Supplementary Material

Python Code

The following code is also available as a Python file at https://osf.io/68kgy/ (see ANN_step-by-step.py)

```
#Load features
import pandas as pd
x_train = pd.read_csv("TrainingFeatures.csv", header = None)
x_train = x_train.values

#Load labels
y_train = pd.read_csv("TrainingLabels.csv", header = None)
y_train = (y_train.values).flatten()

#Import packages and methods
import numpy as np
np.random.seed(48151)
import tensorflow as tf
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
tf.random.set_seed(48151)

#Create an artificial neural network
ann = Sequential()
#Add first hidden layer
ann.add(Dense(12, input_shape=(24, ), activation = "relu"))
#Output layer
ann.add(Dense(1, activation='sigmoid'))
#Loss function
ann.compile(loss='binary_crossentropy')

#Train model
ann.fit(x_train, y_train, epochs = 20, class_weight = {0:1, 1:0.25})

#Import test data
x_test = pd.read_csv("TestFeatures.csv", header = None)
y_test = pd.read_csv("TestLabels.csv", header = None)
x_test = x_test.values
y_test = (y_test.values).flatten()

#Get predictionss on test set
predictions = np.round(ann.predict(x_test))

#Accuracy measure
accuracy = np.sum(predictions.flatten() == y_test)/2000
print(accuracy)

#Type I error rate
idx_noeffect, = np.where(y_test == 0)
typeIerror = np.sum(predictions[idx_noeffect])/len(idx_noeffect)
print(typeIerror)

#Power
idx_effect, = np.where(y_test == 1)
```

```
power = np.sum(predictions[idx_effect])/len(idx_effect)
print(power)

#Split training dataset for hyperparameter tuning
from sklearn.model_selection import train_test_split
x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train,
                                                      test_size=0.5, random_state=48151)

#List of number of neurons
nbneurons_list = range(3,25,3)

#Initialize best accuracy and the optimal number of neurons at 0
best_accuracy = 0
optimal_nbneurons = 0

#Repeat for each number of neurons value
for nbneurons_value in nbneurons_list:
    #Create model
    ann = Sequential()
    ann.add(Dense(nbneurons_value, input_shape=(24, ), activation = "relu"))
    ann.add(Dense(1, activation='sigmoid'))
    ann.compile(loss='binary_crossentropy')
    #Train model
    ann.fit(x_train, y_train, epochs = 20, class_weight = {0:1, 1:0.25})
    #Predict result on validation set
    predictions = np.round(ann.predict(x_valid))
    #Accuracy on validation set
    accuracy = np.sum(predictions.flatten() == y_valid)/len(y_valid)
    #If model improves accuracy, save model, accuracy, and number of neurons
    if accuracy > best_accuracy:
        ann.save('best_ann.h5')
        best_accuracy = accuracy
        optimal_nbneurons = nbneurons_value

#Check predictions and outcomes on test data from Lanovaz & Turgeon(2020)
best_ann = tf.keras.models.load_model('best_ann.h5')
predictions = np.round(best_ann.predict(x_test))
accuracy = np.sum(predictions.flatten() == y_test)/2000
print(accuracy)
typeIerror = np.sum(predictions[idx_noeffect])/len(idx_noeffect)
print(typeIerror)
power = np.sum(predictions[idx_effect])/len(idx_effect)
print(power)

#Import data for a graph
series = (pd.read_excel('application.xlsx', header = None)).values
#Extract the features used by the program
from extract_features import extractFeatures
features = extractFeatures(series)

#Import model
best_ann = tf.keras.models.load_model('best_ann.h5')
#Make prediction on graph
prediction = np.round(best_ann.predict(features.reshape(1,-1)))
print(prediction)
```