



Ain Shams University, Faculty of Engineering

Milestone 2 Report

9 January 2023

Prepared by: Abdulrahman Ayman 19P6458

Ahmed Sameh 19P5861

ElSaeed Ahmed ElSaeed 19P1087

Contents

1)Function Implementation:	3
MLP Classifier:	3
MLP Tuning:	4
SVM Classifier:	5
SVM Tuning:	6
Decision Tree Classifier:	7
Decision Tree Tuning:	8
2)Running Project:	10
MLP Functions:	10
SVM Functions:	13
Decision Tree Functions:	16
3)GitHub Link:	20

1)Function Implementation:

MLP Classifier:

```
def MLP_Train_Test(self, train_sizes, test_size):
    output_text = ""
    X_test, Y_test = self.load_test_data(test_size)
    all_outs = []
    all_percent = []
    for sample_size in train_sizes:
        X, Y = self.load_train_data(sample_size)
        mlp = MLPClassifier(learning_rate_init=self.learningRate, max_iter=self.noEpoch, activation=self.actFunc)
        mlp.fit(X, Y)
        Y_pred = mlp.predict(X_test)
        output_text = output_text + "\n" + "... "
        all_outs.append((sample_size, round(accuracy_score(Y_test, Y_pred) * 100, 2)))
        all_percent.append(round(accuracy_score(Y_test, Y_pred) * 100, 2))
    self.temp_text = output_text
    best_pair = all_outs[all_percent.index(max(all_percent))]
    output_text = output_text + "\n" + f"Best Size is {best_pair[0]} with accuracy: {best_pair[1]}"
    self.output_text = output_text
    self.pairs = all_outs
```

Steps:

1. Load data using self.load_test_data() function same as Milestone 1
2. Initialize a MLPClassifier Using The implemented version from the SkLearn Library
3. Fit X and Y to the MLPClassifier and predict Y_Pred using X_test
4. Check it against Y_test to get both accuracy and precision using function provided by the SkLearn Library
5. Append all outputs to array to get the best one in the end
6. Graph the result using matplotlib Library same as Milestone 1

MLP Tuning:

```
def clearThreads(n_jobs=30):...

def heavy(activationh, learning_rate_inith, max_iterh):
    mlp = MLPClassifier(activation=activationh, learning_rate_init=learning_rate_inith, max_iter=max_iterh)
    mlp.fit(X, Y)
    Y_pred = mlp.predict(X_test)
    all_outs.append(((activationh, learning_rate_inith, max_iterh), round(accuracy_score(Y_test, Y_pred) * 100, 5)))
    all_percents.append(round(accuracy_score(Y_test, Y_pred) * 100, 5))

for activ in activation:
    for learning_rate in learning_rate_init:
        for epoch in max_iter:
            clearThreads(30)
            t = threading.Thread(target=heavy, args=(activ, learning_rate, epoch))
            threads.append(t)
            t.start()

for thread in threads:
    thread.join()

best_pair = all_outs[all_percents.index(max(all_percents))]
```

```
if self.activ_check.isChecked():
    activation = ["identity", "tanh", "relu"]
if self.checkBox_2.isChecked():
    learning_rate_init = [0.001, 0.01, 0.1]
if self.checkBox.isChecked():
    max_iter = [200, 250, 300, 350, 400]
```

Steps:

1. Load data using self.load_test_data() function same as Milestone 1
2. By checking certain checkboxes user adds more hyperparameters to tune
3. We then perform an exhaustive search on these parameters to find the best combination using multithreading for faster computation
 - a. Initialize a MLPClassifier Using The implemented version from the SkLearn Library with current parameters
 - b. Fit X and Y to the MLPClassifier and predict Y_Pred using X_test

- c. Check it against Y_test to get both accuracy and precision using function provided by the SkLearn Library
 - d. Append all outputs to array to get the best one in the end with the best hyperparameters
4. Graph the result using matplotlib Library same as Milestone 1

SVM Classifier:

```
def SVM_Train_Test(self, train_sizes, test_size):
    output_text = ""
    X_test, Y_test = self.load_test_data(test_size)
    all_outs = []
    all_percent = []
    for sample_size in train_sizes:
        X, Y = self.load_train_data(sample_size)
        svm = SVC(C=self.c, gamma=self.gamma)
        svm.fit(X, Y)
        Y_pred = svm.predict(X_test)
        output_text = output_text + "\n" + "... "
        all_outs.append((sample_size, round(accuracy_score(Y_test, Y_pred) * 100, 2)))
        all_percent.append(round(accuracy_score(Y_test, Y_pred) * 100, 2))
        self.temp_text = output_text
    best_pair = all_outs[all_percent.index(max(all_percent))]
    output_text = output_text + "\n" + f"Best Size is {best_pair[0]} with accuracy: {best_pair[1]}"
    self.output_text = output_text
    self.pairs = all_outs
```

Steps:

1. Load data using self.load_test_data() function same as Milestone 1
2. Initialize a SVC Classifier Using The implemented version from the SkLearn Library
3. Fit X and Y to the SVC and predict Y_Pred using X_test
4. Check it against Y_test to get both accuracy and precision using function provided by the SkLearn Library
5. Append all outputs to array to get the best one in the end
6. Graph the result using matplotlib Library same as Milestone 1

SVM Tuning:

```
def clearThreads(n_jobs=30):...

def heavy(gammah, ch):
    svm = SVC(C=ch, gamma=gammah)
    svm.fit(X, Y)
    Y_pred = svm.predict(X_test)
    all_outs.append(((gammah, ch), round(accuracy_score(Y_test, Y_pred) * 100, 5)))
    all_percents.append(round(accuracy_score(Y_test, Y_pred) * 100, 5))

for g in gamma:
    for c in C:
        clearThreads(30)
        t = threading.Thread(target=heavy, args=(g, c))
        threads.append(t)
        t.start()

for thread in threads:
    thread.join()
```

```
if self.gamma_check.isChecked():
    gamma = [1, 0.1, 0.01, 0.001, 'scale', 'auto']
if self.c_check.isChecked():
    C = [0.1, 1, 10, 100]
```

Steps:

1. Load data using self.load_test_data() function same as Milestone 1
2. By checking certain checkboxes user adds more hyperparameters to tune
3. We then perform an exhaustive search on these parameters to find the best combination using multithreading for faster computation
 - a. Initialize an SVC Classifier Using The implemented version from the SkLearn Library with current parameters
 - b. Fit X and Y to the SVC and predict Y_Pred using X_test

- c. Check it against Y_test to get both accuracy and precision using function provided by the SkLearn Library
 - d. Append all outputs to array to get the best one in the end with the best hyperparameters
4. Graph the result using matplotlib Library same as Milestone 1

Decision Tree Classifier:

```
def Decision_Train_Test(self, train_sizes, test_size):
    output_text = ""
    X_test, Y_test = self.load_test_data(test_size)
    all_outs = []
    all_percent = []
    all_trees = []
    for sample_size in train_sizes:
        X, Y = self.load_train_data(sample_size)
        des = DecisionTreeClassifier(criterion=self.criterion, max_depth=self.maxDepth,
                                    max_leaf_nodes=self.maxLeafs)
        des.fit(X, Y)
        all_trees.append(des)
        Y_pred = des.predict(X_test)
        output_text = output_text + "\n" + "... "
        all_outs.append((sample_size, round(accuracy_score(Y_test, Y_pred) * 100, 2)))
        all_percent.append(round(accuracy_score(Y_test, Y_pred) * 100, 2))
        self.temp_text = output_text
    best_pair = all_outs[all_percent.index(max(all_percent))]
    self.tree = all_trees[all_percent.index(max(all_percent))]
    output_text = output_text + "\n" + f"Best Size is {best_pair[0]} with accuracy: {best_pair[1]}"
    self.output_text = output_text
    self.pairs = all_outs
```

Steps:

1. Load data using self.load_test_data() function same as Milestone 1
2. Initialize a DecisionTreeClassifier Using The implemented version from the SkLearn Library
3. Fit X and Y to the DecisionTreeClassifier and predict Y_Pred using X_test

4. Check it against Y_test to get both accuracy and precision using function provided by the SkLearn Library
5. Append all outputs to array to get the best one in the end
6. Graph the result using matplotlib Library same as Milestone 1 for accuracy and using sklearn's plot_tree() function for plotting the decision tree

```
if mode != "normal":
    self.resize(1751, 747)
    self.canvas.setGeometry(QtCore.QRect(20, 10, 1710, 722))
    self.toolbar = NavigationToolbar(self.canvas, self)
    layout.addWidget(self.toolbar)
    plot_tree(pairs, max_depth=4, fontsize=7, label='root', class_names=True, filled=True)
```

Decision Tree Tuning:

```
def clearThreads(n_jobs=30):...

def heavy(criterion, max_depth, max_leaf_nodes):
    decision = DecisionTreeClassifier(criterion=criterion, max_depth=max_depth, max_leaf_nodes=max_leaf_nodes)
    decision.fit(X, Y)
    Y_pred = decision.predict(X_test)
    acc = round(accuracy_score(Y_test, Y_pred) * 100, 5)
    all_outs.append(((criterion, max_depth, max_leaf_nodes, decision), acc))
    all_percents.append(acc)

for crit in self.grid_params['criterion']:
    for depth in self.grid_params['max_depth']:
        for leaf in self.grid_params['max_leaf_nodes']:
            clearThreads(30)
            t = threading.Thread(target=heavy, args=(crit, depth, leaf))
            threads.append(t)
            t.start()

for thread in threads:
    thread.join()

self.grid_params = {'max_depth': [110, 130, 150, 170, 190, 210, 230, 250, 270, 290, 310],
                    'criterion': ['gini', 'entropy'],
                    'max_leaf_nodes': [110, 130, 150, 170, 190, 210, 230, 250, 270, 290, 310]}

self.grid_params = {
    'max_depth': [i for i in range(int(self.start_depth.text()), int(self.end_depth.text()), 20)],
    'max_leaf_nodes': [i for i in range(int(self.start_leaf.text()), int(self.end_leaf.text()), 20)]
}
if self.crit_check.isChecked():
    self.grid_params['criterion'] = ['gini', 'entropy']
```


Steps:

1. Load data using `self.load_test_data()` function same as Milestone 1
2. By inputting specific starts and ends for hyper parameters user can use manual tuning with their choosing values and pressing manual tune, Or by pressing auto tune directly and tuning using pre-existing values
3. We then preform an exhaustive search on these parameters to find the best combination using multithreading for faster computation
 - a. Initialize an `DecisionTreeClassifier` Using The implemented version from the `SkLearn` Library with current parameters
 - b. Fit X and Y to the `DecisionTreeClassifier` and predict `Y_Pred` using `X_test`
 - c. Check it against `Y_test` to get both accuracy and precision using function provided by the `SkLearn` Library
 - d. Append all outputs to array to get the best one in the end with the best hyperparameters
4. Graph the result using `matplotlib` Library same as Milestone 1 for accuracy and using `sklearn's plot_tree()` function for plotting the decision tree

```
if mode != "normal":
    self.resize(1751, 747)
    self.canvas.setGeometry(QtCore.QRect(20, 10, 1710, 722))
    self.toolbar = NavigationToolbar(self.canvas, self)
    layout.addWidget(self.toolbar)
    plot_tree(pairs, max_depth=4, fontsize=7, label='root', class_names=True, filled=True)
```

2)Running Project:

MLP Functions:

Super Classifier

Classifier Type:
MLP

Hyper Parameters Setup:
Setup

Data Sample :
Digit Data

Training Sample Size:
Starting Size: 4750 Ending Size: 5001 Step: 50

Testing Sample Size:
Size: 1000

Controls
Go
Clear Output
Reset

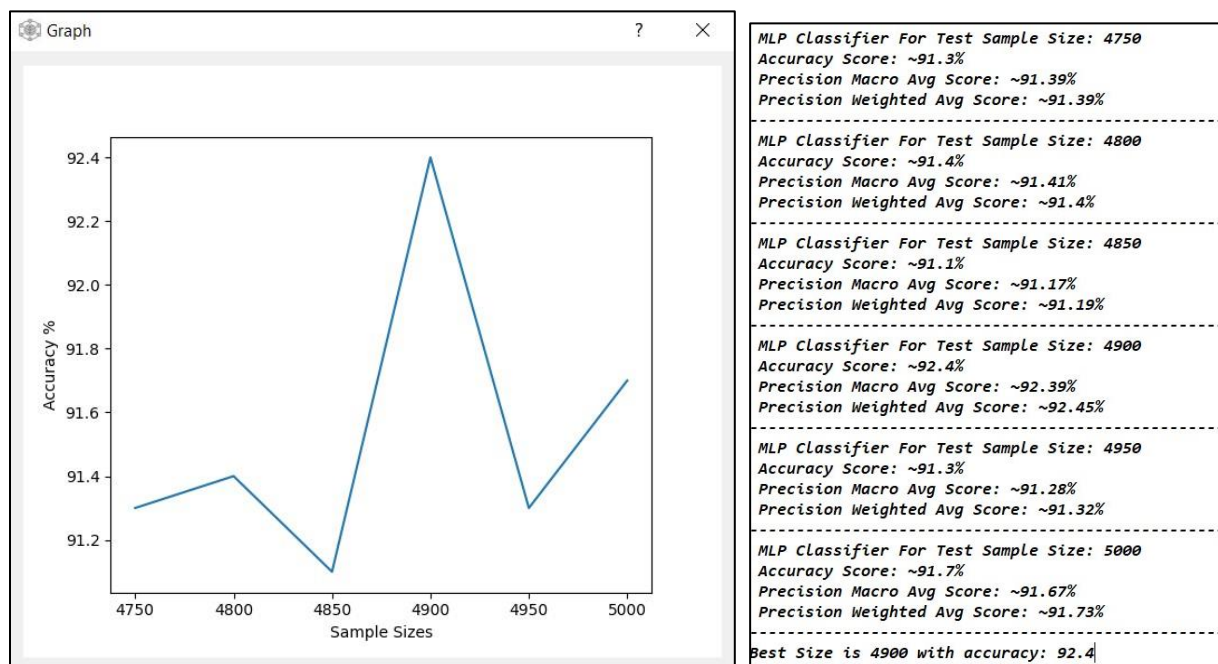
MLP Classifier For Test Sample Size: 4900
Accuracy Score: ~92.4%
Precision Macro Avg Score: ~92.39%
Precision Weighted Avg Score: ~92.45%

MLP Classifier For Test Sample Size: 4950
Accuracy Score: ~91.3%
Precision Macro Avg Score: ~91.28%
Precision Weighted Avg Score: ~91.32%

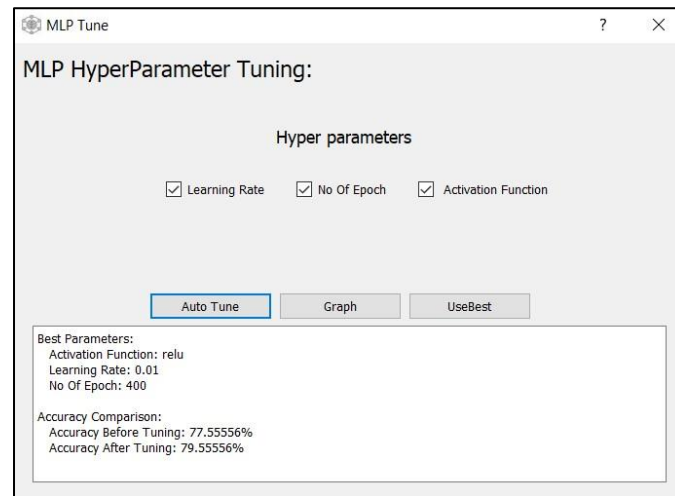
MLP Classifier For Test Sample Size: 5000
Accuracy Score: ~91.7%
Precision Macro Avg Score: ~91.67%
Precision Weighted Avg Score: ~91.73%

Best Size is 4900 with accuracy: 92.4

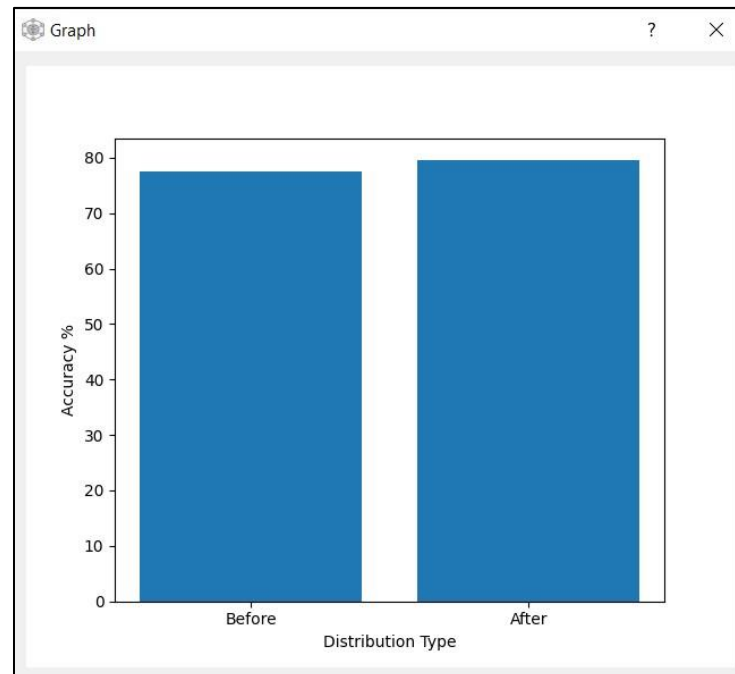
- By pressing Go we get output of MLP Classification results for all training sizes we have set and default hyperparameters:



- By pressing Hyper parameter Setup Button:

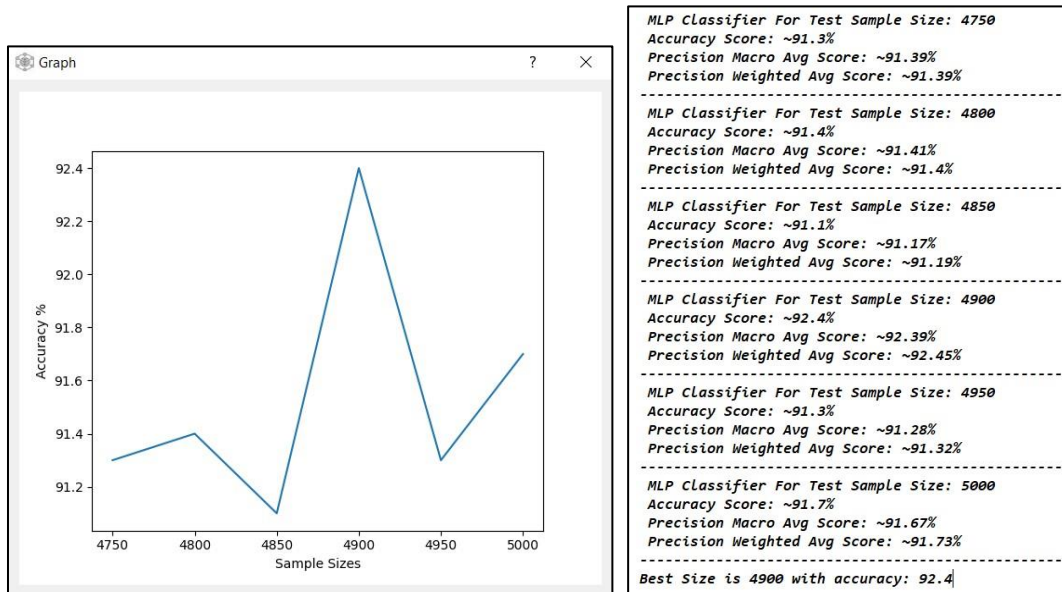


We get the following screen where we can auto tune MLP using list of values defined in code to get best parameters

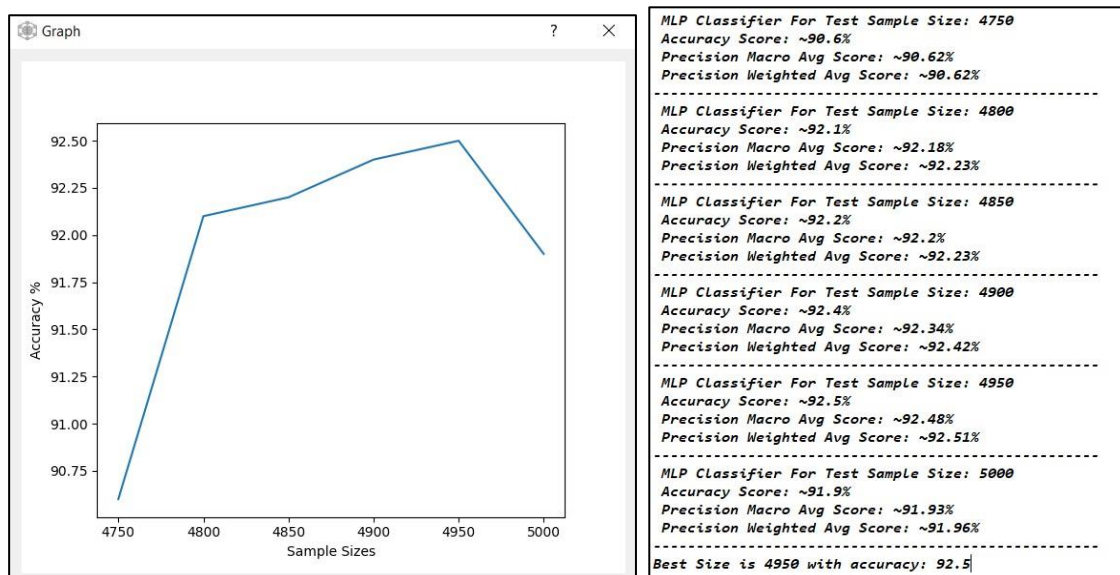


Note: By pressing UseBest Button you will return to main screen and the best parameters will be applied

Before:



After:



SVM Functions:

Super Classifier

Classifier Type: SVM

Hyper Parameters Setup: Setup

Data Sample : Digit Data

Training Sample Size: Starting Size: 4750 Ending Size: 5001 Step: 50

Testing Sample Size: Size: 1000

Controls

Go

Clear Output

Reset

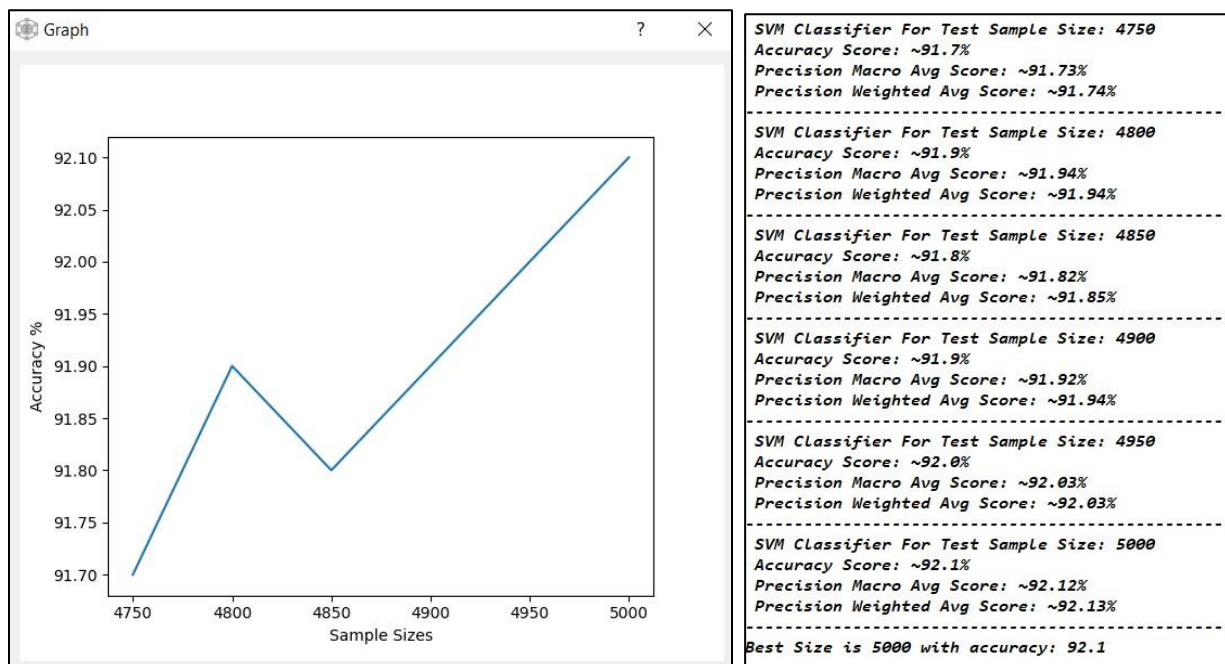
SVM Classifier For Test Sample Size: 4750
Accuracy Score: ~91.7%
Precision Macro Avg Score: ~91.73%
Precision Weighted Avg Score: ~91.74%

SVM Classifier For Test Sample Size: 4800
Accuracy Score: ~91.9%
Precision Macro Avg Score: ~91.94%
Precision Weighted Avg Score: ~91.94%

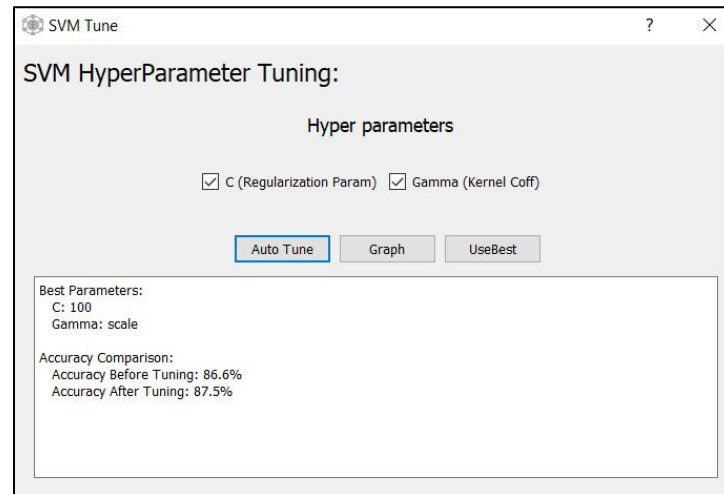
SVM Classifier For Test Sample Size: 4850
Accuracy Score: ~91.8%
Precision Macro Avg Score: ~91.82%
Precision Weighted Avg Score: ~91.85%

SVM Classifier For Test Sample Size: 4900

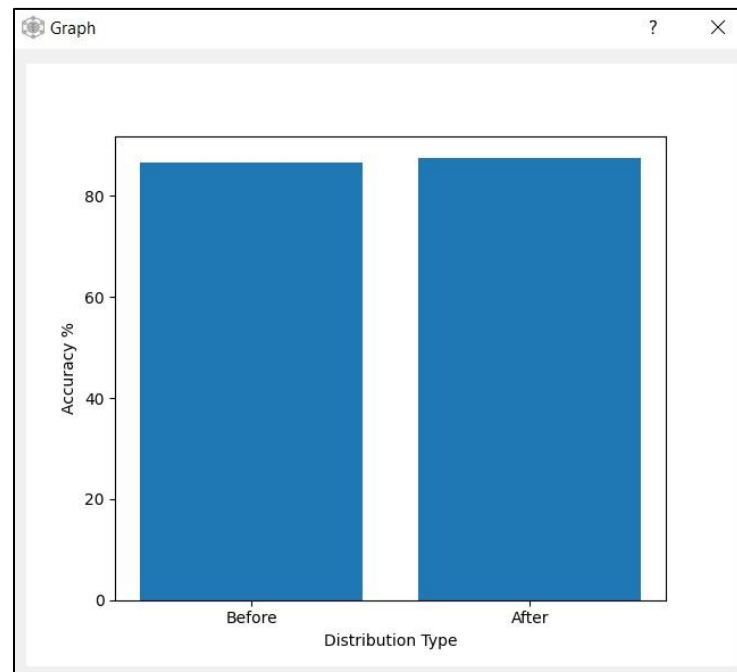
- By pressing Go we get output of SVM Classification results for all training sizes we have set and default hyperparameters:



- By pressing Hyper parameter Setup Button:

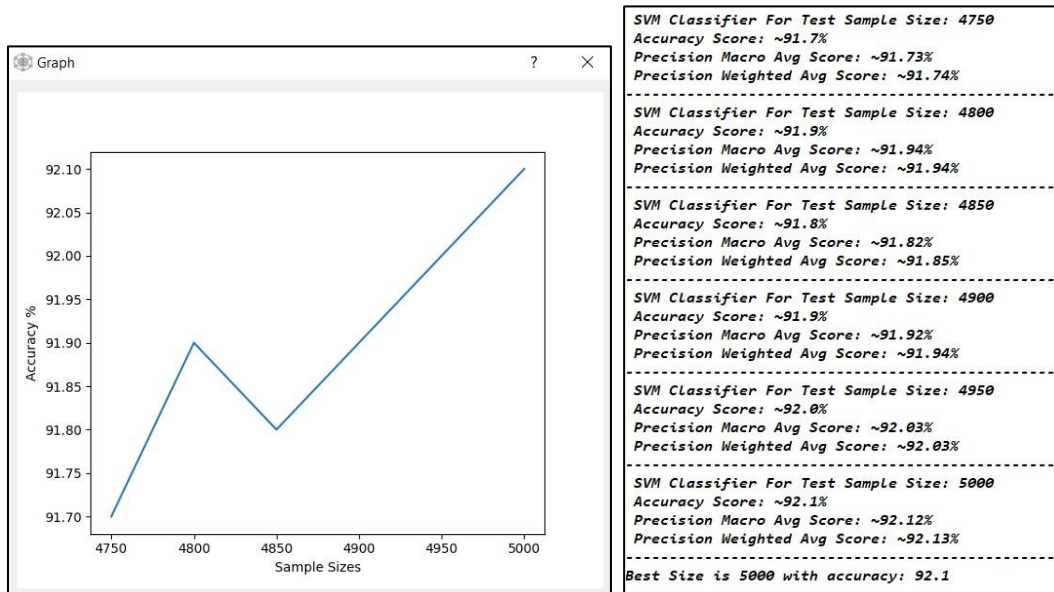


We get the following screen where we can auto tune MLP using list of values defined in code to get best parameters

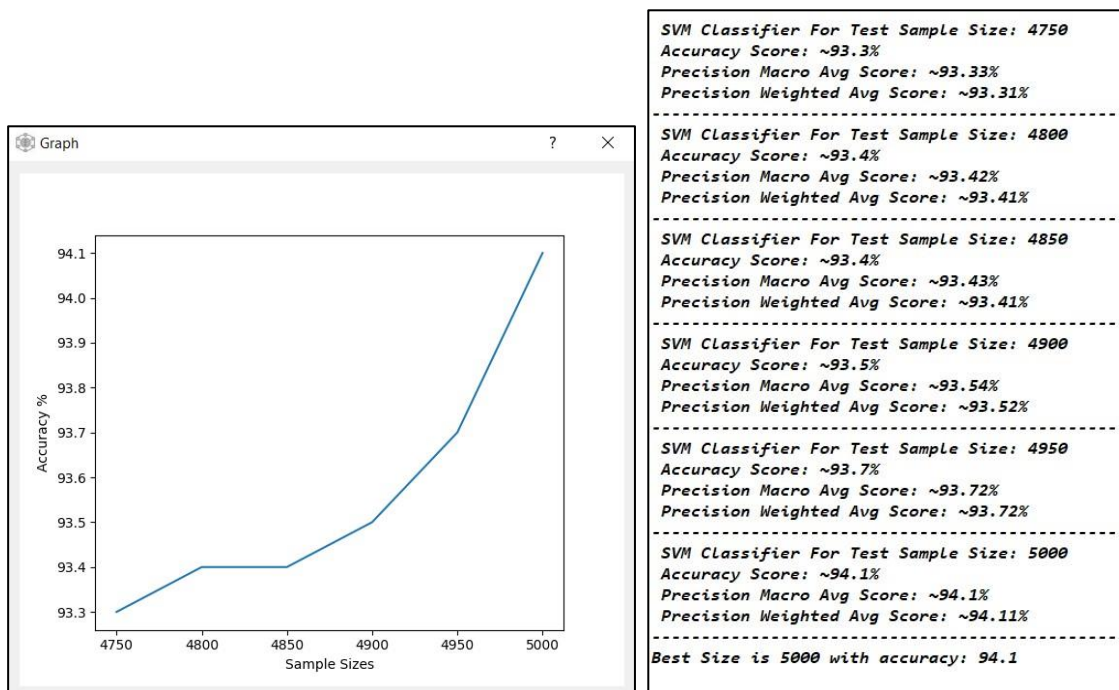


Note: By pressing UseBest Button you will return to main screen and the best parameters will be applied

Before:



After:



Decision Tree Functions:

Super Classifier

Classifier Type: Decision Tree

Hyper Parameters Setup: Setup

Data Sample : Digit Data

Training Sample Size:
 Starting Size: Ending Size: Step:

Testing Sample Size:
 Size:

Controls

Go
Clear Output
Reset

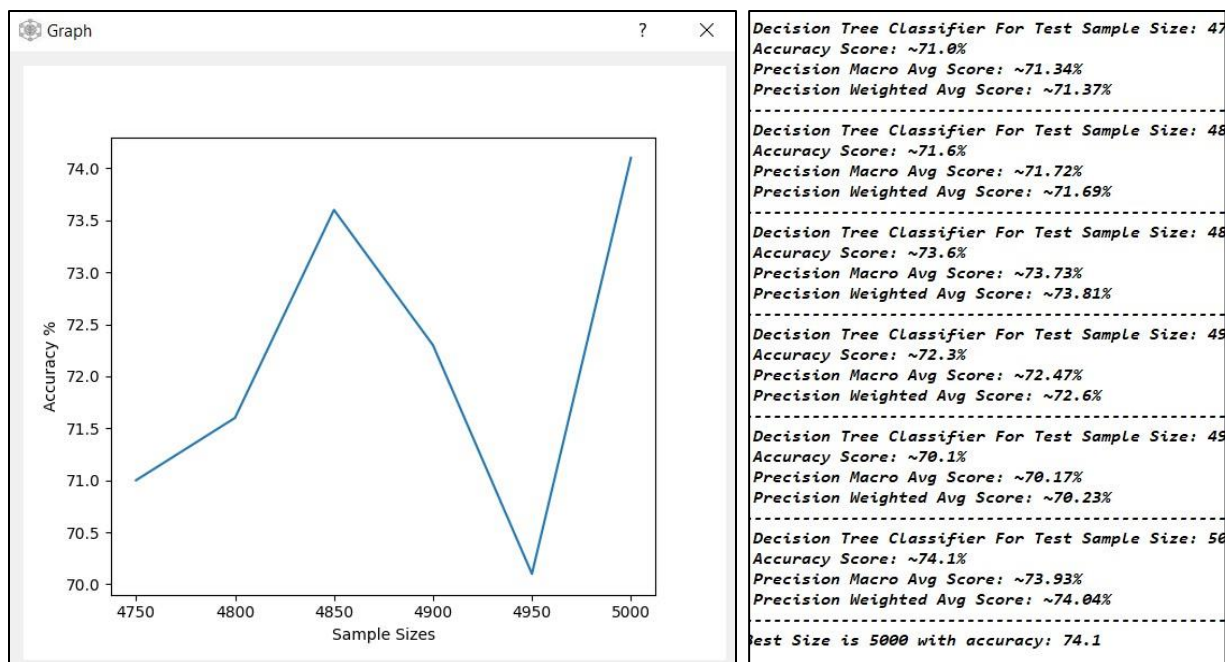
Decision Tree Classifier For Test Sample Size: 4900
 Accuracy Score: ~72.3%
 Precision Macro Avg Score: ~72.47%
 Precision Weighted Avg Score: ~72.6%

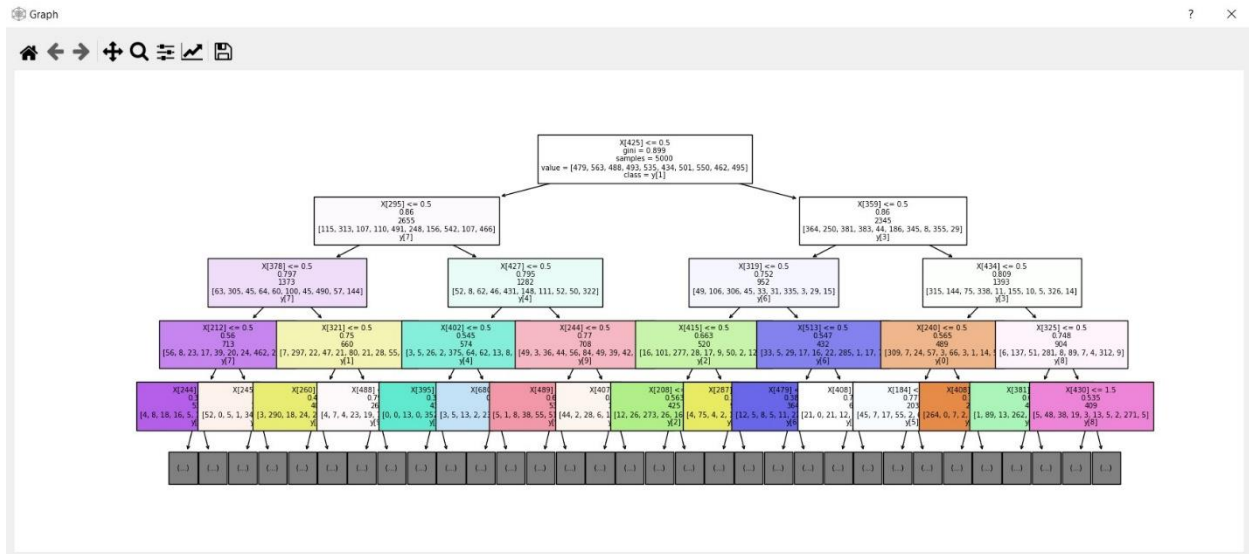
Decision Tree Classifier For Test Sample Size: 4950
 Accuracy Score: ~70.1%
 Precision Macro Avg Score: ~70.17%
 Precision Weighted Avg Score: ~70.23%

Decision Tree Classifier For Test Sample Size: 5000
 Accuracy Score: ~74.1%
 Precision Macro Avg Score: ~73.93%
 Precision Weighted Avg Score: ~74.04%

Best Size is 5000 with accuracy: 74.1

- By pressing Go we get output of Decision Tree Classification results for all training sizes we have set and default hyperparameters:





- By pressing Hyper parameter Setup Button:

Decision Tree HyperParameter Tuning:

Hyper parameters

Max Depth		Max Leaf Nodes	
Start	100	Start	100
End	300	End	300

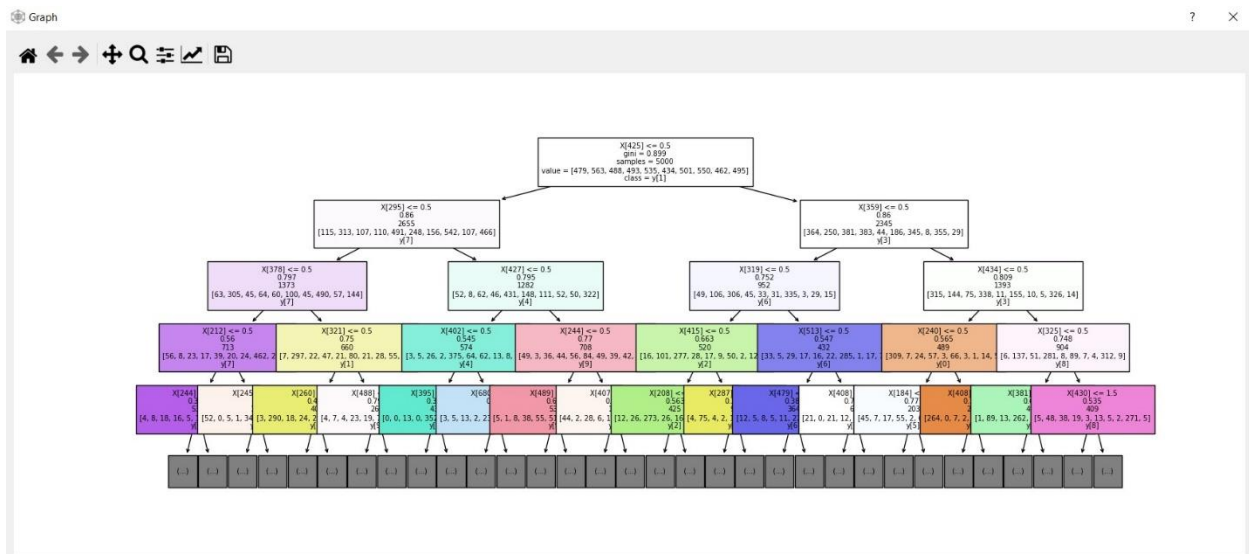
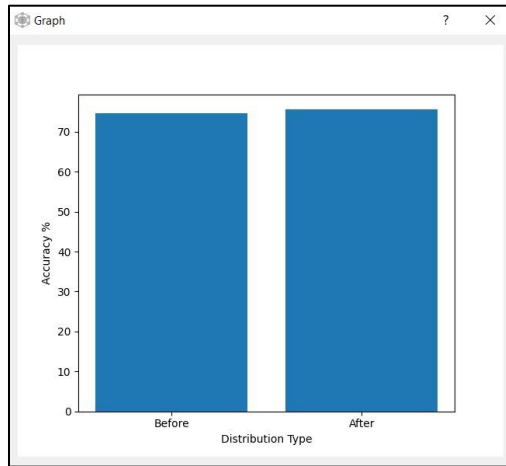
☒ Tune Criterion

Manual Tune Auto Tune Graph UseBest

Best Parameters:
 Criterion: gini
 Max Depth: 180
 Max Leaves: 280

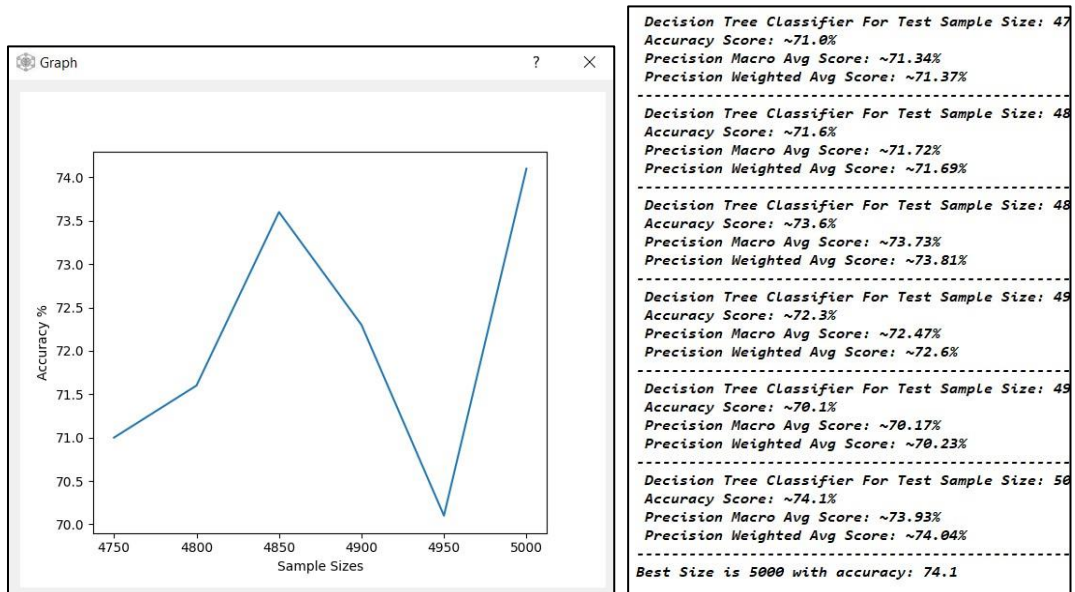
Accuracy Comparison:
 Accuracy Before Tuning: 74.6%
 Accuracy After Tuning: 75.6%

We get the following screen where we can auto tune Decision Tree either using list of values defined in code to get best parameters or using manually inputted values

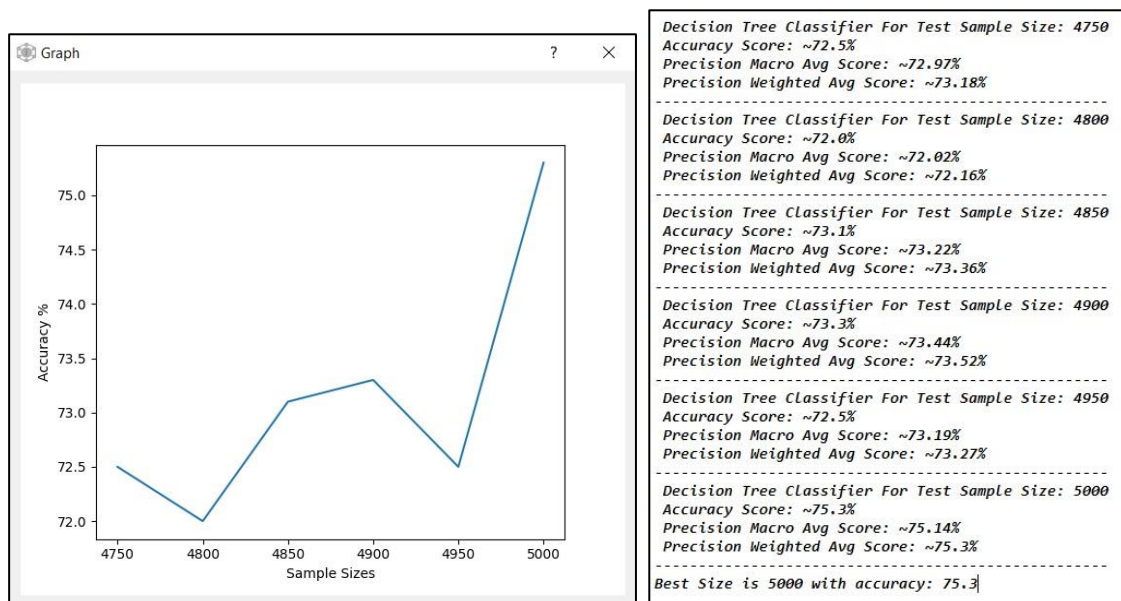


Note: By pressing UseBest Button you will return to main screen and the best parameters will be applied

Before:



After:



3)GitHub Link:

<https://github.com/Abduaws/Machine-Learning-Algorithms>