In [ ]:

```
# Import modules
```

In [7]:

```
import time
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import svm
```

In [ ]:

```
# dataset already splitted to three subsets; train, validation and test set
```

In [3]:

```
# load the data
# split the data into features and target variables
print ('#####...loading training data...####')
X_train = pd.read_excel('C:\\Users\\QuickPass\\Documents\\ML\\Sorted\\X_train.xlsx')
y_train = pd.read_excel('C:\\Users\\QuickPass\\Documents\\ML\\Sorted\\y_train.xlsx').values.ravel()

print ('#####...loading validation data...####')
X_valid = pd.read_excel('C:\\Users\\QuickPass\\Documents\\ML\\Sorted\\X_val.xlsx')
y_valid = pd.read_excel('C:\\Users\\QuickPass\\Documents\\ML\\Sorted\\y_val.xlsx').values.ravel()

print ('#####...loading training data...####')
X_test = pd.read_excel('C:\\Users\\QuickPass\\Documents\\ML\\Sorted\\X_test.xlsx')
y_test = pd.read_excel('C:\\Users\\QuickPass\\Documents\\ML\\Sorted\\y_test.xlsx').values.ravel()

print ("#####.... data subsets are ready for feature extraction...#####")
```

```
#####...loading training data...####
#####...loading validation data...####
#####...loading training data...####
#####.... data subsets are ready for feature extraction...#####
```

In [4]:

```
# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

```
print("#####... data standardization finished! ...#####")
```

```
#####... data standardization finished! ...#####
```

In [ ]:

```
# Create the model object and define parametes for the GridSearchCV
```

In [8]:

```
# Create an MLP classifier object
mlp = MLPClassifier(solver='adam', max_iter=1000)

# Define the parameters for the MLP model
parameters = {'hidden_layer_sizes': [(10,), (50,), (100,), (10, 10), (50, 50), (100, 100)],
              'activation': ['relu', 'logistic'],
              'alpha': [0.0001, 0.001, 0.01, 0.1]}

# Create a GridSearchCV object
clf = GridSearchCV(mlp, parameters)

# Fit the GridSearchCV object on the training set
clf.fit(X_train, y_train)
```

Out[8]:

```
GridSearchCV(estimator=MLPClassifier(max_iter=1000),
             param_grid={'activation': ['relu', 'logistic'],
                         'alpha': [0.0001, 0.001, 0.01, 0.1],
                         'hidden_layer_sizes': [(10,), (50,), (100,), (10, 10),
                                                (50, 50), (100, 100)]})
```

In [ ]:

```
# Check for the best estimator and asscociated parameters
```

In [10]:

```
# Print the best estimator and its score on the validation set
print('Best estimator:', clf.best_estimator_)
print('Best parameters:', clf.best_params_)
print('Accuracy on validation set:', clf.score(X_valid, y_valid))

# Calculate the score on the testing set
test_score = clf.score(X_test, y_test)
print('Accuracy on testing set:', test_score)
```

```
Best estimator: MLPClassifier(activation='logistic', max_iter=1000)
Best parameters: {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100,)}
Accuracy on validation set: 0.9951690821256038
Accuracy on testing set: 0.9947217749150117
```

In [11]:

```
# Extract the results of the grid search
cv_results = clf.cv_results_
```

```
print (cv_results)
```

```
{'mean_fit_time': array([12.72303486,  9.06692982, 11.7329453 , 12.73663144,  8.57287655,
       10.19684825, 13.73420506, 10.75904369, 11.29897962, 15.6505414 ,
        7.85309334, 10.85590854, 13.46751447,  9.03234425,  9.39909697,
       15.43689017, 13.66957712, 24.53363285, 15.08982997, 14.60832858,
       16.77805052, 16.8533577 , 24.54235129, 32.513972  , 44.85297489,
       25.63420243, 26.79834299, 43.78121285, 22.93658767, 29.23388405,
       38.63304825, 26.25831723, 28.49579344, 47.01573563, 23.70889235,
       28.75939736, 38.02882566, 25.65979218, 34.918502  , 37.87926526,
       32.0619792 , 40.51466031, 20.72898188, 26.97130213, 40.66204352,
       26.90215712, 36.53327761, 55.67373853]), 'std_fit_time': array([1.40176845, 0.61335834, 0.94937651, 2.04139027,
1.03945229,
       1.90684997, 2.75813782, 2.05333955, 1.65133904, 1.96021788,
       1.06649535, 1.79669575, 2.67831832, 1.11864621, 0.47643191,
       1.38425352, 2.72068731, 7.60835398, 1.83467947, 2.34434098,
       1.67487591, 3.20663305, 3.9414702 , 4.39406283, 6.83515876,
       2.17103573, 3.63678291, 3.40903829, 1.37525804, 2.03302622,
       2.64535149, 3.50306813, 3.12589835, 6.53702025, 3.14906098,
       2.64985511, 4.50213006, 2.38685839, 3.02130708, 5.09499891,
       5.80465031, 1.75371383, 2.66058315, 3.23082604, 7.89197632,
       1.22714296, 3.0008348 , 6.24986051]), 'mean_score_time': array([0.00250673, 0.00504837, 0.00763397, 0.00282178,
0.00796371,
       0.0156723 , 0.0025003 , 0.00565124, 0.00885754, 0.00300102,
       0.00760121, 0.01422019, 0.00249348, 0.00542789, 0.00920596,
       0.00324821, 0.01482182, 0.02071767, 0.00305438, 0.00884118,
       0.01354938, 0.00476112, 0.01266985, 0.01932626, 0.00401273,
       0.00940666, 0.01767392, 0.00435977, 0.01818309, 0.02745242,
       0.00349956, 0.01489625, 0.01567349, 0.00597644, 0.01873164,
       0.03910937, 0.00450239, 0.00991011, 0.01510296, 0.0070374 ,
       0.01786656, 0.02634096, 0.00475836, 0.00904832, 0.01992674,
       0.00499611, 0.01523757, 0.03283534]), 'std_score_time': array([0.00044212, 0.00133563, 0.0002021 , 0.00041313,
0.00077526,
       0.00269648, 0.00031658, 0.00106589, 0.00274155, 0.00063   ,
       0.00115758, 0.00096485, 0.00030281, 0.00038537, 0.00179878,
       0.00079353, 0.00339922, 0.00564509, 0.00099746, 0.00310069,
       0.00511575, 0.00140693, 0.00380446, 0.00677951, 0.00150259,
       0.00321078, 0.00678528, 0.00121907, 0.00611224, 0.00934202,
       0.00205039, 0.00385996, 0.00405245, 0.00255576, 0.00593888,
       0.01290597, 0.00145294, 0.00382041, 0.0036898 , 0.00268895,
       0.00791643, 0.00644616, 0.00149273, 0.00285299, 0.00503377,
       0.00178374, 0.0057039 , 0.0069716 ]), 'param_activation': masked_array(data=['relu', 'relu', 'relu', 'relu', 're
lu', 'relu', 'relu',
                   'relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'relu',
                   'relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'relu',
                   'relu', 'relu', 'relu', 'logistic', 'logistic',
                   'logistic', 'logistic', 'logistic', 'logistic',
                   'logistic', 'logistic', 'logistic', 'logistic',
                   'logistic', 'logistic', 'logistic', 'logistic',
                   'logistic', 'logistic', 'logistic', 'logistic',
                   'logistic', 'logistic', 'logistic', 'logistic',
                   'logistic', 'logistic'],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object), 'param_alpha': masked_array(data=[0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.001,
                   0.001, 0.001, 0.001, 0.001, 0.001, 0.01, 0.01, 0.01,
                   0.01, 0.01, 0.01, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.0001,
                   0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.001, 0.001,
                   0.001, 0.001, 0.001, 0.001, 0.01, 0.01, 0.01, 0.01,
                   0.01, 0.01, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object), 'param_hidden_layer_sizes': masked_array(data=[(10,), (50,), (100,), (10, 10), (50, 50), (10
0, 100),
                   (10,), (50,), (100,), (10, 10), (50, 50), (100, 100),
                   (10,), (50,), (100,), (10, 10), (50, 50), (100, 100),
                   (10,), (50,), (100,), (10, 10), (50, 50), (100, 100),
                   (10,), (50,), (100,), (10, 10), (50, 50), (100, 100),
                   (10,), (50,), (100,), (10, 10), (50, 50), (100, 100),
                   (10,), (50,), (100,), (10, 10), (50, 50), (100, 100),
                   (10,), (50,), (100,), (10, 10), (50, 50), (100, 100)],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (10,)}, {'activatio
n': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,)}, {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_size
s': (100,)}, {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 10)}, {'activation': 'relu', 'alpha':
0.0001, 'hidden_layer_sizes': (50, 50)}, {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 100)}, {'a
```

```
ctivation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (10,)}, {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_
sizes': (50,)}, {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (100,)}, {'activation': 'relu', 'alpha':
0.001, 'hidden_layer_sizes': (10, 10)}, {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (50, 50)}, {'activ
ation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (100, 100)}, {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_
sizes': (10,)}, {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (50,)}, {'activation': 'relu', 'alpha': 0.0
1, 'hidden_layer_sizes': (100,)}, {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (10, 10)}, {'activation':
'relu', 'alpha': 0.01, 'hidden_layer_sizes': (50, 50)}, {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (10
0, 100)}, {'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (10,)}, {'activation': 'relu', 'alpha': 0.1, 'hidd
en_layer_sizes': (50,)}, {'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (100,)}, {'activation': 'relu', 'al
pha': 0.1, 'hidden_layer_sizes': (10, 10)}, {'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (50, 50)}, {'act
ivation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (100, 100)}, {'activation': 'logistic', 'alpha': 0.0001, 'hidden_
layer_sizes': (10,)}, {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50,)}, {'activation': 'logisti
c', 'alpha': 0.0001, 'hidden_layer_sizes': (100,)}, {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes':
(10, 10)}, {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50)}, {'activation': 'logistic', 'alp
ha': 0.0001, 'hidden_layer_sizes': (100, 100)}, {'activation': 'logistic', 'alpha': 0.001, 'hidden_layer_sizes': (1
0,)}, {'activation': 'logistic', 'alpha': 0.001, 'hidden_layer_sizes': (50,)}, {'activation': 'logistic', 'alpha': 0.00
1, 'hidden_layer_sizes': (100,)}, {'activation': 'logistic', 'alpha': 0.001, 'hidden_layer_sizes': (10, 10)}, {'activat
ion': 'logistic', 'alpha': 0.001, 'hidden_layer_sizes': (50, 50)}, {'activation': 'logistic', 'alpha': 0.001, 'hidden_l
ayer_sizes': (100, 100)}, {'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (10,)}, {'activation': 'logis
tic', 'alpha': 0.01, 'hidden_layer_sizes': (50,)}, {'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (10
0,)}, {'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (10, 10)}, {'activation': 'logistic', 'alpha': 0.
01, 'hidden_layer_sizes': (50, 50)}, {'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (100, 100)}, {'act
ivation': 'logistic', 'alpha': 0.1, 'hidden_layer_sizes': (10,)}, {'activation': 'logistic', 'alpha': 0.1, 'hidden_laye
r_sizes': (50,)}, {'activation': 'logistic', 'alpha': 0.1, 'hidden_layer_sizes': (100,)}, {'activation': 'logistic', 'a
lpha': 0.1, 'hidden_layer_sizes': (10, 10)}, {'activation': 'logistic', 'alpha': 0.1, 'hidden_layer_sizes': (50, 50)},
{'activation': 'logistic', 'alpha': 0.1, 'hidden_layer_sizes': (100, 100)}], 'split0_test_score': array([0.96809304, 0.
99030863, 0.99463247, 0.97599523, 0.99433428,
       0.9928433 , 0.97733711, 0.99224691, 0.99075593, 0.97226778,
       0.99135232, 0.99418518, 0.97495154, 0.9929924 , 0.99358879,
       0.96839123, 0.99209781, 0.99507977, 0.971075  , 0.99179961,
       0.9928433 , 0.9780826 , 0.9925451 , 0.99418518, 0.96093634,
       0.99329059, 0.99478157, 0.97301327, 0.99120322, 0.99433428,
       0.95989265, 0.99433428, 0.99567616, 0.97032951, 0.99030863,
       0.99388698, 0.96615476, 0.992396  , 0.99478157, 0.96511108,
       0.99179961, 0.992396  , 0.96675116, 0.98762487, 0.98747577,
       0.97137319, 0.99045773, 0.99224691]), 'split1_test_score': array([0.96645296, 0.98986134, 0.98121366, 0.9697331
1, 0.98345013,
       0.99329059, 0.96660206, 0.99150142, 0.99060683, 0.97375876,
       0.98673028, 0.9926942 , 0.97331147, 0.98777397, 0.98717758,
       0.97196958, 0.98956314, 0.98702848, 0.96555837, 0.98956314,
       0.98986134, 0.97450425, 0.98673028, 0.98956314, 0.96660206,
       0.99060683, 0.99433428, 0.96108543, 0.98732667, 0.99165051,
       0.9637692 , 0.99150142, 0.99358879, 0.95899806, 0.98941405,
       0.99075593, 0.96287461, 0.99090502, 0.99403608, 0.96526018,
       0.98956314, 0.99150142, 0.95064858, 0.98717758, 0.98807216,
       0.96719845, 0.9856866 , 0.98687938]), 'split2_test_score': array([0.97405309, 0.98941247, 0.99209663, 0.9685356
4, 0.98926335,
       0.99343871, 0.97241276, 0.99239487, 0.99269311, 0.97599165,
       0.99045631, 0.99358783, 0.97062332, 0.99284223, 0.98836863,
       0.97584253, 0.99224575, 0.99224575, 0.97494781, 0.99090367,
       0.99299135, 0.97479869, 0.99030719, 0.99358783, 0.968983  ,
       0.99403519, 0.99522815, 0.96242171, 0.99000895, 0.99194751,
       0.96436027, 0.99299135, 0.99388607, 0.96152699, 0.99164927,
       0.99179839, 0.96271995, 0.99209663, 0.99343871, 0.96048315,
       0.99358783, 0.99463167, 0.95884283, 0.98762302, 0.98732478,
       0.96704444, 0.98926335, 0.99060543]), 'split3_test_score': array([0.97643901, 0.99120191, 0.99194751, 0.9746495
7, 0.99179839,
       0.98985983, 0.96152699, 0.98881599, 0.99015807, 0.9674918 ,
       0.98792126, 0.99254399, 0.96570236, 0.98956159, 0.99179839,
       0.96450939, 0.99135103, 0.99328959, 0.97256188, 0.99075455,
       0.99284223, 0.97181628, 0.99030719, 0.99209663, 0.96152699,
       0.99239487, 0.99492991, 0.97524605, 0.98926335, 0.99314047,
       0.96361467, 0.99120191, 0.99463167, 0.9712198 , 0.99060543,
       0.99269311, 0.96122875, 0.99239487, 0.99418431, 0.97271101,
       0.99328959, 0.99269311, 0.96212347, 0.98702654, 0.99030719,
       0.97360573, 0.99030719, 0.99105279]), 'split4_test_score': array([0.96928124, 0.98687742, 0.99179839, 0.9668953
2, 0.99135103,
       0.99239487, 0.9712198 , 0.98881599, 0.99328959, 0.96540412,
       0.98956159, 0.99090367, 0.97211452, 0.99000895, 0.99150015,
       0.9719654 , 0.98896511, 0.99284223, 0.96779004, 0.98896511,
       0.99090367, 0.97345661, 0.98553534, 0.98553534, 0.96570236,
       0.99120191, 0.99403519, 0.95809723, 0.98911423, 0.98807038,
       0.94750969, 0.99030719, 0.99269311, 0.97464957, 0.99060543,
       0.99209663, 0.96137787, 0.99254399, 0.99239487, 0.96838652,
       0.99254399, 0.99254399, 0.96212347, 0.98553534, 0.98568446,
       0.96943036, 0.98538622, 0.98896511]), 'mean_test_score': array([0.97086387, 0.98953235, 0.99033773, 0.97116177,
0.99003943,
       0.99236546, 0.96981975, 0.99075503, 0.9915007 , 0.97098282,
       0.98920435, 0.99278297, 0.97134064, 0.99063583, 0.99048671,
       0.97053563, 0.99084457, 0.99209716, 0.97038662, 0.99039722,
       0.99188838, 0.97453169, 0.98908502, 0.99099363, 0.96475015,
       0.99230588, 0.99466182, 0.96597274, 0.98938328, 0.99182863,
       0.9598293 , 0.99206723, 0.99409516, 0.96734479, 0.99051656,
       0.99224621, 0.96287119, 0.9920673 , 0.99376711, 0.96639039,
       0.99215683, 0.99275324, 0.9600979 , 0.98699747, 0.98777287,
       0.96973043, 0.98822022, 0.98994992]), 'std_test_score': array([0.00376541, 0.00145336, 0.00468   , 0.00354026,
0.00366827,
       0.001305  , 0.00537249, 0.00161193, 0.00124726, 0.00394346,
       0.00167786, 0.00111452, 0.00315679, 0.00200793, 0.00235827,
       0.00382564, 0.00133902, 0.00270505, 0.00334856, 0.0010097 ,
       0.00127411, 0.00205962, 0.00257311, 0.00316138, 0.00307204,
       0.00127072, 0.00042589, 0.00684226, 0.00126649, 0.00210568,
       0.00635859, 0.00142549, 0.00100531, 0.0060133 , 0.00071521,
       0.00103303, 0.00177383, 0.00059905, 0.00080826, 0.00404337,
       0.00143775, 0.0010267 , 0.00535419, 0.0007689 , 0.00149418,
       0.00250727, 0.00223163, 0.00186119]), 'rank_test_score': array([37, 26, 23, 35, 24,  6, 40, 18, 15, 36, 28,  4,
34, 19, 21, 38, 17,
       10, 39, 22, 13, 33, 29, 16, 45,  7,  1, 44, 27, 14, 48, 12,  2, 42,
       20,  8, 46, 11,  3, 43,  9,  5, 47, 32, 31, 41, 30, 25])}
```

In [14]:

```
mean_test_scores = cv_results['mean_test_score']
params = cv_results['params']
hidden_layer_sizes = [params[i]['hidden_layer_sizes'] for i in range(len(params))]
activation = [params[i]['activation'] for i in range(len(params))]
alpha = [params[i]['alpha'] for i in range(len(params))]

print (mean_test_scores)
```

```
[0.97086387 0.98953235 0.99033773 0.97116177 0.99003943 0.99236546
 0.96981975 0.99075503 0.9915007  0.97098282 0.98920435 0.99278297
 0.97134064 0.99063583 0.99048671 0.97053563 0.99084457 0.99209716
 0.97038662 0.99039722 0.99188838 0.97453169 0.98908502 0.99099363
 0.96475015 0.99230588 0.99466182 0.96597274 0.98938328 0.99182863
 0.9598293  0.99206723 0.99409516 0.96734479 0.99051656 0.99224621
 0.96287119 0.9920673  0.99376711 0.96639039 0.99215683 0.99275324
 0.9600979  0.98699747 0.98777287 0.96973043 0.98822022 0.98994992]
```

In [ ]: