Importing libraries

In [50]:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from scipy.optimize import minimize
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.naive_bayes import GaussianNB
import xgboost as xgb
import numpy as np
from numpy import mean, std
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import StackingClassifier
from sklearn.metrics import cohen_kappa_score, make_scorer
kappa_scorer = make_scorer(cohen_kappa_score)
```

Importing data from Excel and creating initial dataset

In [40]:

```
excel_data = pd.read_excel (r'C:\Users\dania\Desktop\data\Input Dataset.xlsx', sheet_name
='dataset',index_col=0)
raw_data = excel_data.values[:,0:67]
y = excel_data.label
std_data = StandardScaler().fit_transform(raw_data)
```

Determining the optimal number of principal components for PCA

In [3]:

```
pca_k_scores =[]
for i in range (1,21):
    pca = PCA(n_components=i)
    pca_data = pca.fit_transform(std_data[:,0:65])
    data = pd.DataFrame(pca_data,index=excel_data.index)
    data = data.assign(concrete=std_data[:,[65]])
    data = data.assign(time=std_data[:,[66]])
    model = SVC()
    kernel = ['poly', 'rbf', 'sigmoid']
    C = [ 1.0, 2.0, 3.0,5.0,10.0, 0.1, 0.01]
    grid = dict(kernel=kernel,C=C)
    cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
    grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                           cv=cv, scoring='accuracy',error_score=0)
    grid_result = grid_search.fit(data, y)
    pca_k_scores.append([i,(1-grid_result.best_score_)])
pca_k_scores
```

Out[3]:

```
[[1, 0.3018181818181819],
 [2, 0.1903030303030303],
 [3, 0.1333333333333333],
 [4, 0.1515151515151516],
```

```
 [5, 0.10909090909090913],
 [6, 0.11454545454545462],
 [7, 0.1260606060606061],
 [8, 0.12666666666666667],
 [9, 0.13212121212121208],
 [10, 0.13212121212121208],
 [11, 0.13818181818181818],
 [12, 0.1321212121212122],
 [13, 0.13818181818181818],
 [14, 0.13818181818181818],
 [15, 0.13818181818181818],
 [16, 0.13818181818181818],
 [17, 0.13818181818181818],
 [18, 0.14484848484848478],
 [19, 0.13818181818181818],
 [20, 0.13818181818181818]]
```

In [4]:

```
pca_k_scores =[]
for i in range (1,21):
    pca = PCA(n_components=i)
    pca_data = pca.fit_transform(std_data[:,0:65])
    data = pd.DataFrame(pca_data,index=excel_data.index)
    data = data.assign(concrete=std_data[:,[65]])
    data = data.assign(time=std_data[:,[66]])
    model = KNeighborsClassifier()
    n_neighbors = range(1, 21,2)
    leaf_size = [10, 20, 30, 50]
    metric = ['euclidean','minkowski','manhattan']
    grid = dict(n_neighbors=n_neighbors,metric=metric, leaf_size=leaf_size)
    cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
    grid_search = GridSearchCV(estimator=model, param_grid=grid,
                           n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
    grid_result = grid_search.fit(data, y)
    pca_k_scores.append(1-grid_result.best_score_)
pca_k_scores
```

Out[4]:

```
[0.35272727272727267,
 0.20121212121212118,
 0.1527272727272727,
 0.17151515151515162,
 0.13757575757575757,
 0.13151515151515147,
 0.1381818181818183,
 0.1393939393939394,
 0.16969696969696968,
 0.15636363636363637,
 0.16363636363636358,
 0.15090909090909088,
 0.1575757575757577,
 0.1515151515151517,
 0.1642424242424242,
 0.1581818181818181,
 0.171515151515151515,
 0.1648484848484848,
 0.16545454545454552,
 0.16545454545454552]
```

In [5]:

```
pca_k_scores =[]
for i in range (1,21):
    pca = PCA(n_components=i)
    pca_data = pca.fit_transform(std_data[:,0:65])
    data = pd.DataFrame(pca_data,index=excel_data.index)
    data = data.assign(concrete=std_data[:,[65]])
    data = data.assign(time=std_data[:,[66]])
    model = RandomForestClassifier(random_state=1)
    n_estimators = [50, 100, 120, 200]
```

```
    max_features = ['sqrt', 'log2']
    max_depth = [2,6,8,10]
    grid = dict(n_estimators=n_estimators,
                max_features=max_features,max_depth=max_depth)
    cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
    grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                               cv=cv, scoring='accuracy',error_score=0)
    grid_result = grid_search.fit(data, y)
    pca_k_scores.append(1-grid_result.best_score_)
pca_k_scores
```

Out[5]:

```
[0.3284848484848484,
 0.1884848484848486,
 0.18060606060606055,
 0.18727272727272726,
 0.16909090909090907,
 0.18848484848484837,
 0.21272727272727276,
 0.19454545454545447,
 0.19999999999999984,
 0.20060606060606057,
 0.19333333333333336,
 0.20666666666666655,
 0.22484848484848485,
 0.21393939393939398,
 0.21939393939393925,
 0.23272727272727267,
 0.22545454545454535,
 0.20666666666666667,
 0.20181818181818179,
 0.22606060606060607]
```

In [6]:

```
pca_k_scores =[]
for i in range (1,21):
    pca = PCA(n_components=i)
    pca_data = pca.fit_transform(std_data[:,0:65])
    data = pd.DataFrame(pca_data,index=excel_data.index)
    data = data.assign(concrete=std_data[:,[65]])
    data = data.assign(time=std_data[:,[66]])
    model = GaussianNB()
    var_smoothing= np.logspace(0,-9, num=5)
    grid = dict(var_smoothing=var_smoothing)
    cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
    grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                                   cv=cv, scoring='f1_macro',error_score=0)
    grid_result = grid_search.fit(data, y)
    pca_k_scores.append(1-grid_result.best_score_)
pca_k_scores
```

Out[6]:

```
[0.38943562610229276,
 0.1768478435145103,
 0.14100529100529102,
 0.16134038800705475,
 0.13107583774250442,
 0.12469135802469133,
 0.12469135802469133,
 0.12358024691358027,
 0.14007054673721342,
 0.1852380952380952,
 0.17786596119929454,
 0.17827160493827165,
 0.17206349206349203,
 0.18564373897707231,
 0.18564373897707231,
 0.18564373897707231,
 0.18564373897707231,
 0.1925573192239859,
```

```
 0.1925573192239859,
 0.1925573192239859]
```

In [7]:

```
pca_k_scores =[]
for i in range (1,21):
    pca = PCA(n_components=i)
    pca_data = pca.fit_transform(std_data[:,0:65])
    data = pd.DataFrame(pca_data,index=excel_data.index)
    data = data.assign(concrete=std_data[:,[65]])
    data = data.assign(time=std_data[:,[66]])
    model = xgb.XGBClassifier(objective='multi:softmax')
    subsample = [0.5, 0.75]
    colsample_bytree = [0.75, 1]
    min_child_weight= [0.5, 1]
    max_depth =[2,6]
    learning_rate = [0.1]
    n_estimators = [1000, 2000]
    grid = dict(subsample=subsample, colsample_bytree=colsample_bytree,
                min_child_weight=min_child_weight,max_depth=max_depth,
                learning_rate=learning_rate,n_estimators=n_estimators)
    cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
    grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                           cv=cv, scoring='accuracy',error_score=0)
    grid_result = grid_search.fit(data, y)
    pca_k_scores.append(1-grid_result.best_score_)
pca_k_scores
```

Out[7]:

```
[0.4181818181818181,
 0.1581818181818181,
 0.16303030303030297,
 0.18181818181818177,
 0.17515151515151517,
 0.17575757575757567,
 0.19939393939393923,
 0.18787878787878776,
 0.19393939393939374,
 0.21939393939393925,
 0.21939393939393914,
 0.21878787878787864,
 0.21939393939393925,
 0.23212121212121195,
 0.23212121212121195,
 0.22606060606060596,
 0.22606060606060596,
 0.23272727272727267,
 0.22545454545454535,
 0.22060606060606058]
```

Feature extraction with PCA and creating the input dataset

In [41]:

```
pca = PCA(n_components=5)
pca_data = pca.fit_transform(std_data[:,0:65])
pca.explained_variance_ratio_
data = pd.DataFrame(pca_data,index=excel_data.index)
data = data.assign(concrete=std_data[:,[65]])
data = data.assign(time=std_data[:,[66]])
```

Tuning hyperparameters of base classifiers using grid search

In [30]:

```
model = KNeighborsClassifier()
n_neighbors = range(1, 21,2)
leaf_size = [10, 20, 30, 50]
metric = ['euclidean','minkowski','manhattan']
grid = dict(n_neighbors=n_neighbors,metric=metric, leaf_size=leaf_size)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid,
                           n_jobs=-1, cv=cv, scoring='f1_macro',error_score=0)
grid_result = grid_search.fit(data, y)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.848137 using {'leaf_size': 10, 'metric': 'euclidean', 'n_neighbors': 3}
```

In [10]:

```
from sklearn.svm import SVC
model = SVC()
kernel = ['poly', 'rbf', 'sigmoid']
C = [ 1.0, 2.0, 3.0, 5.0, 10.0, 0.1, 0.01]
grid = dict(kernel=kernel,C=C)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                           cv=cv, scoring='f1_macro',error_score=0)
grid_result = grid_search.fit(data, y)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.884374 using {'C': 1.0, 'kernel': 'sigmoid'}
```

In [11]:

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state=1)
n_estimators = [50, 100, 120, 200]
max_features = ['sqrt', 'log2']
max_depth = [2,6,8,10]
grid = dict(n_estimators=n_estimators,
            max_features=max_features,max_depth=max_depth)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                           cv=cv, scoring='f1_macro',error_score=0)
grid_result = grid_search.fit(data, y)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.802257 using {'max_depth': 8, 'max_features': 'sqrt', 'n_estimators': 200}
```

In [12]:

```
model =  GaussianNB()
var_smoothing= np.logspace(0,-9, num=5)
grid = dict(var_smoothing=var_smoothing)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                           cv=cv, scoring='f1_macro',error_score=0)
grid_result = grid_search.fit(data, y)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.868924 using {'var_smoothing': 0.005623413251903491}
```

In [13]:

```
model = xgb.XGBClassifier(objective='multi:softmax')
subsample = [0.5, 0.75, 1]
colsample_bytree = [0.5, 0.75, 1]
min_child_weight= [0.5, 1, 5]
max_depth =[2,6,8]
learning_rate = [0.1, 0.01, 0.05]
n_estimators = [200, 500, 1000, 2000]
grid = dict(subsample=subsample, colsample_bytree=colsample_bytree,
            min_child_weight=min_child_weight,max_depth=max_depth,
            learning_rate=learning_rate,n_estimators=n_estimators)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
```

```
                    cv=cv, scoring='f1_macro',error_score=0)
grid_result = grid_search.fit(data, y)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.805450 using {'colsample_bytree': 1, 'learning_rate': 0.1, 'max_depth': 2, 'min_c
hild_weight': 0.5, 'n_estimators': 2000, 'subsample': 0.75}
```

Training base classifiers

In [42]:

```
model_knn = KNeighborsClassifier(metric= 'euclidean', n_neighbors= 3, leaf_size=10)
model_svm = SVC(C= 1.0, kernel='sigmoid')
model_rf = RandomForestClassifier(random_state=1, max_depth=8 , max_features='sqrt', n_e
stimators=200)
model_nb = GaussianNB(var_smoothing= 0.0056)
model_xgb = xgb.XGBClassifier(objective='multi:softmax', colsample_bytree=1, learning_rat
e= 0.1,
                              max_depth= 2, min_child_weight= 0.5, n_estimators= 2000, s
ubsample= 0.75)
models=[("knn", model_knn),("svm",model_svm),("rf", model_rf),("nb",model_nb),("xgb",mod
el_xgb)]
```

Finding and tuning the best meta learner

In [15]:

```
model_stack =  KNeighborsClassifier()
model = StackingClassifier(estimators=models, final_estimator=model_stack)
n_neighbors = range(1, 21,2)
metric = ['euclidean','minkowski','manhattan']
leaf_size = [10, 20, 30, 50]
grid = dict(final_estimator__n_neighbors=n_neighbors,final_estimator__metric=metric, fina
l_estimator__leaf_size=leaf_size)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid,
                           n_jobs=-1, cv=cv, scoring='f1_macro',error_score=0)
grid_result = grid_search.fit(data, y)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.908131 using {'final_estimator__leaf_size': 10, 'final_estimator__metric': 'manha
ttan', 'final_estimator__n_neighbors': 5}
```

In [16]:

```
model_stack =  SVC()
model = StackingClassifier(estimators=models, final_estimator=model_stack)
kernel = ['poly', 'rbf', 'sigmoid']
C = [ 1.0, 2.0, 3.0, 5.0, 10.0, 0.1, 0.01]
grid = dict(final_estimator__kernel=kernel,final_estimator__C=C)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid,
                           n_jobs=-1, cv=cv, scoring='f1_macro',error_score=0)
grid_result = grid_search.fit(data, y)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.914903 using {'final_estimator__C': 0.1, 'final_estimator__kernel': 'poly'}
```

In [17]:

```
model_stack =  RandomForestClassifier(random_state=1)
model = StackingClassifier(estimators=models, final_estimator=model_stack)
n_estimators = [50, 100, 120, 200]
max_features = ['sqrt', 'log2']
max_depth = [2,6,8,10]
grid = dict(final_estimator__n_estimators=n_estimators,
            final_estimator__max_features=max_features,final_estimator__max_depth=max_de
pth)
```

```
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid,
                           n_jobs=-1, cv=cv, scoring='f1_macro',error_score=0)
grid_result = grid_search.fit(data, y)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.856808 using {'final_estimator__max_depth': 2, 'final_estimator__max_features': '
sqrt', 'final_estimator__n_estimators': 50}
```

In [18]:

```
model_stack = xgb.XGBClassifier(objective='multi:softmax', random_state=1)
model = StackingClassifier(estimators=models, final_estimator=model_stack)
max_depth =[2, 6, 8]
learning_rate = [0.1, 0.01]
n_estimators = [200,600,1000]
subsample = [0.75, 1]
colsample_bytree = [0.75, 1]
min_child_weight= [0.5, 1]
grid = dict(final_estimator__max_depth=max_depth,final_estimator__subsample=subsample,
            final_estimator__colsample_bytree=colsample_bytree, final_estimator__min_chi
ld_weight=min_child_weight,
            final_estimator__learning_rate=learning_rate, final_estimator__n_estimators=
n_estimators)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid,
                           n_jobs=-1, cv=cv, scoring='f1_macro',error_score=0)
grid_result = grid_search.fit(data, y)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.855379 using {'final_estimator__colsample_bytree': 0.75, 'final_estimator__learni
ng_rate': 0.1, 'final_estimator__max_depth': 2, 'final_estimator__min_child_weight': 1, '
final_estimator__n_estimators': 200, 'final_estimator__subsample': 0.75}
```

In [19]:

```
model_stack =  GaussianNB()
model = StackingClassifier(estimators=models, final_estimator=model_stack)
var_smoothing= np.logspace(0,-9, num=5)
grid = dict(final_estimator__var_smoothing=var_smoothing)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                           cv=cv, scoring='f1_macro',error_score=0)
grid_result = grid_search.fit(data, y)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.890317 using {'final_estimator__var_smoothing': 1.0}
```

Training the proposed stacking ensemble model

In [51]:

```
model_stack =  SVC(C= 0.1, kernel='poly')
model_ensemble = StackingClassifier(estimators=models, final_estimator=model_stack)
```

Evaluation of the proposed stacking model and 5 base classifiers using 3-times repeated stratified 5-fold CV

In [44]:

```
model=model_knn
cv= RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
scores = cross_val_score(model, data, y, scoring='accuracy', cv=cv, n_jobs=-1)
print('Accuracy: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='f1_macro', cv=cv, n_jobs=-1)
print('f1_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='precision_macro', cv=cv, n_jobs=-1)
```

```
print('precision_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='recall_macro', cv=cv, n_jobs=-1)
print('recall_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring=kappa_scorer, cv=cv, n_jobs=-1)
print('kappa_scorer: %.4f (%.4f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.8624 (0.0812)
f1_macro: 0.8481 (0.0939)
precision_macro: 0.8875 (0.0699)
recall_macro: 0.8537 (0.0904)
kappa_scorer: 0.7899 (0.1253)
```

In [45]:

```
model=model_svm
cv= RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
scores = cross_val_score(model, data, y, scoring='accuracy', cv=cv, n_jobs=-1)
print('Accuracy: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='f1_macro', cv=cv, n_jobs=-1)
print('f1_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='precision_macro', cv=cv, n_jobs=-1)
print('precision_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='recall_macro', cv=cv, n_jobs=-1)
print('recall_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring=kappa_scorer, cv=cv, n_jobs=-1)
print('kappa_scorer: %.4f (%.4f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.8909 (0.0629)
f1_macro: 0.8844 (0.0694)
precision_macro: 0.9056 (0.0648)
recall_macro: 0.8907 (0.0669)
kappa_scorer: 0.8337 (0.0973)
```

In [46]:

```
model=model_nb
cv= RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
scores = cross_val_score(model, data, y, scoring='accuracy', cv=cv, n_jobs=-1)
print('Accuracy: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='f1_macro', cv=cv, n_jobs=-1)
print('f1_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='precision_macro', cv=cv, n_jobs=-1)
print('precision_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='recall_macro', cv=cv, n_jobs=-1)
print('recall_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring=kappa_scorer, cv=cv, n_jobs=-1)
print('kappa_scorer: %.4f (%.4f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.8794 (0.0897)
f1_macro: 0.8689 (0.0967)
precision_macro: 0.9089 (0.0647)
recall_macro: 0.8778 (0.0900)
kappa_scorer: 0.8190 (0.1337)
```

In [47]:

```
model=model_rf
cv= RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
scores = cross_val_score(model, data, y, scoring='accuracy', cv=cv, n_jobs=-1)
print('Accuracy: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='f1_macro', cv=cv, n_jobs=-1)
print('f1_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='precision_macro', cv=cv, n_jobs=-1)
print('precision_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='recall_macro', cv=cv, n_jobs=-1)
print('recall_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring=kappa_scorer, cv=cv, n_jobs=-1)
print('kappa_scorer: %.4f (%.4f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.8309 (0.0912)
f1_macro: 0.8023 (0.1171)
precision_macro: 0.8322 (0.1275)
```

```
recall_macro: 0.8148 (0.1038)
kappa_scorer: 0.7413 (0.1408)
```

In [48]:

```
model=model_xgb
cv= RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
scores = cross_val_score(model, data, y, scoring='accuracy', cv=cv, n_jobs=-1)
print('Accuracy: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='f1_macro', cv=cv, n_jobs=-1)
print('f1_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='precision_macro', cv=cv, n_jobs=-1)
print('precision_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='recall_macro', cv=cv, n_jobs=-1)
print('recall_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring=kappa_scorer, cv=cv, n_jobs=-1)
print('kappa_scorer: %.4f (%.4f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.8248 (0.1119)
f1_macro: 0.8054 (0.1187)
precision_macro: 0.8570 (0.1021)
recall_macro: 0.8074 (0.1159)
kappa_scorer: 0.7335 (0.1683)
```

In [49]:

```
model = model_ensemble
cv= RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
scores = cross_val_score(model, data, y, scoring='accuracy', cv=cv, n_jobs=-1)
print('Accuracy: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='f1_macro', cv=cv, n_jobs=-1)
print('f1_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='precision_macro', cv=cv, n_jobs=-1)
print('precision_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring='recall_macro', cv=cv, n_jobs=-1)
print('recall_macro: %.4f (%.4f)' % (mean(scores), std(scores)))
scores = cross_val_score(model, data, y, scoring=kappa_scorer, cv=cv, n_jobs=-1)
print('kappa_scorer: %.4f (%.4f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.9236 (0.0530)
f1_macro: 0.9149 (0.0634)
precision_macro: 0.9263 (0.0612)
recall_macro: 0.9222 (0.0619)
kappa_scorer: 0.8841 (0.0817)
```

In [ ]: