In [ ]:
```
#Bibliotecas p/ importação
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

from keras import layers

sc = StandardScaler()# Método de Estandardização por Escalonamento
import warnings
warnings.filterwarnings("ignore")
import numpy as np #Manipulação de vetores, matrizes, operações, etc.
import pandas as pd #Visualização, organização nos DataFrames (tabelas)
import tensorflow as tf #Machine Learning etc.
import matplotlib.pyplot as plt #Plotagem e visualização
from sklearn import svm #SVM padrão
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV #Model tunning
from sklearn.preprocessing import StandardScaler #Estandardização por escalonamento
from sklearn.feature_selection import RFECV #Seleção por eliminação recursiva de atributos
from sklearn.metrics import confusion_matrix #Matriz de confusão p/ discriminar os erros e acertos na rede neural

import DADOS
import random
random.seed(13)
plt.rcParams["figure.figsize"]=18,18
%matplotlib qt
pd.set_option('display.max_colwidth', None)
```

In [ ]:
```
from keras import Sequential
from keras.layers import Dense
```

In [ ]:
```
featureGroup = list(DADOS.Maquinas.keys())
featureVarList = [list(DADOS.Maquinas[str(x)].columns) for x in list(DADOS.Maquinas.keys())]
features = pd.DataFrame([list(DADOS.Maquinas.keys()), [list(DADOS.Maquinas[str(x)].columns) for x in list(DADOS.Maquinas.keys())]]).T
print(features.to_latex(index=False))
```

In [ ]:
```
#Define pasta principal como diretório de trabalho
os.chdir('c:\\Users\\pablo\OneDrive - ufmt.br\\1 - Análise de Estabilidade Transitória Utilizando Aprendizagem de Máquina')
```

In [ ]:
```
parametros = {'Kernel':'linear', #Parâmetro p/ SVM - Tipo de kernel
              'Funcao perda':'binary_crossentropy',#Parâmetro p/ RNA - Função de pontuação de otimização
              'Otimizador':'SGD',#Parâmetro p/ RNA - Forma de cálculo de descida do gradiente do treinamento
              'Ativacao camadas':'relu',#Parâmetro p/ RNA - Função de ativação das camadas de entrada e escondidas
              'Ativacao saida':'sigmoid',#Parâmetro p/ RNA - Função de ativação da camada de saída
              }

conjuntos = DADOS.Maquinas  #Dicionário com cada conjunto associado a um índice

index = list()
for x in conjuntos: index.append(x) #Lista de índices com os conjuntos

y = DADOS.y
```

In [ ]:
```
#Função p/ treinamento e teste de RNA
#caso 0 - Apenas camadas de entrada e saída (2 layers)
#caso 1 - Uma única camada entre camadas de entrada e saída (3 layers)
#caso 2 - Quatro camadas entre camadas de entrada e saída (6 layers)
#A saída da função é uma única tabela resumo dos resultados obtidos na execução
def rna(rfe, conjuntos, index, parametros):
    resultado = pd.DataFrame()
    perda = parametros['Funcao perda']
    otimizador = parametros['Otimizador']
    fatv_camadas = parametros['Ativacao camadas']
    fatv_camada_saida = parametros['Ativacao saida']

    for conjunto in conjuntos: #Para cada índice do dicionário 'conjuntos',
        X = conjuntos[str(conjunto)] # a variável X recebe o índice
        X_treino, X_teste, y_treino, y_teste = train_test_split(abs(X.values), y.values.ravel(),
                                                                test_size=0.3, shuffle = True) #Separação dos dados
        X_treino = sc.fit_transform(X_treino)
        X_teste = sc.fit_transform(X_teste)
        if rfe:
            selec_cv = RFECV(SVC(C=1, kernel='linear'), step=1, cv=StratifiedKFold(2))
            selec_cv = selec_cv.fit(X_treino, y_treino)
            #Retorna os atributos selecionados e os índices da seleção

            X_selected = np.array(X[X.columns[selec_cv.get_support()]])

            selected_inputs = X.columns[selec_cv.get_support()]

            X_treino, X_teste, y_treino, y_teste = train_test_split(abs(X.values), y.values.ravel(),
                                                                    test_size=0.3, shuffle = True) #Separação dos dados
            X_treino = sc.fit_transform(X_treino)
            X_teste = sc.fit_transform(X_teste)
        #Modelagem de camadas de neurônios
        modelo = Sequential()

        #Camada de Entrada  -> dimensão da entrada = número de variáveis
        modelo.add(Dense(64, input_dim=X_treino.shape[1], activation=fatv_camadas))

        modelo.add(Dense(1, activation=fatv_camada_saida))

        #Compila o modelo - baseia-se no dicionário 'parametros'
        modelo.compile(loss=perda,
                       optimizer=otimizador,
                       metrics=['accuracy'])

        #17.5% dos dados de treino p/ CV - 25% dos dados do conjunto
        #Batch size - número de amostras processadas antes de atualizar o modelo da RNA
        batch_size = 128

        #Steps per epoch - número de iterações de batch antes de terminar uma época = nº amostras/tamanho do batch
        steps_per_epoch = int(X_treino.shape[0]/batch_size)

        #Ajuste do modelo
        clf_rna = modelo.fit(X_treino,y_treino, validation_split=0.175, batch_size=None, epochs=30,
                             validation_steps = steps_per_epoch*2, steps_per_epoch=steps_per_epoch)
        #Aavaliação do Modelo
        eval_treino=modelo.evaluate(X_treino, y_treino)
        eval_teste=modelo.evaluate(X_teste, y_teste)

        #Predição com os dados de teste
        y_pred = modelo.predict(X_teste)

        #Classificação p/ construção de matriz de confusão
        #y_pred>0.5 -> True -> Classificado corretamente; y_pred>0.5 -> False -> Classificado incorretamente
        y_pred = (y_pred>0.5)
        cm = confusion_matrix(y_teste, y_pred)

        #Construção da tabela de saída
        resultado = resultado.append({'Teste - Acertos p/ Estável':cm[0][0],
                        'Teste - Erros p/ Estável':cm[0][1],
                        'Teste - Acertos p/ Instável':cm[1][1],
                        'Teste - Erros p/ Instável':cm[1][0],
                        'Precisão de Treino (%)':eval_treino[1]*100,
                        'Precisão de Teste (%)':eval_teste[1]*100}, ignore_index=True)
    resultado.set_index(np.array(index), inplace=True)

    return resultado
```

In [ ]:
```
Resultado_RNA_RFE = rna(True, conjuntos, index, parametros)
Resultado_RNA = rna(False, conjuntos, index, parametros)
```