```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 SpatialMetrics
                                 A QGIS plugin
 This plugins calculates Spatial Metrics for Spatial Indicators
                              -------------------
        begin                : 2013-12-13
        copyright            : (C) 2013 by Leon
        email                : mugwizal@gmail.som
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
"""
# Import the PyQt and QGIS libraries
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from qgis.core import *
from PyQt4.QtCore import QSettings, QVariant, Qt, SIGNAL
from PyQt4.QtGui import QDialog, QDialogButtonBox, QMessageBox
# Initialize Qt resources from file resources.py
from qgis.gui import *

import qgis.utils
import resources_rc
# Import the code for the dialog
from spatialmetricsdialog import SpatialMetricsDialog
from visualizationdialog import VisualizationDialog
import os.path
import matplotlib.pyplot as plt
import os,  glob
import string
import numpy as np
from osgeo import gdal
from pylab import *
import time
from progressbar import ProgressBar
import sys


class SpatialMetrics:

    def __init__(self, iface):
        # Save reference to the QGIS interface
        self.iface = iface
        # initialize plugin directory
        self.plugin_dir = os.path.dirname(__file__)
        # initialize locale
        locale = QSettings().value("/locale/userLocale")[0:2]
        localePath = os.path.join(self.plugin_dir, 'i18n', 'spatialmetrics_{}.qm'.format(locale))

        if os.path.exists(localePath):
            self.translator = QTranslator()
            self.translator.load(localePath)

            if qVersion() > '4.3.3':
                QCoreApplication.installTranslator(self.translator)

        # Create the dialog (after translation) and keep reference
        self.dlg = SpatialMetricsDialog()
        self.dlg2 = VisualizationDialog()
        self.canvas = QgsMapCanvas()

        #Initial  variables for opening files
```

```
        QObject.connect( self.dlg.ui.btnBaseDir, SIGNAL( "clicked()" ), self.selectFiles ) #link the button to the function of selecting the directory
        QObject.connect( self.dlg.ui.btnBaseDir_2, SIGNAL( "clicked()" ), self.selectDir ) #link the button to the function of selecting the directory
        QObject.connect( self.dlg.ui.pushButton, SIGNAL( "clicked()" ), self.LoadLayers) #link the button to the function of loading input files to l
        QObject.connect( self.dlg2.ui.pushButton_4, SIGNAL( "clicked()" ), self.reloadLayers) #link the button to the function of loading input file


        QObject.connect( self.dlg.ui.pushButton_3, SIGNAL( "clicked()" ), self.Preprocessing) #link the button to the function preprocess input fi

        self.loadSM()
        QObject.connect( self.dlg.ui.btnBaseDir_3, SIGNAL( "clicked()" ), self.selectDir ) #link the button to the function of selecting the directory
        QObject.connect( self.dlg.ui.pushButton_4, SIGNAL( "clicked()" ), self.SMcalc) #link the button to the function of calculating SM
        QObject.connect( self.dlg.ui.pushButton_5, SIGNAL( "clicked()" ), self.showClassesTable) #link the button to the function of calculating


        self.loadSI()   #load on combo box list of  spatial indicators
        self.loadES() #load on combo box list of Ecosystem services
        QObject.connect(self.dlg2.ui.comboBox, SIGNAL("currentIndexChanged (const QString&)"),   self.changeCombo) #link the ES list to SI

        QObject.connect( self.dlg2.ui.pushButton_3, SIGNAL( "clicked()" ), self.MapsAnim) #link the button to the function animation
        QObject.connect( self.dlg2.ui.btnBaseDir_3, SIGNAL( "clicked()" ), self.selectDir ) #link the button to the function of selecting the directo
        QObject.connect( self.dlg2.ui.pushButton, SIGNAL( "clicked()" ), self.graphs) #link the button to the function of selecting the viewtss
        QObject.connect( self.dlg2.ui.pushButton_2, SIGNAL( "clicked()" ), self.loadMaps) #link the button to the function of selecting the loadm

    def initGui(self):
        # Create action that will start plugin configuration
        self.action = QAction(
            QIcon(":/plugins/spatialmetrics/icon.png"),
            u"Spatial Metrics Calc", self.iface.mainWindow())
        self.action2 = QAction(
            QIcon(":/plugins/spatialmetrics/icon.png"),
            u"Spatial Indicators View", self.iface.mainWindow())
        # connect the action to the run method
        self.action.triggered.connect(self.run)
        self.action2.triggered.connect(self.view)
        # Add toolbar button and menu item
        self.iface.addToolBarIcon(self.action)
        self.iface.addPluginToMenu(u"&SM", self.action)
        self.iface.addToolBarIcon(self.action2)
        self.iface.addPluginToMenu(u"&Results", self.action2)

    def unload(self):
        # Remove the plugin menu item and icon
        self.iface.removePluginMenu(u"&SM", self.action)
        self.iface.removeToolBarIcon(self.action)
        self.iface.addPluginToMenu(u"&Results", self.action2)
        self.iface.removeToolBarIcon(self.action2)

    def run(self):
        self.dlg.show() # show the dialog
        self.dlg.ui.tableWidget.hide()
#       if self.dlg.ui.checkBox.isChecked():
#       else: self.dlg.ui.tableWidget.hide()

        result = self.dlg.exec_() # Run the dialog event loop


    def view(self):
        self.dlg2.show()# show the dialog

  # Selecting the files in directory
    def selectFiles( self ):
        settings = QSettings()
        path = str(QFileDialog.getOpenFileName(self.iface.mainWindow(), "Select Directory"))
        if path: self.dlg.ui.txtBaseDir.setText( path ) #write the name of the directory path

  # Selecting the directory containg files
    def selectDir( self ):
        settings = QSettings()
        path = QFileDialog.getExistingDirectory( self.iface.mainWindow(), "Select a directory")
        if path: self.dlg.ui.txtBaseDir2.setText( path )
        if path: self.dlg.ui.txtBaseDir2_5.setText( path )
```

```
        if path: self.dlg2.ui.txtBaseDir2_5.setText( path )
        if path: self.dlg.ui.txtBaseDir.setText( path ) #write the name of the directory path $$$$$

    def LoadLayers(self):
        self.dataDir = str(self.dlg.ui.txtBaseDir.text())
        a=string.maketrans(',')
        ch=a.translate(a, string.digits)
        year = str(self.dataDir.translate(a, ch))
        self.dlg.ui.txtBaseDir_2.setText( year ) #write the period for the LU map to load to layer

##        self.iface.addRasterLayer(self.dataDir, os.path.basename(self.dataDir))
##        fileInfo = QFileInfo(self.dataDir)
##        baseName = fileInfo.baseName()
        rlayer = QgsRasterLayer(self.dataDir, baseName)
###         rlayer.setDrawingStyle( QgsRasterLayer.SingleBandPseudoColor )
        QgsMapLayerRegistry.instance().addMapLayer(rlayer)
###         layer = self.iface.mapCanvas().currentLayer()
###         layer.setDrawingStyle( QgsRasterLayer.SINGLE_BAND_PSEUDO_COLOR)


    def reloadLayers(self):
        for layer in self.iface.legendInterface().layers():
            QgsMapLayerRegistry.instance().removeMapLayer( layer.id()  )
        self.dlg2.ui.widget.canvas.ax.clear()




##########################################################################################
    def Preprocessing(self):
        for i in range(0,100+1,50):
            self.dlg.ui.progressBar.setProperty("value",i)
            self.dataDir = str(self.dlg.ui.txtBaseDir2.text())
            os.chdir(self.dataDir )
            reclas_map = 'classified'
            name = str(self.dlg.ui.txtBaseDir2_2.text())
            from formatting import Myformat as prepro
            d = prepro (reclas_map, name)
            self.run()
            self.dlg.ui.progressBar.setProperty("value",i)

    def loadSM(self):
        self.dlg.ui.comboBox.addItem("CA")
        self.dlg.ui.comboBox.addItem("MPS")

    def SMcalc(self):
        self.dataDir = str(self.dlg.ui.txtBaseDir2_5.text())
        os.chdir(self.dataDir )
        classe = str(self.dlg.ui.txtBaseDir2_3.text())
        Naming = str(self.dlg.ui.txtBaseDir2_4.text())
        Metric = self.dlg.ui.comboBox.currentText()

        import csv
        self.dataDir = str(self.dlg.ui.txtBaseDir2_5.text())
        os.chdir(self.dataDir )
        LUclasses = []
        reader = csv.reader(open('legend2.txt'),  delimiter='\t')
        for row in reader:
            LUclasses.append(row)
        rowCount = len(LUclasses)
        colCount = max([len(p) for p in LUclasses])
        self.dlg.ui.tableWidget.setRowCount(rowCount)
        self.dlg.ui.tableWidget.setColumnCount(1)
        headers = []
        headers.append("Value&Class")
        self.dlg.ui.tableWidget.setHorizontalHeaderLabels(headers)
        for row, person in enumerate(LUclasses):
            for column, value in enumerate(person):
                newItem = QTableWidgetItem(value)
                self.dlg.ui.tableWidget.setItem(row, column, newItem)
##        import metrics
```

```
        for i in range(0,100+1,50):
            self.dlg.ui.progressBar_2.setProperty("value",i)
            cmd = "python metrics.py "+classe+"  "+Naming+"  "+Metric
            os.system(cmd)
            self.dlg.ui.progressBar_2.setProperty("value",i)
            self.dlg.ui.tableWidget.hide()

    def showClassesTable (self):
        import csv
        self.dlg.ui.tableWidget.show()
        self.dataDir = str(self.dlg.ui.txtBaseDir2_5.text())
        os.chdir(self.dataDir )
        LUclasses = []
        reader = csv.reader(open('legend2.txt'),  delimiter='\t')
        for row in reader:
            LUclasses.append(row)
        rowCount = len(LUclasses)
        colCount = max([len(p) for p in LUclasses])
        self.dlg.ui.tableWidget.setRowCount(rowCount)
        self.dlg.ui.tableWidget.setColumnCount(1)
        headers = []
        headers.append("Value&Class")
        self.dlg.ui.tableWidget.setHorizontalHeaderLabels(headers)
        for row, person in enumerate(LUclasses):
            for column, value in enumerate(person):
                newItem = QTableWidgetItem(value)
                self.dlg.ui.tableWidget.setItem(row, column, newItem)
##########################################################################
    def loadES(self):
        self.dlg2.ui.comboBox.addItems(["Crop production", "Carbon storage", "Timber/ fuel wood", "Fodder production",
            "Wild capture and collection", "Erosion control"])

    def loadSI(self):
        self.di = {"Crop production":["area of agriculture", "cluster size of agriculture area"],
                   "Carbon storage":["area of forest"],
                   "Timber/ fuel wood":["area of forest", "cluster size of forest"],
                   "Fodder production":["area of grassland", "cluster size of grassland"],
                   "Wild capture and collection":["cluster size of forest"],
                   "Erosion control":["cluster size of grassland", "cluster size of forest"]}

        self.dlg2.ui.comboBox_2.addItems (self.di["Crop production"])

    def changeCombo(self, ind):
        self.dlg2.ui.comboBox_2.clear()
        self.dlg2.ui.comboBox_2.addItems(self.di[ind])

##########################################################################################################
    def MapsAnim(self):# wtih matplotlib
        import numpy
        numpy.seterr(divide='ignore', invalid='ignore', over='ignore')
        ax = plt.subplot(111)

        self.dataDir = str(self.dlg2.ui.txtBaseDir2_5.text())
        os.chdir(self.dataDir )
##        import openmaps
        fig = plt.figure(1)
        plt.ioff()
        ax = fig.add_subplot( 111 )
        ax.set_title("My Title")
        if self.dlg2.ui.comboBox_2.currentText() == "area of agriculture": filename = '*SurfCrop*'
        if self.dlg2.ui.comboBox_2.currentText() == "cluster size of agriculture area": filename = '*MPSCrop*'
        if self.dlg2.ui.comboBox_2.currentText() == "area of forest": filename = '*SurfFore*'
        if self.dlg2.ui.comboBox_2.currentText() == "cluster size of forest": filename = '*MPSFore*'
        if self.dlg2.ui.comboBox_2.currentText() == "area of grassland": filename = '*SurfGras*'
        if self.dlg2.ui.comboBox_2.currentText() == "cluster size of grassland": filename = '*MPSGras*'
##        filename = '*LU*'
##        filename = '*LU*.tiff'
        file_list =  glob.glob(filename)
        for index in file_list:
            if index.endswith(".xml"):
                file_list.remove(index)
```

```
            if index.endswith(".tss"):
                file_list.remove(index)
        file_list.sort()
        s = QSettings()
        self.dlg2.hide()
        oldValidation = s.value( "/Projections/defaultBehaviour", "useGlobal" )
        s.setValue( "/Projections/defaultBehaviour", "useGlobal" )
        self.iface.addRasterLayer(file_list[0], os.path.basename(str(file_list[0]))).setCrs( QgsCoordinateReferenceSystem(4326, QgsCoordinate
        self.iface.messageBar().pushMessage(str(file_list[0]), level=QgsMessageBar.INFO, duration=1)
        for index, file in enumerate(file_list[1:]):
            self.iface.addRasterLayer(file, os.path.basename(str(file))).setCrs( QgsCoordinateReferenceSystem(4326, QgsCoordinateReference
            self.canvas.setFocus()
            self.iface.messageBar().pushMessage(str(file), level=QgsMessageBar.INFO,  duration=3)
#           self.iface.legendInterface().addGroup( 'abc')
            time.sleep(1)
        s.setValue( "/Projections/defaultBehaviour", oldValidation )
        self.dlg2.show()


##########################################################################################
    def graphs(self):# wtih matplotlib
        ax = plt.subplot(111)
        self.dataDir = str(self.dlg2.ui.txtBaseDir2_5.text())
        os.chdir(self.dataDir )
        stripped = []
#       filename = "areaBare.tss"
        if self.dlg2.ui.comboBox_2.currentText() == "area of agriculture":
            filename = "CACrop.tss"
            Title = "area of agriculture"
        if self.dlg2.ui.comboBox_2.currentText() == "cluster size of agriculture area":
            filename = "MPSCrop.tss"
            Title = "cluster size of agriculture area"
        if self.dlg2.ui.comboBox_2.currentText() == "area of forest":
            filename = "CAFore.tss"
            Title = "area of forest"
        if self.dlg2.ui.comboBox_2.currentText() == "cluster size of forest":
            filename = "MPSFore.tss"
            Title = "cluster size of forest"
        if self.dlg2.ui.comboBox_2.currentText() == "area of grassland":
            filename = "CAGras.tss"
            Title = "area of grassland"
        if self.dlg2.ui.comboBox_2.currentText() == "cluster size of grassland":
            filename = "MPSGras.tss"
            Title = "cluster size of grassland"
        stripper =  open(filename, 'r')
        st_lines = stripper.readlines()[4:]
        stripper.close()
        for lines in st_lines:
            stripped_line = " ".join(lines.split())
            stripped.append(stripped_line)
        data = "\n".join(stripped)
        data = data.split('\n')

        values = []
        dates = []
        years = 2004
        yl = []
        for row in data:
            x, y = row.split()
            values.append(float(y))
            year = (int(x.translate(string.maketrans("\n\t\r", "    ")).strip()))
            dates.append(year)
            years = years +1
            yl.append(years)
        xlabels = yl
        self.dlg2.ui.widget.canvas.ax.clear()
        self.dlg2.ui.widget.canvas.ax.set_title(Title)
        self.dlg2.ui.widget.canvas.ax.set_xlabel ('Years')
        self.dlg2.ui.widget.canvas.ax.set_ylabel ('Area in ha')
        self.dlg2.ui.widget.canvas.ax.bar(dates, values, align='center')
        self.dlg2.ui.widget.canvas.ax.set_xticks(dates)  # put the tick markers under your bars
```

```
        self.dlg2.ui.widget.canvas.ax.set_xticklabels(xlabels)    # set the labels to be your formatted years
        self.dlg2.ui.widget.canvas.draw()


#############################################################################################
    def loadMaps(self): # wtih matplotlib
        import numpy
        numpy.seterr(divide='ignore', invalid='ignore', over='ignore')
        self.dataDir = str(self.dlg2.ui.txtBaseDir2_5.text())
        os.chdir(self.dataDir )
        if self.dlg2.ui.comboBox_2.currentText() == "area of agriculture": filename = '*SurfCrop*'
        if self.dlg2.ui.comboBox_2.currentText() == "cluster size of agriculture area": filename = '*MPSCrop*'
        if self.dlg2.ui.comboBox_2.currentText() == "area of forest": filename = '*SurfFore*'
        if self.dlg2.ui.comboBox_2.currentText() == "cluster size of forest": filename = '*MPSFore*'
        if self.dlg2.ui.comboBox_2.currentText() == "area of grassland": filename = '*SurfGras*'
        if self.dlg2.ui.comboBox_2.currentText() == "cluster size of grassland": filename = '*MPSGras*'
        ##        filename = '*LU*'
        file_list =  glob.glob(filename)
        for index in file_list:
            if index.endswith(".xml"):
                file_list.remove(index)
            if index.endswith(".tss"):
                file_list.remove(index)
        file_list.sort()
        ax = plt.subplot(111)
        axnext = plt.axes([0.81, 0.05, 0.1, 0.075])
        bnext = Button(axnext, 'Next')
        def event_fct(event):
            """ evaluate key-press events """
            global counter
            if event is None:
                counter = 0
            else:
                if event.key in ['r', 'enter']:
                    if event.key in 'r':
                        print "reject image"
                    elif event.key in 'enter':
                        print "use image"
                    # go to next image
                    counter += 1
                else:
                    print "Key >%s< is not defined" % (event.key)
                    return
            if counter < len(file_list):
                lc_data = gdal.Open(file_list[counter])
                im = lc_data.ReadAsArray()
                img = ax.imshow(im, interpolation='nearest', vmin=0)
            #    ax.set_title("[u]se or [r]eject image %i by key press" % (counter))
                ax.set_title(str( file_list[counter] ) +"\n Go to next map by pressing [enter] key", fontsize=12)
                plt.draw()
            else:
                event is None
        event_fct(None) # initial setup
        plt.connect('key_press_event', event_fct)
        callback = ax.imshow(gdal.Open(file_list[counter]).ReadAsArray(), interpolation='nearest', vmin=0)
        plt.show()
```