In [18]:
```
# import your preferred ml model.
from pyswarms.utils.functions import single_obj as fx
#build the model with your preferred hyperparameters.
model = RandomForestClassifier(n_estimators=231)

# create the GeneticSelection search with the different parameters available.
selection = (model,
                                cv=5,
                                scoring="accuracy",
                                max_features=12,
                                n_population=120,
                                crossover_proba=0.5,
                                mutation_proba=0.2,
                                n_generations=50,
                                crossover_independent_proba=0.5,
                                mutation_independent_proba=0.05,
                                n_gen_no_change=10,
                                n_jobs=-1)

# fit the GA search to our data.
selection = selection.fit(X_train, y_train)

# print the results.
print(selection.support_)
```

```
  File "<ipython-input-18-ee35fcfa87ae>", line 8
    cv=5,
      ^
SyntaxError: invalid syntax
```

In [19]:
```
selection_support = selection.get_support()
selection_feature = X.loc[:,selection_support].columns.tolist()
print(str(len(selection_feature)), 'selected features')
```

```
12 selected features
```

PSO ALGO

In [20]:
```
from main import PSOxBackpro
from random import random
import pandas as pd
from matplotlib import pyplot as plt
```

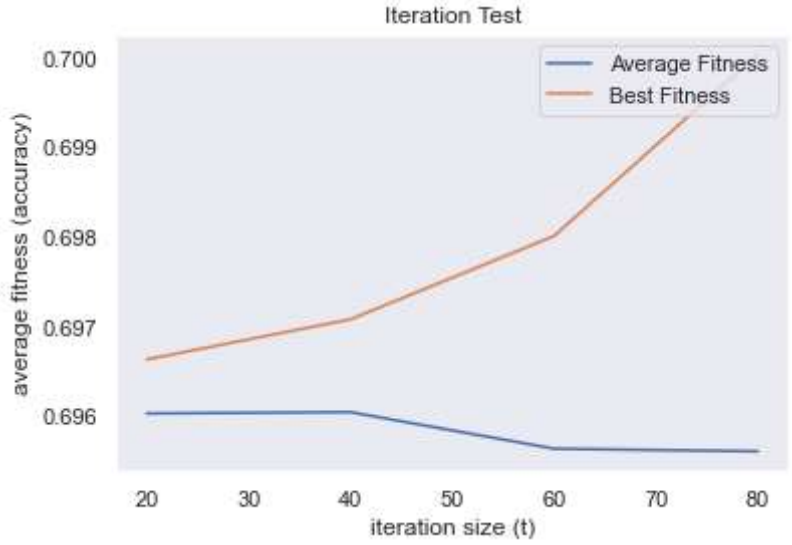
In [21]:

```
df_value = df.iloc[:, 1].values

normalized_dataset = list()
for x in range(len(df_value)):
    norm_value = (df_value[x] - min(df_value)) / (
        max(df_value) - min(df_value))
    normalized_dataset.append(norm_value)

time_series_dataset = [
    normalized_dataset[:len(normalized_dataset) - 5],
    normalized_dataset[1:len(normalized_dataset) - 4],
    normalized_dataset[2:len(normalized_dataset) - 3],
    normalized_dataset[3:len(normalized_dataset) - 2],
    normalized_dataset[4:len(normalized_dataset) - 1],
    normalized_dataset[5:]]

print("Time Series")
print(pd.DataFrame([list(i) for i in zip(*time_series_dataset)]))

df = pd.DataFrame([list(i) for i in zip(*time_series_dataset)])


X_train = df.iloc[:, :5]
Y_train = df.iloc[:, 5]
X_test = df.iloc[:, :5]
Y_test = df.iloc[:, 5]

max_val = 1292.0
min_val = 538.0
```

```
Time Series
       0    1    2    3    4    5
0    1.0  0.0  1.0  1.0  0.0  1.0
1    0.0  1.0  1.0  0.0  1.0  1.0
2    1.0  1.0  0.0  1.0  1.0  1.0
3    1.0  0.0  1.0  1.0  1.0  1.0
4    0.0  1.0  1.0  1.0  1.0  0.0
..   ...  ...  ...  ...  ...  ...
260  0.0  1.0  1.0  1.0  1.0  1.0
261  1.0  1.0  1.0  1.0  1.0  1.0
262  1.0  1.0  1.0  1.0  1.0  0.0
263  1.0  1.0  1.0  1.0  0.0  1.0
264  1.0  1.0  1.0  0.0  1.0  1.0

[265 rows x 6 columns]
```

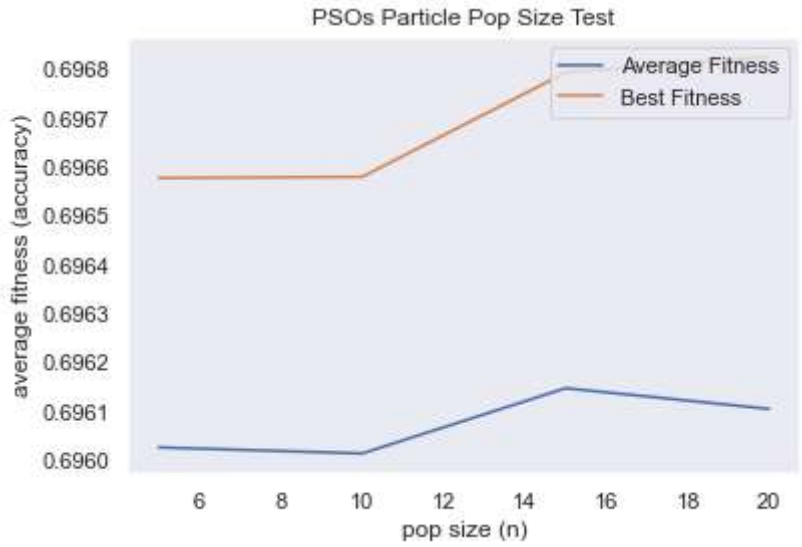
In [22]:
```
g_best_fitness = []
average_fitness = []

iter_param_test = [20, 40, 60, 80]
pso_backpp = PSOxBackpro(pop_size=10, particle_size=15, k=1)
pso_backpp.set_backpro_param(X_train, Y_train, X_test, Y_test, max_val, min_val,
for iterasi in range(len(iter_param_test)):
    g_best_fitness_temp = 0
    average_fitness_temp = 0
    for i in range(5):
        print('.',end=' ')
        pso_backpp.initPops()
        gbest_fitness, avg_fitness = pso_backpp.optimize(
            iter_param_test[iterasi], 1, 1, 1)
        g_best_fitness_temp += gbest_fitness
        average_fitness_temp += avg_fitness
    print(f"\nIter param test for {iter_param_test[iterasi]} is complete")
    g_best_fitness.append(g_best_fitness_temp / 5)
    average_fitness.append(average_fitness_temp / 5)


fig, ax = plt.subplots()
ax.plot(iter_param_test, average_fitness, label="Average Fitness")
ax.plot(iter_param_test, g_best_fitness, label="Best Fitness")
ax.legend(loc="upper right")

ax.set(xlabel='iteration size (t)', ylabel='average fitness (accuracy)',
       title='Iteration Test')
ax.grid()

fig.savefig("result/iter_test.png")

plt.show()
```

```
. . . . .
Iter param test for 20 is complete
. . . . .
Iter param test for 40 is complete
. . . . .
Iter param test for 60 is complete
. . . . .
Iter param test for 80 is complete

---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-22-6c7bbf668dbb> in <module>
     29 ax.grid()
     30
---> 31 fig.savefig("result/iter_test.png")
     32
     33 plt.show()

~\anaconda3\lib\site-packages\matplotlib\figure.py in savefig(self, fname, tran
sparent, **kwargs)
   2309                 patch.set_edgecolor('none')
   2310
```


Iteration Test
Average Fitness
Best Fitness
average fitness (accuracy)
0.700
0.699
0.698
0.697
0.696
20
30
40
50
60
70
80
iteration size (t)

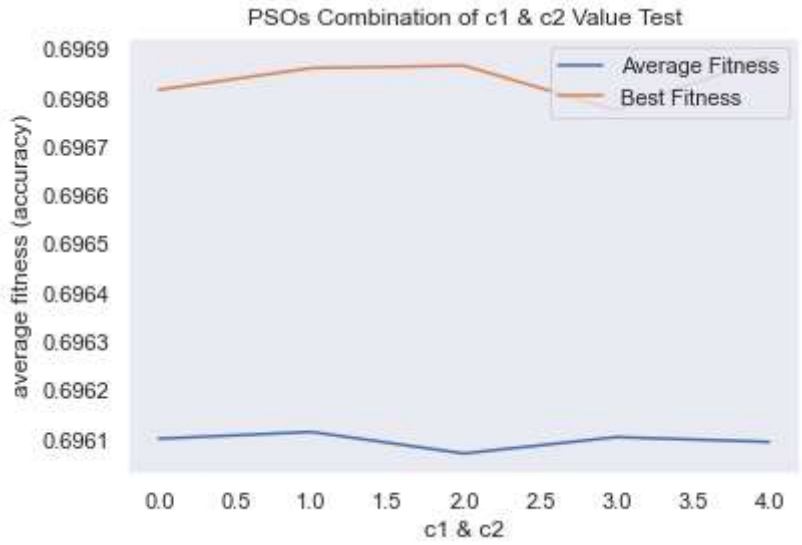
In [23]:
```
g_best_fitness = []
average_fitness = []

part_size_param_test = [5, 10, 15, 20]
for part_size in range(len(part_size_param_test)):
    g_best_fitness_temp = 0
    average_fitness_temp = 0
    for i in range(5):
        print('.',end=' ')
        pso_backpp = PSOxBackpro(part_size_param_test[part_size], 15, 1)
        pso_backpp.set_backpro_param(X_train, Y_train, X_test, Y_test, max_val, m
        pso_backpp.initPops()
        gbest_fitness, avg_fitness = pso_backpp.optimize(25, 1, 1, 1)
        g_best_fitness_temp += gbest_fitness
        average_fitness_temp += avg_fitness
    print(f"\nParticle param test for {part_size_param_test[part_size]} is comple
    g_best_fitness.append(g_best_fitness_temp / 5)
    average_fitness.append(average_fitness_temp / 5)


fig, ax = plt.subplots()
ax.plot(part_size_param_test, average_fitness, label="Average Fitness")
ax.plot(part_size_param_test, g_best_fitness, label="Best Fitness")
ax.legend(loc="upper right")

ax.set(xlabel='pop size (n)', ylabel='average fitness (accuracy)',
       title='PSOs Particle Pop Size Test')
ax.grid()


plt.show()
```

```
. . . . .
Particle param test for 5 is complete
. . . . .
Particle param test for 10 is complete
. . . . .
Particle param test for 15 is complete
. . . . .
Particle param test for 20 is complete
```


PSOs Particle Pop Size Test
Average Fitness
Best Fitness
average fitness (accuracy)
0.6968
0.6967
0.6966
0.6965
0.6964
0.6963
0.6962
0.6961
0.6960
6
8
10
12
14
16
18
20
pop size (n)

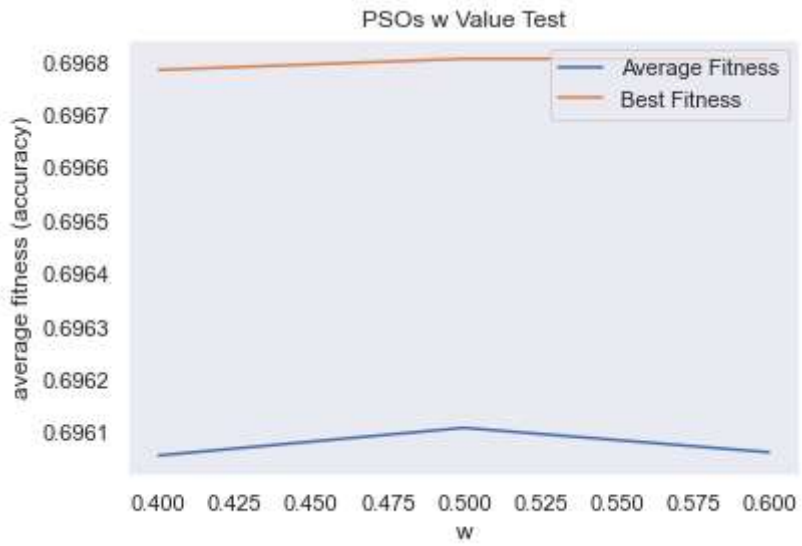
In [24]:
```
g_best_fitness = []
average_fitness = []

c_param_test = [(2.5, 0.5), (2, 1), (1.5, 1.5), (1, 2), (0.5, 2.5)]
param_test_str = []
for c in range(len(c_param_test)):
    g_best_fitness_temp = 0
    average_fitness_temp = 0
    param_test_str.append(c)
    for i in range(5):
        print('.',end=' ')
        pso_backpp = PSOxBackpro(40, 15, 1)
        pso_backpp.set_backpro_param(X_train, Y_train, X_test, Y_test, max_val, m
        pso_backpp.initPops()
        gbest_fitness, avg_fitness = pso_backpp.optimize(25, 1, c_param_test[c][0
        g_best_fitness_temp += gbest_fitness
        average_fitness_temp += avg_fitness
    print(f"\nc1 & c2 param test for {c_param_test[c]} is complete")
    g_best_fitness.append(g_best_fitness_temp / 5)
    average_fitness.append(average_fitness_temp / 5)


fig, ax = plt.subplots()
ax.plot(param_test_str, average_fitness, label="Average Fitness")
ax.plot(param_test_str, g_best_fitness, label="Best Fitness")
ax.legend(loc="upper right")

ax.set(xlabel='c1 & c2', ylabel='average fitness (accuracy)',
       title='PSOs Combination of c1 & c2 Value Test')
ax.grid()


plt.show()
```

```
. . . . .
c1 & c2 param test for (2.5, 0.5) is complete
. . . . .
c1 & c2 param test for (2, 1) is complete
. . . . .
c1 & c2 param test for (1.5, 1.5) is complete
. . . . .
c1 & c2 param test for (1, 2) is complete
. . . . .
c1 & c2 param test for (0.5, 2.5) is complete
```


PSOs Combination of c1 & c2 Value Test
Average Fitness
Best Fitness
average fitness (accuracy)
0.6969
0.6968
0.6967
0.6966
0.6965
0.6964
0.6963
0.6962
0.6961
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
c1 & c2

In [25]:
```
g_best_fitness = []
average_fitness = []

w_param_test = [0.4, 0.5, 0.6]
for w in range(len(w_param_test)):
    g_best_fitness_temp = 0
    average_fitness_temp = 0
    for i in range(5):
        print('.',end=' ')
        pso_backpp = PSOxBackpro(40, 15, 1)
        pso_backpp.set_backpro_param(X_train, Y_train, X_test, Y_test, max_val, m
        pso_backpp.initPops()
        gbest_fitness, avg_fitness = pso_backpp.optimize(25, w_param_test[w], 2.5
        g_best_fitness_temp += gbest_fitness
        average_fitness_temp += avg_fitness
    print(f"\nw param test for {w_param_test[w]} is complete")
    g_best_fitness.append(g_best_fitness_temp / 5)
    average_fitness.append(average_fitness_temp / 5)


fig, ax = plt.subplots()
ax.plot(w_param_test, average_fitness, label="Average Fitness")
ax.plot(w_param_test, g_best_fitness, label="Best Fitness")
ax.legend(loc="upper right")

ax.set(xlabel='w', ylabel='average fitness (accuracy)',
       title='PSOs w Value Test')
ax.grid()

plt.show()
```

```
. . . . .
w param test for 0.4 is complete
. . . . .
w param test for 0.5 is complete
. . . . .
w param test for 0.6 is complete
```

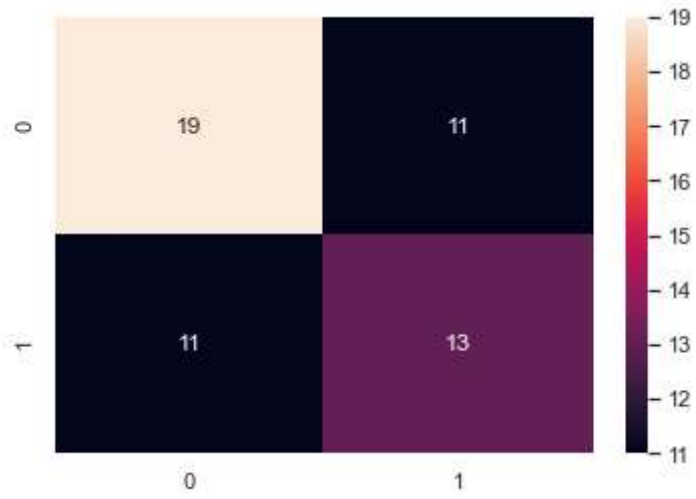


DECISION TREE CLASSIFIER

In [182]:
```
from sklearn.tree import DecisionTreeClassifier
decc = DecisionTreeClassifier()
decc.fit(X_train,y_train)
y_pred_decc = decc.predict(X_test)
```

In [183]:
```
CM=confusion_matrix(y_test,y_pred_decc)
sns.heatmap(CM, annot=True)

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
print(specificity)
```

```
0.6333333333333333
```



In [184]:
```
acc= accuracy_score(y_test,y_pred_decc)
print(acc)
```

```
0.5925925925925926
```

BAGGED TREE

In [304]:
```
from sklearn.ensemble import BaggingClassifier
```

In [305]:
```
bt = BaggingClassifier(base_estimator=decc,n_estimators=100)
```

In [306]:
```
bt.fit(X_train,y_train)
```

Out[306]:
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=100)
```

In [307]:
```
y_pred_ada = bt.predict(X_test)
```

In [308]:
```
from sklearn.metrics import log_loss,roc_auc_score,precision_score,f1_score,recal
from sklearn.metrics import classification_report, confusion_matrix,accuracy_scor
from sklearn import metrics
```

In [309]:
```
CM=confusion_matrix(y_test,y_pred_ada)
sns.heatmap(CM, annot=True)

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
print(specificity)
```

```
0.7666666666666667
```

In [310]:
```
acc= accuracy_score(y_test,y_pred_ada)
print(acc)
```

```
0.7592592592592593
```

In [282]:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[30  0]
 [24  0]]
              precision    recall  f1-score   support

      absent       0.56      1.00      0.71        30
     present       0.00      0.00      0.00        24

    accuracy                           0.56        54
   macro avg       0.28      0.50      0.36        54
weighted avg       0.31      0.56      0.40        54


C:\Users\Priya\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1
245: UndefinedMetricWarning: Precision and F-score are ill-defined and being se
t to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Priya\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1
245: UndefinedMetricWarning: Precision and F-score are ill-defined and being se
t to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Priya\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1
245: UndefinedMetricWarning: Precision and F-score are ill-defined and being se
t to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

RANDOM FOREST

In [320]:

```
rf_ent = RandomForestClassifier(criterion='entropy',n_estimators=100)
rf_ent.fit(X_train, y_train)
y_pred_rfe = rf_ent.predict(X_test)
```

In [321]:
```
CM=confusion_matrix(y_test,y_pred_rfe)
sns.heatmap(CM, annot=True)

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
print(specificity)
```

```
0.8666666666666667
```

In [322]:
```
acc= accuracy_score(y_test,y_pred_rfe)
print(acc)
```

```
0.8333333333333334
```

KNN (K NEAREST NEIGHBOUR)

In [164]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X, y)
```

Out[164]:
```
KNeighborsClassifier()
```

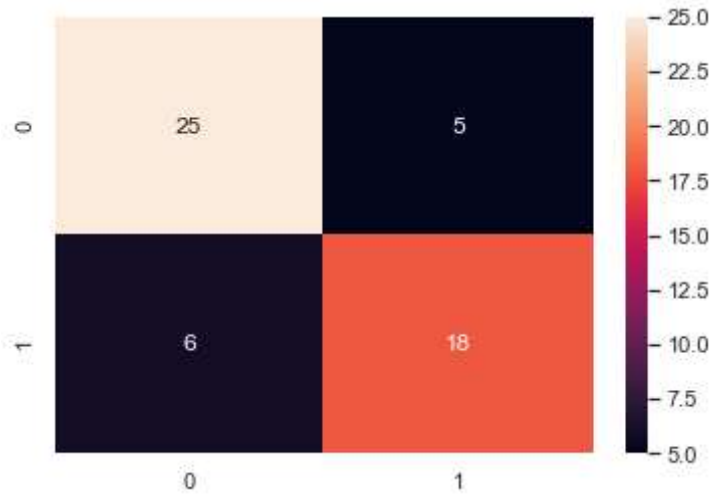
In [201]:
```
y_pred = classifier.predict(X_test)
```

In [207]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[30  0]
 [24  0]]
              precision    recall  f1-score   support

      absent       0.56      1.00      0.71        30
     present       0.00      0.00      0.00        24

    accuracy                           0.56        54
   macro avg       0.28      0.50      0.36        54
weighted avg       0.31      0.56      0.40        54


C:\Users\Priya\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1
245: UndefinedMetricWarning: Precision and F-score are ill-defined and being se
t to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Priya\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1
245: UndefinedMetricWarning: Precision and F-score are ill-defined and being se
t to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Priya\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1
245: UndefinedMetricWarning: Precision and F-score are ill-defined and being se
t to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [208]:
```
CM=confusion_matrix(y_test,y_pred_rfe)
sns.heatmap(CM, annot=True)

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
print(specificity)
```

```
0.8333333333333334
```



In [209]:
```
acc= accuracy_score(y_test,y_pred)
print(acc)
```

```
0.5555555555555556
```

ADABOOST

In [210]:
```
adaboost = AdaBoostClassifier()
adaboost.fit(X_train,y_train)
y_pred_adaboost = adaboost.predict(X_test)
```

In [211]:
```
CM=confusion_matrix(y_test,y_pred_adaboost)
sns.heatmap(CM, annot=True)

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
print(specificity)
```

```
0.7666666666666667
```

In [212]:
```
acc= accuracy_score(y_test,y_pred_adaboost)
print(acc)
```

```
0.7407407407407407
```

In [ ]:
```

```