Grey wolf optimizer

```
#Importing the Libraries
from sklearn.datasets import make_classification
from sklearn.datasets import make_classification
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import load_model


import numpy as np
from sklearn.neighbors import KNeighborsClassifier


# error rate
def error_rate(xtrain, ytrain, x, opts):
    # parameters
    k     = opts['k']
    fold  = opts['fold']
    xt    = fold['xt']
    yt    = fold['yt']
    xv    = fold['xv']
    yv    = fold['yv']

    # Number of instances
    num_train = np.size(xt, 0)
    num_valid = np.size(xv, 0)
    # Define selected features
    xtrain  = xt[:, x == 1]
    ytrain  = yt.reshape(num_train)  # Solve bug
    xvalid  = xv[:, x == 1]
    yvalid  = yv.reshape(num_valid)  # Solve bug
    # Training
    mdl     = KNeighborsClassifier(n_neighbors = k)
    mdl.fit(xtrain, ytrain)
    # Prediction
    ypred   = mdl.predict(xvalid)
    acc     = np.sum(yvalid == ypred) / num_valid
    error   = 1 - acc

    return error


# Error rate & Feature size
def Fun(xtrain, ytrain, x, opts):
    # Parameters
    alpha    = 0.99
    beta     = 1 - alpha
    # Original feature size
    max_feat = len(x)
    # Number of selected features
    num_feat = np.sum(x == 1)
```

```
    # Solve if no feature selected
    if num_feat == 0:
        cost  = 1
    else:
        # Get error rate
        error = error_rate(xtrain, ytrain, x, opts)
        # Objective function
        cost  = alpha * error + beta * (num_feat / max_feat)

    return cost



#[2014]-"Grey wolf optimizer"

import numpy as np
from numpy.random import rand

def init_position(lb, ub, N, dim):
    X = np.zeros([N, dim], dtype='float')
    for i in range(N):
        for d in range(dim):
            X[i,d] = lb[0,d] + (ub[0,d] - lb[0,d]) * rand()
    return X


def binary_conversion(X, thres, N, dim):
    Xbin = np.zeros([N, dim], dtype='int')
    for i in range(N):
        for d in range(dim):
            if X[i,d] > thres:
                Xbin[i,d] = 1
            else:
                Xbin[i,d] = 0

    return Xbin


def boundary(x, lb, ub):
    if x < lb:
        x = lb
    if x > ub:
        x = ub

    return x


def jfs(xtrain, ytrain, opts):
    # Parameters
    ub    = 1
    lb    = 0
    thres = 0.7

    N        = opts['N']
    max_iter = opts['T']

    # Dimension
    dim = np.size(xtrain, 1)
    if np.size(lb) == 1:
        ub = ub * np.ones([1, dim], dtype='float')
```

```
    lb = lb * np.ones([1, dim], dtype='float')

# Initialize position
X      = init_position(lb, ub, N, dim)

# Binary conversion
Xbin   = binary_conversion(X, thres, N, dim)

# Fitness at first iteration
fit    = np.zeros([N, 1], dtype='float')
Xalpha = np.zeros([1, dim], dtype='float')
Xbeta  = np.zeros([1, dim], dtype='float')
Xdelta = np.zeros([1, dim], dtype='float')
Falpha = float('inf')
Fbeta  = float('inf')
Fdelta = float('inf')

for i in range(N):
    fit[i,0] = Fun(xtrain, ytrain, Xbin[i,:], opts)
    if fit[i,0] < Falpha:
        Xalpha[0,:] = X[i,:]
        Falpha      = fit[i,0]

    if fit[i,0] < Fbeta and fit[i,0] > Falpha:
        Xbeta[0,:]  = X[i,:]
        Fbeta       = fit[i,0]

    if fit[i,0] < Fdelta and fit[i,0] > Fbeta and fit[i,0] > Falpha:
        Xdelta[0,:] = X[i,:]
        Fdelta      = fit[i,0]

# Pre
curve = np.zeros([1, max_iter], dtype='float')
t     = 0

curve[0,t] = Falpha.copy()
print("Iteration:", t + 1)
print("Best (GWO):", curve[0,t])

t += 1

while t < max_iter:
    # Coefficient decreases linearly from 2 to 0
    a = 2 - t * (2 / max_iter)

    for i in range(N):
        for d in range(dim):
            # Parameter C (3.4)
            C1     = 2 * rand()
            C2     = 2 * rand()
            C3     = 2 * rand()
            # Compute Dalpha, Dbeta & Ddelta (3.5)
            Dalpha = abs(C1 * Xalpha[0,d] - X[i,d])
            Dbeta  = abs(C2 * Xbeta[0,d] - X[i,d])
            Ddelta = abs(C3 * Xdelta[0,d] - X[i,d])
            # Parameter A (3.3)
            A1     = 2 * a * rand() - a
            A2     = 2 * a * rand() - a
            A3     = 2 * a * rand() - a
            # Compute X1, X2 & X3 (3.6)
```

```
                X1     = Xalpha[0,d] - A1 * Dalpha
                X2     = Xbeta[0,d] - A2 * Dbeta
                X3     = Xdelta[0,d] - A3 * Ddelta
                # Update wolf (3.7)
                X[i,d] = (X1 + X2 + X3) / 3
                # Boundary
                X[i,d] = boundary(X[i,d], lb[0,d], ub[0,d])

        # Binary conversion
        Xbin  = binary_conversion(X, thres, N, dim)

        # Fitness
        for i in range(N):
            fit[i,0] = Fun(xtrain, ytrain, Xbin[i,:], opts)
            if fit[i,0] < Falpha:
                Xalpha[0,:] = X[i,:]
                Falpha      = fit[i,0]

            if fit[i,0] < Fbeta and fit[i,0] > Falpha:
                Xbeta[0,:]  = X[i,:]
                Fbeta       = fit[i,0]

            if fit[i,0] < Fdelta and fit[i,0] > Fbeta and fit[i,0] > Falpha:
                Xdelta[0,:] = X[i,:]
                Fdelta      = fit[i,0]

        curve[0,t] = Falpha.copy()
        print("Iteration:", t + 1)
        print("Best (GWO):", curve[0,t])
        print("Alpha (GWO):", Falpha)
        print("Beta (GWO):", Fbeta)
        print("Delta (GWO):", Fdelta)
        t += 1


    # Best feature subset
    Gbin       = binary_conversion(Xalpha, thres, 1, dim)
    Gbin       = Gbin.reshape(dim)
    pos        = np.asarray(range(0, dim))
    sel_index  = pos[Gbin == 1]
    num_feat   = len(sel_index)
    # Create dictionary
    gwo_data = {'sf': sel_index, 'c': curve, 'nf': num_feat}
    print(gwo_data)
    return gwo_data



import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt


# load data
data  = pd.read_csv('/content/dataset_full.csv')
data.shape
data=data.dropna()
data  = data.values
feat  = np.asarray(data[:, 0:-1])
```

```
label = np.asarray(data[:, -1])

# split data into train & validation (70 -- 30)
xtrain, xtest, ytrain, ytest = train_test_split(feat, label, test_size=0.3, stratify=label
fold = {'xt':xtrain, 'yt':ytrain, 'xv':xtest, 'yv':ytest}

# parameter
k    = 5     # k-value in KNN
N    = 10    # number of particles
T    = 1     # maximum number of iterations
opts = {'k':k, 'fold':fold, 'N':N, 'T':T}

# perform feature selection
fmdl = jfs(feat, label, opts)
sf   = fmdl['sf']

# model with selected features
num_train = np.size(xtrain, 0)
num_valid = np.size(xtest, 0)
x_train   = xtrain[:, sf]
y_train   = ytrain.reshape(num_train)  # Solve bug
x_valid   = xtest[:, sf]
y_valid   = ytest.reshape(num_valid)  # Solve bug

mdl       = KNeighborsClassifier(n_neighbors = k)
mdl.fit(x_train, y_train)

# accuracy
y_pred    = mdl.predict(x_valid)
Acc       = np.sum(y_valid == y_pred)  / num_valid
print("Accuracy:", 100 * Acc)

# number of selected features
num_feat = fmdl['nf']
print("Feature Size:",num_feat)

# plot convergence
curve   = fmdl['c']
curve   = curve.reshape(np.size(curve,1))
x       = np.arange(0, opts['T'], 1.0) + 1.0

fig, ax = plt.subplots()
ax.plot(x, curve, 'o-')
ax.set_xlabel('Number of Iterations')
ax.set_ylabel('Fitness')
ax.set_title('PSO')
ax.grid()
plt.show()
```

```
Iteration: 1
Best (GWO): 0.06987231901952716
{'sf': array([  8,  10,  15,  16,  20,  23,  25,  27,  28,  30,  35,  37,  38,
        39,  41,  44,  57,  58,  60,  67,  74,  78,  80,  91,  96, 101,
       102, 103, 104, 106, 108]), 'c': array([[0.06987232]]), 'nf': 31}
Accuracy: 93.22429028012785
Feature Size: 31
```

PSO

```
selected_features = [ 6,   8,  18,  19,  23,  28,  29,  31,  33,  50,  54,  56,  57,
        58,  63,  64,  65,  69,  71,  73,  74,  76,  80,  87,  88,  89,
        91,  93,  94,  99, 103, 110]
X_train = data[:, selected_features]
```

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

```
# load data
data  = pd.read_csv('/content/dataset_full.csv')
data.head()
```

	qty_dot_url	qty_hyphen_url	qty_underline_url	qty_slash_url	qty_questionma
0	3	0	0	1	
1	5	0	1	3	
2	2	0	0	1	
3	4	0	2	5	
4	2	0	0	0	

5 rows × 112 columns

```
print(data)
```

```
88644                0              0             0             0
88645                0              0             0             0
88646                0              0             0             0

       qty_exclamation_url  qty_space_url  ...  qty_ip_resolved  \
0                        0              0  ...                1
1                        0              0  ...                1
2                        0              0  ...                1
3                        0              0  ...                1
4                        0              0  ...                1
```

```
...                    ...             ...  ...               ...
88642                    0               0  ...                 1
88643                    0               0  ...                 1
88644                    0               0  ...                 1
88645                    0               0  ...                 1
88646                    0               0  ...                 1

       qty_nameservers  qty_mx_servers  ttl_hostname  tls_ssl_certificate  \
0                    2               0           892                    0
1                    2               1          9540                    1
2                    2               3           589                    1
3                    2               0           292                    1
4                    2               1          3597                    0
...                ...             ...           ...                  ...
88642                3               1          3597                    0
88643                2               2           591                    0
88644                2               5         14391                    1
88645                1               1            52                    1
88646                4               0           299                    0

       qty_redirects  url_google_index  domain_google_index  url_shortened  \
0                  0                 0                    0              0
1                  0                 0                    0              0
2                  0                 0                    0              0
3                  0                 0                    0              0
4                  1                 0                    0              0
...              ...               ...                  ...            ...
88642              0                 0                    0              0
88643              2                 0                    0              0
88644              0                 0                    0              0
88645              0                 0                    0              0
88646              0                 0                    0              0

       phishing
0             1
1             1
2             0
3             1
4             0
...         ...
88642         0
88643         0
88644         1
88645         1
88646         0

[88647 rows x 112 columns]
```

```
data.head()
```

	qty_dot_url	qty_hyphen_url	qty_underline_url	qty_slash_url	qty_questionma
0	3	0	0	1	
1	5	0	1	3	
2	2	0	0	1	
3	4	0	2	5	

```
X = data.drop('phishing', axis=1)
y = data['phishing']
print(X,y)
```

```
       qty_dot_url  qty_hyphen_url  qty_underline_url  qty_slash_url  \
0                3               0                  0              1
1                5               0                  1              3
2                2               0                  0              1
3                4               0                  2              5
4                2               0                  0              0
...            ...             ...                ...            ...
88642            3               1                  0              0
88643            2               0                  0              0
88644            2               1                  0              5
88645            2               0                  0              1
88646            2               0                  0              0

       qty_questionmark_url  qty_equal_url  qty_at_url  qty_and_url  \
0                         0              0           0            0
1                         0              3           0            2
2                         0              0           0            0
3                         0              0           0            0
4                         0              0           0            0
...                     ...            ...         ...          ...
88642                     0              0           0            0
88643                     0              0           0            0
88644                     0              0           0            0
88645                     0              0           0            0
88646                     0              0           0            0

       qty_exclamation_url  qty_space_url  ...  time_domain_expiration  \
0                        0              0  ...                      -1
1                        0              0  ...                     150
2                        0              0  ...                      -1
3                        0              0  ...                      -1
4                        0              0  ...                     306
...                    ...            ...  ...                     ...
88642                    0              0  ...                     334
88643                    0              0  ...                     431
88644                    0              0  ...                     712
88645                    0              0  ...                      -1
88646                    0              0  ...                      64

       qty_ip_resolved  qty_nameservers  qty_mx_servers  ttl_hostname  \
0                    1                2               0           892
1                    1                2               1          9540
```

```
2                     1                  2                 3            589
3                     1                  2                 0            292
4                     1                  2                 1           3597
...                 ...                ...               ...            ...
88642                 1                  3                 1           3597
88643                 1                  2                 2            591
88644                 1                  2                 5          14391
88645                 1                  1                 1             52
88646                 1                  4                 0            299

       tls_ssl_certificate  qty_redirects  url_google_index  \
0                        0              0                 0
1                        1              0                 0
2                        1              0                 0
3                        1              0                 0
4                        0              1                 0
```

```
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=101


import time

start_time = time.time()




import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score


from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
from keras.layers import Input, Dense
from keras.models import Model, Sequential
from keras import regularizers


print(X_train)
```

```
       qty_dot_url  qty_hyphen_url  qty_underline_url  qty_slash_url  \
15357            1               0                  0              0
13293            1               0                  0              1
20366            2               0                  0              0
5760             2               0                  0              0
1862             3               0                  1              4
...            ...             ...                ...            ...
5695             2               1                  1              9
73542            2               0                  0              0
83281            2               0                  0              0
83467            2               0                  0              0
```

```
45919             2                0                  0             0

       qty_questionmark_url  qty_equal_url  qty_at_url  qty_and_url  \
15357                     0              0           0            0
13293                     0              0           0            0
20366                     0              0           0            0
5760                      0              0           0            0
1862                      0              1           0            1
...                     ...            ...         ...          ...
5695                      0              0           0            0
73542                     0              0           0            0
83281                     0              0           0            0
83467                     0              0           0            0
45919                     0              0           0            0

       qty_exclamation_url  qty_space_url  ...  time_domain_expiration  \
15357                    0              0  ...                      -1
13293                    0              0  ...                     229
20366                    0              0  ...                      97
5760                     0              0  ...                     363
1862                     0              0  ...                      81
...                    ...            ...  ...                     ...
5695                     0              0  ...                     236
73542                    0              0  ...                    1134
83281                    0              0  ...                     326
83467                    0              0  ...                      -1
45919                    0              0  ...                      33

       qty_ip_resolved  qty_nameservers  qty_mx_servers  ttl_hostname  \
15357                1                1               1          9175
13293                1                4               1          3590
20366                1                2               2          3600
5760                 4                2               2            54
1862                 1                2               7         21547
...                ...              ...             ...           ...
5695                 1                2               1         14365
73542                1                4               1           289
83281                1                2               1          3597
83467                1                2               3           291
45919                1                2               0           298

       tls_ssl_certificate  qty_redirects  url_google_index  \
15357                    1              1                 0
13293                    0             -1                 0
20366                    1              1                 0
5760                     1              1                 0
1862                     0              1                 0
```

```
ann = tf.keras.models.Sequential()
ann.add(tf.keras.layers.Dense(64, activation = 'relu'))
ann.add(tf.keras.layers.Dense(32, activation = 'Softmax'))
ann.add(tf.keras.layers.Dense(1, activation = 'sigmoid'))
ann.compile(optimizer = 'adam', loss = 'msle', metrics = ['accuracy'])
ann.fit(X_train,y_train,epochs=10,batch_size=1024)
```

```
Epoch 1/10
```

```
59/59 [==============================] - 1s 6ms/step - loss: 0.1206 - accuracy:
Epoch 2/10
59/59 [==============================] - 0s 6ms/step - loss: 0.1138 - accuracy:
Epoch 3/10
59/59 [==============================] - 0s 5ms/step - loss: 0.1113 - accuracy:
Epoch 4/10
59/59 [==============================] - 0s 6ms/step - loss: 0.1098 - accuracy:
Epoch 5/10
59/59 [==============================] - 0s 6ms/step - loss: 0.1089 - accuracy:
Epoch 6/10
59/59 [==============================] - 0s 6ms/step - loss: 0.1084 - accuracy:
Epoch 7/10
59/59 [==============================] - 0s 6ms/step - loss: 0.1083 - accuracy:
Epoch 8/10
59/59 [==============================] - 0s 6ms/step - loss: 0.1080 - accuracy:
Epoch 9/10
59/59 [==============================] - 0s 6ms/step - loss: 0.1079 - accuracy:
Epoch 10/10
59/59 [==============================] - 0s 6ms/step - loss: 0.1078 - accuracy:
<keras.callbacks.History at 0x7fb33c8a1b70>
```

```
y_pred=ann.predict(X_test)
```

```
915/915 [==============================] - 2s 2ms/step
```

```
y_pred=(y_pred)
```

```
y_pred=y_pred.astype(int)
```

```
from sklearn.metrics import accuracy_score
print('ACCURACY SCORE : '+str(accuracy_score(y_test,y_pred)*100))
from sklearn.metrics import recall_score
print('RECALL SCORE : '+str(recall_score(y_test,y_pred,average='macro')*100))

from sklearn.metrics import precision_score
print('PRECISION SCORE : '+str(precision_score(y_test,y_pred,average='macro')*100))

from sklearn.metrics import f1_score
print('F1 SCORE : '+str(f1_score(y_test,y_pred,average='macro')*100))
```

```
ACCURACY SCORE : 65.37909345730498
RECALL SCORE : 50.0
PRECISION SCORE : 32.68954672865249
F1 SCORE : 39.53286482017362
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344
  _warn_prf(average, modifier, msg_start, len(result))
```

RNN

```
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN
model2 = Sequential()
```

```
model2.add(SimpleRNN(32, input_shape=(111, 1), return_sequences=True))
model2.add(SimpleRNN(32))
model2.add(Dense(1, activation='sigmoid'))
model2.compile(loss='mse', optimizer='adam', metrics=['accuracy'])
# Train the model on your data
model2.fit(X_train, y_train, epochs=2, batch_size=1024)
```

```
Epoch 1/2
59/59 [==============================] - 12s 179ms/step - loss: 0.2117 - accura
Epoch 2/2
59/59 [==============================] - 10s 163ms/step - loss: 0.1216 - accura
<keras.callbacks.History at 0x7fb33c7b70a0>
```

```
y_pr=model2.predict(X_test)
y_pr=y_pr.astype(int)
```

```
915/915 [==============================] - 13s 14ms/step
```

```
from sklearn.metrics import accuracy_score
print('ACCURACY SCORE : '+str(accuracy_score(y_test,y_pr)*100))
from sklearn.metrics import recall_score
print('RECALL SCORE : '+str(recall_score(y_test,y_pr,average='macro')*100))

from sklearn.metrics import precision_score
print('PRECISION SCORE : '+str(precision_score(y_test,y_pr,average='macro')*100))

from sklearn.metrics import f1_score
print('F1 SCORE : '+str(f1_score(y_test,y_pr,average='macro')*100))
```

```
ACCURACY SCORE : 65.37909345730498
RECALL SCORE : 50.0
PRECISION SCORE : 32.68954672865249
F1 SCORE : 39.53286482017362
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344
  _warn_prf(average, modifier, msg_start, len(result))
```

Autoencoder

```
# AutoEncoder Model Preparation
batch_size=1024
n_inputs = X.shape[1]
# define encoder
input_data_shape= Input(shape=(n_inputs,))
# encoder level 1
encoder= Dense(n_inputs*2)(input_data_shape)
encoder = BatchNormalization()(encoder)
encoder= LeakyReLU()(encoder)
# encoder level 2
encoder= Dense(n_inputs)(encoder)
encoder= BatchNormalization()(encoder)
encoder= LeakyReLU()(encoder)
# bottleneck
n_bottleneck = round(float(n_inputs) / 2.0)
bottleneck = Dense(n_bottleneck)(encoder)
# define decoder, level 1
```

```
decoder = Dense(n_inputs)(bottleneck)
decoder = BatchNormalization()(decoder)
decoder = LeakyReLU()(decoder)
# decoder level 2
decoder = Dense(n_inputs*2)(decoder)
decoder = BatchNormalization()(decoder)
decoder = LeakyReLU()(decoder)
```

```
# output layer
output = Dense(n_inputs, activation='linear')(decoder)
# define autoencoder model
model = Model(inputs=input_data_shape, outputs=output)
# compile autoencoder model
model.compile(optimizer='adam', loss='mse',metrics=['accuracy'])
```

```
# fit the autoencoder model to reconstruct input
history = model.fit(X_train,y_train,epochs=10,batch_size=1024)
```

```
Epoch 1/10
59/59 [==============================] - 5s 33ms/step - loss: 0.2316 - accuracy
Epoch 2/10
59/59 [==============================] - 2s 33ms/step - loss: 0.1302 - accuracy
Epoch 3/10
59/59 [==============================] - 2s 33ms/step - loss: 0.0819 - accuracy
Epoch 4/10
59/59 [==============================] - 2s 32ms/step - loss: 0.0701 - accuracy
Epoch 5/10
59/59 [==============================] - 2s 34ms/step - loss: 0.0685 - accuracy
Epoch 6/10
59/59 [==============================] - 3s 55ms/step - loss: 0.0648 - accuracy
Epoch 7/10
59/59 [==============================] - 2s 32ms/step - loss: 0.0628 - accuracy
Epoch 8/10
59/59 [==============================] - 2s 32ms/step - loss: 0.0616 - accuracy
Epoch 9/10
59/59 [==============================] - 2s 32ms/step - loss: 0.0605 - accuracy
Epoch 10/10
59/59 [==============================] - 2s 36ms/step - loss: 0.0597 - accuracy
```

```
# define an encoder model (without the decoder)
encoder = Model(inputs=input_data_shape, outputs=bottleneck)
```

```
# Defining the classification model
input_encoded = Input(shape=(111,))
hidden1 = Dense(15, activation='relu')(input_encoded)
output = Dense(1, activation='sigmoid')(hidden1)
classifier = Model(input_encoded, output)
# Compiling the model
classifier.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
# Training the model
classifier.fit(X_train, y_train, epochs=10, batch_size=1024)
```

```
Epoch 1/10
59/59 [==============================] - 1s 4ms/step - loss: 0.3661 - accuracy:
Epoch 2/10
59/59 [==============================] - 0s 4ms/step - loss: 0.3575 - accuracy:
Epoch 3/10
59/59 [==============================] - 0s 3ms/step - loss: 0.3461 - accuracy:
Epoch 4/10
59/59 [==============================] - 0s 4ms/step - loss: 0.3414 - accuracy:
Epoch 5/10
59/59 [==============================] - 0s 6ms/step - loss: 0.3413 - accuracy:
Epoch 6/10
59/59 [==============================] - 0s 5ms/step - loss: 0.3412 - accuracy:
Epoch 7/10
59/59 [==============================] - 0s 4ms/step - loss: 0.3412 - accuracy:
Epoch 8/10
59/59 [==============================] - 0s 5ms/step - loss: 0.3412 - accuracy:
Epoch 9/10
59/59 [==============================] - 0s 4ms/step - loss: 0.3411 - accuracy:
Epoch 10/10
59/59 [==============================] - 0s 5ms/step - loss: 0.3411 - accuracy:
<keras.callbacks.History at 0x7fb33763b760>
```

```
y_p=classifier.predict(X_test)
y_p=(y_p)
y_p=y_p.astype(int)
```

```
915/915 [==============================] - 2s 2ms/step
```

```
from sklearn.metrics import accuracy_score
print('ACCURACY SCORE : '+str(accuracy_score(y_test,y_p)*100))
from sklearn.metrics import recall_score
print('RECALL SCORE : '+str(recall_score(y_test,y_p,average='macro')*100))

from sklearn.metrics import precision_score
print('PRECISION SCORE : '+str(precision_score(y_test,y_p,average='macro')*100))

from sklearn.metrics import f1_score
print('F1 SCORE : '+str(f1_score(y_test,y_p,average='macro')*100))
```

```
ACCURACY SCORE : 65.48164353592671
RECALL SCORE : 50.14810426540285
PRECISION SCORE : 82.72310429783741
F1 SCORE : 39.85272772917989
```

```
X_ens = np.hstack((ann.predict(X_train),classifier.predict(X_train),model2.predict(X_train
y_ens = y_train
```

```
1857/1857 [==============================] - 3s 1ms/step
1857/1857 [==============================] - 2s 1ms/step
1857/1857 [==============================] - 28s 15ms/step
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
# Creating the Random Forest classifier for ensemble learning
```

```
rfc = RandomForestClassifier(n_estimators=200, random_state=1024)

# Fitting the Random Forest classifier to the data
rfc.fit(X_ens, y_ens)

# Making predictions on the test data
y_pred = rfc.predict(X_ens)

# Calculating the accuracy of the model
accuracy = accuracy_score(y_ens, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9896620813900628
```

```
from sklearn.metrics import accuracy_score
print('ACCURACY SCORE : '+str(accuracy_score(y_ens, y_pred)*100))
from sklearn.metrics import recall_score
print('RECALL SCORE : '+str(recall_score(y_ens, y_pred,average='macro')*100))

from sklearn.metrics import precision_score
print('PRECISION SCORE : '+str(precision_score(y_ens, y_pred,average='macro')*100))

from sklearn.metrics import f1_score
print('F1 SCORE : '+str(f1_score(y_ens, y_pred,average='macro')*100))
```

```
ACCURACY SCORE : 98.96620813900628
RECALL SCORE : 98.57170871560325
PRECISION SCORE : 99.1478180204525
F1 SCORE : 98.85039824028905
```

```
end_time = time.time()

delay = end_time - start_time
print(f"Program took {delay} seconds to execute.")
```