In [ ]:
```
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

from itertools import product
sc = StandardScaler()

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

In [ ]:
```
#Classe básica da SVM, agrega as entradas e processa o treino e teste baseando-se nelas
#RFECV_select() processa o algoritmo de seleção recursiva de atributos por validação cruzada
#Otimiza_C() processa uma busca em grid do paramêtro de regularização C ótimo
#Reporta() emite um relatório da SVM para apresentação.
#A lista de todos os atributos da classe Build_SVM() podem ser acessados com a passagem do método .__dict__
class Build_SVM:
    def __init__(self, X, y, labels, conjunto, C, kernel):
        #Armazena as entradas
        self.conjunto = conjunto
        self.labels = labels
        self.X_raw = X
        self.X = np.array(self.X_raw)
        self.y = np.array(y)
        # Separação dos dados em conjuntos de treino e teste
        self.X_treino, self.X_teste, self.y_treino, self.y_teste = train_test_split(self.X, self.y, test_size=0.3, shuffle = True) #Separação dos dados
        #Shuffle true pra ser aleatorio com random.seed=13
        #Estandardização por escalonamento
        #z = (x - u) / s
        #u é a média da amostra de treino; s desvio padrão da amostra de treino
        #Padroniza as amostras removendo a média e escalando o desvio padrão
        self.X_treino_norm, self.X_teste_norm = sc.fit_transform(self.X_treino), sc.fit_transform(self.X_teste)

        # Construção do modelo de classificador SVM
        self.Model = SVC(C=C, kernel=kernel) # É de interesse apenas C e kernel, com o resto dos parâmetros em default
        self.Model.fit(self.X_treino_norm, self.y_treino) # O classificador se ajusta aos dados de treinamento


        self.Precisao_Treino = self.Model.score(self.X_treino_norm, self.y_treino) #Pontuação de precisão de treino
        self.Precisao_Teste = self.Model.score(self.X_teste_norm, self.y_teste) #Pontuação de precisão de teste


    #'RFECV_selected()' função para aplicar RFECV e retornar os dados selecionados e os índices das variáveis
    def RFECV_select(self, cv):
        #Aplicação do selecionador RFECV, com remoção de 1 atributo para cada iteração do método
        self.selec_cv = RFECV(SVC(C=self.Model.C, kernel=self.Model.kernel), step=1, cv=StratifiedKFold(cv))
        self.selec_cv = self.selec_cv.fit(self.X_treino_norm, self.y_treino)

        #Retorna os atributos selecionados e os índices da seleção
        self.X_selected = np.array(self.X_raw[self.X_raw.columns[self.selec_cv.get_support()]])
        self.selected_inputs = self.X_raw.columns[self.selec_cv.get_support()]

        self.X_selected_treino, self.X_selected_teste, self.y_treino, self.y_teste = train_test_split(self.X_selected, self.y, test_size=0.3, shuffle = T
        self.X_s_treino_norm, self.X_s_teste_norm = sc.fit_transform(self.X_selected_treino), sc.fit_transform(self.X_selected_teste)


        self.Model_selected = SVC(C=self.Model.C, kernel='linear')
        self.Model_selected.fit(self.X_s_treino_norm, self.y_treino) # O classificador se ajusta aos dados de treinamento

        self.Precisao_Treino_selected = self.Model_selected.score(self.X_s_treino_norm, self.y_treino) #Pontuação de precisão de treino
        self.Precisao_Teste_selected = self.Model_selected.score(self.X_s_teste_norm, self.y_teste) #Pontuação de precisão de teste

        return self.X_selected, self.selected_inputs

    def Otimiza_C(self):
        # Arranjo de parâmetros para teste. O melhor parâmetro C será escolhido para o modelo de SVM
        param_grid = {'kernel':['linear'],'C':[10, 1, 0.1, 0.01, 0.001]}

        # Declara a função GridSearch e adequa o modelo para o conjunto de dados de treino
        # A função irá testar os dados com todos os itens do arranjo de parâmetros declarados
        self.grid_search = GridSearchCV(self.Model, param_grid)
        self.grid_search.fit(self.X_treino_norm, self.y_treino)

        #Muda os parâmetro do modelo principal de acordo com o resultado da busca em grid
        self.Model.set_params(**self.grid_search.best_params_)

    def Reporta(self, rfe):
        #Declara dados de exibição - Armazenados em self.df_OUT
        self.report_RFECV = {'Precisão de Treino: ': '{0:.3f}%'.format(self.Precisao_Treino_selected*100), 'Precisão de Teste: ': '{0:.3f}%'.format(self.

        self.report = {'Precisão de Treino: ': '{0:.3f}%'.format(self.Precisao_Treino*100), 'Precisão de Teste: ': '{0:.3f}%'.format(self.Precisao_Teste*

        self.df_OUT = pd.DataFrame([x for x in (self.report.keys(), self.report.values())])
        self.df_OUT_RFECV = pd.DataFrame([x for x in (self.report_RFECV.keys(), self.report_RFECV.values())])
        print('\nResultados SVM Padrão:')
        print(self.df_OUT)
        print('\nResultados SVM com Seleção Recursiva de Atributos por Validação Cruzada:')
        print(self.df_OUT_RFECV)

        #print('\nDados: {}'.format(self.labels))
        print(r'=============================================================================================================')

def SVM_LOOP(cv_rfecv=2, rfecv=True, report=True):
    out = list()
    for n in DADOS.Maquinas.keys():
        print('Construindo Modelo para o conjunto {} - C = 0.001 ...'.format(n))
        model = Build_SVM(DADOS.Maquinas[n], DADOS.y, list(DADOS.Maquinas[n].columns), n, 0.001, 'linear')
        model.Otimiza_C()
        print('Ok! \nC otimizado: {}'.format(model.Model.C))

        if rfecv:
            print(r'-------------------------------------------------------------------------------------------------------------')
            print(r'Seleção de Atributos por Eliminação Recursiva com Validação Cruzada')
            out_RFE = model.RFECV_select(cv_rfecv)
            print('Atributos do Conjunto:\n ', list(DADOS.Maquinas[n].columns))
            print('Atributos Selecionados:\n ', list(out_RFE[1]))
            print('Redução de {} atributos'.format(len(list(DADOS.Maquinas[n].columns))-len(list(out_RFE[1]))))

        if report:
            model.Reporta(True)

        out.append(model)
        #print([list(DADOS.Maquinas[n].columns), n])
    return dict(zip(DADOS.Maquinas.keys(), out))

SVM_maquinas = SVM_LOOP()
```

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

In [ ]:
```
res = list()
outs = list()
outsrfe = list()
for key in SVM_maquinas.keys():
    res.append([key, SVM_maquinas[key].Model_selected.C])
    outs.append(SVM_maquinas[key].df_OUT[1:])
    outsrfe.append(SVM_maquinas[key].df_OUT_RFECV[1:])
featureC = pd.DataFrame(res)
```

In [ ]:
```
outDF = pd.concat(outs)
outDF = outDF.drop(2, axis=1)
outDF_RFE = pd.concat(outsrfe)
outDF_RFE = outDF_RFE.drop(2, axis=1)
outDF = outDF[outDF.columns[::-1]]
outDF_RFE = outDF_RFE[outDF_RFE.columns[::-1]]
```