 0.0 & 826.0 \end{pmatrix}$$

25 hidden layer units

- *Training accuracy* : 86.4167%
- *Test accuracy* : 84.72%
- *Training time* : 74.39s

- *Test data confusion matrix* :

$$\text{confusion matrix} = \begin{pmatrix} 943.0 & 1.0 & 13.0 & 3.0 & 0.0 & 1.0 & 0.0 & 1.0 & 0.0 & 2.0 \\ 14.0 & 734.0 & 18.0 & 92.0 & 0.0 & 131.0 & 0.0 & 12.0 & 0.0 & 16.0 \\ 29.0 & 11.0 & 854.0 & 38.0 & 2.0 & 48.0 & 0.0 & 9.0 & 0.0 & 64.0 \\ 8.0 & 155.0 & 40.0 & 773.0 & 0.0 & 118.0 & 0.0 & 1.0 & 0.0 & 7.0 \\ 0.0 & 2.0 & 1.0 & 0.0 & 893.0 & 1.0 & 39.0 & 5.0 & 17.0 & 2.0 \\ 0.0 & 67.0 & 33.0 & 88.0 & 0.0 & 494.0 & 0.0 & 15.0 & 0.0 & 68.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 66.0 & 0.0 & 913.0 & 5.0 & 42.0 & 0.0 \\ 2.0 & 11.0 & 6.0 & 6.0 & 6.0 & 34.0 & 0.0 & 952.0 & 1.0 & 20.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 32.0 & 0.0 & 48.0 & 0.0 & 940.0 & 1.0 \\ 4.0 & 19.0 & 34.0 & 0.0 & 1.0 & 173.0 & 0.0 & 0.0 & 0.0 & 820.0 \end{pmatrix}$$



(a) Accuracy variance

(b) Training time variance

Figure 23: Single hidden layer neural network with adptive learning rate

Observations

- As would be expected, training accuracy increases with unit count, which causes the model to overfit. Training accuracy also seems to stabilise with larger numbers of units, albeit more slowly.
- In the above models, the observed reduction in accuracies from the models in section 2.2 is

an example of underfitting due to the lack of training epochs and requires more time to fit on the data properly.

2.4 ReLU activation

The sigmoid activation was compared with the ReLU activation in the same hidden layer architecture : [100,100] with an adaptive learning rate.

Sigmoid Activation

- *Training accuracy* : 86.7%
- *Test accuracy* : 84.8%
- *Training time* : 948.2s
- *Test data confusion matrix* :

$$\text{confusion matrix} = \begin{pmatrix} 936.0 & 3.0 & 17.0 & 2.0 & 0.0 & 5.0 & 0.0 & 1.0 & 0.0 & 5.0 \\ 15.0 & 703.0 & 15.0 & 106.0 & 0.0 & 145.0 & 0.0 & 16.0 & 0.0 & 18.0 \\ 34.0 & 11.0 & 853.0 & 43.0 & 2.0 & 61.0 & 0.0 & 11.0 & 0.0 & 59.0 \\ 9.0 & 160.0 & 38.0 & 757.0 & 0.0 & 114.0 & 0.0 & 2.0 & 0.0 & 8.0 \\ 0.0 & 2.0 & 2.0 & 1.0 & 890.0 & 6.0 & 46.0 & 9.0 & 17.0 & 4.0 \\ 0.0 & 82.0 & 32.0 & 80.0 & 0.0 & 418.0 & 0.0 & 10.0 & 0.0 & 54.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 63.0 & 0.0 & 880.0 & 6.0 & 46.0 & 0.0 \\ 2.0 & 18.0 & 5.0 & 9.0 & 9.0 & 37.0 & 0.0 & 941.0 & 1.0 & 21.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 36.0 & 0.0 & 74.0 & 1.0 & 936.0 & 1.0 \\ 4.0 & 21.0 & 37.0 & 2.0 & 0.0 & 214.0 & 0.0 & 3.0 & 0.0 & 830.0 \end{pmatrix}$$

ReLU Activation

- *Training accuracy* : 94.1%
- *Test accuracy* : 87.21%
- *Training time* : 901.95s

- *Test data confusion matrix* :

$$\text{confusion matrix} = \begin{pmatrix} 964.0 & 1.0 & 12.0 & 1.0 & 0.0 & 2.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & 801.0 & 12.0 & 87.0 & 0.0 & 92.0 & 0.0 & 8.0 & 0.0 & 13.0 \\ 21.0 & 11.0 & 878.0 & 33.0 & 1.0 & 33.0 & 0.0 & 8.0 & 0.0 & 32.0 \\ 4.0 & 106.0 & 41.0 & 825.0 & 0.0 & 79.0 & 0.0 & 7.0 & 0.0 & 7.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 953.0 & 0.0 & 19.0 & 3.0 & 6.0 & 2.0 \\ 3.0 & 57.0 & 28.0 & 47.0 & 0.0 & 629.0 & 0.0 & 8.0 & 0.0 & 87.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 33.0 & 0.0 & 947.0 & 5.0 & 39.0 & 0.0 \\ 1.0 & 5.0 & 4.0 & 6.0 & 2.0 & 18.0 & 0.0 & 955.0 & 1.0 & 11.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 11.0 & 0.0 & 34.0 & 1.0 & 954.0 & 0.0 \\ 6.0 & 19.0 & 24.0 & 0.0 & 0.0 & 147.0 & 0.0 & 4.0 & 0.0 & 848.0 \end{pmatrix}$$

Observation

- The above comparison clearly indicates the superiority of the ReLU activation function of the sigmoid activation function, with ReLU performing better in test accuracy by 2.51%..

2.5 Multi-layer networks

Keeping the number of units in each hidden layer set to 50, observations were made on the effect of number of hidden layers on the test performance and training time.

Sigmoid Activation

2 Hidden layers

- *Training accuracy* : 84.58%
- *Test accuracy* : 82.92%

3 Hidden layers

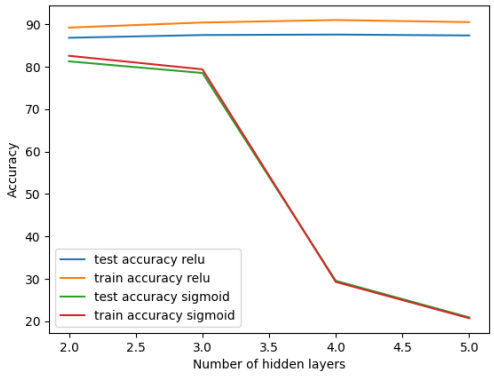
- *Training accuracy* : 82.4533%
- *Test accuracy* : 81.74%

4 Hidden layers

- *Training accuracy* : 29.285%
- *Test accuracy* : 29.54%

5 Hidden layers

- *Training accuracy* : 20.73%
- *Test accuracy* : 20.92%



(a) Accuracy variance

ReLU Activation

2 Hidden layers

- *Training accuracy* : 89.255%
- *Test accuracy* : 86.4%

3 Hidden layers

- *Training accuracy* : 90.445%
- *Test accuracy* : 87.5%

4 Hidden layers

- *Training accuracy* : 91.033%
- *Test accuracy* : 87.61%

5 Hidden layers

- *Training accuracy* : 90.535%
- *Test accuracy* : 87.4%

Observation

- In multi-layer models with sigmoid activation, the decline in training and test accuracy points to underfitting from a shortage of training epochs. This enables us to draw the conclusion that as the number of hidden layers increases, more training epochs are required in order for the model to match the training data correctly.
- ReLU activation models exhibit an improvement in training accuracy up to 4 layers, in contrast to sigmoid activation models, which show a drop in test accuracy. This demonstrates unequivocally that a model with ReLU activation can train to overfitting with the same amount of epochs, but sigmoid activation models can only train to underfitting. This enables us to conclude that, in order to get the same fit, a model with ReLU activation function needs much fewer epochs than a model with sigmoid activation. In comparison to the 4-layer network, the 5-layer model with ReLU activation exhibits a decline in training accuracy and a minor improvement in test accuracy. If you take into account the limited amount of training epochs, this may also be the result of an unexpected early stop brought on by disappearing gradients.

Experimenting with number of hidden layers

ReLU activated [25,50] architecture

- *Training accuracy* : 91.8%
- *Test accuracy* : 87.2%

ReLU activated [50,25] architecture

- *Training accuracy* : 90.86%
- *Test accuracy* : 86.75%

ReLU activated [50,75] architecture

- *Training accuracy* : 93.4%
- *Test accuracy* : 87.6%

ReLU activated [25,50,25] architecture

- *Training accuracy* : 91.53%
- *Test accuracy* : 87.22%

The best architecture is one with layers [50,50,50,50]

2.6 Binary cross entropy loss

Considering the backpropagation was programmed to implement a stochastic gradient descent on a desired cost function J, we can formulate the following :

Let the output of neurons be $\mathcal{O}_j$ and ψ be the activation function.

$$\delta_j = \frac{\partial J}{\partial \mathcal{O}_j}\frac{\partial \mathcal{O}_j}{\partial net_j} = \begin{cases} \frac{\partial J(\mathcal{O}_j,y)}{\partial \mathcal{O}_j}\frac{\partial \psi(net_j)}{\partial net_j} & \text{if } j \text{ is and output neuron} \\ (\sum_{l\in L} w_{jl}\delta_l)\frac{\partial \psi(net_j)}{\partial net_j} & \text{if } j \text{ is and inner neuron} \end{cases}$$

The only change to implement gradient descent on binary cross entropy loss will be in the δ_j calculation at the output layer, where $\partial J(\mathcal{O}_j, y)/\partial \mathcal{O}_j$ will now be evaluated as follows.

$$\begin{aligned} J(\mathcal{O}, y) &= -\sum_{j=1}^{C} y_j log(\mathcal{O}_j) + (1-y_j)log(1-\mathcal{O}_j) \\ \frac{\partial J(\mathcal{O}, y)}{\partial \mathcal{O}_j} &= \frac{\mathcal{O}_j - y_j}{\mathcal{O}_j(1-\mathcal{O}_j)} \\ \frac{\partial J(\mathcal{O}_j, y)}{\partial \mathcal{O}_j}\frac{\partial \psi(net_j)}{\partial net_j} &= \mathcal{O}_j - y_j \end{aligned}$$

Implementing the above with 3 hidden layers with 50 units each(as it performed the best in section 2.5) with ReLU activation and adaptive learning rate, the following observations were made.

- *Training accuracy* : 90.64%
- *Test accuracy* : 87.53%

2.7 MLPClassifier by scikit-learn

The multi-layer perceptron classifier *MLPClassifier* by scikit-learn was used to train(SGD) a neural network provide with 4 hidden layers (50,50,50,50) with ReLU activation. MLPClassifier by default uses binary cross entropy loss, hence can be compare to the model in section 2.6. The results of the training were as follows :

- *Training accuracy* : 90.58%
- *Test accuracy* : 85.31%
- *Training time* : 153.67s

Observations

- While both models use the same neural network architecture and activation, the MLPClassifier model requires more time to train and performs marginally better than the model created entirely from scratch.