Assignment 1

1. Datasets

The 2 datasets chosen for this assignment are the Heart dataset and the Digit Recognizer (MNIST) dataset. The Heart dataset contains 303 samples, while the Digit Recognizer dataset contains 42000 samples.
The Heart dataset has 2 labels [heart disease, not heart disease], while the Digit Recognizer dataset has 10 labels [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].
I chose the datasets with no missing data or any data pre-processing is needed before using the data to focus on tuning the hyperparameters for the algorithms.

1.1 Heart Dataset

I tend to choose medical/health datasets in most of my machine learning research/projects. While getting my master's degree in electrical engineering, I worked on a deep learning model for breast cancer classification problem in my master's thesis using novel raw dataset provided to me by my thesis advisor and I achieved very good and promising results. Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worldwide. Four out of 5 CVD deaths are due to heart attacks and strokes, and one-third of these deaths occur prematurely in people under 70 years of age. Heart failure is a common event caused by CVDs and this dataset contains 11 features that can be used to predict a possible heart disease.
Initially, the dataset contains 76 features or attributes from 303 patients; however, published studies chose only 14 features that are relevant in predicting heart disease. Hence, here we will be using the dataset consisting of 303 patients with 14 features set.
To introduce the problem; It is a classification problem where we need to train our model with the Heart dataset and be able to classify between Heart Disease or Not Heart Disease. For these reasons I chose the Heart dataset and I want to explore all possible machine learning algorithms to achieve a good model with high classification accuracy.
For the Heart Dataset Attribute Information, please visit the below link:
UCI Machine Learning Repository: Statlog (Heart) Data Set

1.2 Digit Recognizer (MNIST) Dataset

MNIST is a large dataset of 28x28 grayscale images of handwritten single digits between 0 to 9. The dataset contains 42000 samples, all images are labeled with the respective digit that they represent. There are total of 10 classes of digits from (0 to 9).

The challenge is to classify a handwritten digit based on 28x28 grayscale image. MNIST is often credited as one of the first datasets to prove the effectiveness of neural networks. Since we are exploring Neural Network algorithms in this assignment, I wanted to work on a dataset that is known to work well on the Neural Network classifiers and test this dataset on the different classifiers that we are implementing in this assignment, such as

SVM, Decision Tree and KNN. Please visit the below link for more information about MNIST dataset

MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges

2. Introduction to the Problem

In this section we will be identifying the problem we are trying to solve for both datasets, I will also go through my approach in solving the problem and optimizing the algorithms I am using to solve the problem.
What we are trying to solve for the Heart and MNIST datasets is a Classification problem. For the Heart dataset, we have 2 classes [Heart Disease, Not Heart Disease] which we need to classify an input to be one these 2 classes. For the MNIST dataset, we have 10 classes which are grayscale images of digits from 0 to 9, our goal is to classify an input image to be one of the 10 classes with the highest accuracy possible.

3. The Proposed Approach to Solve the Classification Problem

My approach to solve this classification problem will be the same for each algorithm, except for the Neural Network algorithm where I used the GridSearchCV library to find the best value for each hyperparameter I need to tune.
The following steps summarize my approach:

1. Import the dataset and split it into train and test sets, test set = 30 % of the dataset.
2. Get an insight on the classifier default parameters performance on the train and test sets by fitting the model into the train set using the default parameters, then test it on the test set.
3. In most cases, the results are not good. Therefore, hyperparameters tuning is the next step.
4. Using the cross-validation technique on the training set and by focusing on each hyperparameter solely, the validation curves shows the best value for each hyperparameter.
5. Using the best value for each hyperparameter in the classification algorithm and re-fit the model with the tuned hyperparameters into the train set, then test it once again on the test set. In most cases, we get better classification results.

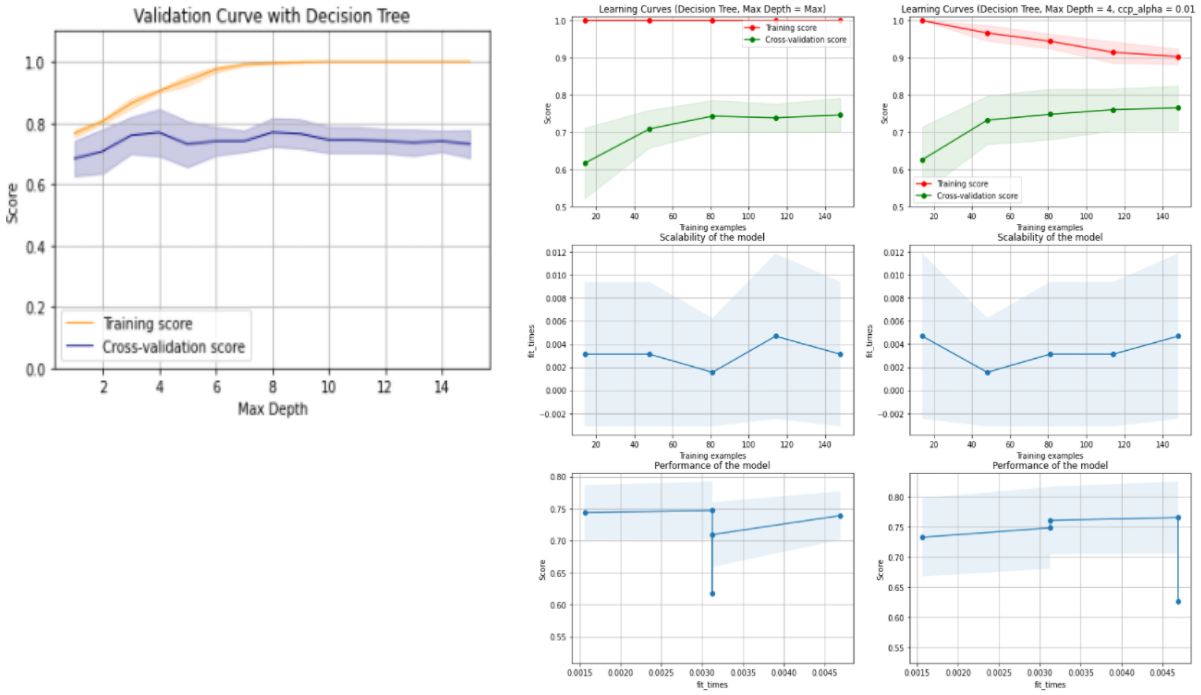
4. Results

After I gave a quick overview on my approach in section 3, section 4 will walk you through my approach in details with visual aids such as graphs and reports, and I will provide a brief discussion and analysis in this section as well.

4.1 Decision Tree Algorithm

A. Heart Dataset

Fitting the model with the default hyperparameters resulted in 100% accuracy for train set and 76.9% accuracy for the test set and that is a sign of overfitting.
By using the cross-validation technique, Max Depth = 4 hyperparameter gives us the best cross-validation score as shown below
After applying the Max Depth = 4 to our tuned model. Also, I applied different values for the ccp_alpha hyperparameter which is a complexity parameter used for minimal cost-complexity pruning, ccp_alpha = 0.01 was the best value for our model as shown below in the learning curves. This tuned model when tested on the test set gives us 75.8% accuracy.



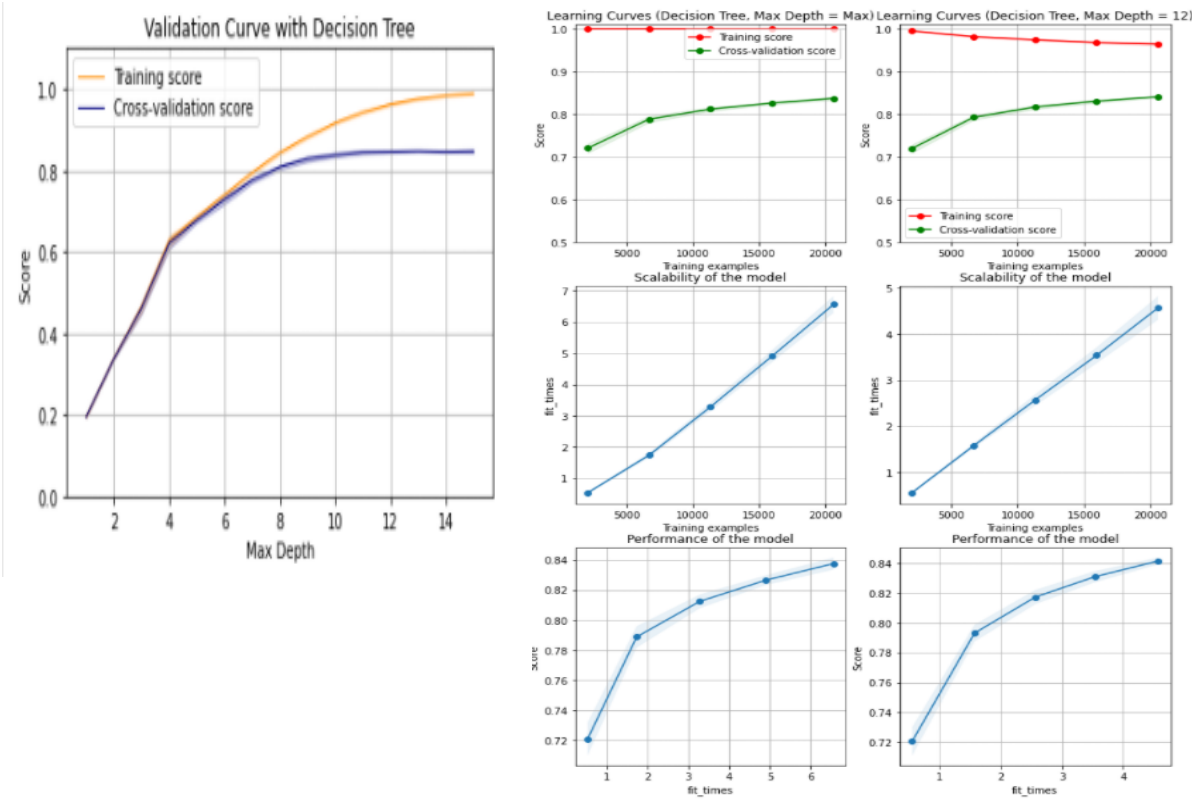
Note: On the left the Validation Curve graph. On the right the Learning Curve graphs, the first column represents the learning curves of the model with the default hyperparameters, the second column represent the learning curves of the model with the tuned hyperparameters. The second plots in the second row show the times required by the models to train with various sizes of training sets. The plots in the third row show how much time was required to train the models for each training size.

B. MNIST Dataset

Fitting the model with the default hyperparameters resulted in 100% accuracy for train set and 85.1% accuracy for the test set and that is a sign of overfitting.

By using the cross-validation technique, Max Depth = 12 gives us the best cross-validation score as shown below

By applying Max Depth = 12 to our tuned model. This tuned model when tested on the test set gives us 85.4% accuracy on the test. The learning curves are shown below



4.2 **Boosting Algorithm (AdaBoost)**

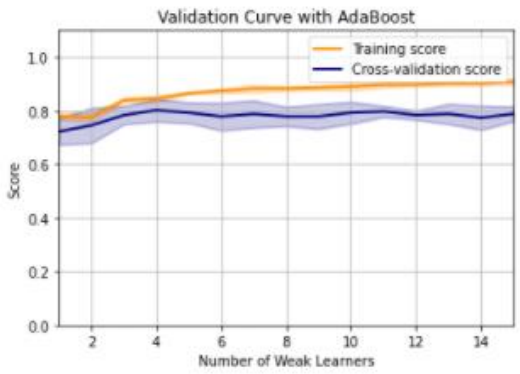
A. Heart Dataset

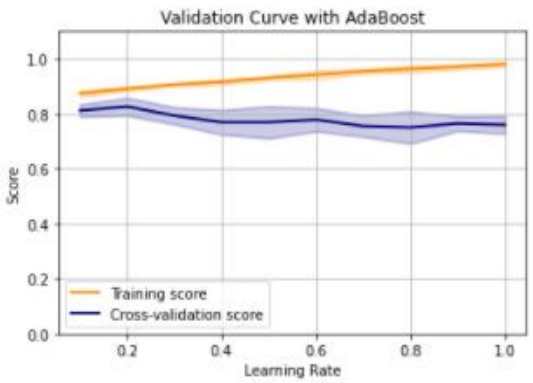
Fitting the model with the default hyperparameters resulted in 93.8% accuracy for train set and 83.5% accuracy for the test set.

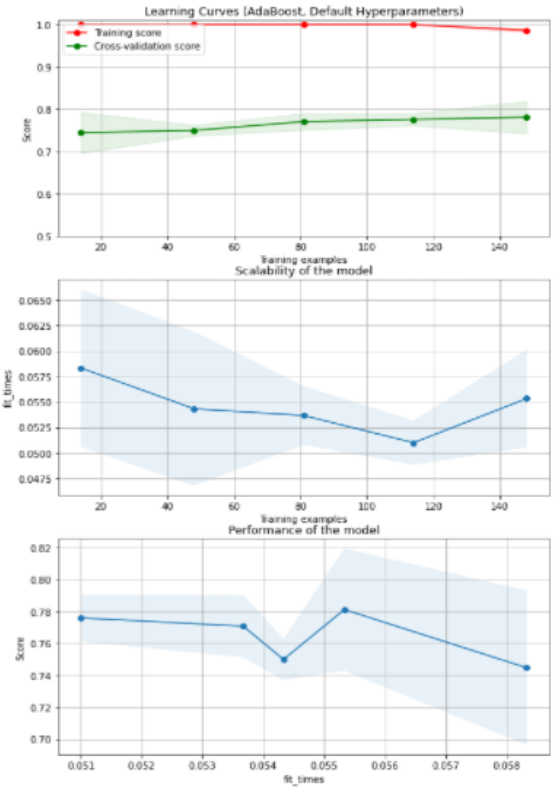
By using the cross-validation technique, Number of Weak Learners = 4 gives us the best cross-validation score as shown below. Also, the validation curve for the best learning rate = 0.2 hyperparameter is shown below

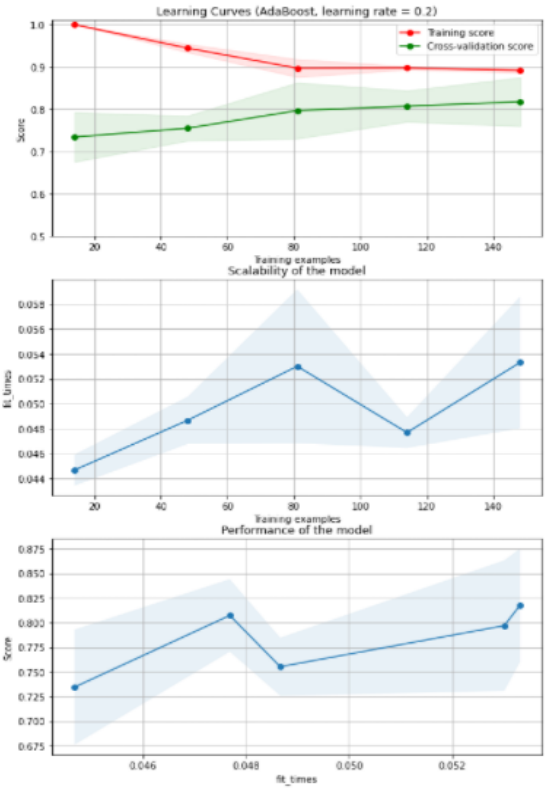
After examining both tuned hyperparameters, weak learner = 4 and learning rate = 0.2, the results show the weak learners have less influence on the model performance than the learning rate. So ,I decided to use only learning rate to tune my AdaBoost model and ignore the number of weak learners hyperparameter.

The learning curves are shown below. The tuned model gives us 85.7% accuracy on the test set.









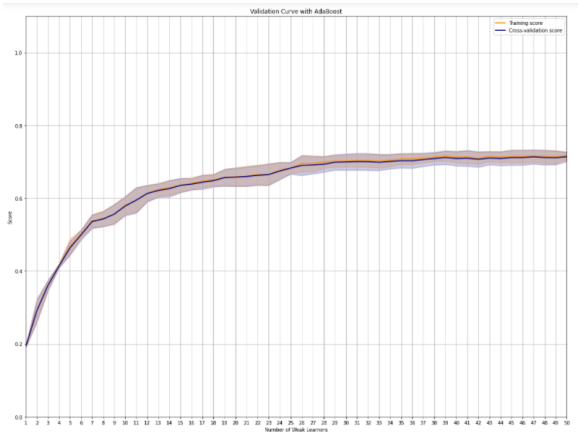
B. MNIST Dataset

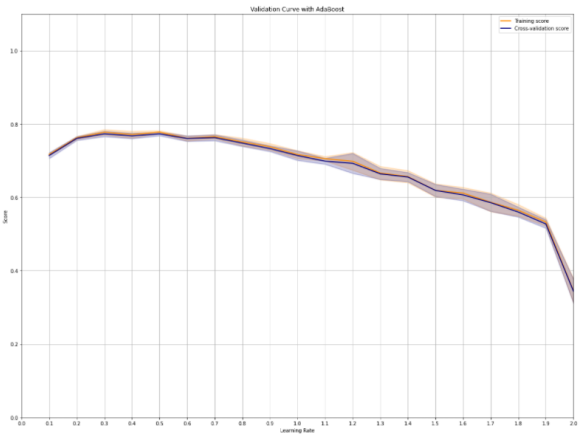
Fitting the model with the default hyperparameters resulted in 73.6% accuracy for train set and 72.9% accuracy for the test set.

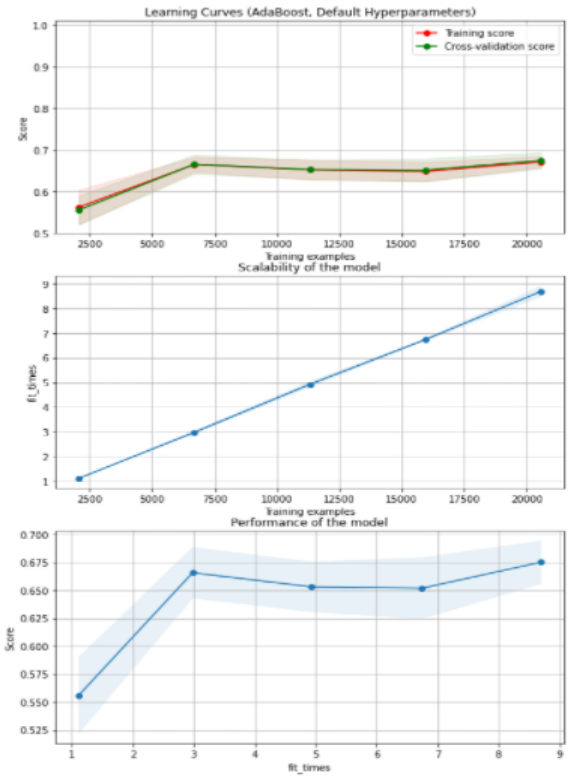
By using the cross-validation technique, Training and Validation scores become stable (70-75%) at number of weak learners between 25-30 learners as shown below. Also, the validation curve for the best learning rate = 0.3 hyperparameter is shown below

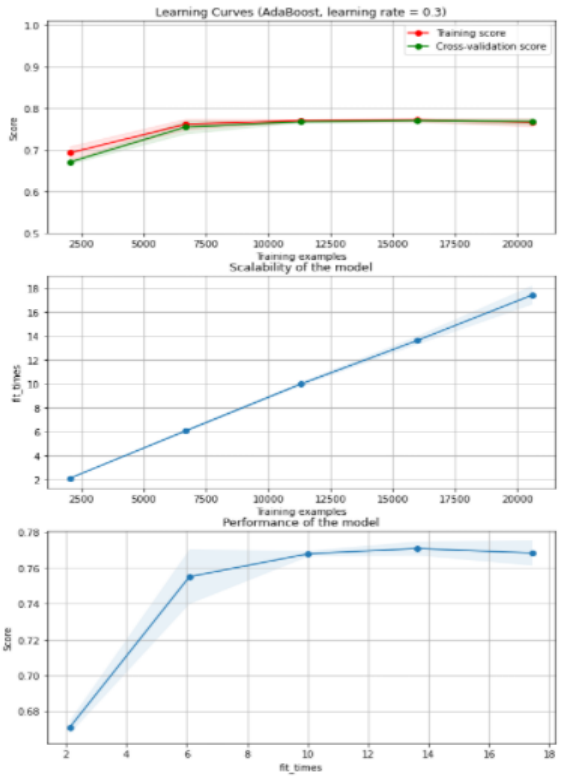
After examining both tuned hyperparameters, weak learner = 25 and learning rate = 0.3, the results show the weak learners have less influence on the model

performance than the learning rate. So ,I decided to use only learning rate to tune my AdaBoost model and ignore the weak learners number hyperparameter to reduce computational cost. The learning curves are shown below. The tuned model gives us 77.7% accuracy on the test set.









4.3 **Support Vector Machine (SVM)**

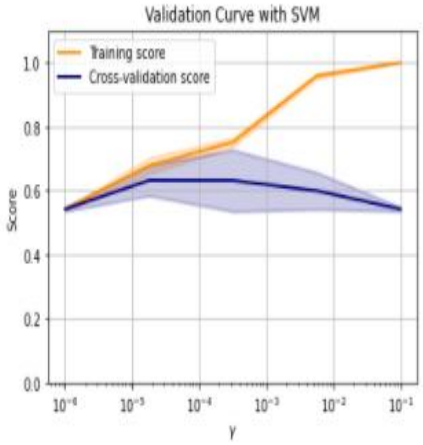
A. Heart Dataset

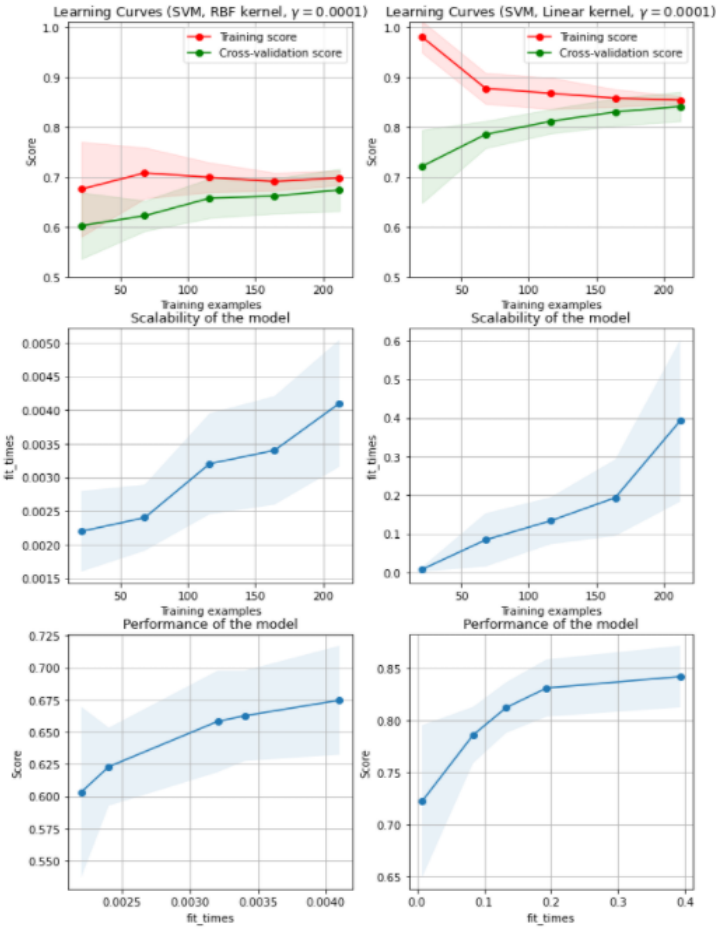
Fitting the model with the default hyperparameters resulted in 67.4% accuracy for train set and 60.4% accuracy for the test set.

By using the cross-validation technique, the 'Linear' Kernel hyperparameter and hyperparameter gamma = 0.0001, where gamma is the 'Kernel Coefficient' gives us

the best cross-validation score when tested against the 'RBF' Kernel with the same gamma value as shown below.

The tuned model gives us 81.3% accuracy on the test set.



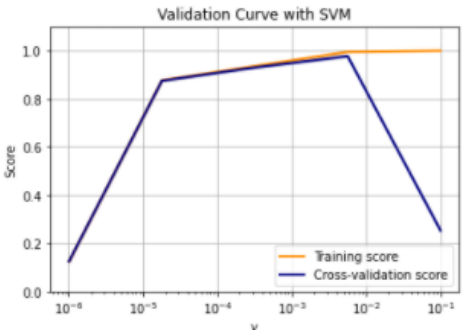


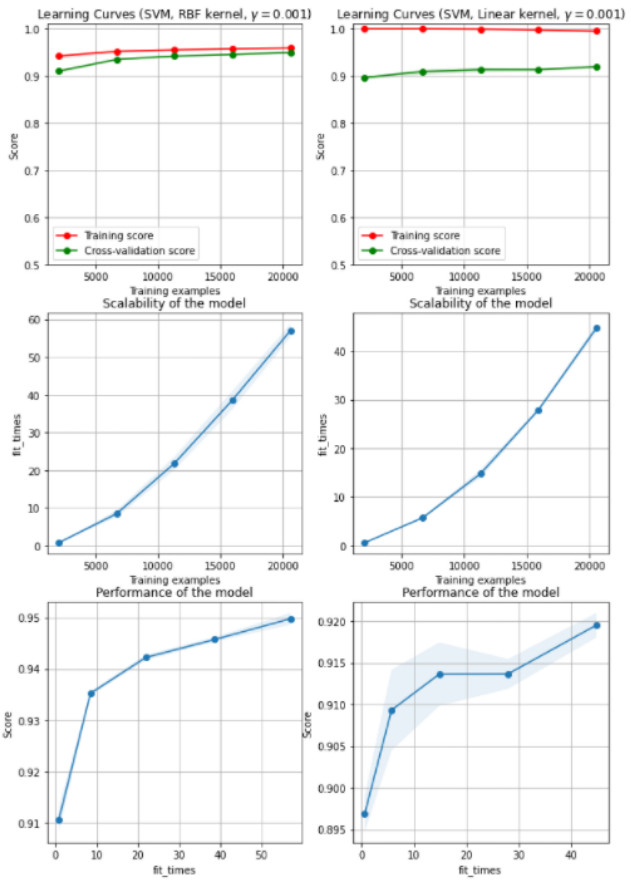
B. MNIST Dataset

Fitting the model with the default hyperparameters resulted in 98.8% accuracy for train set and 97% accuracy for the test set.

By using the cross-validation technique, the 'RBF' Kernel hyperparameter and hyperparameter gamma = 0.001 gives us the best cross-validation score when tested against the 'Linear' Kernel with the same gamma value as shown below.

The tuned model gives us 95.5% accuracy on the test set.





4.4 **Neural Network Algorithm**

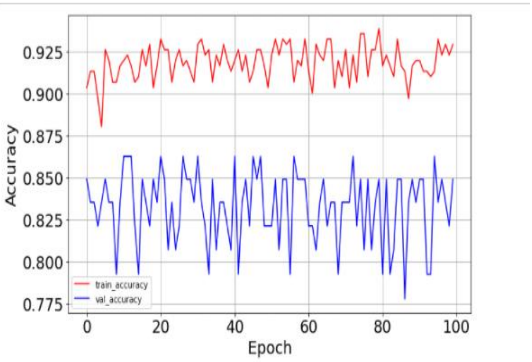
A. Heart Dataset

Fitting the model with the default hyperparameters resulted in 54.2% accuracy for train set and 54.9% accuracy for the test set.
By using the GridSearchCV method to tune hyperparameters, I got the following best value for Neural Network hyperparameters: Batch Size = 10, Epochs = 100, Optimizer = Adam, Learning rate = 0.001, Input Activation Function = ReLU, Dropout Rate = 0.1, Weight Constraint = 2, Number of Neuron in the Hidden Layer = 8, Output Activation Function = Sigmoid. Examples these results from the GridSearchCV is shown below

Applying these tuned hyperparameters to the model resulted in 84.6% accuracy on test set. The learning curves are shown below

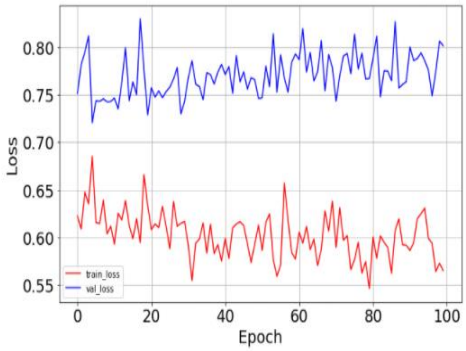
```
Best: 0.763917 using {'batch_size': 10, 'epochs': 100}
0.542857 (0.063016) with: {'batch_size': 10, 'epochs': 10}
0.678873 (0.088721) with: {'batch_size': 10, 'epochs': 50}
0.763917 (0.067364) with: {'batch_size': 10, 'epochs': 100}
```

```
Best: 0.782562 using {'optimizer': 'Adam'}
0.452917 (0.064590) with: {'optimizer': 'SGD'}
0.754058 (0.123918) with: {'optimizer': 'RMSprop'}
0.782562 (0.097215) with: {'optimizer': 'Adam'}
```




```
Best: 0.810999 using {'neurons': 8}
0.711133 (0.171050) with: {'neurons': 1}
0.810999 (0.059261) with: {'neurons': 8}
0.773239 (0.065590) with: {'neurons': 16}
0.773441 (0.024493) with: {'neurons': 32}
0.768545 (0.075202) with: {'neurons': 64}
0.726157 (0.048209) with: {'neurons': 128}
```

```
Best: 0.797049 using {'learn_rate': 0.001}
0.797049 (0.033982) with: {'learn_rate': 0.001}
0.778068 (0.035000) with: {'learn_rate': 0.01}
0.542119 (0.067931) with: {'learn_rate': 0.1}
0.542119 (0.067931) with: {'learn_rate': 0.2}
0.452917 (0.064590) with: {'learn_rate': 0.3}
```



B. MNIST Dataset

Fitting the model with the default hyperparameters resulted in 10.4% accuracy for train set and 10.2% accuracy for the test set.

By using the GridSearchCV method to tune hyperparameters, I got the following best value for Neural Network hyperparameters: Batch Size = 60, Epochs = 50, Optimizer = Adam, Learning rate = 0.001, Input Activation Function = ReLU, Dropout Rate = 0, Weight Constraint = 1, Number of Neuron in the Hidden Layer = 128, Output Activation Function = Softmax. Examples these results from the GridSearchCV is shown below

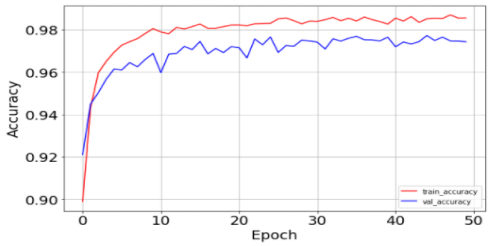
Applying these tuned hyperparameters to the model resulted in 94.4% accuracy on test set. The learning curves are shown below

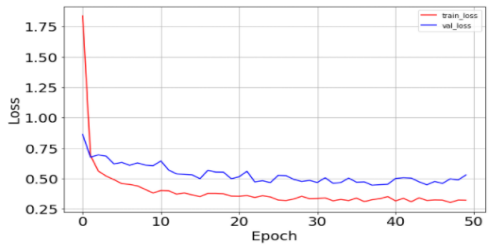
```
Best: 0.838367 using {'batch_size': 60, 'epochs': 50}
0.394660 (0.064633) with: {'batch_size': 10, 'epochs': 10}
0.545238 (0.099963) with: {'batch_size': 10, 'epochs': 20}
0.670204 (0.055631) with: {'batch_size': 10, 'epochs': 30}
0.552143 (0.121680) with: {'batch_size': 10, 'epochs': 40}

Best: 0.784864 using {'optimizer': 'Adam'}
0.111565 (0.002097) with: {'optimizer': 'SGD'}
0.695544 (0.132050) with: {'optimizer': 'RMSprop'}
0.784864 (0.010683) with: {'optimizer': 'Adam'}

Best: 0.836429 using {'learn_rate': 0.001}
0.836429 (0.025614) with: {'learn_rate': 0.001}
0.123605 (0.019045) with: {'learn_rate': 0.01}
0.105748 (0.006069) with: {'learn_rate': 0.1}
0.099286 (0.001041) with: {'learn_rate': 0.2}
0.101837 (0.005161) with: {'learn_rate': 0.3}

Best: 0.953401 using {'neurons': 128}
0.316905 (0.028872) with: {'neurons': 1}
0.876395 (0.013132) with: {'neurons': 8}
0.922891 (0.005246) with: {'neurons': 16}
0.941531 (0.002433) with: {'neurons': 32}
0.947449 (0.003573) with: {'neurons': 64}
0.953401 (0.004817) with: {'neurons': 128}
```





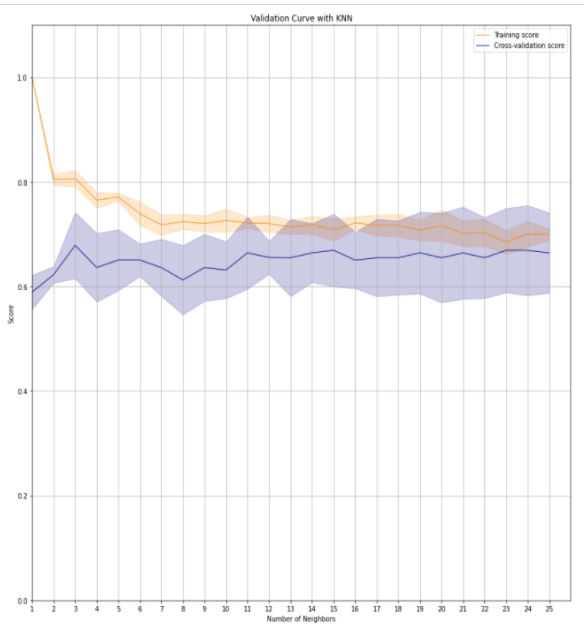
4.5 **K-Nearest Neighbor Algorithm**

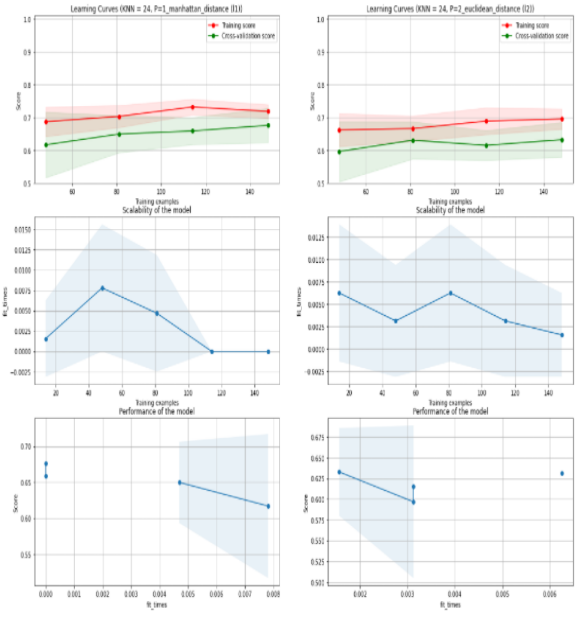
A. Heart Dataset

Fitting the model with the default hyperparameters resulted in 78.7% accuracy for train set and 63.7% accuracy for the test set.

By using the cross-validation technique, the 'Number of Neighbors' hyperparameter n_neighbors = 24 and p = 2 hyperparameter gives us the best cross-validation score when tested against hyperparameter p = 1, where 'p = 2' is equivalent to using euclidean_distance (l2) and 'p=1' equivalent to using manhattan_distance (l1) as shown below

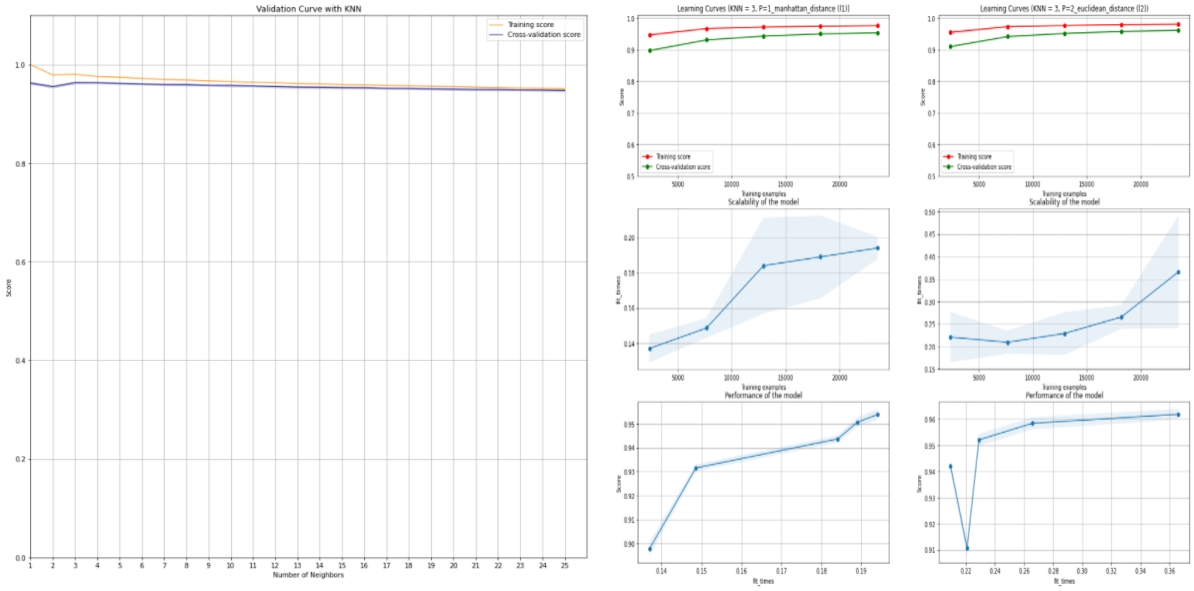
The tuned model gives us 68.1% accuracy on the test set.





B. MNIST Dataset

Fitting the model with the default hyperparameters resulted in 97.7% accuracy for train set and 96.1% accuracy for the test set.
By using the cross-validation technique, the 'Number of Neighbors' hyperparameter n_neighbors = 3 and p = 2 hyperparameter gives us the best cross-validation score when tested against hyperparameter p = 1
The tuned model gives us 96.4% accuracy on the test set.



5. Analysis and Conclusion

We saw 5 supervised learning algorithms and how they perform in a classification problem using the Heart and MNIST datasets. One of the interesting observations is the Decision Tree algorithm tend to overfit and we need to apply hyperparamet tuning in order for this algorithm to be able to generalize well on unseen data.
One of the reasons I chose the MNIST dataset that I wanted to test the Neural Network and Support Vector Machines algorithms on this dataset due to its popularity of giving good results when tested with NN and SVM, and yes the results were great, NN algorithm scored 94.4% and SVM scored 95.5% accuracy on the MNIST test set.
The computational costs for the Neural Network and Support Vector Machines with big datasets such as MNIST are huge, both algorithms took over 4 hours to execute. However, on the other hand the results are great, so it is considered a trade off between the computational cost and accuracy.
One of the interesting and unexpected results that I observed was K-Nearest Neighbor algorithm scored 96.4% accuracy on the MNIST test set, which is the highest so far, which can be an indicator that K-Nearest Neighbor algorithm perform well on image classification problems.
The below table summarize all the results I got on the test sets.

Algorithm	Test Accuracy Before Hyperparameter tuning_Heart Dataset	Test Accuracy After Hyperparameter tuning_Heart Dataset	Test Accuracy Before Hyperparameter tuning_MNIST Dataset	Test Accuracy After Hyperparameter tuning_MNIST Dataset
Decision Tree	76.9%	75.8%	85.1%	85.4%
Boosting	83.5%	85.7%	72.9%	77.7%
Support Vector Machine	60.4%	81.3%	97%	95.5%
Neural Network	54.9%	84.6%	10.2%	94.4%
K-Nearest Neighbor	63.7%	68.1%	96.1%	96.4%

6. **References**

1. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 2nd Edition
2. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2, 3rd Edition
3. https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html
4. https://scikit-learn.org/stable/auto_examples/model_selection/plot_validation_curve.html
5. How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras (machinelearningmastery.com)
6. Abdelmalak, Peter, "Deep learning for quantitative motion tracking based on optical coherence tomography" (2020). *Theses*. 1772. https://digitalcommons.njit.edu/theses/1772