SAMPLE CODING

BAYESIAN OPTIMIZATION

```
from google.colab import drive

drive.mount('/content/gdrive')

import pandas as pd
import numpy as np
import keras as ke
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection  import train_test_split
from sklearn.model_selection  import cross_val_score
from keras.models  import Sequential
from keras.layers import Dense, BatchNormalization, Dropout
from tensorflow.keras.optimizers import SGD, Adam, RMSprop, Adadelta, Adagrad, Adama
x, Nadam, Ftrl
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.wrappers.scikit_learn import KerasClassifier
from math import floor
from sklearn.metrics import make_scorer, accuracy_score
#from bayes_opt import BayesianOptimization
import BayesianOptimization
#import bayes_opt.BayesianOptimization
from bayes_opt import BayesianOptimization
from sklearn.model_selection import StratifiedKFold
from keras.layers import LeakyReLU
LeakyReLU = LeakyReLU(alpha=0.1)
```

```
import warnings
warnings.filterwarnings('ignore')
pd.set_option("display.max_columns", None)


d1=pd.read_csv('/content/gdrive/MyDrive/dataset-org.csv',header=None)
print(d1)

score_acc = make_scorer(accuracy_score)


X=pd.DataFrame(d1.iloc[:,:-1].values)
Y=d1.iloc[:,-1].values


X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2, random_state = 0 )


def nn_cl_bo(neurons,activation,optimizer,learning_rate, batch_size, epochs ):
    optimizerL = ['SGD', 'Adam', 'RMSprop', 'Adadelta', 'Adagrad', 'Adamax', 'Nadam', 'Ftrl','SGD']
    optimizerD= {'Adam':Adam(lr=learning_rate), 'SGD':SGD(lr=learning_rate), 'RMSprop':RMSprop(lr=learning_rate), 'Adadelta':Adadelta(lr=learning_rate), 'Adagrad':Adagrad(lr=learning_rate), 'Adamax':Adamax(lr=learning_rate), 'Nadam':Nadam(lr=learning_rate), 'Ftrl':Ftrl(lr=learning_rate)}
    activationL = ['relu', 'sigmoid', 'softplus', 'softsign', 'tanh', 'selu', 'elu', 'exponential', 'LeakyReLU','relu']
    neurons = round(neurons)
    activation = activationL[round(activation)]
    batch_size = round(batch_size)
    epochs = round(epochs)
    def nn_cl_fun():
        opt = Adam(lr = learning_rate)
        nn = Sequential()
        nn.add(Dense(neurons, input_dim=178, activation=activation))
```

```
        nn.add(Dense(neurons, activation=activation))
        nn.add(Dense(1, activation='sigmoid'))
        nn.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
        return nn
    es = EarlyStopping(monitor='accuracy', mode='max', verbose=0, patience=20)
    nn = KerasClassifier(build_fn=nn_cl_fun, epochs=epochs, batch_size=batch_size, verbose
=0)
    kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=123)
    score = cross_val_score(nn, X_train, Y_train, scoring=score_acc, cv=kfold, fit_params={'c
allbacks':[es]}).mean()
    return score


params_nn={'neurons':(10, 100),'activation':(0, 9),'optimizer':(0,7),'learning_rate':(0.01, 1),'ba
tch_size':(200,1000),'epochs':(20, 100)}
# Run Bayesian Optimization
nn_bo = BayesianOptimization(nn_cl_bo, params_nn, random_state=111)
nn_bo.maximize(init_points=25, n_iter=4)


params_nn_ = nn_bo.max['params']
activationL = ['relu', 'sigmoid', 'softplus', 'softsign', 'tanh', 'selu','elu', 'exponential', LeakyReL
U,'relu']
params_nn_['activation']=activationL[round(params_nn_['activation'])]
```

BAYESIAN INTO ANN

```
from google.colab import drive
drive.mount('/content/gdrive')
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
d1=pd.read_csv('/content/drive/MyDrive/dataset-org.csv',header=None)
print(d1)


X=pd.DataFrame(d1.iloc[:,:-1].values)
Y=d1.iloc[:,-1].values


X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2, random_state = 0


print(X_train)


print(X_test)


print(Y_train)


print(Y_test)


from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


import tensorflow as tf


import keras  as ke


from keras.models import Sequential
from keras.layers import Dense


classifier = Sequential()
```

```
classifier.add(Dense(units= 100, kernel_initializer = 'uniform', activation = 'relu', input_dim =
 178))

classifier.add(Dense(units = 100,kernel_initializer = 'uniform', activation = 'relu'))

classifier.add(Dense(units = 1,kernel_initializer   = 'uniform', activation = 'sigmoid'))

classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

classifier.fit(X_train, Y_train, batch_size = 200, epochs = 100)

y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

y_pred

from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(Y_test, y_pred)
print(cm)
accuracy_score(Y_test,y_pred)
```

RANDOM SEARCH OPTIMIZATION

```
import numpy
import keras
import pandas as pd
from sklearn.model_selection import RandomizedSearchCV
from keras.models import Sequential
from keras.layers import Dense
import warnings
```

```
warnings.filterwarnings('ignore')
from keras.wrappers.scikit_learn import KerasClassifier

data = pd.read_csv('/content/drive/MyDrive/dataset-org.csv')

data.head()

X = data.iloc[:,0:178]
Y = data.iloc[:,178]

def create_my_model(optimizer='adam', activationL='relu', neurons=0.01):
  mymodel=Sequential()
  mymodel.add(Dense(16,input_dim=178,activation='relu'))
  mymodel.add(Dense(neurons, activation=activationL))
  mymodel.add(Dense(1,activation='sigmoid'))

  mymodel.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
  return mymodel

model=KerasClassifier(build_fn=create_my_model)


batchSize=[10,20,40,60,80,100]
epochs=[10,30,50]
optimizer=['SGD', 'Adadelta', 'RMSprop', 'Adagrad', 'Adam']
activationL= ['relu', 'sigmoid', 'selu', 'exponential', 'LeakyReLU','elu','softplus', 'softsign', 'tanh
']
neurons=(10,100)

parameter_rdm=dict(batch_size=batchSize, epochs=epochs, optimizer=opt
imizer, activationL=activationL, neurons=neurons)
```

```
rdm_search =RandomizedSearchCV(estimator=model,param_distributions= parameter_rdm,
n_jobs=-1,cv=3)
rdm_result =rdm_search.fit(X,Y)

print("Best:%f using %s"%(grid_result.best_score_,grid_result.best_params_))
```

Random search into ANN

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

d1=pd.read_csv('/content/drive/MyDrive/dataset-org.csv',header=None)
print(d1)

X=pd.DataFrame(d1.iloc[:,:-1].values)
Y=d1.iloc[:,-1].values

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2, random_state = 0 )

print(X_train)

print(Y_train)

print(Y_test)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)

import tensorflow as tf

import keras  as ke

from keras.models import Sequential
from keras.layers import Dense
classifier = Sequential()

classifier.add(Dense(units= 100, kernel_initializer = 'uniform', activation = 'relu', input_dim =
 178))

classifier.add(Dense(units = 100,kernel_initializer = 'uniform', activation = 'relu'))

classifier.add(Dense(units = 1,kernel_initializer   = 'uniform', activation = 'sigmoid'))

classifier.compile(optimizer = 'sgd', loss = 'binary_crossentropy', metrics = ['accuracy'])

classifier.fit(X_train, Y_train, batch_size = 10, epochs = 30)

y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

y_pred

from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(Y_test, y_pred)
print(cm)
accuracy_score(Y_test,y_pred)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(Y_test, y_pred)
print(cm)
accuracy_score(Y_test,y_pred)
```