Introduction to Machine learning
Project Phase 2
Team 8

Name	ID
AbdulRaouf Monir Kamal Mahmoud	19P4442
Abdallah Amin Fahmy	20p5880
Michael Joseph Adeeb	1901075

Submission Date: 30/12/2023

Submitted to: Prof. Dr. Mahmoud Khalil

Eng. Omar Elessawy

Decision Trees

Steps on how to run project

1) Open Decision tree.ipynb and then run the first cell to import all the required libraries
2) Project is ready to run!

Screenshots of the Code including the output

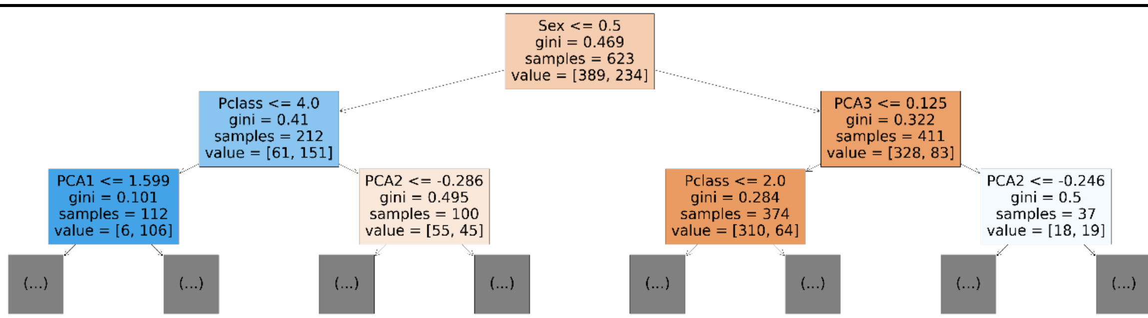
```
%pip install fast-ml
%pip install seaborn
%pip install tqdm
```

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
```

```python
train_dataset = pd.read_csv('updated_train.csv')
label_encoder = LabelEncoder()
train_dataset['Sex'] = label_encoder.fit_transform(train_dataset['Sex'])
```

```python
# hyperparameter optemization
from sklearn.model_selection import GridSearchCV
from fast_ml.model_development import train_valid_test_split
X_train, y_train, X_valid, y_valid, X_test, y_test = train_valid_test_split(train_dataset, target = 'Survived',
                                                                             train_size=0.7, valid_size=0.15, test_size=0.15)
```

```python
#defining and Fitting decision tree model
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.tree import plot_tree, export_text
plt.figure(figsize =(80,20))

plot_tree(classifier, feature_names=X_train.columns, max_depth=2, filled=True);
```




```python
#Tuning the hyperparameters
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50, 60, 70, 80],
    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 3, 4, 5, 6]
}
gs = GridSearchCV(classifier, param_grid=param_grid,scoring=["accuracy","f1","precision","recall","roc_auc"],cv=10,n_jobs=-1,refit="accuracy")
g_res = gs.fit(X_valid, y_valid)
g_res.best_score_
```

```
0.8137362637362637
```

```python
# get the hyperparameters with the best score
g_res.best_params_
```

```
{'criterion': 'gini',
 'max_depth': None,
 'min_samples_leaf': 3,
 'min_samples_split': 10}
```

These are the optimal hyperparameters for the Decision tree classifier

```python
y_pred = g_res.predict(X_test)
```

```python
# precesion score
from sklearn.metrics import precision_score
print("Precesion Score:",precision_score(y_test, y_pred, average='macro'))
```

```
Precesion Score: 0.7868487043744776
```

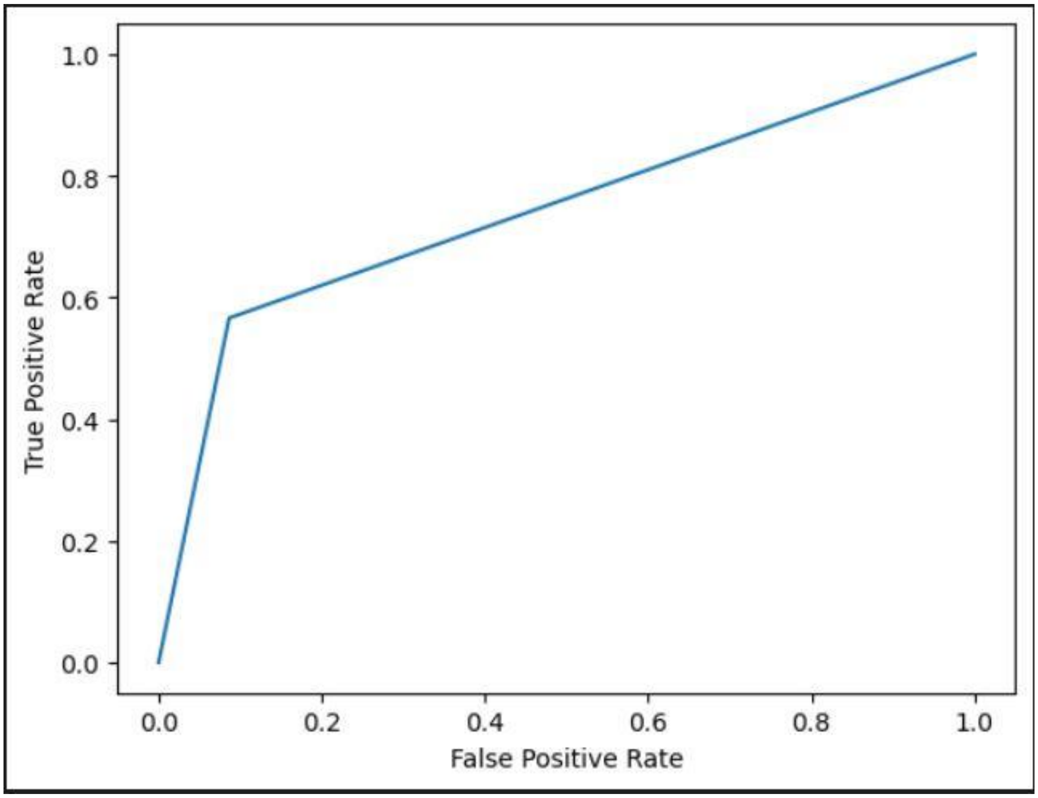
```
# Recall Score
from sklearn.metrics import recall_score
print("Recall Score:",recall_score(y_test, y_pred, average='macro'))
```

```
Recall Score: 0.7398089913813184
```

```
# F1-Score
from sklearn.metrics import f1_score
print("F1-Score:",f1_score(y_test, y_pred, average='macro'))
```

```
F1-Score: 0.749063670411985
```

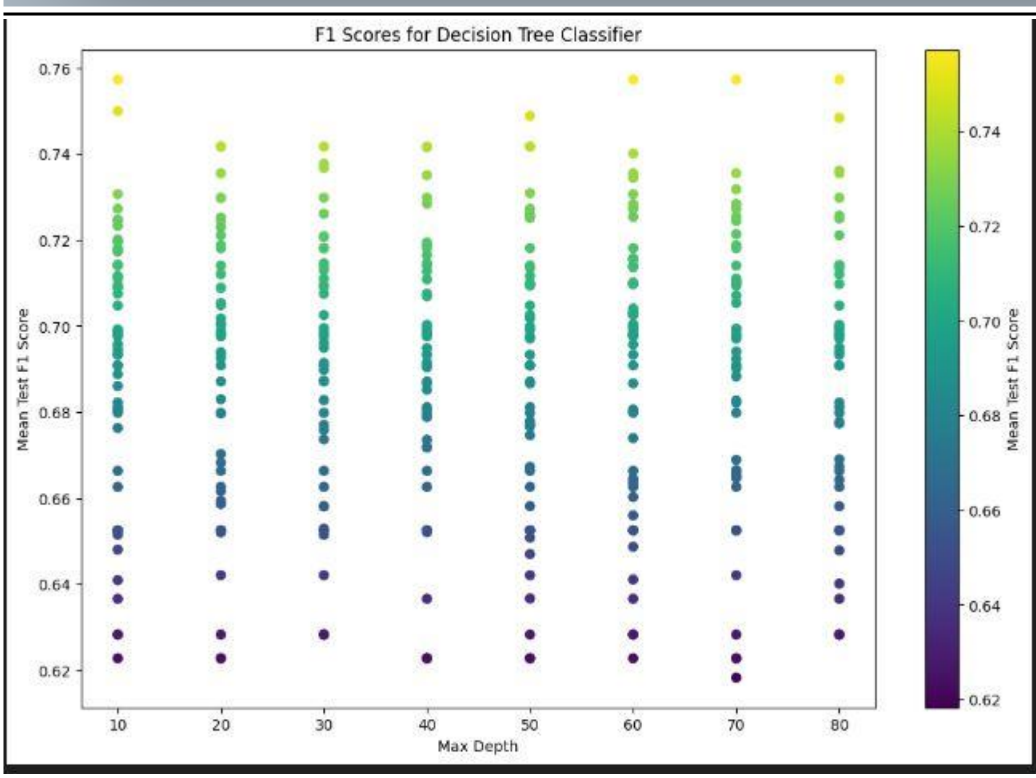
```
# ROC/AUC Curves
%matplotlib inline
from sklearn.metrics import roc_curve
def plot_roc_curve(true_y, y_prob):
    fpr, tpr, thresholds = roc_curve(true_y, y_prob)
    plt.plot(fpr, tpr)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
plot_roc_curve(y_test,y_pred)
```



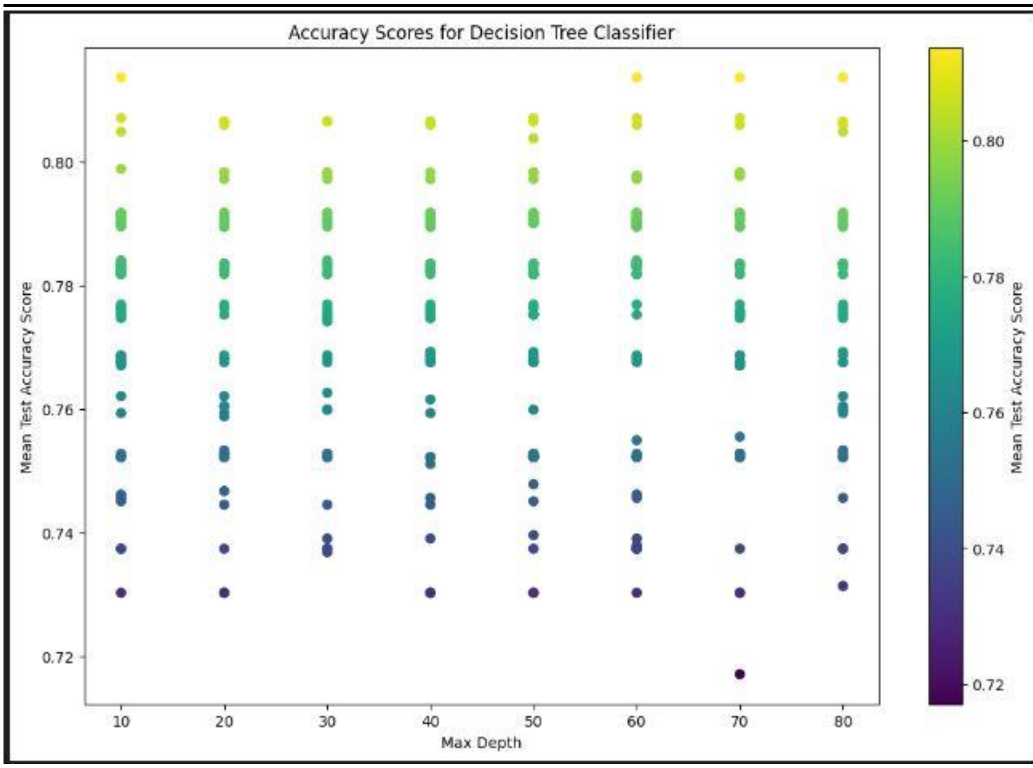

```python
# ROC AUC Score
from sklearn.metrics import roc_auc_score
r_a_score = roc_auc_score(y_test, y_pred)
print("ROC-AUC-Score:", r_a_score)
```

```
ROC-AUC-Score: 0.7398089913813184
```

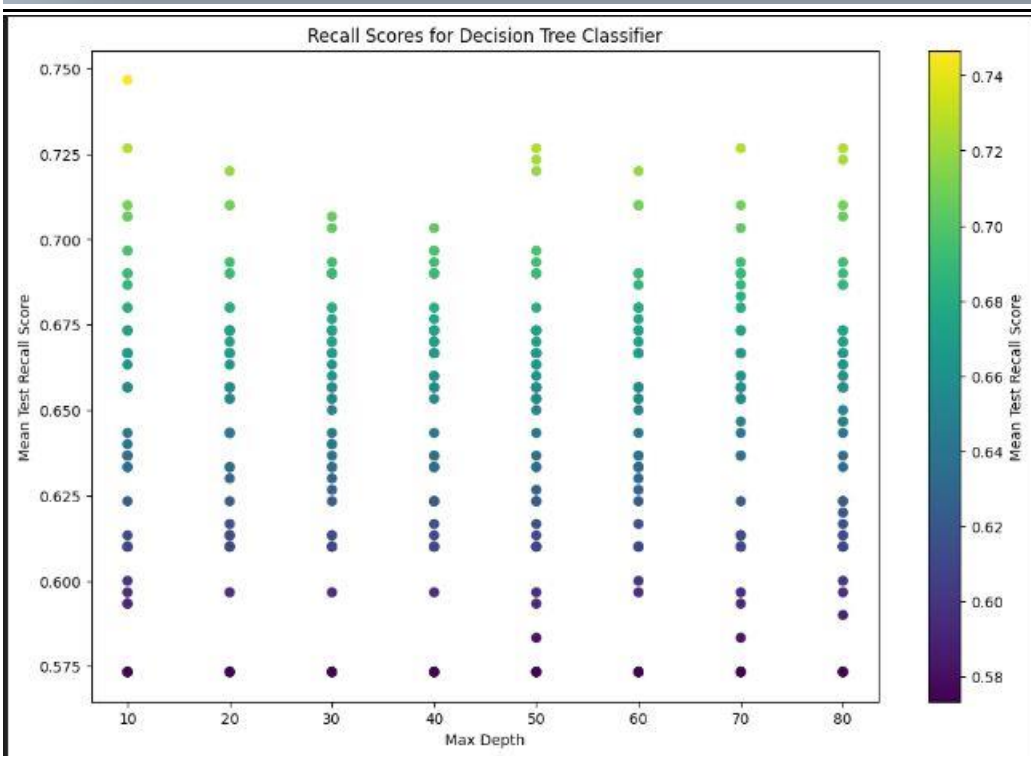
```python
# Extract the F1 scores and hyperparameters from the cv_results_
f1_scores = g_res.cv_results_['mean_test_f1']
params = g_res.cv_results_['params']

# Extract the values for each hyperparameter
max_depth_values = [param['max_depth'] for param in params]

# Create a 2D scatter plot to visualize the F1 scores
plt.figure(figsize=(12, 8))
plt.scatter(max_depth_values, f1_scores, c=f1_scores, cmap='viridis', marker='o')

plt.xlabel('Max Depth')
plt.ylabel('Mean Test F1 Score')
plt.title('F1 Scores for Decision Tree Classifier')

# Add a colorbar to the right of the plot
cbar = plt.colorbar()
cbar.set_label('Mean Test F1 Score')

plt.show()
```



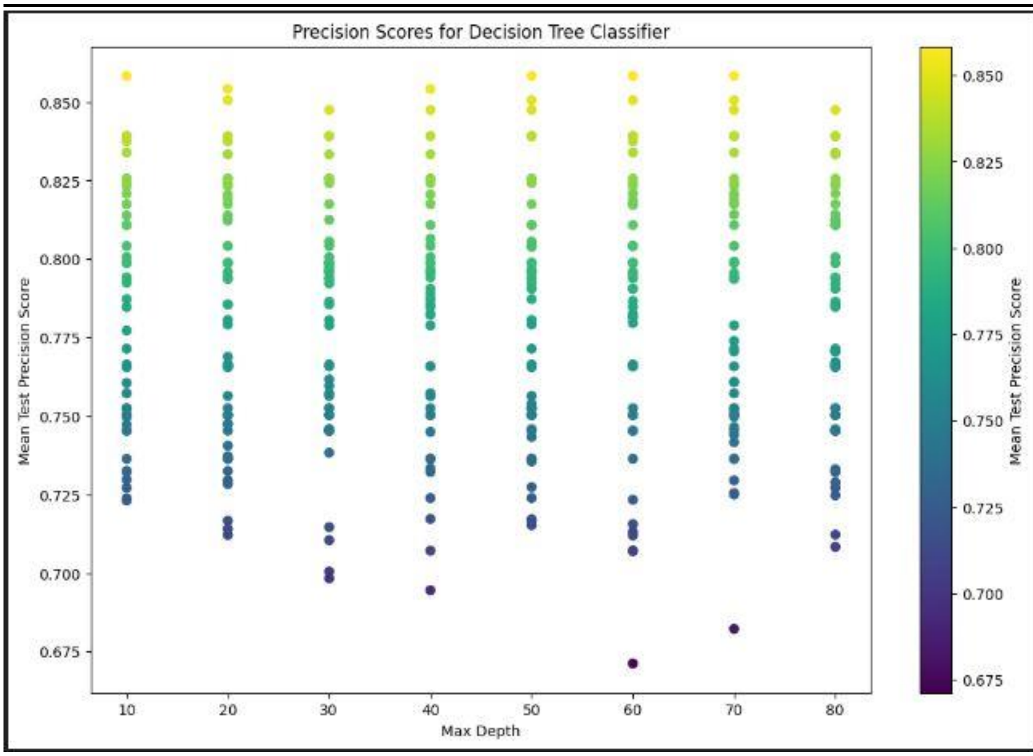
```python
# Extract the Accuracy scores and hyperparameters from the cv_results_
accuracy_scores = g_res.cv_results_['mean_test_accuracy']
params = g_res.cv_results_['params']

# Extract the values for each hyperparameter
max_depth_values = [param['max_depth'] for param in params]

# Create a 2D scatter plot to visualize the Accuracy scores
plt.figure(figsize=(12, 8))
plt.scatter(max_depth_values, accuracy_scores, c=accuracy_scores, cmap='viridis', marker='o')

plt.xlabel('Max Depth')
plt.ylabel('Mean Test Accuracy Score')
plt.title('Accuracy Scores for Decision Tree Classifier')

# Add a colorbar to the right of the plot
cbar = plt.colorbar()
cbar.set_label('Mean Test Accuracy Score')

plt.show()
```



```python
# Extract the Recall scores and hyperparameters from the cv_results_
recall_scores = g_res.cv_results_['mean_test_recall']
params = g_res.cv_results_['params']

# Extract the values for each hyperparameter
max_depth_values = [param['max_depth'] for param in params]

# Create a 2D scatter plot to visualize the Recall scores
plt.figure(figsize=(12, 8))
plt.scatter(max_depth_values, recall_scores, c=recall_scores, cmap='viridis', marker='o')

plt.xlabel('Max Depth')
plt.ylabel('Mean Test Recall Score')
plt.title('Recall Scores for Decision Tree Classifier')

# Add a colorbar to the right of the plot
cbar = plt.colorbar()
cbar.set_label('Mean Test Recall Score')

plt.show()

```



```python
# Extract the Precision scores and hyperparameters from the cv_results_
precision_scores = g_res.cv_results_['mean_test_precision']
params = g_res.cv_results_['params']

# Extract the values for each hyperparameter
max_depth_values = [param['max_depth'] for param in params]

# Create a 2D scatter plot to visualize the Precision scores
plt.figure(figsize=(12, 8))
plt.scatter(max_depth_values, precision_scores, c=precision_scores, cmap='viridis', marker='o')

plt.xlabel('Max Depth')
plt.ylabel('Mean Test Precision Score')
plt.title('Precision Scores for Decision Tree Classifier')

# Add a colorbar to the right of the plot
cbar = plt.colorbar()
cbar.set_label('Mean Test Precision Score')

plt.show()
```



Multilayer Perceptron

Steps on how to run project

1) Open Multilayer-Perceptron.ipynb and then run the first cell to import all the required libraries
2) Project is ready to run!

Code Screenshots and Details

```
%pip install fast-ml
%pip install scikit-learn
%pip install numpy
%pip install pandas
%pip install seaborn
```

First, we install all the required libraries for initializing and training our multilayer perceptron model.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

train_dataset = pd.read_csv('../updated_train.csv')
label_encoder = LabelEncoder()
train_dataset['Sex'] = label_encoder.fit_transform(train_dataset['Sex'])

X  = train_dataset.drop("Survived",axis=1)
y = train_dataset["Survived"]
```

Then we import our preprocessed dataset and encode the "Sex" Column as Multilayer perceptron only accepts numerical features. Afterwards, we split the data into input features (X) and the output (y)

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.85,test_size=0.15)
```

Then we split the input and output features into training and testing sets, but this time we won't use the fast-ml library, as there is no need to create a validation set, which we will see as progress on, will be generated by the multilayer perceptron model via the validation_factor parameter.

```
def generate_hidden_layer_sizes(num_sizes, min_nodes, max_nodes):
    return [(np.random.randint(min_nodes, max_nodes),) for _ in
range(num_sizes)]
```

This is a method we will use to generate a list of different hidden_layer_sizes (the number of hidden nodes per layer) that will be supplied to GridSearchCV to determine the optimal hyperparameter as per the supplied parameter grid.

```
import numpy as np

num_hidden_sizes = 10


min_nodes_per_layer = 50
max_nodes_per_layer = 250


hidden_layer_sizes = generate_hidden_layer_sizes(num_hidden_sizes,
min_nodes_per_layer, max_nodes_per_layer)
```

Now, we will call the generate_hidden_layer_sizes method to generate different hidden layer sizes as described above

```python
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    "activation": ["identity","logistic","tanh","relu",],
    "solver": ["lbfgs","sgd","adam",],
    "learning_rate": ["constant","invscaling","adaptive",],
    "early_stopping": [True, False,],
    "validation_fraction": [0.15,],
    "hidden_layer_sizes": hidden_layer_sizes
}

mlp = GridSearchCV(
    MLPClassifier(),
    param_grid=param_grid,
    scoring=["accuracy","f1","precision","recall","roc_auc"],
    n_jobs=-1,
    cv=2,
    refit="f1",
    verbose=True
      )

mlp.fit(X_train,y_train)
mlp.best_score_
```

Finally, we will call the GridSearchCV object to supply it with a multilayer perceptron object, and parameter grid that act our hyperparameters with almost each and every possible value of each hyperparameter, and only two folds for simplicity, and the refit is going to be based primarily on F1-score

Then we will print our best achieved score, which is approximately 75%

```
0.7466949109284875
```

```python
y_pred = mlp.predict(X_test)
```

In this line of code, we just create a variable that will be used for calculating precision, recall, f1-score, and roc/auc measures

```python
from sklearn.metrics import precision_score
print(f"Multilayer perceptron precision score:{precision_score(y_test, y_pred)}")
```

✓ 0.0s

```
Multilayer perceptron precision score:0.8333333333333334
```

```python
from sklearn.metrics import recall_score
print(f"Multilayer perceptron recall score:{recall_score(y_test, y_pred)}")
```

✓ 0.0s

```
Multilayer perceptron recall score:0.6862745098039216
```

```python
from sklearn.metrics import f1_score
print(f"Multilayer perceptron f1 score:{f1_score(y_test, y_pred)}")
```

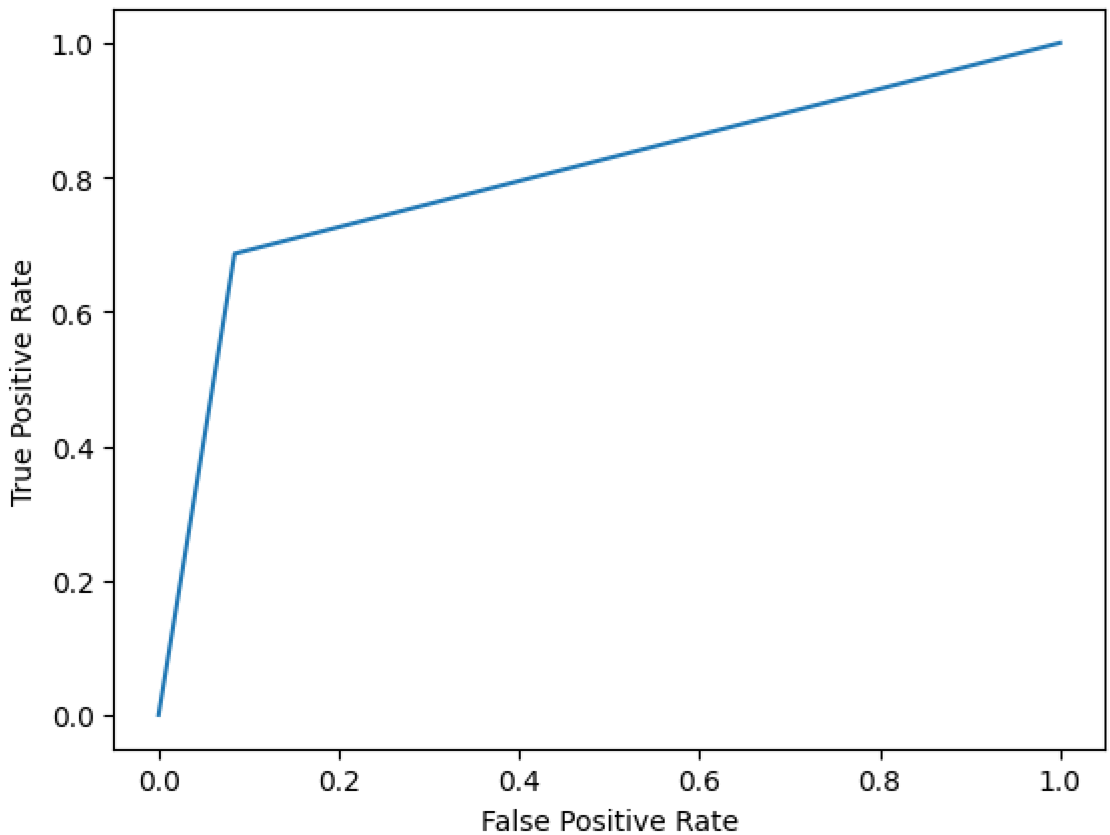
✓ 0.0s

```
Multilayer perceptron f1 score:0.7526881720430109
```

```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve

def plot_roc_curve(true_y, y_prob):

    fpr, tpr, thresholds = roc_curve(true_y, y_prob)
    plt.plot(fpr, tpr)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
```

Above is a function to plot roc/auc curve

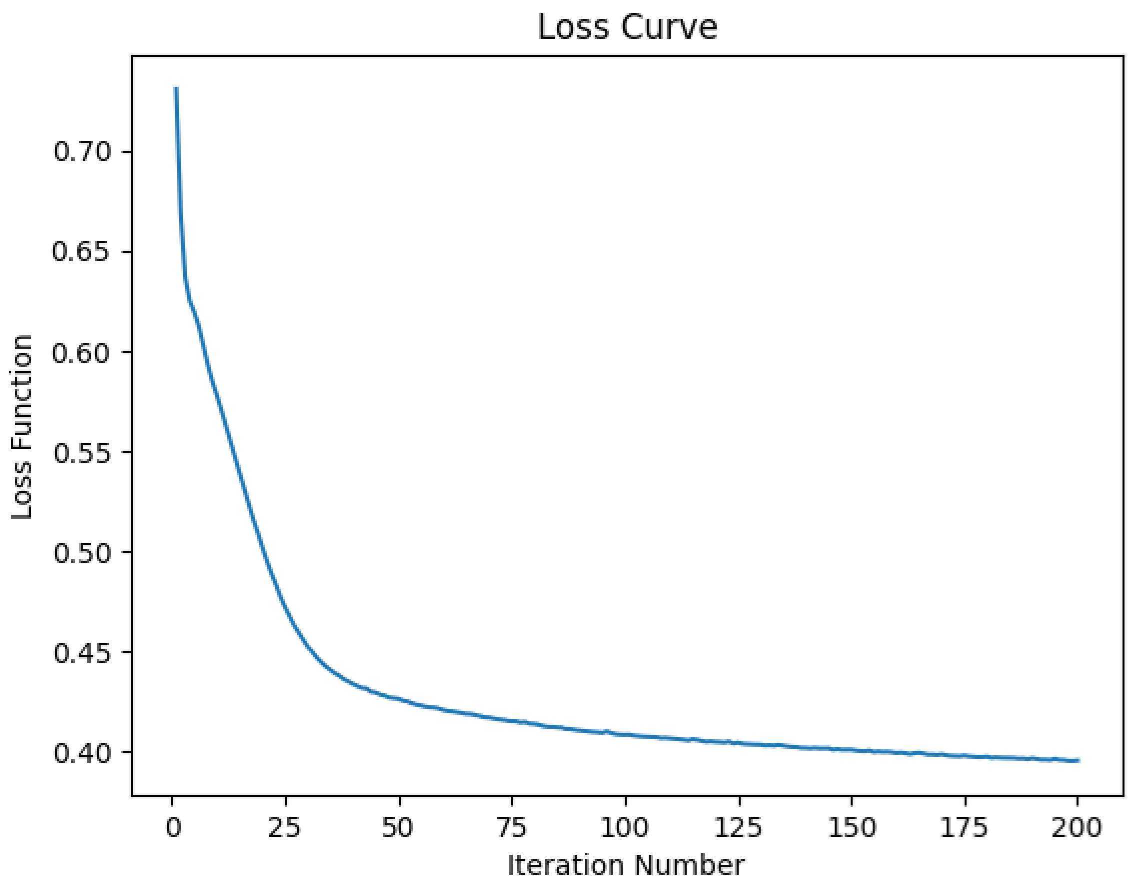
```python
from sklearn.metrics import roc_auc_score
plot_roc_curve(y_test, y_pred)
plt.show()
print(f'Multilayer perceptron AUC score: {roc_auc_score(y_test, y_pred)}')
```




```
Multilayer perceptron AUC score: 0.8009685802031656
```

```
import matplotlib.pyplot as plt
iterations = [i for i in range(1,mlp.best_estimator_.n_iter_+1)]
loss_curve = mlp.best_estimator_.loss_curve_

plt.plot(iterations,loss_curve)
plt.xlabel("Iteration Number")
plt.ylabel("Loss Function")
plt.title("Loss Curve")
```

This is a function to visualize the loss curve of our best estimator(model)

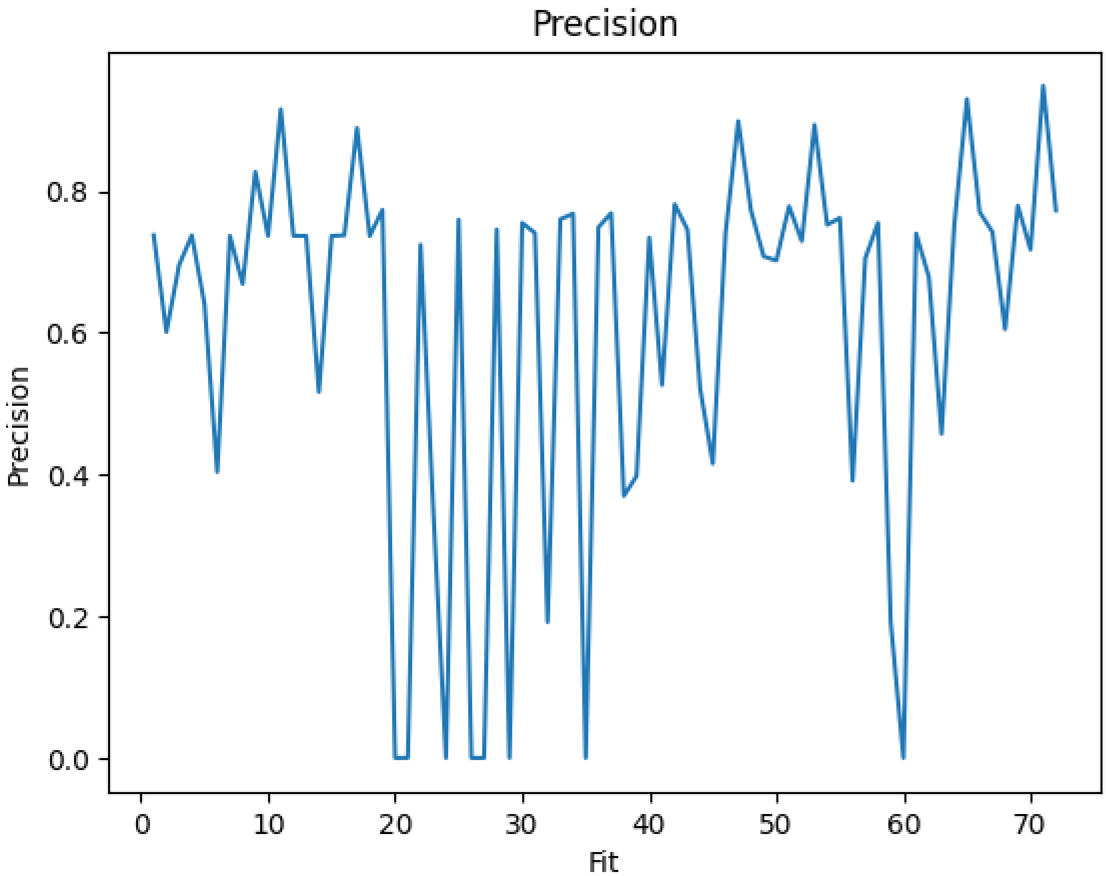



```
MLPClassifier
MLPClassifier(hidden_layer_sizes=(206,), learning_rate='invscaling',
              validation_fraction=0.15)
```

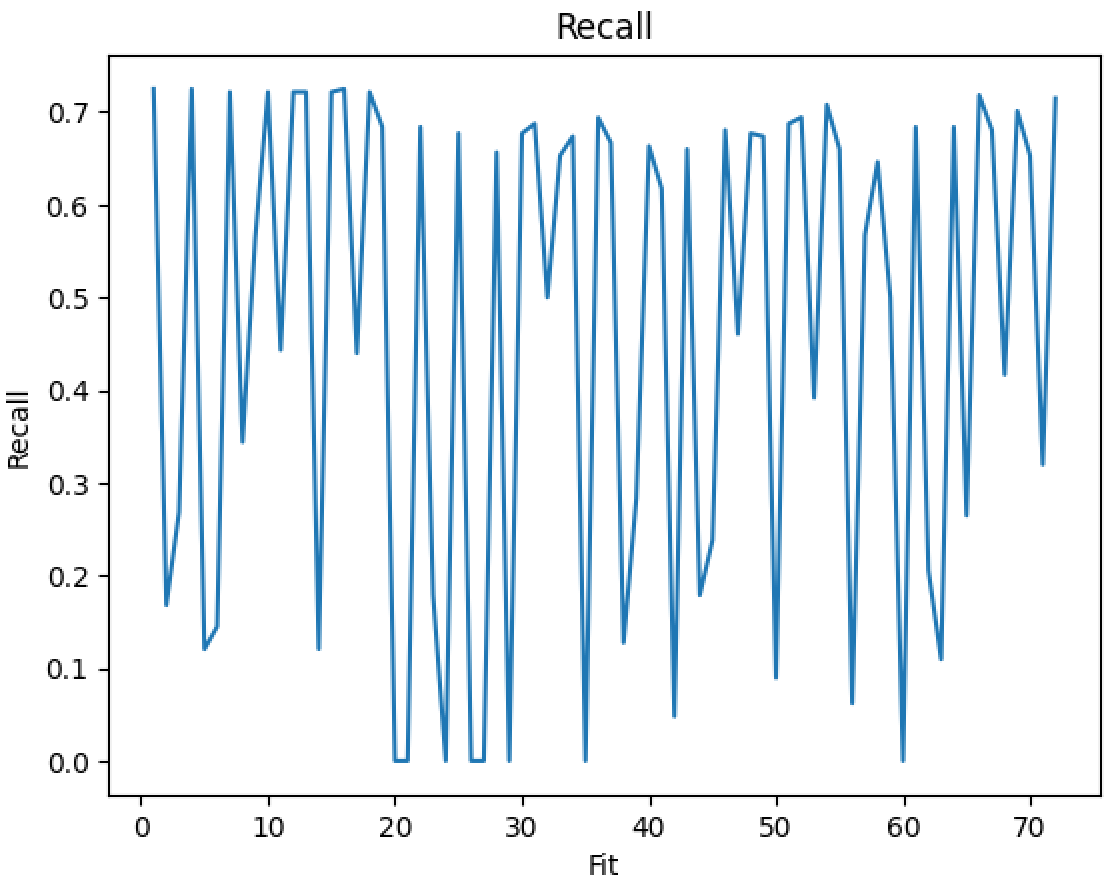
These are the properties of the best estimator as per the parameters in parameter grid, and are the **optimal hyperparameters**

Now we will visualize the fit against accuracy metric (precision, recall, f1-score, and roc/auc)

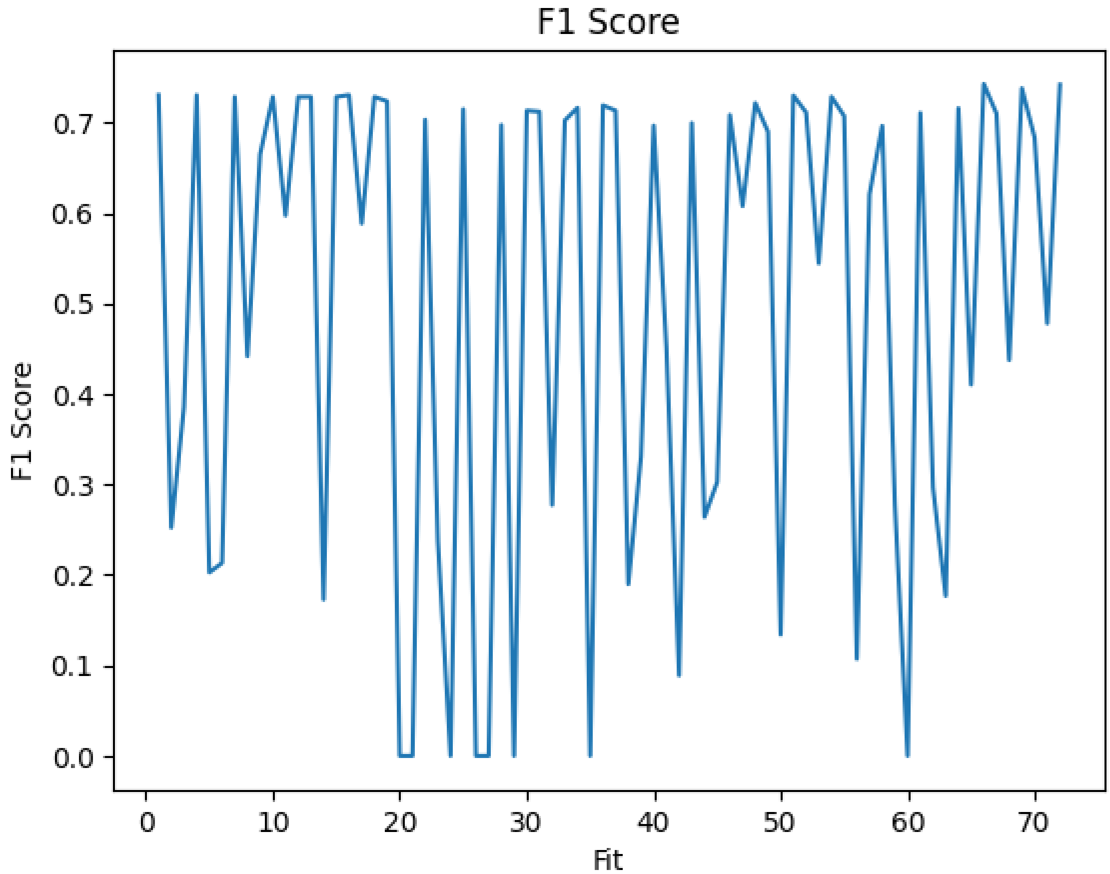
```
fits = [fit for fit in range(1,73)]
accuracy = mlp.cv_results_["mean_test_precision"][::10]
plt.plot(fits,accuracy)
plt.xlabel("Fit")
plt.ylabel("Precision")
plt.title("Precision")
```

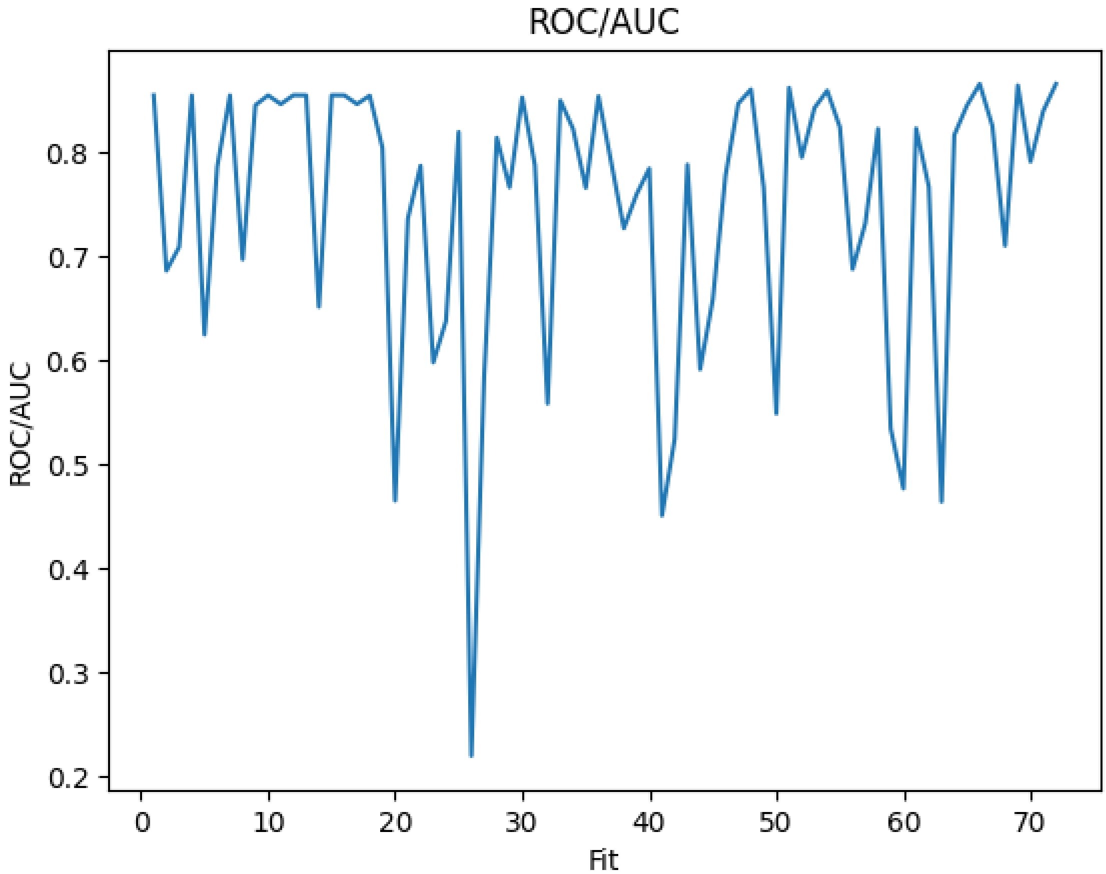


The code is similar for all other accuracy change metrics, note that many of the fits have been removed for better visualization of the accuracy change



Recall
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Recall
0
10
20
30
40
50
60
70
Fit


F1 Score
F1 Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
10
20
30
40
50
60
70
Fit


ROC/AUC
ROC/AUC
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
10
20
30
40
50
60
70
Fit