from multiprocessing import process

import os

from PyQt6 import QtCore, QtGui, QtWidgets

from UI\_Dialog import ErrorDialog

from UI\_ProgressDialog import ProgressDialog

import subprocess

import csv

import sys

import sys

import numpy as np

from intuition\_fuzzy2 import IntuitiveFuzzy

from sklearn.preprocessing import LabelEncoder

from sklearn import preprocessing

from tabulate import tabulate

import warnings, os, time

from sklearn.model\_selection import KFold

from operator import itemgetter

class Worker(*QtCore*.QThread):

    finished = QtCore.pyqtSignal()

    def \_\_init\_\_(*self*, *path*, *col*, *row*, *row\_selected*, *alpha*, *delta*):

        super().\_\_init\_\_()

*self*.path = *path*

*self*.col = *col*

*self*.row = *row*

*self*.row\_selected = *row\_selected*

*self*.delta = *delta*

*self*.alpha = *alpha*

    def run(*self*):

        shell = False

        command = [

            "python",

            "app.py",

            str(*self*.path),

            str(*self*.col),

            str(*self*.row),

            str(*self*.alpha),

            str(*self*.delta),

            str(*self*.row\_selected),

        ]

        # Sử dụng subprocess.Popen() với tham số creationflags để ẩn cửa sổ cmd

        subprocess\_process = subprocess.Popen(command)

        subprocess\_process.wait()  # Chờ cho quá trình kết thúc

*self*.finished.emit()

class Ui\_ReducedData(object):

    def \_\_init\_\_(*self*) -> None:

*self*.path = None

*self*.col = None

*self*.row = None

*self*.delta = None

*self*.alpha = 0

*self*.row\_selected = 0

    "process"

    arr\_data = []

    min\_max\_scaler = preprocessing.MinMaxScaler()

    def preprocessing(*self*, *name\_file*, *att\_nominal\_cate*):

        DS = np.genfromtxt(*name\_file*, *delimiter*=",", *dtype*=object)[:, :]

        att = DS[0].astype(int)

*att\_nominal\_cate* = np.array(*att\_nominal\_cate*)

        att\_real = np.setdiff1d(att, *att\_nominal\_cate*)

        DS[0] = att

        for i in *att\_nominal\_cate*:

            DS[1:, i] = LabelEncoder().fit\_transform(DS[1:, i])

        DS[1:, :] = DS[1:, :]

        DS[1:, att\_real] = *self*.min\_max\_scaler.fit\_transform(DS[1:, att\_real])

        return DS[1:]

    def split\_data\_icr(*self*,*data*, *row\_selected*):

        arrs = []

        data\_1 = *data*[:*row\_selected*]

        data\_2 = *data*[*row\_selected*:]

        arrs.append(data\_1)

        arrs.append(data\_2)

        return arrs

    def mainRun(*self*):

        path = *self*.path

        col = int(*self*.col)

        alpha = float(*self*.alpha)

        delta = float(*self*.delta.currentText())

        row\_selected = int(*self*.row\_selected)

        new\_data = [path, [col - 1], delta]

        arr\_data = []

        arr\_data.append(new\_data)

        start = time.time()

        a\_sc = [["Data","|C|", "|R\_F|", "Acc\_O","std\_O", "Acc\_F", "std\_F", "T\_F", "Reduct", "Alpha"]]

        num\_prev = 0

        dis\_tg = 0

        X = [alpha]

        for arr in arr\_data:

            for x in X:

                F = []

        DS = *self*.preprocessing(arr[0], arr[1])

        st = time.time()

        DS = *self*.split\_data\_icr(DS,row\_selected)

        # step 1: Compute IFPDs on original dataset.

        IF = IntuitiveFuzzy(DS[0], arr[0], arr[1], arr[2], x, F, num\_prev, dis\_tg)

        F, dis\_tg, time\_filter, dis\_tg\_C, dis\_tg\_B= IF.filter()

        # print("F", F)

        sc = IF.evaluate(arr[0], F, time\_filter)

        a\_sc.append(sc)

        # os.system('cls')

        # print (tabulate(a\_sc, headers='firstrow', tablefmt='pipe', stralign='center'))

        # os.system('cls')

        U = DS[0]

        column\_order = ["Reduct", "Size of the reduct", "Acc\_O ± std\_O", "Acc\_F ± std\_F", "Runtime", "Alpha", "Dis\_Tg","Row\_select", "Delta", "Dis\_Tg\_B", "Dis\_Tg\_C"]

        file\_name = os.path.splitext(os.path.basename(arr[0]))[0] + '\_output.txt'

        with open(file\_name, 'w') as f:

            f.write('\t'.join(column\_order) + '\n')

            for i, row in enumerate(a\_sc):

                if i >= 1:

                    f.write("{}\t{}\t{}\t{}\t{}\t{}\t{}\t{}\t{}\t{}\t{}\n".format(row[8], row[2], str(row[3]) + " ± " + str(row[4]), str(row[5]) + " ± " + str(row[6]), row[7], row[9], dis\_tg, row\_selected, delta, dis\_tg\_B, dis\_tg\_C))  # Thêm dis\_tg vào hàng ghi

    def go\_back(*self*):

        from UI\_Home import Ui\_MainWindow

*self*.window = QtWidgets.QMainWindow()

*self*.ui = Ui\_MainWindow()

*self*.ui.setupUi(*self*.window)

*self*.window.show()

        QtCore.QTimer.singleShot(

            0, QtWidgets.QApplication.instance().activeWindow().close

        )

    def increase\_alpha(*self*):

        current\_alpha = float(*self*.ledAlpha.text())

        new\_alpha = min(current\_alpha + *self*.delta, 1.0)

*self*.ledAlpha.setText(str(new\_alpha))

    def decrease\_alpha(*self*):

        current\_alpha = float(*self*.ledAlpha.text())

        new\_alpha = max(current\_alpha - *self*.delta, 0.0)

*self*.ledAlpha.setText(str(new\_alpha))

    def importData(*self*):

        file\_path, \_ = QtWidgets.QFileDialog.getOpenFileName(

            None, "Import Data", "", "CSV Files (\*.csv)"

        )

*self*.path = file\_path

        data = []

        if file\_path:

            file\_name = os.path.basename(file\_path)

            with open(file\_path, "r") as file:

                csv\_reader = csv.reader(file)

                data = list(csv\_reader)

            if data:

                headers = data[0]

                num\_rows = len(data) - 1

                num\_rows\_View = len(data) - 1

                num\_cols = len(headers)

*self*.col = num\_cols

*self*.row = num\_rows

                decision\_classes = set(row[-1] for row in data[1:])

                num\_decision\_classes = len(decision\_classes)

*self*.data = data

*self*.tabAns.setColumnCount(num\_cols)

*self*.tabAns.setRowCount(9)

*self*.tabAns.setHorizontalHeaderLabels(map(str, headers))

                for row\_idx, row\_data in enumerate(data[1:10:]):

                    for col\_idx, cell\_value in enumerate(row\_data):

                        item = QtWidgets.QTableWidgetItem(str(cell\_value))

                        # Set font and background color for cells

                        item.setFont(QtGui.QFont("Arial", 10))

                        item.setBackground(QtGui.QColor(240, 240, 240))

*self*.tabAns.setItem(row\_idx, col\_idx, item)

*self*.labInfor.setText(

                    f"Data imported successfully from file: \n{file\_name}.\nRows: {num\_rows}\nColumns: {num\_cols}\nDecision Classes: {num\_decision\_classes}"

                )

*self*.tabAns.resizeColumnsToContents()

            else:

*self*.labInfor.setText("No file selected.")

*self*.tabAns.resizeColumnsToContents()

    def run\_app(*self*):

        alpha\_text = *self*.ledAlpha.text()

        if not *self*.path:

            error\_dialog = ErrorDialog("")

            error\_dialog.show\_error("No file selected. Please import a CSV file.")

            return

        if not alpha\_text:

            error\_dialog = ErrorDialog("")

            error\_dialog.show\_error("Please enter the value of alpha.")

            return

        try:

            alpha = float(alpha\_text)

            if not 0 <= alpha <= 1:

                error\_dialog = ErrorDialog("")

                error\_dialog.show\_error(

                    "The value of alpha is not within the range from 0 to 1."

                )

                return

        except ValueError:

            error\_dialog = ErrorDialog("")

            error\_dialog.show\_error("The value of alpha is invalid.")

            return

        row\_selected\_text = *self*.ledSelectedRow.text()

        if not row\_selected\_text:

            error\_dialog = ErrorDialog("")

            error\_dialog.show\_error("Please enter the number of the selected row..")

            return

        try:

            row\_selected = int(row\_selected\_text)

*self*.row\_selected = row\_selected

            row = int(*self*.row)

            if row\_selected <= 0 or row\_selected > row:

                error\_dialog = ErrorDialog("")

                error\_dialog.show\_error(f"The selected row is invalid.")

                return

        except ValueError:

            error\_dialog = ErrorDialog("")

            error\_dialog.show\_error("The value of the selected row is invalid.")

            return

        delta = float(*self*.delta.currentText())

*self*.mainRun()

    def on\_process\_finished(*self*):

*self*.progress\_dialog.close()

*self*.tabAns.setRowCount(0)

        file\_name\_with\_ext = os.path.basename(*self*.path)

        file\_name, \_ = os.path.splitext(file\_name\_with\_ext)

        output\_file\_name = file\_name + "\_output.txt"

        with open(output\_file\_name, "r") as file:

            reader = csv.reader(file, *delimiter*="\t")

            column\_names = next(reader)

*self*.tabAns.setColumnCount(len(column\_names))

*self*.tabAns.setHorizontalHeaderLabels(column\_names)

            for row\_idx, row in enumerate(reader):

*self*.tabAns.insertRow(row\_idx)

                for col\_idx, cell\_value in enumerate(row):

                    item = QtWidgets.QTableWidgetItem(str(cell\_value))

                    item.setFont(QtGui.QFont("Arial", 10))

                    item.setBackground(QtGui.QColor(240, 240, 240))

*self*.tabAns.setItem(row\_idx, col\_idx, item)

        num\_columns = *self*.tabAns.columnCount()

        # Loop through all columns and set bold font for headers

        for col\_idx in range(min(num\_columns, 7)):  # Only for the first 7 columns

            header\_item = *self*.tabAns.horizontalHeaderItem(col\_idx)

            if header\_item is not None:

                font = QtGui.QFont()

                font.setBold(True)

                header\_item.setFont(font)

*self*.tabAns.setWordWrap(True)

        # Set the width for the first 7 columns

        first\_column\_width = round(0.4 \* *self*.tabAns.width())

*self*.tabAns.setColumnWidth(0, first\_column\_width)

*self*.tabAns.setColumnWidth(1, round(0.1 \* *self*.tabAns.width()))

*self*.tabAns.setColumnWidth(2, round(0.15 \* *self*.tabAns.width()))

*self*.tabAns.setColumnWidth(3, round(0.15 \* *self*.tabAns.width()))

*self*.tabAns.setColumnWidth(4, round(0.0932 \* *self*.tabAns.width()))

*self*.tabAns.setColumnWidth(5, round(0.0932 \* *self*.tabAns.width()))

        # Hide the remaining columns

        for col\_idx in range(6, num\_columns):

*self*.tabAns.setColumnHidden(col\_idx, True)

    def setupUi(*self*, *mainWindow*):

*mainWindow*.setObjectName("mainWindow")

*mainWindow*.setFixedSize(1300, 616)

        sizePolicy = QtWidgets.QSizePolicy(

            QtWidgets.QSizePolicy.*Policy*.Fixed, QtWidgets.QSizePolicy.*Policy*.Fixed

        )

        sizePolicy.setHorizontalStretch(0)

        sizePolicy.setVerticalStretch(0)

        sizePolicy.setHeightForWidth(*mainWindow*.sizePolicy().hasHeightForWidth())

*mainWindow*.setSizePolicy(sizePolicy)

*self*.centralwidget = QtWidgets.QWidget(*parent*=*mainWindow*)

*self*.centralwidget.setObjectName("centralwidget")

        # Button to import data

*self*.btnImportData = QtWidgets.QPushButton(

*parent*=*self*.centralwidget, *clicked*=lambda: *self*.importData()

        )

*self*.btnImportData.setGeometry(QtCore.QRect(40, 40, 251, 23))

        font = QtGui.QFont()

        font.setFamily("Tahoma")

        font.setPointSize(12)

        font.setBold(False)

        font.setWeight(50)

*self*.btnImportData.setFont(font)

*self*.btnImportData.setObjectName("btnImportData")

*self*.btnBack = QtWidgets.QPushButton(

*parent*=*self*.centralwidget, *clicked*=lambda: *self*.go\_back()

        )

*self*.btnBack.setGeometry(QtCore.QRect(40, 10, 60, 20))

        font.setPointSize(12)

*self*.btnBack.setFont(font)

*self*.btnBack.setObjectName("btnBack")

        icon = QtGui.QIcon("icon\_back.png")

*self*.btnBack.setIcon(icon)

*self*.btnBack.clicked.connect(*self*.go\_back)

        # Button to process

*self*.btnProcess = QtWidgets.QPushButton(

*parent*=*self*.centralwidget, *clicked*=lambda: *self*.run\_app()

        )

*self*.btnProcess.setGeometry(QtCore.QRect(950, 40, 301, 23))

        font = QtGui.QFont()

        font.setFamily("Tahoma")

        font.setPointSize(12)

        font.setBold(False)

        font.setWeight(50)

*self*.btnProcess.setFont(font)

*self*.btnProcess.setObjectName("btnProcess")

        # Label and Line Edit for Alpha

*self*.label = QtWidgets.QLabel(*parent*=*self*.centralwidget)

*self*.label.setGeometry(QtCore.QRect(40, 90, 111, 21))

        font = QtGui.QFont()

        font.setFamily("Tahoma")

        font.setPointSize(12)

*self*.label.setFont(font)

*self*.label.setObjectName("label")

*self*.ledAlpha = QtWidgets.QLineEdit(*parent*=*self*.centralwidget)

*self*.ledAlpha.setGeometry(QtCore.QRect(160, 90, 131, 20))

*self*.ledAlpha.setObjectName("ledAlpha")

        # self.btnIncreaseAlpha = QtWidgets.QPushButton(parent=self.centralwidget, text="▲", clicked=self.increase\_alpha)

        # self.btnIncreaseAlpha.setGeometry(QtCore.QRect(310, 90, 20, 20))

        # self.btnDecreaseAlpha = QtWidgets.QPushButton(parent=self.centralwidget, text="▼", clicked=self.decrease\_alpha)

        # self.btnDecreaseAlpha.setGeometry(QtCore.QRect(340, 90, 20, 20))

        # Label and Line Edit for Selected Row

*self*.label\_2 = QtWidgets.QLabel(*parent*=*self*.centralwidget)

*self*.label\_2.setGeometry(QtCore.QRect(40, 140, 111, 21))

        font = QtGui.QFont()

        font.setFamily("Tahoma")

        font.setPointSize(12)

*self*.label\_2.setFont(font)

*self*.label\_2.setObjectName("label\_2")

*self*.ledSelectedRow = QtWidgets.QLineEdit(*parent*=*self*.centralwidget)

*self*.ledSelectedRow.setGeometry(QtCore.QRect(160, 140, 131, 20))

*self*.ledSelectedRow.setObjectName("ledSelectedRow")

        # Combo Box for selecting values

*self*.delta = QtWidgets.QComboBox(*parent*=*self*.centralwidget)

*self*.delta.setGeometry(QtCore.QRect(160, 190, 131, 20))

*self*.delta.setObjectName("delta")

*self*.delta.addItem("")

*self*.delta.addItem("")

*self*.delta.addItem("")

*self*.labelDelta = QtWidgets.QLabel(*parent*=*self*.centralwidget)

*self*.labelDelta.setGeometry(QtCore.QRect(40, 190, 111, 21))

        font = QtGui.QFont()

        font.setFamily("Tahoma")

        font.setPointSize(12)

*self*.labelDelta.setFont(font)

*self*.labelDelta.setObjectName("labelDelta")

*self*.labelDelta.setText("Delta")

        # Information Label

*self*.labInfor = QtWidgets.QLabel(*parent*=*self*.centralwidget)

*self*.labInfor.setGeometry(QtCore.QRect(950, 80, 350, 100))

        font = QtGui.QFont()

        font.setFamily("Tahoma")

        font.setPointSize(12)

*self*.labInfor.setFont(font)

*self*.labInfor.setText("")

*self*.labInfor.setObjectName("labInfor")

        # Table Widget

*self*.tabAns = QtWidgets.QTableWidget(*parent*=*self*.centralwidget)

*self*.tabAns.setGeometry(QtCore.QRect(30, 240, 1230, 341))

*self*.tabAns.setObjectName("tabAns")

*self*.tabAns.setColumnCount(0)

*self*.tabAns.setRowCount(0)

        # Setting up the main window

*mainWindow*.setCentralWidget(*self*.centralwidget)

*self*.menubar = QtWidgets.QMenuBar(*parent*=*mainWindow*)

*self*.menubar.setGeometry(QtCore.QRect(0, 0, 1032, 21))

*self*.menubar.setObjectName("menubar")

*mainWindow*.setMenuBar(*self*.menubar)

*self*.statusbar = QtWidgets.QStatusBar(*parent*=*mainWindow*)

*self*.statusbar.setObjectName("statusbar")

*mainWindow*.setStatusBar(*self*.statusbar)

*self*.retranslateUi(*mainWindow*)

        QtCore.QMetaObject.connectSlotsByName(*mainWindow*)

    def retranslateUi(*self*, *mainWindow*):

        \_translate = QtCore.QCoreApplication.translate

*mainWindow*.setWindowTitle(\_translate("mainWindow", "MainWindow"))

*self*.btnImportData.setText(\_translate("mainWindow", "Import Data"))

*self*.btnProcess.setText(\_translate("mainWindow", "Process"))

*self*.label.setText(\_translate("mainWindow", "Alpha Level"))

*self*.label\_2.setText(\_translate("mainWindow", "Rows Selected"))

*self*.delta.setItemText(0, \_translate("mainWindow", "0.01"))

*self*.delta.setItemText(1, \_translate("mainWindow", "0.1"))

*self*.delta.setItemText(2, \_translate("mainWindow", "0"))

if \_\_name\_\_ == "\_\_main\_\_":

    import sys

    app = QtWidgets.QApplication(sys.argv)

    mainWindow = QtWidgets.QMainWindow()

    ui = Ui\_ReducedData()

    ui.setupUi(mainWindow)

    mainWindow.show()

    sys.exit(app.exec())