# File Name:requirements.txt

GDAL==3.4.3

hyperopt==0.2.7

lightgbm==4.3.0

matplotlib==3.8.0

numpy==1.26.4

pandas==2.1.4

PyQt5==5.15.10

PyQt5\_sip==12.13.0

SciencePlots==2.1.1

scikit\_learn==1.2.2

seaborn==0.12.2

xgboost==2.0.3

# -\*- coding: utf-8 -\*-

# File Name：main.py

# description: 主程序入口

import sys

import os

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

try:

from PyQt5 import QtCore, QtGui, QtWidgets

from \_\_UI\_\_ import ClassificationUI

except ImportError:

# 读取requirements.txt

with open("requirements.txt", "r") as f:

requirements = f.readlines()

# 安装依赖

for requirement in requirements:

os.system(

f"pip install {requirement} -i http://mirrors.aliyun.com/pypi/simple --trusted-host mirrors.aliyun.com"

)

# 重新导入

from PyQt5 import QtCore, QtGui, QtWidgets

from \_\_UI\_\_ import ClassificationUI

try:

app = QtWidgets.QApplication(sys.argv)

MainWindow = QtWidgets.QMainWindow()

ui = ClassificationUI()

ui.setupUi(MainWindow)

# 设置图标

MainWindow.setWindowIcon(QtGui.QIcon("icon.png"))

MainWindow.show()

sys.exit(app.exec\_())

except Exception as e:

print(e)

os.system("pause")

# -\*- coding: utf-8 -\*-

# File name: \_\_UI\_\_.py

from PyQt5 import QtCore, QtGui, QtWidgets

from classes import \*

class ClassificationUI(object):

def ThreadLog(self, msg):

self.log\_text\_browser.append(msg)

def SaveLogButton\_clicked(self):

# 获取当前text browser的文本

# 打开一个保存文件选择的框

try:

# 获取当前text browser的文本

text = self.log\_text\_browser.toPlainText()

# 新建文件的输出路径文件名

out\_path, \_ = QtWidgets.QFileDialog.getSaveFileName(

None, "Save File", "", "Text Files(\*.txt)"

)

open(out\_path, "w").write(text)

QtWidgets.QMessageBox.information(None, "成功", "成功保存日志文件!", QtWidgets.QMessageBox.Ok)

except Exception as e:

QtWidgets.QMessageBox.critical(None, "Error", str(e), QtWidgets.QMessageBox.Ok)

def get\_page\_space(self):

if self.tabWidgetPages.currentIndex() == 0:

space = {

"n\_estimators": hp.choice(

"n\_estimators",

range(

int(self.RF\_n\_estimators\_min.text()),

int(self.RF\_n\_estimators\_max.text()),

int(self.RF\_n\_estimators\_interval.text()),

),

),

"max\_depth": hp.choice(

"max\_depth",

range(

int(self.RF\_max\_depth\_min.text()),

int(self.RF\_max\_depth\_max.text()),

int(self.RF\_max\_depth\_interval.text()),

),

),

"min\_samples\_leaf": hp.choice(

"min\_samples\_leaf",

range(

int(self.RF\_min\_samples\_leaf\_min.text()),

int(self.RF\_min\_samples\_leaf\_max.text()),

int(self.RF\_min\_samples\_leaf\_interval.text()),

),

),

"min\_samples\_split": hp.choice(

"min\_samples\_split",

range(

int(self.RF\_min\_samples\_split\_min.text()),

int(self.RF\_min\_samples\_split\_max.text()),

int(self.RF\_min\_samples\_split\_interval.text()),

),

),

}

max\_features = []

if self.sqrt\_cb.isChecked():

max\_features.append("sqrt")

if self.log2\_cb.isChecked():

max\_features.append("log2")

if self.None\_cb.isChecked():

max\_features.append(None)

space["max\_features"] = hp.choice("max\_features", max\_features)

bootstrap = []

if self.True\_cb.isChecked():

bootstrap.append(True)

if self.False\_cb.isChecked():

bootstrap.append(False)

space["bootstrap"] = hp.choice("bootstrap", bootstrap)

return space

elif self.tabWidgetPages.currentIndex() == 1:

space = {

"n\_estimators": hp.choice(

"n\_estimators",

range(

int(self.LGB\_n\_estimators\_min.text()),

int(self.LGB\_n\_estimators\_max.text()),

int(self.LGB\_n\_estimators\_interval.text()),

),

),

"max\_depth": hp.choice(

"max\_depth",

range(

int(self.LGB\_max\_depth\_min.text()),

int(self.LGB\_max\_depth\_max.text()),

int(self.LGB\_max\_depth\_interval.text()),

),

),

"learning\_rate": hp.choice(

"learning\_rate",

float\_range(

float(self.LGB\_learning\_rate\_min.text()),

float(self.LGB\_learning\_rate\_Max.text()),

float(self.LGB\_learning\_rate\_Interval.text()),

),

),

"min\_child\_weight": hp.choice(

"min\_child\_weight",

range(

int(self.LGB\_min\_child\_weight\_min.text()),

int(self.LGB\_min\_child\_weight\_max.text()),

int(self.LGB\_min\_child\_weight\_interval.text()),

),

),

"subsample": hp.choice(

"subsample",

float\_range(

float(self.LGB\_subsample\_min.text()),

float(self.LGB\_subsample\_max.text()),

float(self.LGB\_subsample\_interval.text()),

),

),

"colsample\_bytree": hp.choice(

"colsample\_bytree",

float\_range(

float(self.LGB\_colsample\_bytree\_min.text()),

float(self.LGB\_colsample\_bytree\_max.text()),

float(self.LGB\_colsample\_bytree\_interval.text()),

),

),

"reg\_alpha": hp.choice(

"reg\_alpha",

range(

int(self.LGB\_reg\_alpha\_min.text()),

int(self.LGB\_reg\_alpha\_max.text()),

int(self.LGB\_reg\_alpha\_interval.text()),

),

),

"reg\_lambda": hp.choice(

"reg\_lambda",

range(

int(self.LGB\_reg\_lambda\_min.text()),

int(self.LGB\_reg\_lambda\_max.text()),

int(self.LGB\_reg\_lambda\_interval.text()),

),

),

"objective": "multiclass",

# "metric": "multi\_logloss",

# "boosting\_type": "gbdt",

}

# print(space['learning\_rate'])

return space

elif self.tabWidgetPages.currentIndex() == 2:

space = {

"n\_estimators": hp.choice(

"n\_estimators",

range(

int(self.XGB\_n\_estimators\_min.text()),

int(self.XGB\_n\_estimators\_max.text()),

int(self.XGB\_n\_estimators\_interval.text()),

),

),

"max\_depth": hp.choice(

"max\_depth",

range(

int( self.XGB\_max\_depth\_min.text()),

int(self.XGB\_max\_depth\_max.text()),

int(self.XGB\_max\_depth\_interval.text()),

),

),

"learning\_rate": hp.choice(

"learning\_rate",

float\_range(

float(self.XGB\_learning\_rate\_min.text()),

float(self.XGB\_learning\_rate\_Max.text()),

float(self.XGB\_learning\_rate\_Interval.text()),

),

),

"min\_child\_weight": hp.choice(

"min\_child\_weight",

range(

int(self.XGB\_min\_child\_weight\_min.text()),

int(self.XGB\_min\_child\_weight\_max.text()),

int(self.XGB\_min\_child\_weight\_interval.text()),

),

),

"subsample": hp.choice(

"subsample",

float\_range(

float(self.XGB\_subsample\_min.text()),

float(self.XGB\_subsample\_max.text()),

float(self.XGB\_subsample\_interval.text()),

),

),

"colsample\_bytree": hp.choice(

"colsample\_bytree",

float\_range(

float(self.XGB\_colsample\_bytree\_min.text()),

float(self.XGB\_colsample\_bytree\_max.text()),

float(self.XGB\_colsample\_bytree\_interval.text()),

),

),

"gamma": hp.choice(

"gamma",

range(

int(self.XGB\_gamma\_min.text()),

int(self.XGB\_gamma\_max.text()),

int(self.XGB\_gamma\_interval.text()),

),

),

"reg\_alpha": hp.choice(

"reg\_alpha",

range(

int(self.XGB\_reg\_alpha\_min.text()),

int(self.XGB\_reg\_alpha\_max.text()),

int(self.XGB\_reg\_alpha\_interval.text()),

),

),

"reg\_lambda": hp.choice(

"reg\_lambda",

range(

int(self.XGB\_reg\_lambda\_min.text()),

int(self.XGB\_reg\_lambda\_max.text()),

int(self.XGB\_reg\_lambda\_interval.text()),

),

),

"objective": "multiclass",

# "metric": "multi\_logloss",

# "boosting\_type": "gbdt",

}

return space

# 随机森林数据获取部分的按钮点击链接

def RF\_select\_image\_pushbutton\_clicked(self):

selectImgPath(self.RF\_img\_path, self.log\_text\_browser)

def RF\_select\_label\_pushbutton\_clicked(self):

selectLabelPath(self.RF\_label\_path, self.log\_text\_browser)

def RF\_select\_numclass\_path\_pushButton\_clicked(self):

selectNumClassPath(self.RF\_class\_num\_path, self.log\_text\_browser)

def RF\_select\_sample\_path\_pushButton\_clicked(self):

selectOutSamplePath(self.RF\_output\_samples\_path, self.log\_text\_browser)

def select\_RF\_train\_Save\_model\_path\_pushButton\_clicked(self):

selectSaveModelPath(self.RF\_train\_model\_save\_path, self.log\_text\_browser)

def select\_RF\_model\_path\_pushButton\_clicked(self):

selectModelPath(self.RF\_model\_path, self.log\_text\_browser)

def select\_RF\_class\_num\_pushButton\_clicked(self):

selectNumClassPath(self.RF\_class\_num\_path\_2, self.log\_text\_browser)

def select\_RF\_predict\_result\_output\_path\_pushButton\_clicked(self):

selectSaveImgPath(self.RF\_output\_predict\_result\_path, self.log\_text\_browser)

def select\_RF\_img\_predict\_path\_pushButton\_clicked(self):

selectImgPath(self.RF\_img\_predict\_path, self.log\_text\_browser)

def RF\_get\_sample\_pushButton\_clicked(self):

# 判断是否有路径

if (

not self.RF\_img\_path.text()

or not self.RF\_label\_path.text()

or not self.RF\_output\_samples\_path.text()

or not self.RF\_class\_num\_path.text()

):

self.log\_text\_browser.append("请检查路径!!!!")

return

# 开启线程

self.getSamplesFThread = GetSamplesFThread(

self.RF\_img\_path.text(),

self.RF\_label\_path.text(),

self.RF\_output\_samples\_path.text(),

self.RF\_class\_num\_path.text(),

)

self.getSamplesFThread.error.connect(self.ThreadLog)

self.getSamplesFThread.msg.connect(self.ThreadLog)

self.getSamplesFThread.start()

def RF\_train\_pushButton\_clicked(self):

# 判断是否有路径

if (

not self.RF\_output\_samples\_path.text()

or not self.RF\_train\_model\_save\_path.text()

or not self.RF\_val\_datasets\_size.text()

):

self.log\_text\_browser.append("请检查路径!!!!")

return

# 判断data\_size是否为浮点数字

try:

float(self.RF\_val\_datasets\_size.text())

except:

self.log\_text\_browser.append("请输入正确的数据集大小!!!!")

return

# 开启线程

self.trainThread = Train(

self.RF\_output\_samples\_path.text(),

self.RF\_train\_model\_save\_path.text(),

self.get\_page\_space(),

float(self.RF\_val\_datasets\_size.text()),

"RandomForest",

)

self.trainThread.error.connect(self.ThreadLog)

self.trainThread.msg.connect(self.ThreadLog)

self.trainThread.start()

def RF\_predict\_pushButton\_clicked(self):

# 判断是否有路径

if (

not self.RF\_model\_path.text()

or not self.RF\_img\_predict\_path.text()

or not self.RF\_output\_predict\_result\_path.text()

or not self.RF\_class\_num\_path\_2.text()

):

self.log\_text\_browser.append("请检查路径!!!!")

return

# 开启线程

self.predictThread = Predict(

self.RF\_model\_path.text(),

self.RF\_img\_predict\_path.text(),

self.RF\_output\_predict\_result\_path.text(),

self.RF\_class\_num\_path\_2.text(),

"RandomForest",

)

self.predictThread.error.connect(self.ThreadLog)

self.predictThread.msg.connect(self.ThreadLog)

self.predictThread.start()

# XGB数据获取部分的按钮点击链接

def XGB\_select\_image\_pushbutton\_clicked(self):

selectImgPath(self.XGB\_img\_path, self.log\_text\_browser)

def XGB\_select\_label\_pushbutton\_clicked(self):

selectLabelPath(self.XGB\_label\_path, self.log\_text\_browser)

def XGB\_select\_numclass\_path\_pushbutton\_clicked(self):

selectNumClassPath(self.XGB\_class\_num\_path, self.log\_text\_browser)

def XGB\_select\_sample\_path\_pushbutton\_clicked(self):

selectOutSamplePath(self.XGB\_output\_samples\_path, self.log\_text\_browser)

def select\_XGB\_train\_Save\_model\_path\_pushButton\_clicked(self):

selectSaveModelPath(self.XGB\_train\_model\_save\_path, self.log\_text\_browser)

def select\_XGB\_model\_path\_pushButton\_clicked(self):

selectModelPath(self.XGB\_model\_path, self.log\_text\_browser)

def select\_XGB\_class\_num\_button\_2\_clicked(self):

selectNumClassPath(self.XGB\_class\_num\_path\_2, self.log\_text\_browser)

def XGB\_select\_predict\_result\_output\_path\_pushButton\_clicked(self):

selectSaveImgPath(self.XGB\_output\_predict\_result\_path, self.log\_text\_browser)

def select\_XGB\_img\_predict\_path\_pushButton\_clicked(self):

selectImgPath(self.XGB\_img\_predict\_path, self.log\_text\_browser)

def XGB\_get\_sample\_pushButton\_clicked(self):

# 判断是否有路径

if (

not self.XGB\_img\_path.text()

or not self.XGB\_label\_path.text()

or not self.XGB\_output\_samples\_path.text()

or not self.XGB\_class\_num\_path.text()

):

self.log\_text\_browser.append("请检查路径!!!!")

return

# 开启线程

self.getSamplesFThread = GetSamplesFThread(

self.XGB\_img\_path.text(),

self.XGB\_label\_path.text(),

self.XGB\_output\_samples\_path.text(),

self.XGB\_class\_num\_path.text(),

)

self.getSamplesFThread.error.connect(self.ThreadLog)

self.getSamplesFThread.msg.connect(self.ThreadLog)

self.getSamplesFThread.start()

def XGB\_train\_pushButton\_clicked(self):

# 判断是否有路径

if (

not self.XGB\_output\_samples\_path.text()

or not self.XGB\_train\_model\_save\_path.text()

or not self.XGB\_val\_datasets\_size.text()

):

self.log\_text\_browser.append("请检查路径!!!!")

return

# 判断data\_size是否为浮点数字

try:

float(self.XGB\_val\_datasets\_size.text())

except:

self.log\_text\_browser.append("请输入正确的数据集大小!!!!")

return

# 开启线程

self.trainThread = Train(

self.XGB\_output\_samples\_path.text(),

self.XGB\_train\_model\_save\_path.text(),

self.get\_page\_space(),

float(self.XGB\_val\_datasets\_size.text()),

"XGBoost",

)

self.trainThread.error.connect(self.ThreadLog)

self.trainThread.msg.connect(self.ThreadLog)

self.trainThread.start()

def XGB\_predict\_pushButton\_clicked(self):

# 判断是否有路径

if (

not self.XGB\_model\_path.text()

or not self.XGB\_img\_predict\_path.text()

or not self.XGB\_output\_predict\_result\_path.text()

or not self.XGB\_class\_num\_path\_2.text()

):

self.log\_text\_browser.append("请检查路径!!!!")

return

# 开启线程

self.predictThread = Predict(

self.XGB\_model\_path.text(),

self.XGB\_img\_predict\_path.text(),

self.XGB\_output\_predict\_result\_path.text(),

self.XGB\_class\_num\_path\_2.text(),

"XGBoost",

)

self.predictThread.error.connect(self.ThreadLog)

self.predictThread.msg.connect(self.ThreadLog)

self.predictThread.start()

# LGB数据获取部分的按钮点击链接

def LGB\_select\_image\_pushbutton\_clicked(self):

selectImgPath(self.LGB\_img\_path, self.log\_text\_browser)

def LGB\_select\_label\_pushbutton\_clicked(self):

selectLabelPath(self.LGB\_label\_path, self.log\_text\_browser)

def LGB\_select\_numclass\_path\_pushbutton\_clicked(self):

selectNumClassPath(self.LGB\_class\_num\_path, self.log\_text\_browser)

def LGB\_select\_sample\_path\_pushbutton\_clicked(self):

selectOutSamplePath(self.LGB\_output\_samples\_path, self.log\_text\_browser)

def select\_LGB\_train\_Save\_model\_path\_pushButton\_clicked(self):

selectSaveModelPath(self.LGB\_train\_model\_save\_path, self.log\_text\_browser)

def select\_LGB\_model\_path\_pushbutton\_clicked(self):

selectModelPath(self.LGB\_model\_path, self.log\_text\_browser)

def select\_LGB\_class\_num\_button\_2\_clicked(self):

selectNumClassPath(self.LGB\_class\_num\_path\_2, self.log\_text\_browser)

def LGB\_select\_predict\_result\_output\_path\_pushbutton\_clicked(self):

selectSaveImgPath(self.LGB\_output\_predict\_result\_path, self.log\_text\_browser)

def select\_LGB\_img\_predict\_path\_button\_clicked(self):

selectImgPath(self.LGB\_img\_predict\_path, self.log\_text\_browser)

def LGB\_get\_sample\_pushButton\_clicked(self):

# 判断是否有路径

if (

not self.LGB\_img\_path.text()

or not self.LGB\_label\_path.text()

or not self.LGB\_output\_samples\_path.text()

or not self.LGB\_class\_num\_path.text()

):

self.log\_text\_browser.append("请检查路径!!!!")

return

# 开启线程

self.getSamplesFThread = GetSamplesFThread(

self.LGB\_img\_path.text(),

self.LGB\_label\_path.text(),

self.LGB\_output\_samples\_path.text(),

self.LGB\_class\_num\_path.text(),

)

self.getSamplesFThread.error.connect(self.ThreadLog)

self.getSamplesFThread.msg.connect(self.ThreadLog)

self.getSamplesFThread.start()

def LGB\_train\_pushButton\_clicked(self):

# 判断是否有路径

if (

not self.LGB\_output\_samples\_path.text()

or not self.LGB\_train\_model\_save\_path.text()

or not self.LGB\_val\_datasets\_size.text()

):

self.log\_text\_browser.append("请检查路径!!!!")

return

# 判断data\_size是否为浮点数字

try:

float(self.LGB\_val\_datasets\_size.text())

except:

self.log\_text\_browser.append("请输入正确的数据集大小!!!!")

return

# 开启线程

self.trainThread = Train(

self.LGB\_output\_samples\_path.text(),

self.LGB\_train\_model\_save\_path.text(),

self.get\_page\_space(),

float(self.LGB\_val\_datasets\_size.text()),

"LightGBM",

)

self.trainThread.error.connect(self.ThreadLog)

self.trainThread.msg.connect(self.ThreadLog)

self.trainThread.start()

def LGB\_predict\_pushButton\_clicked(self):

# 判断是否有路径

if (

not self.LGB\_model\_path.text()

or not self.LGB\_img\_predict\_path.text()

or not self.LGB\_output\_predict\_result\_path.text()

or not self.LGB\_class\_num\_path\_2.text()

):

self.log\_text\_browser.append("请检查路径!!!!")

return

# 开启线程

self.predictThread = Predict(

self.LGB\_model\_path.text(),

self.LGB\_img\_predict\_path.text(),

self.LGB\_output\_predict\_result\_path.text(),

self.LGB\_class\_num\_path\_2.text(),

"LightGBM",

)

self.predictThread.error.connect(self.ThreadLog)

self.predictThread.msg.connect(self.ThreadLog)

self.predictThread.start()

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(1300, 910)

# 固定窗口大小

MainWindow.setFixedSize(1300, 910)

self.centralWidget = QtWidgets.QWidget(MainWindow)

self.centralWidget.setLayoutDirection(QtCore.Qt.LeftToRight)

self.centralWidget.setObjectName("centralWidget")

self.tabWidgetPages = QtWidgets.QTabWidget(self.centralWidget)

self.tabWidgetPages.setGeometry(QtCore.QRect(20, 10, 480, 890))

self.tabWidgetPages.setObjectName("tabWidgetPages")

# RF\_tab\_page

self.RF\_tab\_page = QtWidgets.QWidget()

self.RF\_tab\_page.setObjectName("RF\_tab\_page")

self.RF\_tab\_page\_init()

self.tabWidgetPages.addTab(self.RF\_tab\_page, "")

# LGBM\_tab\_page

self.LGB\_tab\_page = QtWidgets.QWidget()

self.LGB\_tab\_page.setObjectName("LGB\_tab\_page")

self.LightGBM\_tab\_page\_init()

self.tabWidgetPages.addTab(self.LGB\_tab\_page, "")

#### ======================================== ========================================

# XGB\_tab\_page

self.XGB\_tab\_page = QtWidgets.QWidget()

self.XGB\_tab\_page.setObjectName("XGB\_tab\_page")

self.XGB\_tab\_page\_init()

self.tabWidgetPages.addTab(self.XGB\_tab\_page, "")

# 设置日志界面

self.log\_group = QtWidgets.QGroupBox(self.centralWidget)

self.log\_group.setGeometry(QtCore.QRect(530, 10, 750, 890))

self.log\_group.setObjectName("log\_group")

self.log\_text\_browser = QtWidgets.QTextBrowser(self.log\_group)

self.log\_text\_browser.setGeometry(QtCore.QRect(10, 20, 730, 830))

self.log\_text\_browser.setObjectName("log\_text\_browser")

self.SaveLogButton = QtWidgets.QPushButton(self.log\_group)

self.SaveLogButton.setGeometry(QtCore.QRect(330, 855, 90, 30))

self.SaveLogButton.setObjectName("SaveLogButton")

self.SaveLogButton.clicked.connect(self.SaveLogButton\_clicked)

MainWindow.setCentralWidget(self.centralWidget)

# 绑定tabWidgetPagesChanged

self.tabWidgetPages.setCurrentIndex(0)

self.tabWidgetPages.currentChanged.connect(self.tabWidgetPagesChanged)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def RF\_tab\_page\_init(self):

# ======================================== RF\_get\_samples\_group ========================================

self.RF\_get\_samples\_group = QtWidgets.QGroupBox(self.RF\_tab\_page)

self.RF\_get\_samples\_group.setGeometry(QtCore.QRect(10, 10, 460, 225))

self.RF\_get\_samples\_group.setObjectName("RF\_get\_samples\_group")

self.RF\_img\_path\_label = QtWidgets.QLabel(self.RF\_get\_samples\_group)

self.RF\_img\_path\_label.setGeometry(QtCore.QRect(10, 20, 80, 30))

self.RF\_img\_path\_label.setObjectName("RF\_img\_path\_label")

self.RF\_img\_path = QtWidgets.QLineEdit(self.RF\_get\_samples\_group)

self.RF\_img\_path.setGeometry(QtCore.QRect(90, 24, 235, 22))

self.RF\_img\_path.setObjectName("RF\_img\_path")

self.RF\_img\_path.setReadOnly(True)

self.select\_RF\_img\_button = QtWidgets.QPushButton(self.RF\_get\_samples\_group)

self.select\_RF\_img\_button.setGeometry(QtCore.QRect(350, 20, 100, 30))

self.select\_RF\_img\_button.setObjectName("select\_RF\_img\_button")

self.select\_RF\_img\_button.clicked.connect(self.RF\_select\_image\_pushbutton\_clicked)

self.RF\_label\_path\_label = QtWidgets.QLabel(self.RF\_get\_samples\_group)

self.RF\_label\_path\_label.setGeometry(QtCore.QRect(10, 60, 80, 30))

self.RF\_label\_path\_label.setObjectName("RF\_label\_path\_label")

self.RF\_label\_path = QtWidgets.QLineEdit(self.RF\_get\_samples\_group)

self.RF\_label\_path.setGeometry(QtCore.QRect(90, 64, 235, 22))

self.RF\_label\_path.setObjectName("RF\_label\_path")

self.RF\_label\_path.setReadOnly(True)

self.select\_RF\_label\_button = QtWidgets.QPushButton(self.RF\_get\_samples\_group)

self.select\_RF\_label\_button.setGeometry(QtCore.QRect(350, 60, 100, 30))

self.select\_RF\_label\_button.setObjectName("select\_RF\_label\_button")

self.select\_RF\_label\_button.clicked.connect(self.RF\_select\_label\_pushbutton\_clicked)

self.RF\_class\_num\_label = QtWidgets.QLabel(self.RF\_get\_samples\_group)

self.RF\_class\_num\_label.setGeometry(QtCore.QRect(10, 100, 80, 30))

self.RF\_class\_num\_label.setObjectName("RF\_class\_num\_label")

self.RF\_class\_num\_path = QtWidgets.QLineEdit(self.RF\_get\_samples\_group)

self.RF\_class\_num\_path.setGeometry(QtCore.QRect(90, 104, 235, 22))

self.RF\_class\_num\_path.setObjectName("RF\_class\_num\_path")

self.RF\_class\_num\_path.setReadOnly(True)

self.select\_RF\_class\_num\_button = QtWidgets.QPushButton(self.RF\_get\_samples\_group)

self.select\_RF\_class\_num\_button.setGeometry(QtCore.QRect(350, 100, 100, 30))

self.select\_RF\_class\_num\_button.setObjectName("select\_RF\_class\_num\_button")

self.select\_RF\_class\_num\_button.clicked.connect(self.RF\_select\_numclass\_path\_pushButton\_clicked)

self.RF\_output\_samples\_path\_label = QtWidgets.QLabel(self.RF\_get\_samples\_group)

self.RF\_output\_samples\_path\_label.setGeometry(QtCore.QRect(10, 140, 80, 30))

self.RF\_output\_samples\_path\_label.setObjectName("RF\_output\_samples\_path\_label")

self.RF\_output\_samples\_path = QtWidgets.QLineEdit(self.RF\_get\_samples\_group)

self.RF\_output\_samples\_path.setGeometry(QtCore.QRect(90, 145, 235, 22))

self.RF\_output\_samples\_path.setObjectName("RF\_output\_samples\_path")

self.RF\_output\_samples\_path.setReadOnly(True)

self.select\_RF\_output\_samples\_path\_button = QtWidgets.QPushButton(self.RF\_get\_samples\_group)

self.select\_RF\_output\_samples\_path\_button.setGeometry(QtCore.QRect(350, 141, 100, 30))

self.select\_RF\_output\_samples\_path\_button.setObjectName("select\_RF\_output\_samples\_path\_button")

self.select\_RF\_output\_samples\_path\_button.clicked.connect(

self.RF\_select\_sample\_path\_pushButton\_clicked

)

self.RF\_get\_samples\_button = QtWidgets.QPushButton(self.RF\_get\_samples\_group)

self.RF\_get\_samples\_button.setGeometry(QtCore.QRect(160, 180, 100, 30))

self.RF\_get\_samples\_button.setObjectName("get\_RF\_samples\_button")

self.RF\_get\_samples\_button.clicked.connect(self.RF\_get\_sample\_pushButton\_clicked)

# ======================================== RF\_train\_group ========================================

self.RF\_train\_group = QtWidgets.QGroupBox(self.RF\_tab\_page)

self.RF\_train\_group.setGeometry(QtCore.QRect(10, 240, 460, 320))

self.RF\_train\_group.setObjectName("RF\_train\_group")

# model save path

self.RF\_train\_model\_save\_path\_label = QtWidgets.QLabel(self.RF\_train\_group)

self.RF\_train\_model\_save\_path\_label.setGeometry(QtCore.QRect(10, 245, 90, 25))

self.RF\_train\_model\_save\_path\_label.setObjectName("RF\_train\_model\_save\_path\_label")

self.RF\_train\_model\_save\_path = QtWidgets.QLineEdit(self.RF\_train\_group)

self.RF\_train\_model\_save\_path.setGeometry(QtCore.QRect(110, 246, 230, 22))

self.RF\_train\_model\_save\_path.setObjectName("RF\_train\_model\_save\_path")

self.RF\_train\_model\_save\_path.setReadOnly(True)

self.select\_RF\_train\_model\_save\_path\_button = QtWidgets.QPushButton(self.RF\_train\_group)

self.select\_RF\_train\_model\_save\_path\_button.setGeometry(QtCore.QRect(350, 243, 100, 30))

self.select\_RF\_train\_model\_save\_path\_button.setObjectName("select\_RF\_train\_model\_save\_path\_button")

self.select\_RF\_train\_model\_save\_path\_button.clicked.connect(

self.select\_RF\_train\_Save\_model\_path\_pushButton\_clicked

)

self.RF\_train\_button = QtWidgets.QPushButton(self.RF\_train\_group)

self.RF\_train\_button.setGeometry(QtCore.QRect(160, 280, 100, 30))

self.RF\_train\_button.setObjectName("RF\_train\_button")

self.RF\_train\_button.clicked.connect(self.RF\_train\_pushButton\_clicked)

self.RF\_val\_datasets\_size\_label = QtWidgets.QLabel(self.RF\_train\_group)

self.RF\_val\_datasets\_size\_label.setGeometry(QtCore.QRect(10, 20, 100, 20))

self.RF\_val\_datasets\_size\_label.setObjectName("RF\_val\_datasets\_size\_label")

self.RF\_val\_datasets\_size = QtWidgets.QLineEdit(self.RF\_train\_group)

self.RF\_val\_datasets\_size.setGeometry(QtCore.QRect(130, 20, 100, 20))

self.RF\_val\_datasets\_size.setObjectName("val\_datasets\_size")

self.RF\_n\_estimators\_label = QtWidgets.QLabel(self.RF\_train\_group)

self.RF\_n\_estimators\_label.setGeometry(QtCore.QRect(10, 65, 130, 20))

self.RF\_n\_estimators\_label.setObjectName("RF\_n\_estimators\_label")

self.RF\_n\_estimators\_min = QtWidgets.QLineEdit(self.RF\_train\_group)

self.RF\_n\_estimators\_min.setGeometry(QtCore.QRect(160, 65, 80, 20))

self.RF\_n\_estimators\_min.setObjectName("RF\_n\_estimators\_min")

self.RF\_n\_estimators\_interval = QtWidgets.QLineEdit(self.RF\_train\_group)

self.RF\_n\_estimators\_interval.setGeometry(QtCore.QRect(340, 65, 80, 20))

self.RF\_n\_estimators\_interval.setObjectName("RF\_n\_estimators\_interval")

self.RF\_n\_estimators\_max = QtWidgets.QLineEdit(self.RF\_train\_group)

self.RF\_n\_estimators\_max.setGeometry(QtCore.QRect(250, 65, 80, 20))

self.RF\_n\_estimators\_max.setObjectName("RF\_n\_estimators\_max")

self.RF\_Min\_label = QtWidgets.QLabel(self.RF\_train\_group)

self.RF\_Min\_label.setGeometry(QtCore.QRect(188, 45, 80, 20))

self.RF\_Min\_label.setObjectName("Min\_label")

self.RF\_Max\_label = QtWidgets.QLabel(self.RF\_train\_group)

self.RF\_Max\_label.setGeometry(QtCore.QRect(278, 45, 80, 20))

self.RF\_Max\_label.setObjectName("RF\_Max\_label")

self.RF\_Interval\_label = QtWidgets.QLabel(self.RF\_train\_group)

self.RF\_Interval\_label.setGeometry(QtCore.QRect(355, 45, 80, 20))

self.RF\_Interval\_label.setObjectName("RF\_Interval\_label")

self.RF\_max\_depth\_min = QtWidgets.QLineEdit(self.RF\_train\_group)

self.RF\_max\_depth\_min.setGeometry(QtCore.QRect(160, 95, 80, 20))

self.RF\_max\_depth\_min.setObjectName("max\_depth\_min")

self.RF\_max\_depth\_max = QtWidgets.QLineEdit(self.RF\_train\_group)

self.RF\_max\_depth\_max.setGeometry(QtCore.QRect(250, 95, 80, 20))

self.RF\_max\_depth\_max.setObjectName("max\_depth\_max")

self.RF\_max\_depth\_interval = QtWidgets.QLineEdit(self.RF\_train\_group)

self.RF\_max\_depth\_interval.setGeometry(QtCore.QRect(340, 95, 80, 20))

self.RF\_max\_depth\_interval.setObjectName("max\_depth\_interval")

self.RF\_max\_depth\_label = QtWidgets.QLabel(self.RF\_train\_group)

self.RF\_max\_depth\_label.setGeometry(QtCore.QRect(10, 95, 140, 20))

self.RF\_max\_depth\_label.setObjectName("max\_depth\_label")

self.RF\_min\_samples\_leaf\_min = QtWidgets.QLineEdit(self.RF\_train\_group)

self.RF\_min\_samples\_leaf\_min.setGeometry(QtCore.QRect(160, 125, 80, 20))

self.RF\_min\_samples\_leaf\_min.setObjectName("RF\_min\_samples\_leaf\_min")

self.RF\_min\_samples\_leaf\_max = QtWidgets.QLineEdit(self.RF\_train\_group)

self.RF\_min\_samples\_leaf\_max.setGeometry(QtCore.QRect(250, 125, 80, 20))

self.RF\_min\_samples\_leaf\_max.setObjectName("RF\_min\_samples\_leaf\_max")

self.RF\_min\_samples\_leaf\_interval = QtWidgets.QLineEdit(self.RF\_train\_group)

self.RF\_min\_samples\_leaf\_interval.setGeometry(QtCore.QRect(340, 125, 80, 20))

self.RF\_min\_samples\_leaf\_interval.setObjectName("RF\_min\_samples\_leaf\_interval")

self.RF\_min\_samples\_leaf\_label = QtWidgets.QLabel(self.RF\_train\_group)

self.RF\_min\_samples\_leaf\_label.setGeometry(QtCore.QRect(10, 125, 140, 20))

self.RF\_min\_samples\_leaf\_label.setObjectName("RF\_min\_samples\_leaf\_label")

self.RF\_min\_samples\_split\_min = QtWidgets.QLineEdit(self.RF\_train\_group)

self.RF\_min\_samples\_split\_min.setGeometry(QtCore.QRect(160, 155, 80, 20))

self.RF\_min\_samples\_split\_min.setObjectName("RF\_min\_samples\_split\_min")

self.RF\_min\_samples\_split\_max = QtWidgets.QLineEdit(self.RF\_train\_group)

self.RF\_min\_samples\_split\_max.setGeometry(QtCore.QRect(250, 155, 80, 20))

self.RF\_min\_samples\_split\_max.setObjectName("RF\_min\_samples\_split\_max")

self.RF\_min\_samples\_split\_interval = QtWidgets.QLineEdit(self.RF\_train\_group)

self.RF\_min\_samples\_split\_interval.setGeometry(QtCore.QRect(340, 155, 80, 20))

self.RF\_min\_samples\_split\_interval.setObjectName("RF\_min\_samples\_split\_interval")

self.RF\_min\_samples\_split\_label = QtWidgets.QLabel(self.RF\_train\_group)

self.RF\_min\_samples\_split\_label.setGeometry(QtCore.QRect(10, 155, 140, 20))

self.RF\_min\_samples\_split\_label.setObjectName("min\_samples\_split\_label")

self.sqrt\_cb = QtWidgets.QCheckBox(self.RF\_train\_group)

self.sqrt\_cb.setGeometry(QtCore.QRect(160, 185, 80, 20))

self.sqrt\_cb.setObjectName("sqrt\_cb")

# 选中

self.sqrt\_cb.setChecked(True)

self.log2\_cb = QtWidgets.QCheckBox(self.RF\_train\_group)

self.log2\_cb.setGeometry(QtCore.QRect(250, 185, 80, 20))

self.log2\_cb.setObjectName("log2\_cb")

# 选中

self.log2\_cb.setChecked(True)

self.None\_cb = QtWidgets.QCheckBox(self.RF\_train\_group)

self.None\_cb.setGeometry(QtCore.QRect(340, 185, 80, 20))

self.None\_cb.setObjectName("None\_cb")

# 选中

self.None\_cb.setChecked(True)

self.RF\_max\_features\_label = QtWidgets.QLabel(self.RF\_train\_group)

self.RF\_max\_features\_label.setGeometry(QtCore.QRect(10, 185, 140, 20))

self.RF\_max\_features\_label.setObjectName("max\_features\_label")

self.False\_cb = QtWidgets.QCheckBox(self.RF\_train\_group)

self.False\_cb.setGeometry(QtCore.QRect(250, 215, 90, 20))

self.False\_cb.setObjectName("False\_cb")

# 选中

self.False\_cb.setChecked(True)

self.True\_cb = QtWidgets.QCheckBox(self.RF\_train\_group)

self.True\_cb.setGeometry(QtCore.QRect(160, 215, 90, 20))

self.True\_cb.setObjectName("True\_cb")

# 选中

self.True\_cb.setChecked(True)

self.bootstrap\_label = QtWidgets.QLabel(self.RF\_train\_group)

self.bootstrap\_label.setGeometry(QtCore.QRect(10, 215, 130, 20))

self.bootstrap\_label.setObjectName("bootstrap\_label")

# ======================================== RF\_predict\_group ========================================

self.RF\_predict\_group = QtWidgets.QGroupBox(self.RF\_tab\_page)

self.RF\_predict\_group.setGeometry(QtCore.QRect(10, 580, 460, 260))

self.RF\_predict\_group.setObjectName("RF\_predict\_group")

self.RF\_model\_path\_label = QtWidgets.QLabel(self.RF\_predict\_group)

self.RF\_model\_path\_label.setGeometry(QtCore.QRect(10, 40, 80, 30))

self.RF\_model\_path\_label.setObjectName("RF\_model\_path\_label")

self.RF\_model\_path = QtWidgets.QLineEdit(self.RF\_predict\_group)

self.RF\_model\_path.setGeometry(QtCore.QRect(90, 44, 235, 22))

self.RF\_model\_path.setObjectName("RF\_model\_path")

self.RF\_model\_path.setReadOnly(True)

self.select\_RF\_model\_path\_button = QtWidgets.QPushButton(self.RF\_predict\_group)

self.select\_RF\_model\_path\_button.setGeometry(QtCore.QRect(350, 40, 100, 30))

self.select\_RF\_model\_path\_button.setObjectName("select\_RF\_model\_path\_button")

self.select\_RF\_model\_path\_button.clicked.connect(self.select\_RF\_model\_path\_pushButton\_clicked)

self.RF\_img\_predict\_path\_label = QtWidgets.QLabel(self.RF\_predict\_group)

self.RF\_img\_predict\_path\_label.setGeometry(QtCore.QRect(10, 80, 80, 30))

self.RF\_img\_predict\_path\_label.setObjectName("RF\_img\_predict\_path\_label")

self.RF\_img\_predict\_path = QtWidgets.QLineEdit(self.RF\_predict\_group)

self.RF\_img\_predict\_path.setGeometry(QtCore.QRect(90, 84, 235, 22))

self.RF\_img\_predict\_path.setObjectName("RF\_img\_predict\_path")

self.RF\_img\_predict\_path.setReadOnly(True)

self.select\_RF\_img\_predict\_path\_button = QtWidgets.QPushButton(self.RF\_predict\_group)

self.select\_RF\_img\_predict\_path\_button.setGeometry(QtCore.QRect(350, 80, 100, 30))

self.select\_RF\_img\_predict\_path\_button.setObjectName("select\_RF\_img\_predict\_path\_button")

self.select\_RF\_img\_predict\_path\_button.clicked.connect(

self.select\_RF\_img\_predict\_path\_pushButton\_clicked

)

self.RF\_class\_num\_label\_2 = QtWidgets.QLabel(self.RF\_predict\_group)

self.RF\_class\_num\_label\_2.setGeometry(QtCore.QRect(10, 120, 80, 30))

self.RF\_class\_num\_label\_2.setObjectName("RF\_class\_num\_label\_2")

self.RF\_class\_num\_path\_2 = QtWidgets.QLineEdit(self.RF\_predict\_group)

self.RF\_class\_num\_path\_2.setGeometry(QtCore.QRect(90, 124, 235, 22))

self.RF\_class\_num\_path\_2.setObjectName("RF\_class\_num\_path\_2")

self.RF\_class\_num\_path\_2.setReadOnly(True)

self.select\_RF\_class\_num\_button\_2 = QtWidgets.QPushButton(self.RF\_predict\_group)

self.select\_RF\_class\_num\_button\_2.setGeometry(QtCore.QRect(350, 120, 100, 30))

self.select\_RF\_class\_num\_button\_2.setObjectName("select\_RF\_class\_num\_button\_2")

self.select\_RF\_class\_num\_button\_2.clicked.connect(self.select\_RF\_class\_num\_pushButton\_clicked)

self.RF\_predict\_output\_result\_path\_label = QtWidgets.QLabel(self.RF\_predict\_group)

self.RF\_predict\_output\_result\_path\_label.setGeometry(QtCore.QRect(10, 160, 80, 30))

self.RF\_predict\_output\_result\_path\_label.setObjectName("RF\_predict\_output\_result\_path\_label")

self.RF\_output\_predict\_result\_path = QtWidgets.QLineEdit(self.RF\_predict\_group)

self.RF\_output\_predict\_result\_path.setGeometry(QtCore.QRect(90, 164, 235, 22))

self.RF\_output\_predict\_result\_path.setObjectName("RF\_output\_predict\_result\_path")

self.RF\_output\_predict\_result\_path.setReadOnly(True)

self.RF\_select\_predict\_result\_output\_path\_button = QtWidgets.QPushButton(self.RF\_predict\_group)

self.RF\_select\_predict\_result\_output\_path\_button.setGeometry(QtCore.QRect(350, 160, 100, 30))

self.RF\_select\_predict\_result\_output\_path\_button.setObjectName(

"RF\_select\_predict\_result\_output\_path\_button"

)

self.RF\_select\_predict\_result\_output\_path\_button.clicked.connect(

self.select\_RF\_predict\_result\_output\_path\_pushButton\_clicked

)

self.RF\_predict\_button = QtWidgets.QPushButton(self.RF\_predict\_group)

self.RF\_predict\_button.setGeometry(QtCore.QRect(160, 200, 100, 30))

self.RF\_predict\_button.setObjectName("RF\_predict\_button")

self.RF\_predict\_button.clicked.connect(self.RF\_predict\_pushButton\_clicked)

def LightGBM\_tab\_page\_init(self):

# ======================================== LGB\_get\_samples\_group ========================================

self.LGB\_get\_samples\_group = QtWidgets.QGroupBox(self.LGB\_tab\_page)

self.LGB\_get\_samples\_group.setGeometry(QtCore.QRect(10, 10, 460, 220))

self.LGB\_get\_samples\_group.setObjectName("LGB\_get\_samples\_group")

self.LGB\_img\_path\_label = QtWidgets.QLabel(self.LGB\_get\_samples\_group)

self.LGB\_img\_path\_label.setGeometry(QtCore.QRect(10, 20, 460, 30))

self.LGB\_img\_path\_label.setObjectName("LGB\_img\_path\_label")

self.LGB\_img\_path = QtWidgets.QLineEdit(self.LGB\_get\_samples\_group)

self.LGB\_img\_path.setGeometry(QtCore.QRect(90, 24, 235, 22))

self.LGB\_img\_path.setObjectName("LGB\_img\_path")

self.LGB\_img\_path.setReadOnly(True)

self.select\_LGB\_img\_button = QtWidgets.QPushButton(self.LGB\_get\_samples\_group)

self.select\_LGB\_img\_button.setGeometry(QtCore.QRect(350, 20, 100, 30))

self.select\_LGB\_img\_button.setObjectName("select\_LGB\_img\_button")

self.select\_LGB\_img\_button.clicked.connect(self.LGB\_select\_image\_pushbutton\_clicked)

self.LGB\_label\_path\_label = QtWidgets.QLabel(self.LGB\_get\_samples\_group)

self.LGB\_label\_path\_label.setGeometry(QtCore.QRect(10, 60, 80, 30))

self.LGB\_label\_path\_label.setObjectName("LGB\_label\_path\_label")

self.LGB\_label\_path = QtWidgets.QLineEdit(self.LGB\_get\_samples\_group)

self.LGB\_label\_path.setGeometry(QtCore.QRect(90, 64, 235, 22))

self.LGB\_label\_path.setObjectName("LGB\_label\_path")

self.LGB\_label\_path.setReadOnly(True)

self.select\_LGB\_lable\_button = QtWidgets.QPushButton(self.LGB\_get\_samples\_group)

self.select\_LGB\_lable\_button.setGeometry(QtCore.QRect(350, 60, 100, 30))

self.select\_LGB\_lable\_button.setObjectName("LGB\_label\_path\_label")

self.select\_LGB\_lable\_button.clicked.connect(self.LGB\_select\_label\_pushbutton\_clicked)

self.LGB\_class\_num\_label = QtWidgets.QLabel(self.LGB\_get\_samples\_group)

self.LGB\_class\_num\_label.setGeometry(QtCore.QRect(10, 100, 80, 30))

self.LGB\_class\_num\_label.setObjectName("LGB\_class\_num\_label")

self.LGB\_class\_num\_path = QtWidgets.QLineEdit(self.LGB\_get\_samples\_group)

self.LGB\_class\_num\_path.setGeometry(QtCore.QRect(90, 104, 235, 22))

self.LGB\_class\_num\_path.setObjectName("LGB\_class\_num\_path")

self.LGB\_class\_num\_path.setReadOnly(True)

self.select\_LGB\_class\_num\_button = QtWidgets.QPushButton(self.LGB\_get\_samples\_group)

self.select\_LGB\_class\_num\_button.setGeometry(QtCore.QRect(350, 100, 100, 30))

self.select\_LGB\_class\_num\_button.setObjectName("select\_LGB\_class\_num\_button")

self.select\_LGB\_class\_num\_button.clicked.connect(self.LGB\_select\_numclass\_path\_pushbutton\_clicked)

self.LGB\_output\_samples\_path\_label = QtWidgets.QLabel(self.LGB\_get\_samples\_group)

self.LGB\_output\_samples\_path\_label.setGeometry(QtCore.QRect(10, 140, 80, 30))

self.LGB\_output\_samples\_path\_label.setObjectName("LGB\_output\_samples\_path\_label")

self.LGB\_output\_samples\_path = QtWidgets.QLineEdit(self.LGB\_get\_samples\_group)

self.LGB\_output\_samples\_path.setGeometry(QtCore.QRect(90, 144, 235, 22))

self.LGB\_output\_samples\_path.setObjectName("LGB\_output\_samples\_path")

self.LGB\_output\_samples\_path.setReadOnly(True)

self.select\_LGB\_output\_samples\_path\_button = QtWidgets.QPushButton(self.LGB\_get\_samples\_group)

self.select\_LGB\_output\_samples\_path\_button.setGeometry(QtCore.QRect(350, 140, 100, 30))

self.select\_LGB\_output\_samples\_path\_button.setObjectName("select\_LGB\_output\_samples\_path\_button")

self.select\_LGB\_output\_samples\_path\_button.clicked.connect(

self.LGB\_select\_sample\_path\_pushbutton\_clicked

)

self.LGB\_get\_samples\_button = QtWidgets.QPushButton(self.LGB\_get\_samples\_group)

self.LGB\_get\_samples\_button.setGeometry(QtCore.QRect(160, 180, 100, 30))

self.LGB\_get\_samples\_button.setObjectName("get\_LGB\_samples\_button")

self.LGB\_get\_samples\_button.clicked.connect(self.LGB\_get\_sample\_pushButton\_clicked)

# ======================================== LGB\_train\_group ========================================

self.LGB\_train\_group = QtWidgets.QGroupBox(self.LGB\_tab\_page)

self.LGB\_train\_group.setGeometry(QtCore.QRect(10, 235, 460, 400))

self.LGB\_train\_group.setObjectName("LGB\_train\_group")

# model save path

self.LGB\_train\_model\_save\_path\_label = QtWidgets.QLabel(self.LGB\_train\_group)

self.LGB\_train\_model\_save\_path\_label.setGeometry(QtCore.QRect(10, 326, 90, 25))

self.LGB\_train\_model\_save\_path\_label.setObjectName("LGB\_train\_model\_save\_path\_label")

self.LGB\_train\_model\_save\_path = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_train\_model\_save\_path.setGeometry(QtCore.QRect(110, 327, 235, 22))

self.LGB\_train\_model\_save\_path.setObjectName("LGB\_train\_model\_save\_path")

self.LGB\_train\_model\_save\_path.setReadOnly(True)

self.select\_LGB\_train\_model\_save\_path\_button = QtWidgets.QPushButton(self.LGB\_train\_group)

self.select\_LGB\_train\_model\_save\_path\_button.setGeometry(QtCore.QRect(350, 323, 100, 30))

self.select\_LGB\_train\_model\_save\_path\_button.setObjectName("select\_LGB\_train\_model\_save\_path\_button")

self.select\_LGB\_train\_model\_save\_path\_button.clicked.connect(

self.select\_LGB\_train\_Save\_model\_path\_pushButton\_clicked

)

self.LGB\_train\_button = QtWidgets.QPushButton(self.LGB\_train\_group)

self.LGB\_train\_button.setGeometry(QtCore.QRect(160, 360, 100, 30))

self.LGB\_train\_button.setObjectName("LGB\_train\_button")

self.LGB\_train\_button.clicked.connect(self.LGB\_train\_pushButton\_clicked)

# 训练的参数设置

# 训练参数比例

self.LGB\_val\_datasets\_size\_label = QtWidgets.QLabel(self.LGB\_train\_group)

self.LGB\_val\_datasets\_size\_label.setGeometry(QtCore.QRect(10, 30, 100, 20))

self.LGB\_val\_datasets\_size\_label.setObjectName("LGB\_val\_datasets\_size\_label")

self.LGB\_val\_datasets\_size = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_val\_datasets\_size.setGeometry(QtCore.QRect(130, 30, 100, 20))

self.LGB\_val\_datasets\_size.setObjectName("val\_datasets\_size")

self.LGB\_Min\_label = QtWidgets.QLabel(self.LGB\_train\_group)

self.LGB\_Min\_label.setGeometry(QtCore.QRect(188, 55, 80, 20))

self.LGB\_Min\_label.setObjectName("LGB\_Min\_label")

self.LGB\_Max\_label = QtWidgets.QLabel(self.LGB\_train\_group)

self.LGB\_Max\_label.setGeometry(QtCore.QRect(278, 55, 80, 20))

self.LGB\_Max\_label.setObjectName("LGB\_Max\_label")

self.LGB\_Interval\_label = QtWidgets.QLabel(self.LGB\_train\_group)

self.LGB\_Interval\_label.setGeometry(QtCore.QRect(355, 55, 80, 20))

self.LGB\_Interval\_label.setObjectName("LGB\_Interval\_label")

self.LGB\_n\_estimators\_min = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_n\_estimators\_min.setGeometry(QtCore.QRect(160, 75, 80, 20))

self.LGB\_n\_estimators\_min.setObjectName("LGB\_n\_estimators\_min")

self.LGB\_n\_estimators\_interval = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_n\_estimators\_interval.setGeometry(QtCore.QRect(340, 75, 80, 20))

self.LGB\_n\_estimators\_interval.setObjectName("LGB\_n\_estimators\_interval")

self.LGB\_n\_estimators\_max = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_n\_estimators\_max.setGeometry(QtCore.QRect(250, 75, 80, 20))

self.LGB\_n\_estimators\_max.setObjectName("LGB\_n\_estimators\_max")

self.LGB\_n\_estimators\_label = QtWidgets.QLabel(self.LGB\_train\_group)

self.LGB\_n\_estimators\_label.setGeometry(QtCore.QRect(10, 75, 130, 20))

self.LGB\_n\_estimators\_label.setObjectName("LGB\_n\_estimators\_label")

self.LGB\_max\_depth\_min = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_max\_depth\_min.setGeometry(QtCore.QRect(160, 105, 80, 20))

self.LGB\_max\_depth\_min.setObjectName("max\_depth\_min")

self.LGB\_max\_depth\_max = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_max\_depth\_max.setGeometry(QtCore.QRect(250, 105, 80, 20))

self.LGB\_max\_depth\_max.setObjectName("max\_depth\_max")

self.LGB\_max\_depth\_interval = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_max\_depth\_interval.setGeometry(QtCore.QRect(340, 105, 80, 20))

self.LGB\_max\_depth\_interval.setObjectName("max\_depth\_interval")

self.LGB\_max\_depth\_label = QtWidgets.QLabel(self.LGB\_train\_group)

self.LGB\_max\_depth\_label.setGeometry(QtCore.QRect(10, 105, 140, 20))

self.LGB\_max\_depth\_label.setObjectName("max\_depth\_label")

self.LGB\_learning\_rate\_min = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_learning\_rate\_min.setGeometry(QtCore.QRect(160, 135, 80, 20))

self.LGB\_learning\_rate\_min.setObjectName("learning\_rate\_min")

self.LGB\_learning\_rate\_Max = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_learning\_rate\_Max.setGeometry(QtCore.QRect(250, 135, 80, 20))

self.LGB\_learning\_rate\_Max.setObjectName("learning\_rate\_Max")

self.LGB\_learning\_rate\_Interval = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_learning\_rate\_Interval.setGeometry(QtCore.QRect(340, 135, 80, 20))

self.LGB\_learning\_rate\_Interval.setObjectName("learning\_rate\_Interval")

self.LGB\_learning\_rate\_label = QtWidgets.QLabel(self.LGB\_train\_group)

self.LGB\_learning\_rate\_label.setGeometry(QtCore.QRect(10, 135, 140, 20))

self.LGB\_learning\_rate\_label.setObjectName("learning\_rate\_label")

self.LGB\_subsample\_min = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_subsample\_min.setGeometry(QtCore.QRect(160, 165, 80, 20))

self.LGB\_subsample\_min.setObjectName("subsample\_min")

self.LGB\_subsample\_max = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_subsample\_max.setGeometry(QtCore.QRect(250, 165, 80, 20))

self.LGB\_subsample\_max.setObjectName("subsample\_max")

self.LGB\_subsample\_interval = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_subsample\_interval.setGeometry(QtCore.QRect(340, 165, 80, 20))

self.LGB\_subsample\_interval.setObjectName("subsample\_interval")

self.LGB\_subsample\_label = QtWidgets.QLabel(self.LGB\_train\_group)

self.LGB\_subsample\_label.setGeometry(QtCore.QRect(10, 165, 140, 20))

self.LGB\_subsample\_label.setObjectName("LGB\_subsample\_label")

self.LGB\_colsample\_bytree\_min = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_colsample\_bytree\_min.setGeometry(QtCore.QRect(160, 195, 80, 20))

self.LGB\_colsample\_bytree\_min.setObjectName("LGB\_colsample\_bytree\_min")

self.LGB\_colsample\_bytree\_max = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_colsample\_bytree\_max.setGeometry(QtCore.QRect(250, 195, 80, 20))

self.LGB\_colsample\_bytree\_max.setObjectName("LGB\_colsample\_bytree\_max")

self.LGB\_colsample\_bytree\_interval = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_colsample\_bytree\_interval.setGeometry(QtCore.QRect(340, 195, 80, 20))

self.LGB\_colsample\_bytree\_interval.setObjectName("LGB\_colsample\_bytree\_interval")

self.LGB\_colsample\_bytree\_label = QtWidgets.QLabel(self.LGB\_train\_group)

self.LGB\_colsample\_bytree\_label.setGeometry(QtCore.QRect(10, 195, 140, 20))

self.LGB\_colsample\_bytree\_label.setObjectName("LGB\_colsample\_bytree\_label")

self.LGB\_reg\_alpha\_min = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_reg\_alpha\_min.setGeometry(QtCore.QRect(160, 225, 80, 20))

self.LGB\_reg\_alpha\_min.setObjectName("LGB\_reg\_alpha\_min")

self.LGB\_reg\_alpha\_max = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_reg\_alpha\_max.setGeometry(QtCore.QRect(250, 225, 80, 20))

self.LGB\_reg\_alpha\_max.setObjectName("LGB\_reg\_alpha\_max")

self.LGB\_reg\_alpha\_interval = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_reg\_alpha\_interval.setGeometry(QtCore.QRect(340, 225, 80, 20))

self.LGB\_reg\_alpha\_interval.setObjectName("LGB\_reg\_alpha\_interval")

self.LGB\_reg\_alpha\_label = QtWidgets.QLabel(self.LGB\_train\_group)

self.LGB\_reg\_alpha\_label.setGeometry(QtCore.QRect(10, 225, 140, 20))

self.LGB\_reg\_alpha\_label.setObjectName("LGB\_reg\_alpha\_label")

self.LGB\_reg\_lambda\_min = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_reg\_lambda\_min.setGeometry(QtCore.QRect(160, 255, 80, 20))

self.LGB\_reg\_lambda\_min.setObjectName("LGB\_reg\_lambda\_min")

self.LGB\_reg\_lambda\_max = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_reg\_lambda\_max.setGeometry(QtCore.QRect(250, 255, 80, 20))

self.LGB\_reg\_lambda\_max.setObjectName("LGB\_reg\_lambda\_max")

self.LGB\_reg\_lambda\_interval = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_reg\_lambda\_interval.setGeometry(QtCore.QRect(340, 255, 80, 20))

self.LGB\_reg\_lambda\_interval.setObjectName("LGB\_reg\_lambda\_interval")

self.LGB\_reg\_lambda\_label = QtWidgets.QLabel(self.LGB\_train\_group)

self.LGB\_reg\_lambda\_label.setGeometry(QtCore.QRect(10, 255, 140, 20))

self.LGB\_reg\_lambda\_label.setObjectName("LGB\_reg\_lambda\_label")

self.LGB\_min\_child\_weight\_min = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_min\_child\_weight\_min.setGeometry(QtCore.QRect(160, 285, 80, 20))

self.LGB\_min\_child\_weight\_min.setObjectName("LGB\_min\_child\_weight\_min")

self.LGB\_min\_child\_weight\_max = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_min\_child\_weight\_max.setGeometry(QtCore.QRect(250, 285, 80, 20))

self.LGB\_min\_child\_weight\_max.setObjectName("LGB\_min\_child\_weight\_max")

self.LGB\_min\_child\_weight\_interval = QtWidgets.QLineEdit(self.LGB\_train\_group)

self.LGB\_min\_child\_weight\_interval.setGeometry(QtCore.QRect(340, 285, 80, 20))

self.LGB\_min\_child\_weight\_interval.setObjectName("LGB\_min\_child\_weight\_interval")

self.LGB\_min\_child\_weight\_label = QtWidgets.QLabel(self.LGB\_train\_group)

self.LGB\_min\_child\_weight\_label.setGeometry(QtCore.QRect(10, 285, 140, 20))

self.LGB\_min\_child\_weight\_label.setObjectName("LGB\_min\_child\_weight\_label")

# ======================================== LGB\_predict\_group ========================================

self.LGB\_predict\_group = QtWidgets.QGroupBox(self.LGB\_tab\_page)

self.LGB\_predict\_group.setGeometry(QtCore.QRect(10, 640, 460, 220))

self.LGB\_predict\_group.setObjectName("LGB\_predict\_group")

# 模型路径

self.LGB\_model\_path\_label = QtWidgets.QLabel(self.LGB\_predict\_group)

self.LGB\_model\_path\_label.setGeometry(QtCore.QRect(10, 20, 80, 30))

self.LGB\_model\_path\_label.setObjectName("LGB\_model\_path\_label")

self.LGB\_model\_path = QtWidgets.QLineEdit(self.LGB\_predict\_group)

self.LGB\_model\_path.setGeometry(QtCore.QRect(90, 24, 235, 22))

self.LGB\_model\_path.setObjectName("LGB\_model\_path")

self.LGB\_model\_path.setReadOnly(True)

self.select\_LGB\_model\_path\_button = QtWidgets.QPushButton(self.LGB\_predict\_group)

self.select\_LGB\_model\_path\_button.setGeometry(QtCore.QRect(350, 20, 100, 30))

self.select\_LGB\_model\_path\_button.setObjectName("select\_LGB\_model\_path\_button")

self.select\_LGB\_model\_path\_button.clicked.connect(self.select\_LGB\_model\_path\_pushbutton\_clicked)

# 输出路径

self.LGB\_predict\_output\_result\_path\_label = QtWidgets.QLabel(self.LGB\_predict\_group)

self.LGB\_predict\_output\_result\_path\_label.setGeometry(QtCore.QRect(10, 140, 80, 30))

self.LGB\_predict\_output\_result\_path\_label.setObjectName("LGB\_predict\_output\_result\_path\_label")

self.LGB\_output\_predict\_result\_path = QtWidgets.QLineEdit(self.LGB\_predict\_group)

self.LGB\_output\_predict\_result\_path.setGeometry(QtCore.QRect(90, 144, 235, 22))

self.LGB\_output\_predict\_result\_path.setObjectName("LGB\_output\_predict\_result\_path")

self.LGB\_output\_predict\_result\_path.setReadOnly(True)

self.LGB\_select\_predict\_result\_output\_path\_button = QtWidgets.QPushButton(self.LGB\_predict\_group)

self.LGB\_select\_predict\_result\_output\_path\_button.setGeometry(QtCore.QRect(350, 140, 100, 30))

self.LGB\_select\_predict\_result\_output\_path\_button.setObjectName(

"LGB\_select\_predict\_result\_output\_path\_button"

)

self.LGB\_select\_predict\_result\_output\_path\_button.clicked.connect(

self.LGB\_select\_predict\_result\_output\_path\_pushbutton\_clicked

)

# 影像路径

self.LGB\_img\_predict\_path\_label = QtWidgets.QLabel(self.LGB\_predict\_group)

self.LGB\_img\_predict\_path\_label.setGeometry(QtCore.QRect(10, 60, 80, 30))

self.LGB\_img\_predict\_path\_label.setObjectName("LGB\_img\_predict\_path\_label")

self.LGB\_img\_predict\_path = QtWidgets.QLineEdit(self.LGB\_predict\_group)

self.LGB\_img\_predict\_path.setGeometry(QtCore.QRect(90, 64, 235, 22))

self.LGB\_img\_predict\_path.setObjectName("LGB\_img\_predict\_path")

self.LGB\_img\_predict\_path.setReadOnly(True)

self.select\_LGB\_img\_predict\_path\_button = QtWidgets.QPushButton(self.LGB\_predict\_group)

self.select\_LGB\_img\_predict\_path\_button.setGeometry(QtCore.QRect(350, 60, 100, 30))

self.select\_LGB\_img\_predict\_path\_button.setObjectName("select\_LGB\_img\_predict\_path\_button")

self.select\_LGB\_img\_predict\_path\_button.clicked.connect(

self.select\_LGB\_img\_predict\_path\_button\_clicked

)

# 类别路径

self.LGB\_class\_num\_label\_2 = QtWidgets.QLabel(self.LGB\_predict\_group)

self.LGB\_class\_num\_label\_2.setGeometry(QtCore.QRect(10, 100, 80, 30))

self.LGB\_class\_num\_label\_2.setObjectName("LGB\_class\_num\_label\_2")

self.LGB\_class\_num\_path\_2 = QtWidgets.QLineEdit(self.LGB\_predict\_group)

self.LGB\_class\_num\_path\_2.setGeometry(QtCore.QRect(90, 104, 235, 22))

self.LGB\_class\_num\_path\_2.setObjectName("LGB\_class\_num\_path\_2")

self.LGB\_class\_num\_path\_2.setReadOnly(True)

self.select\_LGB\_class\_num\_button\_2 = QtWidgets.QPushButton(self.LGB\_predict\_group)

self.select\_LGB\_class\_num\_button\_2.setGeometry(QtCore.QRect(350, 100, 100, 30))

self.select\_LGB\_class\_num\_button\_2.setObjectName("select\_LGB\_class\_num\_button\_2")

self.select\_LGB\_class\_num\_button\_2.clicked.connect(self.select\_LGB\_class\_num\_button\_2\_clicked)

self.LGB\_predict\_button = QtWidgets.QPushButton(self.LGB\_predict\_group)

self.LGB\_predict\_button.setGeometry(QtCore.QRect(160, 180, 100, 30))

self.LGB\_predict\_button.setObjectName("LGB\_predict\_button")

self.LGB\_predict\_button.clicked.connect(self.LGB\_predict\_pushButton\_clicked)

def XGB\_tab\_page\_init(self):

# ======================================== XGB\_get\_samples\_group ========================================

self.XGB\_get\_samples\_group = QtWidgets.QGroupBox(self.XGB\_tab\_page)

self.XGB\_get\_samples\_group.setGeometry(QtCore.QRect(10, 10, 460, 215))

self.XGB\_get\_samples\_group.setObjectName("XGB\_get\_samples\_group")

self.XGB\_img\_path\_label = QtWidgets.QLabel(self.XGB\_get\_samples\_group)

self.XGB\_img\_path\_label.setGeometry(QtCore.QRect(10, 20, 460, 30))

self.XGB\_img\_path\_label.setObjectName("XGB\_img\_path\_label")

self.XGB\_img\_path = QtWidgets.QLineEdit(self.XGB\_get\_samples\_group)

self.XGB\_img\_path.setGeometry(QtCore.QRect(90, 24, 235, 22))

self.XGB\_img\_path.setObjectName("XGB\_img\_path")

self.XGB\_img\_path.setReadOnly(True)

self.select\_XGB\_img\_button = QtWidgets.QPushButton(self.XGB\_get\_samples\_group)

self.select\_XGB\_img\_button.setGeometry(QtCore.QRect(350, 20, 100, 30))

self.select\_XGB\_img\_button.setObjectName("select\_XGB\_img\_button")

self.select\_XGB\_img\_button.clicked.connect(self.XGB\_select\_image\_pushbutton\_clicked)

self.XGB\_label\_path\_label = QtWidgets.QLabel(self.XGB\_get\_samples\_group)

self.XGB\_label\_path\_label.setGeometry(QtCore.QRect(10, 60, 80, 30))

self.XGB\_label\_path\_label.setObjectName("XGB\_label\_path\_label")

self.select\_XGB\_lable\_button = QtWidgets.QPushButton(self.XGB\_get\_samples\_group)

self.select\_XGB\_lable\_button.setGeometry(QtCore.QRect(350, 60, 100, 30))

self.select\_XGB\_lable\_button.setObjectName("XGB\_label\_path\_label")

self.select\_XGB\_lable\_button.clicked.connect(self.XGB\_select\_label\_pushbutton\_clicked)

self.XGB\_label\_path = QtWidgets.QLineEdit(self.XGB\_get\_samples\_group)

self.XGB\_label\_path.setGeometry(QtCore.QRect(90, 64, 235, 22))

self.XGB\_label\_path.setObjectName("XGB\_label\_path")

self.XGB\_label\_path.setReadOnly(True)

self.XGB\_class\_num\_label = QtWidgets.QLabel(self.XGB\_get\_samples\_group)

self.XGB\_class\_num\_label.setGeometry(QtCore.QRect(10, 100, 80, 30))

self.XGB\_class\_num\_label.setObjectName("XGB\_class\_num\_label")

self.XGB\_class\_num\_path = QtWidgets.QLineEdit(self.XGB\_get\_samples\_group)

self.XGB\_class\_num\_path.setGeometry(QtCore.QRect(90, 104, 235, 22))

self.XGB\_class\_num\_path.setObjectName("XGB\_class\_num\_path")

self.XGB\_class\_num\_path.setReadOnly(True)

self.select\_XGB\_class\_num\_button = QtWidgets.QPushButton(self.XGB\_get\_samples\_group)

self.select\_XGB\_class\_num\_button.setGeometry(QtCore.QRect(350, 100, 100, 30))

self.select\_XGB\_class\_num\_button.setObjectName("select\_XGB\_class\_num\_button")

self.select\_XGB\_class\_num\_button.clicked.connect(self.XGB\_select\_numclass\_path\_pushbutton\_clicked)

self.XGB\_output\_samples\_path\_label = QtWidgets.QLabel(self.XGB\_get\_samples\_group)

self.XGB\_output\_samples\_path\_label.setGeometry(QtCore.QRect(10, 140, 80, 30))

self.XGB\_output\_samples\_path\_label.setObjectName("XGB\_output\_samples\_path\_label")

self.XGB\_output\_samples\_path = QtWidgets.QLineEdit(self.XGB\_get\_samples\_group)

self.XGB\_output\_samples\_path.setGeometry(QtCore.QRect(90, 144, 235, 22))

self.XGB\_output\_samples\_path.setObjectName("XGB\_output\_samples\_path")

self.XGB\_output\_samples\_path.setReadOnly(True)

self.select\_XGB\_output\_samples\_path\_button = QtWidgets.QPushButton(self.XGB\_get\_samples\_group)

self.select\_XGB\_output\_samples\_path\_button.setGeometry(QtCore.QRect(350, 140, 100, 30))

self.select\_XGB\_output\_samples\_path\_button.setObjectName("select\_XGB\_output\_samples\_path\_button")

self.select\_XGB\_output\_samples\_path\_button.clicked.connect(

self.XGB\_select\_sample\_path\_pushbutton\_clicked

)

self.XGB\_get\_samples\_button = QtWidgets.QPushButton(self.XGB\_get\_samples\_group)

self.XGB\_get\_samples\_button.setGeometry(QtCore.QRect(160, 175, 100, 30))

self.XGB\_get\_samples\_button.setObjectName("get\_XGB\_samples\_button")

self.XGB\_get\_samples\_button.clicked.connect(self.XGB\_get\_sample\_pushButton\_clicked)

# ======================================== XGB\_train\_group ========================================

self.XGB\_train\_group = QtWidgets.QGroupBox(self.XGB\_tab\_page)

self.XGB\_train\_group.setGeometry(QtCore.QRect(10, 230, 460, 401))

self.XGB\_train\_group.setObjectName("XGB\_train\_group")

# model save path

self.XGB\_train\_model\_save\_path\_label = QtWidgets.QLabel(self.XGB\_train\_group)

self.XGB\_train\_model\_save\_path\_label.setGeometry(QtCore.QRect(10, 336, 90, 25))

self.XGB\_train\_model\_save\_path\_label.setObjectName("XGB\_train\_model\_save\_path\_label")

self.XGB\_train\_model\_save\_path = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_train\_model\_save\_path.setGeometry(QtCore.QRect(110, 339, 230, 22))

self.XGB\_train\_model\_save\_path.setObjectName("XGB\_train\_model\_save\_path")

self.XGB\_train\_model\_save\_path.setReadOnly(True)

self.select\_XGB\_train\_model\_save\_path\_button = QtWidgets.QPushButton(self.XGB\_train\_group)

self.select\_XGB\_train\_model\_save\_path\_button.setGeometry(QtCore.QRect(350, 335, 100, 30))

self.select\_XGB\_train\_model\_save\_path\_button.setObjectName("select\_XGB\_train\_model\_save\_path\_button")

self.select\_XGB\_train\_model\_save\_path\_button.clicked.connect(

self.select\_XGB\_train\_Save\_model\_path\_pushButton\_clicked

)

self.XGB\_train\_button = QtWidgets.QPushButton(self.XGB\_train\_group)

self.XGB\_train\_button.setGeometry(QtCore.QRect(160, 366, 100, 30))

self.XGB\_train\_button.setObjectName("XGB\_train\_button")

self.XGB\_train\_button.clicked.connect(self.XGB\_train\_pushButton\_clicked)

# 训练的参数设置

# 训练参数比例

self.XGB\_val\_datasets\_size\_label = QtWidgets.QLabel(self.XGB\_train\_group)

self.XGB\_val\_datasets\_size\_label.setGeometry(QtCore.QRect(10, 20, 100, 20))

self.XGB\_val\_datasets\_size\_label.setObjectName("XGB\_val\_datasets\_size\_label")

self.XGB\_val\_datasets\_size = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_val\_datasets\_size.setGeometry(QtCore.QRect(130, 20, 100, 20))

self.XGB\_val\_datasets\_size.setObjectName("val\_datasets\_size")

self.XGB\_Min\_label = QtWidgets.QLabel(self.XGB\_train\_group)

self.XGB\_Min\_label.setGeometry(QtCore.QRect(188, 45, 80, 20))

self.XGB\_Min\_label.setObjectName("XGB\_Min\_label")

self.XGB\_Max\_label = QtWidgets.QLabel(self.XGB\_train\_group)

self.XGB\_Max\_label.setGeometry(QtCore.QRect(278, 45, 80, 20))

self.XGB\_Max\_label.setObjectName("XGB\_Max\_label")

self.XGB\_Interval\_label = QtWidgets.QLabel(self.XGB\_train\_group)

self.XGB\_Interval\_label.setGeometry(QtCore.QRect(355, 45, 80, 20))

self.XGB\_Interval\_label.setObjectName("XGB\_Interval\_label")

self.XGB\_n\_estimators\_min = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_n\_estimators\_min.setGeometry(QtCore.QRect(160, 65, 80, 20))

self.XGB\_n\_estimators\_min.setObjectName("XGB\_n\_estimators\_min")

self.XGB\_n\_estimators\_interval = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_n\_estimators\_interval.setGeometry(QtCore.QRect(340, 65, 80, 20))

self.XGB\_n\_estimators\_interval.setObjectName("XGB\_n\_estimators\_interval")

self.XGB\_n\_estimators\_max = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_n\_estimators\_max.setGeometry(QtCore.QRect(250, 65, 80, 20))

self.XGB\_n\_estimators\_max.setObjectName("XGB\_n\_estimators\_max")

self.XGB\_n\_estimators\_label = QtWidgets.QLabel(self.XGB\_train\_group)

self.XGB\_n\_estimators\_label.setGeometry(QtCore.QRect(10, 65, 130, 20))

self.XGB\_n\_estimators\_label.setObjectName("XGB\_n\_estimators\_label")

self.XGB\_max\_depth\_min = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_max\_depth\_min.setGeometry(QtCore.QRect(160, 95, 80, 20))

self.XGB\_max\_depth\_min.setObjectName("max\_depth\_min")

self.XGB\_max\_depth\_max = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_max\_depth\_max.setGeometry(QtCore.QRect(250, 95, 80, 20))

self.XGB\_max\_depth\_max.setObjectName("max\_depth\_max")

self.XGB\_max\_depth\_interval = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_max\_depth\_interval.setGeometry(QtCore.QRect(340, 95, 80, 20))

self.XGB\_max\_depth\_interval.setObjectName("max\_depth\_interval")

self.XGB\_max\_depth\_label = QtWidgets.QLabel(self.XGB\_train\_group)

self.XGB\_max\_depth\_label.setGeometry(QtCore.QRect(10, 95, 140, 20))

self.XGB\_max\_depth\_label.setObjectName("max\_depth\_label")

self.XGB\_learning\_rate\_min = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_learning\_rate\_min.setGeometry(QtCore.QRect(160, 125, 80, 20))

self.XGB\_learning\_rate\_min.setObjectName("learning\_rate\_min")

self.XGB\_learning\_rate\_Max = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_learning\_rate\_Max.setGeometry(QtCore.QRect(250, 125, 80, 20))

self.XGB\_learning\_rate\_Max.setObjectName("learning\_rate\_Max")

self.XGB\_learning\_rate\_Interval = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_learning\_rate\_Interval.setGeometry(QtCore.QRect(340, 125, 80, 20))

self.XGB\_learning\_rate\_Interval.setObjectName("learning\_rate\_Interval")

self.XGB\_learning\_rate\_label = QtWidgets.QLabel(self.XGB\_train\_group)

self.XGB\_learning\_rate\_label.setGeometry(QtCore.QRect(10, 125, 140, 20))

self.XGB\_learning\_rate\_label.setObjectName("learning\_rate\_label")

self.XGB\_subsample\_min = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_subsample\_min.setGeometry(QtCore.QRect(160, 155, 80, 20))

self.XGB\_subsample\_min.setObjectName("subsample\_min")

self.XGB\_subsample\_max = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_subsample\_max.setGeometry(QtCore.QRect(250, 155, 80, 20))

self.XGB\_subsample\_max.setObjectName("subsample\_max")

self.XGB\_subsample\_interval = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_subsample\_interval.setGeometry(QtCore.QRect(340, 155, 80, 20))

self.XGB\_subsample\_interval.setObjectName("subsample\_interval")

self.XGB\_subsample\_label = QtWidgets.QLabel(self.XGB\_train\_group)

self.XGB\_subsample\_label.setGeometry(QtCore.QRect(10, 155, 140, 20))

self.XGB\_subsample\_label.setObjectName("XGB\_subsample\_label")

self.XGB\_colsample\_bytree\_min = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_colsample\_bytree\_min.setGeometry(QtCore.QRect(160, 185, 80, 20))

self.XGB\_colsample\_bytree\_min.setObjectName("XGB\_colsample\_bytree\_min")

self.XGB\_colsample\_bytree\_max = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_colsample\_bytree\_max.setGeometry(QtCore.QRect(250, 185, 80, 20))

self.XGB\_colsample\_bytree\_max.setObjectName("XGB\_colsample\_bytree\_max")

self.XGB\_colsample\_bytree\_interval = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_colsample\_bytree\_interval.setGeometry(QtCore.QRect(340, 185, 80, 20))

self.XGB\_colsample\_bytree\_interval.setObjectName("XGB\_colsample\_bytree\_interval")

self.XGB\_colsample\_bytree\_label = QtWidgets.QLabel(self.XGB\_train\_group)

self.XGB\_colsample\_bytree\_label.setGeometry(QtCore.QRect(10, 185, 140, 20))

self.XGB\_colsample\_bytree\_label.setObjectName("XGB\_colsample\_bytree\_label")

self.XGB\_gamma\_min = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_gamma\_min.setGeometry(QtCore.QRect(160, 215, 80, 20))

self.XGB\_gamma\_min.setObjectName("XGB\_gamma\_min")

self.XGB\_gamma\_max = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_gamma\_max.setGeometry(QtCore.QRect(250, 215, 80, 20))

self.XGB\_gamma\_max.setObjectName("XGB\_gamma\_max")

self.XGB\_gamma\_interval = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_gamma\_interval.setGeometry(QtCore.QRect(340, 215, 80, 20))

self.XGB\_gamma\_interval.setObjectName("XGB\_gamma\_interval")

self.XGB\_gamma\_label = QtWidgets.QLabel(self.XGB\_train\_group)

self.XGB\_gamma\_label.setGeometry(QtCore.QRect(10, 215, 140, 20))

self.XGB\_gamma\_label.setObjectName("XGB\_gamma\_label")

self.XGB\_reg\_alpha\_min = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_reg\_alpha\_min.setGeometry(QtCore.QRect(160, 245, 80, 20))

self.XGB\_reg\_alpha\_min.setObjectName("XGB\_reg\_alpha\_min")

self.XGB\_reg\_alpha\_max = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_reg\_alpha\_max.setGeometry(QtCore.QRect(250, 245, 80, 20))

self.XGB\_reg\_alpha\_max.setObjectName("XGB\_reg\_alpha\_max")

self.XGB\_reg\_alpha\_interval = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_reg\_alpha\_interval.setGeometry(QtCore.QRect(340, 245, 80, 20))

self.XGB\_reg\_alpha\_interval.setObjectName("XGB\_reg\_alpha\_interval")

self.XGB\_reg\_alpha\_label = QtWidgets.QLabel(self.XGB\_train\_group)

self.XGB\_reg\_alpha\_label.setGeometry(QtCore.QRect(10, 245, 140, 20))

self.XGB\_reg\_alpha\_label.setObjectName("XGB\_reg\_alpha\_label")

self.XGB\_reg\_lambda\_min = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_reg\_lambda\_min.setGeometry(QtCore.QRect(160, 275, 80, 20))

self.XGB\_reg\_lambda\_min.setObjectName("XGB\_reg\_lambda\_min")

self.XGB\_reg\_lambda\_max = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_reg\_lambda\_max.setGeometry(QtCore.QRect(250, 275, 80, 20))

self.XGB\_reg\_lambda\_max.setObjectName("XGB\_reg\_lambda\_max")

self.XGB\_reg\_lambda\_interval = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_reg\_lambda\_interval.setGeometry(QtCore.QRect(340, 275, 80, 20))

self.XGB\_reg\_lambda\_interval.setObjectName("XGB\_reg\_lambda\_interval")

self.XGB\_reg\_lambda\_label = QtWidgets.QLabel(self.XGB\_train\_group)

self.XGB\_reg\_lambda\_label.setGeometry(QtCore.QRect(10, 275, 140, 20))

self.XGB\_reg\_lambda\_label.setObjectName("XGB\_reg\_lambda\_label")

self.XGB\_min\_child\_weight\_min = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_min\_child\_weight\_min.setGeometry(QtCore.QRect(160, 305, 80, 20))

self.XGB\_min\_child\_weight\_min.setObjectName("XGB\_min\_child\_weight\_min")

self.XGB\_min\_child\_weight\_max = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_min\_child\_weight\_max.setGeometry(QtCore.QRect(250, 305, 80, 20))

self.XGB\_min\_child\_weight\_max.setObjectName("XGB\_min\_child\_weight\_max")

self.XGB\_min\_child\_weight\_interval = QtWidgets.QLineEdit(self.XGB\_train\_group)

self.XGB\_min\_child\_weight\_interval.setGeometry(QtCore.QRect(340, 305, 80, 20))

self.XGB\_min\_child\_weight\_interval.setObjectName("XGB\_min\_child\_weight\_interval")

self.XGB\_min\_child\_weight\_label = QtWidgets.QLabel(self.XGB\_train\_group)

self.XGB\_min\_child\_weight\_label.setGeometry(QtCore.QRect(10, 305, 140, 20))

self.XGB\_min\_child\_weight\_label.setObjectName("XGB\_min\_child\_weight\_label")

# ======================================== XGB\_predict\_group ========================================

self.XGB\_predict\_group = QtWidgets.QGroupBox(self.XGB\_tab\_page)

self.XGB\_predict\_group.setGeometry(QtCore.QRect(10, 640, 460, 220))

self.XGB\_predict\_group.setObjectName("XGB\_predict\_group")

self.XGB\_model\_path\_label = QtWidgets.QLabel(self.XGB\_predict\_group)

self.XGB\_model\_path\_label.setGeometry(QtCore.QRect(10, 20, 80, 30))

self.XGB\_model\_path\_label.setObjectName("XGB\_model\_path\_label")

self.XGB\_model\_path = QtWidgets.QLineEdit(self.XGB\_predict\_group)

self.XGB\_model\_path.setGeometry(QtCore.QRect(90, 24, 235, 22))

self.XGB\_model\_path.setObjectName("XGB\_model\_path")

self.XGB\_model\_path.setReadOnly(True)

self.select\_XGB\_model\_path\_button = QtWidgets.QPushButton(self.XGB\_predict\_group)

self.select\_XGB\_model\_path\_button.setGeometry(QtCore.QRect(350, 20, 100, 30))

self.select\_XGB\_model\_path\_button.setObjectName("select\_XGB\_model\_path\_button")

self.select\_XGB\_model\_path\_button.clicked.connect(self.select\_XGB\_model\_path\_pushButton\_clicked)

self.XGB\_predict\_output\_result\_path\_label = QtWidgets.QLabel(self.XGB\_predict\_group)

self.XGB\_predict\_output\_result\_path\_label.setGeometry(QtCore.QRect(10, 140, 80, 30))

self.XGB\_predict\_output\_result\_path\_label.setObjectName("XGB\_predict\_output\_result\_path\_label")

self.XGB\_output\_predict\_result\_path = QtWidgets.QLineEdit(self.XGB\_predict\_group)

self.XGB\_output\_predict\_result\_path.setGeometry(QtCore.QRect(90, 144, 235, 22))

self.XGB\_output\_predict\_result\_path.setObjectName("XGB\_output\_predict\_result\_path")

self.XGB\_output\_predict\_result\_path.setReadOnly(True)

self.XGB\_select\_predict\_result\_output\_path\_button = QtWidgets.QPushButton(self.XGB\_predict\_group)

self.XGB\_select\_predict\_result\_output\_path\_button.setGeometry(QtCore.QRect(350, 140, 100, 30))

self.XGB\_select\_predict\_result\_output\_path\_button.setObjectName(

"XGB\_select\_predict\_result\_output\_path\_button"

)

self.XGB\_select\_predict\_result\_output\_path\_button.clicked.connect(

self.XGB\_select\_predict\_result\_output\_path\_pushButton\_clicked

)

self.XGB\_img\_predict\_path\_label = QtWidgets.QLabel(self.XGB\_predict\_group)

self.XGB\_img\_predict\_path\_label.setGeometry(QtCore.QRect(10, 60, 80, 30))

self.XGB\_img\_predict\_path\_label.setObjectName("XGB\_img\_predict\_path\_label")

self.XGB\_img\_predict\_path = QtWidgets.QLineEdit(self.XGB\_predict\_group)

self.XGB\_img\_predict\_path.setGeometry(QtCore.QRect(90, 64, 235, 22))

self.XGB\_img\_predict\_path.setObjectName("XGB\_img\_predict\_path")

self.XGB\_img\_predict\_path.setReadOnly(True)

self.select\_XGB\_img\_predict\_path\_button = QtWidgets.QPushButton(self.XGB\_predict\_group)

self.select\_XGB\_img\_predict\_path\_button.setGeometry(QtCore.QRect(350, 60, 100, 30))

self.select\_XGB\_img\_predict\_path\_button.setObjectName("select\_XGB\_img\_predict\_path\_button")

self.select\_XGB\_img\_predict\_path\_button.clicked.connect(

self.select\_XGB\_img\_predict\_path\_pushButton\_clicked

)

self.XGB\_class\_num\_label\_2 = QtWidgets.QLabel(self.XGB\_predict\_group)

self.XGB\_class\_num\_label\_2.setGeometry(QtCore.QRect(10, 100, 80, 30))

self.XGB\_class\_num\_label\_2.setObjectName("XGB\_class\_num\_label\_2")

self.XGB\_class\_num\_path\_2 = QtWidgets.QLineEdit(self.XGB\_predict\_group)

self.XGB\_class\_num\_path\_2.setGeometry(QtCore.QRect(90, 104, 235, 22))

self.XGB\_class\_num\_path\_2.setObjectName("XGB\_class\_num\_path\_2")

self.XGB\_class\_num\_path\_2.setReadOnly(True)

self.select\_XGB\_class\_num\_button\_2 = QtWidgets.QPushButton(self.XGB\_predict\_group)

self.select\_XGB\_class\_num\_button\_2.setGeometry(QtCore.QRect(350, 100, 100, 30))

self.select\_XGB\_class\_num\_button\_2.setObjectName("select\_XGB\_class\_num\_button\_2")

self.select\_XGB\_class\_num\_button\_2.clicked.connect(self.select\_XGB\_class\_num\_button\_2\_clicked)

self.XGB\_predict\_button = QtWidgets.QPushButton(self.XGB\_predict\_group)

self.XGB\_predict\_button.setGeometry(QtCore.QRect(160, 180, 100, 30))

self.XGB\_predict\_button.setObjectName("XGB\_predict\_button")

self.XGB\_predict\_button.clicked.connect(self.XGB\_predict\_pushButton\_clicked)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "遥感影像机器学习地物分类工具箱"))

# RF\_tab\_page

self.tabWidgetPages.setTabText(

self.tabWidgetPages.indexOf(self.RF\_tab\_page), \_translate("MainWindow", "随机森林算法")

)

self.RF\_get\_samples\_group.setTitle(\_translate("MainWindow", "获取样本"))

self.RF\_img\_path\_label.setText(\_translate("MainWindow", "影像路径:"))

self.select\_RF\_img\_button.setText(\_translate("MainWindow", "选择路径"))

self.RF\_label\_path\_label.setText(\_translate("MainWindow", "标签路径:"))

self.select\_RF\_label\_button.setText(\_translate("MainWindow", "选择路径"))

self.RF\_class\_num\_label.setText(\_translate("MainWindow", "类别数目:"))

self.select\_RF\_class\_num\_button.setText(\_translate("MainWindow", "选择路径"))

self.RF\_output\_samples\_path\_label.setText(\_translate("MainWindow", "样本路径:"))

self.select\_RF\_output\_samples\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.RF\_get\_samples\_button.setText(\_translate("MainWindow", "获取样本"))

self.RF\_train\_group.setTitle(\_translate("MainWindow", "训练寻优参数"))

self.RF\_train\_button.setText(\_translate("MainWindow", "训练"))

self.RF\_val\_datasets\_size\_label.setText(\_translate("MainWindow", "验证集比例:"))

self.RF\_val\_datasets\_size.setText(\_translate("MainWindow", "0.3"))

self.RF\_n\_estimators\_label.setText(\_translate("MainWindow", "n\_estimators:"))

self.RF\_n\_estimators\_min.setText(\_translate("MainWindow", "50"))

self.RF\_n\_estimators\_interval.setText(\_translate("MainWindow", "1"))

self.RF\_n\_estimators\_max.setText(\_translate("MainWindow", "150"))

self.RF\_Min\_label.setText(\_translate("MainWindow", "Min"))

self.RF\_Max\_label.setText(\_translate("MainWindow", "Max"))

self.RF\_Interval\_label.setText(\_translate("MainWindow", "Interval"))

self.RF\_max\_depth\_min.setText(\_translate("MainWindow", "5"))

self.RF\_max\_depth\_max.setText(\_translate("MainWindow", "20"))

self.RF\_max\_depth\_interval.setText(\_translate("MainWindow", "1"))

self.RF\_max\_depth\_label.setText(\_translate("MainWindow", "max\_depth:"))

self.RF\_min\_samples\_leaf\_min.setText(\_translate("MainWindow", "1"))

self.RF\_min\_samples\_leaf\_max.setText(\_translate("MainWindow", "20"))

self.RF\_min\_samples\_leaf\_interval.setText(\_translate("MainWindow", "1"))

self.RF\_min\_samples\_leaf\_label.setText(\_translate("MainWindow", "min\_samples\_leaf:"))

self.RF\_min\_samples\_split\_min.setText(\_translate("MainWindow", "2"))

self.RF\_min\_samples\_split\_max.setText(\_translate("MainWindow", "10"))

self.RF\_min\_samples\_split\_interval.setText(\_translate("MainWindow", "1"))

self.RF\_min\_samples\_split\_label.setText(\_translate("MainWindow", "min\_samples\_split:"))

self.RF\_max\_features\_label.setText(\_translate("MainWindow", "max\_features:"))

self.sqrt\_cb.setText(\_translate("MainWindow", "sqrt"))

self.log2\_cb.setText(\_translate("MainWindow", "log2"))

self.None\_cb.setText(\_translate("MainWindow", "None"))

self.True\_cb.setText(\_translate("MainWindow", "True"))

self.bootstrap\_label.setText(\_translate("MainWindow", "bootstrap:"))

self.False\_cb.setText(\_translate("MainWindow", "False"))

self.RF\_train\_model\_save\_path\_label.setText(\_translate("MainWindow", "模型保存路径:"))

self.select\_RF\_train\_model\_save\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.RF\_predict\_group.setTitle(\_translate("MainWindow", "预测参数"))

self.select\_RF\_img\_predict\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.RF\_select\_predict\_result\_output\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.RF\_class\_num\_label\_2.setText(\_translate("MainWindow", "类别数目:"))

self.RF\_img\_predict\_path\_label.setText(\_translate("MainWindow", "影像路径:"))

self.select\_RF\_class\_num\_button\_2.setText(\_translate("MainWindow", "选择路径"))

self.RF\_model\_path\_label.setText(\_translate("MainWindow", "模型路径:"))

self.select\_RF\_model\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.RF\_predict\_button.setText(\_translate("MainWindow", "预测"))

self.RF\_predict\_output\_result\_path\_label.setText(\_translate("MainWindow", "输出路径:"))

# XGB\_tab\_page

self.tabWidgetPages.setTabText(

self.tabWidgetPages.indexOf(self.XGB\_tab\_page), \_translate("MainWindow", "XGBoost算法")

)

self.XGB\_get\_samples\_group.setTitle(\_translate("MainWindow", "获取样本"))

self.XGB\_img\_path\_label.setText(\_translate("MainWindow", "影像路径:"))

self.select\_XGB\_img\_button.setText(\_translate("MainWindow", "选择路径"))

self.XGB\_label\_path\_label.setText(\_translate("MainWindow", "标签路径:"))

self.select\_XGB\_lable\_button.setText(\_translate("MainWindow", "选择路径"))

self.XGB\_class\_num\_label.setText(\_translate("MainWindow", "类别数目:"))

self.select\_XGB\_class\_num\_button.setText(\_translate("MainWindow", "选择路径"))

self.XGB\_output\_samples\_path\_label.setText(\_translate("MainWindow", "样本路径:"))

self.select\_XGB\_output\_samples\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.XGB\_get\_samples\_button.setText(\_translate("MainWindow", "获取样本"))

self.XGB\_train\_group.setTitle(\_translate("MainWindow", "训练寻优参数"))

self.XGB\_train\_button.setText(\_translate("MainWindow", "训练"))

self.XGB\_val\_datasets\_size\_label.setText(\_translate("MainWindow", "验证集比例:"))

self.XGB\_val\_datasets\_size.setText(\_translate("MainWindow", "0.3"))

self.XGB\_n\_estimators\_label.setText(\_translate("MainWindow", "n\_estimators:"))

self.XGB\_n\_estimators\_min.setText(\_translate("MainWindow", "50"))

self.XGB\_n\_estimators\_max.setText(\_translate("MainWindow", "150"))

self.XGB\_n\_estimators\_interval.setText(\_translate("MainWindow", "1"))

self.XGB\_Min\_label.setText(\_translate("MainWindow", "Min"))

self.XGB\_Max\_label.setText(\_translate("MainWindow", "Max"))

self.XGB\_Interval\_label.setText(\_translate("MainWindow", "Interval"))

self.XGB\_max\_depth\_label.setText(\_translate("MainWindow", "max\_depth:"))

self.XGB\_max\_depth\_min.setText(\_translate("MainWindow", "5"))

self.XGB\_max\_depth\_max.setText(\_translate("MainWindow", "10"))

self.XGB\_max\_depth\_interval.setText(\_translate("MainWindow", "1"))

self.XGB\_learning\_rate\_label.setText(\_translate("MainWindow", "learning\_rate:"))

self.XGB\_learning\_rate\_min.setText(\_translate("MainWindow", "0.01"))

self.XGB\_learning\_rate\_Max.setText(\_translate("MainWindow", "0.1"))

self.XGB\_learning\_rate\_Interval.setText(\_translate("MainWindow", "0.01"))

self.XGB\_subsample\_label.setText(\_translate("MainWindow", "subsample:"))

self.XGB\_subsample\_min.setText(\_translate("MainWindow", "0.5"))

self.XGB\_subsample\_max.setText(\_translate("MainWindow", "1"))

self.XGB\_subsample\_interval.setText(\_translate("MainWindow", "0.05"))

self.XGB\_colsample\_bytree\_label.setText(\_translate("MainWindow", "colsample\_bytree:"))

self.XGB\_colsample\_bytree\_min.setText(\_translate("MainWindow", "0.5"))

self.XGB\_colsample\_bytree\_max.setText(\_translate("MainWindow", "1"))

self.XGB\_colsample\_bytree\_interval.setText(\_translate("MainWindow", "0.05"))

self.XGB\_gamma\_label.setText(\_translate("MainWindow", "gamma:"))

self.XGB\_gamma\_min.setText(\_translate("MainWindow", "0"))

self.XGB\_gamma\_max.setText(\_translate("MainWindow", "10"))

self.XGB\_gamma\_interval.setText(\_translate("MainWindow", "1"))

self.XGB\_reg\_alpha\_label.setText(\_translate("MainWindow", "reg\_alpha:"))

self.XGB\_reg\_alpha\_min.setText(\_translate("MainWindow", "0"))

self.XGB\_reg\_alpha\_max.setText(\_translate("MainWindow", "10"))

self.XGB\_reg\_alpha\_interval.setText(\_translate("MainWindow", "1"))

self.XGB\_reg\_lambda\_label.setText(\_translate("MainWindow", "reg\_lambda:"))

self.XGB\_reg\_lambda\_min.setText(\_translate("MainWindow", "0"))

self.XGB\_reg\_lambda\_max.setText(\_translate("MainWindow", "10"))

self.XGB\_reg\_lambda\_interval.setText(\_translate("MainWindow", "1"))

self.XGB\_min\_child\_weight\_label.setText(\_translate("MainWindow", "min\_child\_weight:"))

self.XGB\_min\_child\_weight\_interval.setText(\_translate("MainWindow", "1"))

self.XGB\_min\_child\_weight\_min.setText(\_translate("MainWindow", "1"))

self.XGB\_min\_child\_weight\_max.setText(\_translate("MainWindow", "10"))

self.XGB\_train\_model\_save\_path\_label.setText(\_translate("MainWindow", "模型保存路径:"))

self.select\_XGB\_train\_model\_save\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.XGB\_predict\_group.setTitle(\_translate("MainWindow", "预测参数"))

self.select\_XGB\_img\_predict\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.XGB\_select\_predict\_result\_output\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.XGB\_class\_num\_label\_2.setText(\_translate("MainWindow", "类别数目:"))

self.XGB\_img\_predict\_path\_label.setText(\_translate("MainWindow", "影像路径:"))

self.select\_XGB\_class\_num\_button\_2.setText(\_translate("MainWindow", "选择路径"))

self.XGB\_model\_path\_label.setText(\_translate("MainWindow", "模型路径:"))

self.select\_XGB\_model\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.XGB\_predict\_button.setText(\_translate("MainWindow", "预测"))

self.XGB\_predict\_output\_result\_path\_label.setText(\_translate("MainWindow", "输出路径:"))

# LGB\_tab\_page

self.tabWidgetPages.setTabText(

self.tabWidgetPages.indexOf(self.LGB\_tab\_page), \_translate("MainWindow", "LightGBM算法")

)

self.LGB\_get\_samples\_group.setTitle(\_translate("MainWindow", "获取样本"))

self.LGB\_img\_path\_label.setText(\_translate("MainWindow", "影像路径:"))

self.select\_LGB\_img\_button.setText(\_translate("MainWindow", "选择路径"))

self.LGB\_label\_path\_label.setText(\_translate("MainWindow", "标签路径:"))

self.select\_LGB\_lable\_button.setText(\_translate("MainWindow", "选择路径"))

self.LGB\_class\_num\_label.setText(\_translate("MainWindow", "类别数目:"))

self.select\_LGB\_class\_num\_button.setText(\_translate("MainWindow", "选择路径"))

self.LGB\_output\_samples\_path\_label.setText(\_translate("MainWindow", "样本路径:"))

self.select\_LGB\_output\_samples\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.LGB\_get\_samples\_button.setText(\_translate("MainWindow", "获取样本"))

self.LGB\_train\_group.setTitle(\_translate("MainWindow", "训练寻优参数"))

self.LGB\_train\_button.setText(\_translate("MainWindow", "训练"))

self.LGB\_val\_datasets\_size\_label.setText(\_translate("MainWindow", "验证集比例:"))

self.LGB\_val\_datasets\_size.setText(\_translate("MainWindow", "0.3"))

self.LGB\_n\_estimators\_label.setText(\_translate("MainWindow", "n\_estimators:"))

self.LGB\_n\_estimators\_min.setText(\_translate("MainWindow", "50"))

self.LGB\_n\_estimators\_max.setText(\_translate("MainWindow", "150"))

self.LGB\_n\_estimators\_interval.setText(\_translate("MainWindow", "1"))

self.LGB\_Min\_label.setText(\_translate("MainWindow", "Min"))

self.LGB\_Max\_label.setText(\_translate("MainWindow", "Max"))

self.LGB\_Interval\_label.setText(\_translate("MainWindow", "Interval"))

self.LGB\_max\_depth\_label.setText(\_translate("MainWindow", "max\_depth:"))

self.LGB\_max\_depth\_min.setText(\_translate("MainWindow", "5"))

self.LGB\_max\_depth\_max.setText(\_translate("MainWindow", "10"))

self.LGB\_max\_depth\_interval.setText(\_translate("MainWindow", "1"))

self.LGB\_learning\_rate\_label.setText(\_translate("MainWindow", "learning\_rate:"))

self.LGB\_learning\_rate\_min.setText(\_translate("MainWindow", "0.01"))

self.LGB\_learning\_rate\_Max.setText(\_translate("MainWindow", "0.1"))

self.LGB\_learning\_rate\_Interval.setText(\_translate("MainWindow", "0.01"))

self.LGB\_subsample\_label.setText(\_translate("MainWindow", "subsample:"))

self.LGB\_subsample\_min.setText(\_translate("MainWindow", "0.5"))

self.LGB\_subsample\_max.setText(\_translate("MainWindow", "1"))

self.LGB\_subsample\_interval.setText(\_translate("MainWindow", "0.05"))

self.LGB\_colsample\_bytree\_label.setText(\_translate("MainWindow", "colsample\_bytree:"))

self.LGB\_colsample\_bytree\_min.setText(\_translate("MainWindow", "0.5"))

self.LGB\_colsample\_bytree\_max.setText(\_translate("MainWindow", "1"))

self.LGB\_colsample\_bytree\_interval.setText(\_translate("MainWindow", "0.05"))

self.LGB\_reg\_alpha\_label.setText(\_translate("MainWindow", "reg\_alpha:"))

self.LGB\_reg\_alpha\_min.setText(\_translate("MainWindow", "0"))

self.LGB\_reg\_alpha\_max.setText(\_translate("MainWindow", "10"))

self.LGB\_reg\_alpha\_interval.setText(\_translate("MainWindow", "1"))

self.LGB\_reg\_lambda\_label.setText(\_translate("MainWindow", "reg\_lambda:"))

self.LGB\_reg\_lambda\_min.setText(\_translate("MainWindow", "0"))

self.LGB\_reg\_lambda\_max.setText(\_translate("MainWindow", "10"))

self.LGB\_reg\_lambda\_interval.setText(\_translate("MainWindow", "1"))

self.LGB\_min\_child\_weight\_label.setText(\_translate("MainWindow", "min\_child\_weight:"))

self.LGB\_min\_child\_weight\_interval.setText(\_translate("MainWindow", "1"))

self.LGB\_min\_child\_weight\_min.setText(\_translate("MainWindow", "1"))

self.LGB\_min\_child\_weight\_max.setText(\_translate("MainWindow", "10"))

self.LGB\_train\_model\_save\_path\_label.setText(\_translate("MainWindow", "模型保存路径:"))

self.select\_LGB\_train\_model\_save\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.LGB\_predict\_group.setTitle(\_translate("MainWindow", "预测参数"))

self.select\_LGB\_img\_predict\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.LGB\_select\_predict\_result\_output\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.LGB\_class\_num\_label\_2.setText(\_translate("MainWindow", "类别数目:"))

self.LGB\_img\_predict\_path\_label.setText(\_translate("MainWindow", "影像路径:"))

self.select\_LGB\_class\_num\_button\_2.setText(\_translate("MainWindow", "选择路径"))

self.LGB\_model\_path\_label.setText(\_translate("MainWindow", "模型路径:"))

self.select\_LGB\_model\_path\_button.setText(\_translate("MainWindow", "选择路径"))

self.LGB\_predict\_button.setText(\_translate("MainWindow", "预测"))

self.LGB\_predict\_output\_result\_path\_label.setText(\_translate("MainWindow", "输出路径:"))

self.log\_group.setTitle(\_translate("MainWindow", "日志"))

self.SaveLogButton.setText(\_translate("MainWindow", "保存日志"))

# 初始化log\_text\_browser内容为

self.log\_text\_browser.append("\*" \* 20 + " 欢迎使用遥感影像机器学习地物分类工具箱 " + "\*" \* 20)

self.log\_text\_browser.append("\*" \* 20 + " 当前页面为:随机森林 " + "\*" \* 20)

def tabWidgetPagesChanged(self, index):

if index == 0:

self.log\_text\_browser.append("\*" \* 20 + " 当前页面为:随机森林 " + "\*" \* 20)

elif index == 1:

self.log\_text\_browser.append("\*" \* 20 + " 当前页面为:LightGBM " + "\*" \* 20)

elif index == 2:

self.log\_text\_browser.append("\*" \* 20 + " 当前页面为:XGBoost " + "\*" \* 20)

# -\*- coding:utf-8 -\*-

# File name: utils.py

# description: 工具函数

# 设置全局变量

import os

os.environ["LOKY\_MAX\_CPU\_COUNT"] = "4"

# GUI所需库

from PyQt5 import QtCore, QtGui, QtWidgets

import time

# 运行所需库

from osgeo import gdal

from sklearn import datasets

from sklearn.ensemble import RandomForestClassifier

import numpy as np

from sklearn import model\_selection

import csv

import pandas as pd

import xgboost as xgb

import lightgbm as lgb

from sklearn.metrics import (

f1\_score,

accuracy\_score,

recall\_score,

f1\_score,

mean\_absolute\_error,

)

from sklearn.model\_selection import learning\_curve

from sklearn.cluster import KMeans

from hyperopt import hp, fmin, tpe, Trials, space\_eval, STATUS\_OK

# 模型保存所需要库

import pickle

# 绘图所需库

import scienceplots

import matplotlib

import matplotlib.pyplot as plt

from seaborn import heatmap

from warnings import simplefilter

from matplotlib.patches import Patch

from matplotlib.colors import ListedColormap

simplefilter(action="ignore", category=FutureWarning)

plt.style.use(["science", "grid", "no-latex"])

# 设置中文字体

plt.rcParams["font.sans-serif"] = ["SimHei"]

# 设置负号正常显示

plt.rcParams["axes.unicode\_minus"] = False

plt.figure()

# ===================== get\_sample.py =====================

def printLog(msg, blog\_textBrowser):

try:

blog\_textBrowser.append(msg)

except Exception as e:

QtWidgets.QMessageBox.critical(None, "Error", str(e), QtWidgets.QMessageBox.Ok)

def get\_info(file\_name, msg):

def read\_tif(file\_name):

dataset = gdal.Open(file\_name)

if dataset is None:

msg.emit(f"{file\_name}文件无法打开")

return dataset

dataset = read\_tif(file\_name)

msg.emit("Dataset Name:" + file\_name)

msg.emit("Driver: " + dataset.GetDriver().ShortName + "/" + dataset.GetDriver().LongName)

msg.emit(

"Size is："

+ "W"

+ str(dataset.RasterXSize)

+ "\*"

+ "H"

+ str(dataset.RasterYSize)

+ "\*"

+ "B"

+ str(dataset.RasterCount)

)

msg.emit("Projection is：" + dataset.GetProjection())

width, height, bands = dataset.RasterXSize, dataset.RasterYSize, dataset.RasterCount

geotrans = dataset.GetGeoTransform()

if geotrans is not None:

msg.emit(f"Origin = ({geotrans[0]}, {geotrans[3]})")

msg.emit(f"Pixel Size = ({geotrans[1]}, {-geotrans[5]})")

\_data\_ = dataset.ReadAsArray(0, 0, width, height)

return width, height, bands, geotrans, \_data\_, dataset.GetProjection()

def ReadNumClass(numclass\_path):

# 规定一个字典

num\_class = {}

if numclass\_path.endswith(".txt"):

# 文件第一行为表头，{类别 数量}

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

# 跳过第一行

file\_read\_obj.readline()

# 读取其余行

for line in file\_read\_obj.readlines():

# 去除空格

line = line.strip()

# 以空格分割

line = line.split(",")

# 以字典形式存储

num\_class[int(line[0])] = int(line[1])

return num\_class

elif numclass\_path.endswith(".csv"):

# 文件第一行为表头，{类别 数量}

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

csv\_read\_obj = csv.reader(file\_read\_obj)

# 跳过第一行

next(csv\_read\_obj)

# 读取其余行

for line in csv\_read\_obj:

# 以字典形式存储

num\_class[int(line[0])] = int(line[1])

return num\_class

def adjustLocationByGeoTransform(img, label, img\_geotrans, label\_geotrans, msg):

# 判断两个影像的坐标是否一致

if img\_geotrans == label\_geotrans:

return img

msg.emit("Warning: \_img\_ and \_label\_ are not in the same location")

try:

# 获取\_label\_图像的地理坐标范围

label\_min\_x, label\_max\_y = gdal.ApplyGeoTransform(label\_geotrans, 0, 0)

label\_max\_x, label\_min\_y = gdal.ApplyGeoTransform(label\_geotrans, label.shape[1], label.shape[0])

# 获取\_img\_图像的地理坐标范围

img\_min\_x, img\_max\_y = gdal.ApplyGeoTransform(img\_geotrans, 0, 0)

img\_max\_x, img\_min\_y = gdal.ApplyGeoTransform(img\_geotrans, img.shape[2], img.shape[1])

except Exception as e:

msg.emit(f"Error calculating geo transforms: {e}")

return img

# 确保\_label\_包含在\_img\_内

if (

label\_min\_x >= img\_min\_x

and label\_max\_x <= img\_max\_x

and label\_min\_y >= img\_min\_y

and label\_max\_y <= img\_max\_y

):

# 计算在\_img\_中对应\_label\_的区域

label\_start\_col = int((label\_min\_x - img\_geotrans[0]) / img\_geotrans[1])

label\_start\_row = int((img\_max\_y - label\_max\_y) / abs(img\_geotrans[5]))

label\_end\_col = int((label\_max\_x - img\_geotrans[0]) / img\_geotrans[1])

label\_end\_row = int((img\_max\_y - label\_min\_y) / abs(img\_geotrans[5]))

# 提取对应区域的\_img\_

adjusted\_img = img[

:,

np.s\_[label\_start\_row:label\_end\_row],

np.s\_[label\_start\_col:label\_end\_col],

]

return adjusted\_img

else:

msg.emit("Error: \_label\_ is not in \_img\_")

return img

def WriteSample(\_img\_, \_label\_, sample\_path, class\_num, msg):

# 读取\_label\_数据中的唯一值确定类别

classes = np.unique(\_label\_)

msg.emit("当前标签数组所有值：")

for i, cls in enumerate(classes):

if cls != 0:

msg.emit("Value:" + str(cls) + " Class:" + str(i))

else:

msg.emit("Value:" + str(cls) + " Class:" + "Nan")

msg.emit("目标类别及对应数量：")

for key, value in class\_num.items():

msg.emit("Class:" + str(key) + " Num:" + str(value))

# 提前检查文件存在性，存在就删除

if os.path.exists(sample\_path):

os.remove(sample\_path)

# 统计每个类别的样本数量

class\_counts = {cls: 0 for cls in classes}

# 读取\_img\_的波段数

bands = \_img\_.shape[0]

# 打开文件，如果文件不存在，就会自动创建,

# 如果sample\_path是csv文件，就用csv模块打开，如果是txt文件，就用open打开

if sample\_path.endswith(".csv"):

file\_write\_obj = open(sample\_path, "a+", newline="", encoding="utf-8")

csv\_write\_obj = csv.writer(file\_write\_obj)

# 将波段数以及类别写入文件

csv\_write\_obj.writerow(list(str("Band") + str(i) for i in range(bands)) + ["class"])

for i, row in enumerate(\_label\_):

for j, \_class in enumerate(row):

if \_class != 0:

# 判断每个类别的样本数量是否达到要求

if class\_counts[\_class] >= class\_num[\_class]:

break

else:

# 获取该像元的所有波段像素值

pixel\_values = [\_img\_[k, i, j] for k in range(\_img\_.shape[0])]

# 将像素值和类别写入文件

csv\_write\_obj.writerow(pixel\_values + [\_class])

# 统计每个类别的样本数量

class\_counts[\_class] += 1

elif sample\_path.endswith(".txt"):

with open(sample\_path, "a+", encoding="utf-8") as file\_write\_obj:

# 将波段数以及类别写入文件

file\_write\_obj.writelines(

",".join(

map(

str,

list(str("Band") + str(i) for i in range(bands)) + ["class"],

)

)

+ "\n"

)

for i, row in enumerate(\_label\_):

for j, \_class in enumerate(row):

if \_class != 0:

# 判断每个类别的样本数量是否达到要求

if class\_counts[\_class] >= class\_num[\_class]:

break

else:

# 获取该像元的所有波段像素值

pixel\_values = [\_img\_[k, i, j] for k in range(\_img\_.shape[0])]

# 将像素值和类别写入文件

line = ",".join(map(str, pixel\_values + [\_class]))

file\_write\_obj.writelines(line + "\n")

# 统计每个类别的样本数量

class\_counts[\_class] += 1

def get\_samples(img\_path, label\_path, sample\_path, numclass\_path, msg):

try:

start\_time = time.time()

# 判断img\_path和label\_path是否为栅格数据

if not img\_path.endswith(".tif") and not img\_path.endswith(".tiff") and not img\_path.endswith(".pix"):

msg.emit("Error: img\_path is not a raster data")

return

if (

not label\_path.endswith(".tif")

and not label\_path.endswith(".tiff")

and not label\_path.endswith(".pix")

):

msg.emit("Error: label\_path is not a raster data")

return

# 读取影像数据

try:

msg.emit("\*" \* 10 + " 读取影像数据 " + "\*" \* 10)

(

img\_width,

img\_height,

img\_bands,

img\_geotrans,

\_img\_,

img\_projection,

) = get\_info(img\_path, msg)

msg.emit("原始影像数据形状：" + str(\_img\_.shape))

msg.emit("\*" \* 10 + " 读取完毕 " + "\*" \* 10)

except Exception as e:

msg.emit("Error: " + str(e) + " 读取影像数据失败，检查数据!!!")

return False

# 读取标签数据

try:

msg.emit("\*" \* 10 + " 读取标签数据 " + "\*" \* 10)

# 读取标签数据

(

label\_width,

label\_height,

label\_bands,

label\_geotrans,

\_label\_,

label\_projection,

) = get\_info(label\_path, msg)

msg.emit("原始标签数据形状：" + str(\_label\_.shape))

msg.emit("\*" \* 10 + " 读取完毕 " + "\*" \* 10)

except Exception as e:

msg.emit("Error: " + str(e) + " 读取标签数据失败，检查数据!!!")

return False

# 调整label和img的大小一致

msg.emit("\*" \* 10 + " 调整数据 " + "\*" \* 10)

\_img\_ = adjustLocationByGeoTransform(\_img\_, \_label\_, img\_geotrans, label\_geotrans, msg)

msg.emit("调整后影像数据形状：" + str(\_img\_.shape))

msg.emit("\*" \* 10 + " 调整完毕 " + "\*" \* 10)

# 写入数据

try:

msg.emit("\*" \* 10 + " 写入数据 " + "\*" \* 10)

class\_num = ReadNumClass(numclass\_path)

WriteSample(\_img\_, \_label\_, sample\_path, class\_num, msg)

msg.emit("\*" \* 10 + " 写入完毕 " + "\*" \* 10)

except Exception as e:

msg.emit("Error: " + str(e) + " 写入数据失败，检查数据!!!")

return False

end\_time = time.time()

msg.emit("\*" \* 10 + " 样本获取运行时间: " + str(end\_time - start\_time) + " 秒 " + "\*" \* 10)

return True

except Exception as e:

msg.emit("Error: " + str(e))

return False

# ===================== RF\_train.py =====================

# 定义字典，便于解析样本数据集txt

def label\_dict(s):

# 如果是数字，直接返回

if s.isdigit():

return int(s)

# 如果是字符串，返回对应的数字

else:

it = {"Vegetation": 1, "Non-Vegetation": 2}

return it[s]

def PlotHeatmap(corr):

# 保留下三角矩阵

mask = np.zeros\_like(corr)

mask[np.triu\_indices\_from(mask)] = False

# 绘制热力图

heatmap(corr, mask=mask, cmap="RdBu\_r", annot=True, fmt=".2f")

plt.savefig("corr.png", dpi=500)

# plt.show()

def Split\_train\_test\_dataset(x, y, val\_size=0.2, random\_state=1):

# 分割训练集和验证集

train\_data, test\_data, train\_label, test\_label = model\_selection.train\_test\_split(

x,

y,

random\_state=random\_state,

test\_size=val\_size,

)

# 输出训练集和测试集的样本数量和特征数量

# print("训练集样本数量：", train\_data.shape[0])

# print("测试集样本数量：", test\_data.shape[0])

return train\_data, test\_data, train\_label, test\_label

def plot\_learning\_curve(estimator, title, X, y, model\_name, ylim=None, cv=None, n\_jobs=1):

plt.title(title)

if ylim is not None:

plt.ylim(\*ylim)

plt.xlabel("Training sample size")

plt.ylabel("Score")

train\_sizes, train\_scores, test\_scores = learning\_curve(estimator, X, y, cv=cv, n\_jobs=n\_jobs)

train\_scores\_mean = np.mean(train\_scores, axis=1)

train\_scores\_std = np.std(train\_scores, axis=1)

test\_scores\_mean = np.mean(test\_scores, axis=1) # 计算测试集上的平均得分

test\_scores\_std = np.std(test\_scores, axis=1) # 计算测试集上的标准差

plt.grid()

plt.fill\_between(

train\_sizes,

train\_scores\_mean - train\_scores\_std,

train\_scores\_mean + train\_scores\_std,

alpha=0.1,

color="g",

)

plt.fill\_between(

train\_sizes,

test\_scores\_mean - test\_scores\_std,

test\_scores\_mean + test\_scores\_std,

alpha=0.1,

color="b",

)

plt.plot(

train\_sizes,

train\_scores\_mean,

"o-",

color="g",

label="the score on the training set",

)

plt.plot(

train\_sizes,

test\_scores\_mean,

"o-",

color="b",

label="the score on the validation set",

)

plt.legend(loc="lower right")

plt.savefig(f"learning curve of {model\_name}.png", dpi=800)

# 清除图像

plt.clf()

# plt.savefig("学习曲线\_XBM.png", dpi=800)

def PlotImportance\_RF(importances):

indices = np.argsort(importances)[::-1]

plt.title("Feature importances by RandomTreeClassifier")

plt.bar(range(len(indices)), importances[indices], color="darkorange", align="center")

plt.xticks(range(len(indices)), indices)

plt.xlim([-1, len(indices)])

plt.xlabel("Feature")

plt.ylabel("Importance")

plt.savefig("Feature importances by RandomTreeClassifier.png", dpi=500)

# 清楚图像

plt.clf()

# plt.show()

def SavePickle(classifier, SavePath):

# 以二进制的方式打开文件：

file = open(SavePath, "wb")

# 将模型写入文件：

pickle.dump(classifier, file)

# 最后关闭文件：

file.close()

# ===================== RF\_predict.py =====================

def Predict\_RF\_func(model\_path, \_img\_):

# 以读二进制的方式打开文件

file = open(model\_path, "rb")

# 把模型从文件中读取出来

rf\_model = pickle.load(file)

# 关闭文件

file.close()

# 用读入的模型进行预测

# 在与测试前要调整一下数据的格式

data = np.zeros((\_img\_.shape[0], \_img\_.shape[1] \* \_img\_.shape[2]))

# 使用tqdm显示进度

for i in range(\_img\_.shape[0]):

data[i] = \_img\_[i].flatten()

data = data.swapaxes(0, 1)

# 对调整好格式的数据进行预测

pred = rf\_model.predict(data)

# 同样地，我们对预测好的数据调整为我们图像的格式

pred = pred.reshape(\_img\_.shape[1], \_img\_.shape[2])

pred = pred.astype(np.uint8)

return pred

def PlotPredictResult(pred, model, classPath=r"data\\ClassDefine.txt"):

labels = []

colors = []

# 设置颜色和标签

with open(classPath, "r") as file\_read\_obj:

# 跳过第一行

file\_read\_obj.readline()

# 读取其余行

for line in file\_read\_obj.readlines():

# 去除空格

line = line.strip()

# 以空格分割

line = line.split(",")

labels.append(line[2])

colors.append(line[3])

# 值为1的像素点为绿色，值为2的像素点为白色

cmap = ListedColormap(colors)

# 为图例添加标签和样式

legend\_elements = [Patch(color=colors[i], label=labels[i]) for i in range(len(colors))]

# 添加图例

plt.legend(handles=legend\_elements, loc="best")

plt.title("预测结果")

# 保存

plt.imshow(pred, cmap=cmap)

plt.savefig(f"PredictResult\_{model}.png", dpi=500)

# plt.show()

# ===================== K\_Means\_Classify.py =====================

def KMeansExecute(data, K, img\_width, img\_height, n\_init=10, random\_state=0):

# K-Means

kmeans = KMeans(n\_clusters=K, n\_init=n\_init, random\_state=random\_state)

kmeans.fit(data)

idx = kmeans.labels\_

labels = idx.reshape(img\_width, img\_height)

return labels

def PlotKMeansResult(labels):

# 显示聚类结果

plt.imshow(labels, cmap="rainbow")

# plt.show()

# ===================== SVM\_Classify.py =====================

def Predict\_SVM(model\_path, \_img\_):

# 以读二进制的方式打开文件

file = open(model\_path, "rb")

# 把模型从文件中读取出来

svm\_model = pickle.load(file)

# 关闭文件

file.close()

# 用读入的模型进行预测

# 在与测试前要调整一下数据的格式

data = np.zeros((\_img\_.shape[0], \_img\_.shape[1] \* \_img\_.shape[2]))

# 使用tqdm显示进度

for i in range(\_img\_.shape[0]):

data[i] = \_img\_[i].flatten()

data = data.swapaxes(0, 1)

# 对调整好格式的数据进行预测

pred = svm\_model.predict(data)

# 同样地，我们对预测好的数据调整为我们图像的格式

pred = pred.reshape(\_img\_.shape[1], \_img\_.shape[2])

pred = pred.astype(np.uint8)

return pred

# 保存tif文件函数

def writeTiff(im\_data, im\_geotrans, im\_proj, path):

if "int8" in im\_data.dtype.name:

datatype = gdal.GDT\_Byte

elif "int16" in im\_data.dtype.name:

datatype = gdal.GDT\_UInt16

else:

datatype = gdal.GDT\_Float32

if len(im\_data.shape) == 3:

im\_bands, im\_height, im\_width = im\_data.shape

elif len(im\_data.shape) == 2:

im\_data = np.array([im\_data])

im\_bands, im\_height, im\_width = im\_data.shape

# 创建文件

driver = gdal.GetDriverByName("GTiff")

dataset = driver.Create(path, int(im\_width), int(im\_height), int(im\_bands), datatype)

if dataset != None:

dataset.SetGeoTransform(im\_geotrans) # 写入仿射变换参数

dataset.SetProjection(im\_proj) # 写入投影

for i in range(im\_bands):

dataset.GetRasterBand(i + 1).WriteArray(im\_data[i])

del dataset

# ===================== XGB\_train.py =====================

def CheckY(y):

# XGBoost要求标签从0开始

if y.min() == 1:

y = y - 1

return y

def PlotImportance\_XGB(importances):

indices = np.argsort(importances)[::-1]

# print("Feature ranking:")

# for f in range(len(indices)):

# print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

plt.title("Feature importances by XGBoost")

plt.bar(range(len(indices)), importances[indices], color="darkorange", align="center")

plt.xticks(range(len(indices)), indices)

plt.xlim([-1, len(indices)])

plt.xlabel("Feature")

plt.ylabel("Importance")

plt.legend().set\_visible(False)

plt.savefig("Feature importances by XGBoost.png", dpi=500)

# plt.show()

def Predict\_XGB\_func(model\_path, \_img\_):

# 以读二进制的方式打开文件

file = open(model\_path, "rb")

# 把模型从文件中读取出来

xgb\_model = pickle.load(file)

# 关闭文件

file.close()

# 用读入的模型进行预测

# 在与测试前要调整一下数据的格式

data = np.zeros((\_img\_.shape[0], \_img\_.shape[1] \* \_img\_.shape[2]))

# 使用tqdm显示进度

for i in range(\_img\_.shape[0]):

data[i] = \_img\_[i].flatten()

data = data.swapaxes(0, 1)

# 对调整好格式的数据进行预测

pred = xgb\_model.predict(data)

# 同样地，我们对预测好的数据调整为我们图像的格式

pred = pred.reshape(\_img\_.shape[1], \_img\_.shape[2])

pred = pred.astype(np.uint8)

return pred

# ===================== lightGBM.py =====================

def PlotImportance\_LGBM(importances, feature\_names):

indices = np.argsort(importances)[::-1]

# print("Feature ranking:")

# for f in range(len(indices)):

# print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

plt.title("Feature importances by LightGBM")

plt.bar(range(len(indices)), importances[indices], color="darkorange", align="center")

plt.xticks(range(len(indices)), list(np.array(feature\_names)[indices]), rotation="vertical")

plt.xlim([-1, len(indices)])

plt.xlabel("Feature")

plt.ylabel("Importance")

plt.legend().set\_visible(False)

plt.savefig("Feature importances by LightGBM.png", dpi=500)

# plt.show()

def Predict\_LGBM\_func(model\_path, \_img\_):

# 以读二进制的方式打开文件

file = open(model\_path, "rb")

# 把模型从文件中读取出来

lgbm\_model = pickle.load(file)

# 关闭文件

file.close()

# 用读入的模型进行预测

# 在与测试前要调整一下数据的格式

data = np.zeros((\_img\_.shape[0], \_img\_.shape[1] \* \_img\_.shape[2]))

# 使用tqdm显示进度

for i in range(\_img\_.shape[0]):

data[i] = \_img\_[i].flatten()

data = data.swapaxes(0, 1)

# 对调整好格式的数据进行预测

pred = lgbm\_model.predict(data)

# 同样地，我们对预测好的数据调整为我们图像的格式

pred = pred.reshape(\_img\_.shape[1], \_img\_.shape[2])

pred = pred.astype(np.uint8)

return pred

# ===================== ui\_functions.py =====================

# 选择的遥感影像数据路径

def selectImgPath(lineEdit, log\_textBrowser):

try:

img, \_ = QtWidgets.QFileDialog.getOpenFileName(

None, "Select Image", "", "Image Files(\*.tif \*.tiff \*.pix)"

)

if img:

lineEdit.setText(str(img))

printLog("选择的遥感影像数据路径:" + str(img), log\_textBrowser)

except Exception as e:

QtWidgets.QMessageBox.critical(None, "Error", str(e), QtWidgets.QMessageBox.Ok)

# 选择的标签数据路径

def selectLabelPath(lineEdit, log\_textBrowser):

try:

label, \_ = QtWidgets.QFileDialog.getOpenFileName(

None, "Select Label", "", "Image Files(\*.tif \*.tiff \*.pix)"

)

if label:

lineEdit.setText(str(label))

printLog("选择的标签数据路径:" + str(label), log\_textBrowser)

except Exception as e:

QtWidgets.QMessageBox.critical(None, "Error", str(e), QtWidgets.QMessageBox.Ok)

# 选择的输出样本数据路径

def selectOutSamplePath(lineEdit, log\_textBrowser):

try:

# 新建文件的输出路径文件名

out\_sample, \_ = QtWidgets.QFileDialog.getSaveFileName(

None, "Save File", "", "Image Files(\*.csv \*.txt)"

)

if out\_sample:

lineEdit.setText(str(out\_sample))

printLog("选择的输出样本数据路径:" + str(out\_sample), log\_textBrowser)

except Exception as e:

QtWidgets.QMessageBox.critical(None, "Error", str(e), QtWidgets.QMessageBox.Ok)

# 选择的NumClass数据路径

def selectNumClassPath(lineEdit, log\_textBrowser):

try:

NumClassPath, \_ = QtWidgets.QFileDialog.getOpenFileName(

None, "Select NumClass", "", "Image Files(\*.csv \*.txt)"

)

if NumClassPath:

lineEdit.setText(str(NumClassPath))

printLog("选择的NumClass数据路径:" + str(NumClassPath), log\_textBrowser)

except Exception as e:

QtWidgets.QMessageBox.critical(None, "Error", str(e), QtWidgets.QMessageBox.Ok)

# 选择的模型保存路径

def selectSaveModelPath(lineEdit, log\_textBrowser):

try:

model, \_ = QtWidgets.QFileDialog.getSaveFileName(

None, "Save Model", "", "Model Files(\*.pkl \*.pickle)"

)

if model:

lineEdit.setText(str(model))

printLog("选择的模型保存路径:" + str(model), log\_textBrowser)

except Exception as e:

QtWidgets.QMessageBox.critical(None, "Error", str(e), QtWidgets.QMessageBox.Ok)

# 选择的模型路径

def selectModelPath(lineEdit, log\_textBrowser):

try:

model, \_ = QtWidgets.QFileDialog.getOpenFileName(

None, "Select Model", "", "Model Files(\*.pkl \*.pickle)"

)

if model:

lineEdit.setText(str(model))

printLog("选择的模型路径:" + str(model), log\_textBrowser)

except Exception as e:

QtWidgets.QMessageBox.critical(None, "Error", str(e), QtWidgets.QMessageBox.Ok)

# 选择的保存影像数据路径

def selectSaveImgPath(lineEdit, log\_textBrowser):

try:

img, \_ = QtWidgets.QFileDialog.getSaveFileName(

None, "Save Image", "", "Image Files(\*.tif \*.tiff \*.pix)"

)

if img:

lineEdit.setText(str(img))

printLog("选择的保存影像数据路径:" + str(img), log\_textBrowser)

except Exception as e:

QtWidgets.QMessageBox.critical(None, "Error", str(e), QtWidgets.QMessageBox.Ok)

# get samples clicked

# 获取样本

def GetSamplesF(

img\_lineEdit,

label\_lineEdit,

out\_sample\_lineEdit,

numclass\_lineEdit,

msg,

):

try:

img\_path = img\_lineEdit

label\_path = label\_lineEdit

sample\_path = out\_sample\_lineEdit

numclass\_path = numclass\_lineEdit

if os.path.exists(img\_path) and os.path.exists(label\_path):

msg.emit("\*" \* 10 + " 开始获取样本数据 " + "\*" \* 10)

res = get\_samples(img\_path, label\_path, sample\_path, numclass\_path, msg)

if res:

msg.emit("\*" \* 10 + " 样本数据获取成功! " + "\*" \* 10)

else:

msg.emit("\*" \* 10 + " 样本数据获取失败! " + "\*" \* 10)

else:

msg.emit("\*" \* 10 + " 请检查输入路径是否正确 " + "\*" \* 10)

except Exception as e:

QtWidgets.QMessageBox.critical(None, "Error", str(e), QtWidgets.QMessageBox.Ok)

def float\_range(start, end, step):

return [start + step \* i for i in range(int((end - start) / step))]

# -\*- coding: utf-8 -\*-

# File Name：classes.py

# description: 线程类

from utils import \*

from train import RF\_train, XGB\_train, LightGBM\_train

from predict import RF\_Predict, XGB\_Predict, LightGBM\_Predict

# 获取样本数据的线程

class GetSamplesFThread(QtCore.QThread):

error = QtCore.pyqtSignal(str)

msg = QtCore.pyqtSignal(str)

def \_\_init\_\_(self, img\_path, mark\_path, out\_sample\_path, numclass\_path):

super(GetSamplesFThread, self).\_\_init\_\_()

self.img\_path = img\_path

self.mark\_path = mark\_path

self.out\_sample\_path = out\_sample\_path

self.numclass\_path = numclass\_path

def run(self):

try:

GetSamplesF(self.img\_path, self.mark\_path, self.out\_sample\_path, self.numclass\_path, self.msg)

except Exception as e:

self.error.emit(str(e))

class Train(QtCore.QThread):

error = QtCore.pyqtSignal(str)

msg = QtCore.pyqtSignal(str)

def \_\_init\_\_(self, sample\_path, SavePath, space, test\_size, model):

super(Train, self).\_\_init\_\_()

self.sample\_path = sample\_path

self.SavePath = SavePath

self.test\_size = test\_size

self.space = space

self.model = model

def run(self):

try:

if self.model == "LightGBM":

LightGBM\_train(

self.sample\_path,

self.SavePath,

self.space,

self.msg,

test\_size=self.test\_size,

)

elif self.model == "XGBoost":

XGB\_train(

self.sample\_path,

self.SavePath,

self.space,

self.msg,

test\_size=self.test\_size,

)

elif self.model == "RandomForest":

RF\_train(

self.sample\_path,

self.SavePath,

self.space,

self.msg,

test\_size=self.test\_size,

)

except Exception as e:

self.error.emit(str(e))

class Predict(QtCore.QThread):

error = QtCore.pyqtSignal(str)

msg = QtCore.pyqtSignal(str)

def \_\_init\_\_(self, RF\_model\_path, img\_path, save\_path, class\_path, model):

super(Predict, self).\_\_init\_\_()

self.RF\_model\_path = RF\_model\_path

self.img\_path = img\_path

self.save\_path = save\_path

self.class\_path = class\_path

self.model = model

def run(self):

try:

if self.model == "RandomForest":

RF\_Predict(self.RF\_model\_path, self.img\_path, self.save\_path, self.class\_path, self.msg)

elif self.model == "XGBoost":

XGB\_Predict(self.RF\_model\_path, self.img\_path, self.save\_path, self.class\_path, self.msg)

elif self.model == "LightGBM":

LightGBM\_Predict(self.RF\_model\_path, self.img\_path, self.save\_path, self.class\_path, self.msg)

except Exception as e:

self.error.emit(str(e))

# -\*- coding: utf-8 -\*-

# File Name：train.py

# description: 训练模型函数

from utils import \*

import numpy as np

def RF\_train(sample\_path, SavePath, space, msg, test\_size=0.5):

msg.emit("\*" \* 30 + " 开始训练 " + "\*" \* 30)

msg.emit("模型：RandomForest")

msg.emit("样本路径：" + sample\_path)

msg.emit("保存路径：" + SavePath)

msg.emit("验证集比例：" + str(test\_size))

msg.emit("\*" \* 30 + " 获取样本中... " + "\*" \* 30)

# 读取第一行以获取列名

if sample\_path.endswith(".txt"):

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

header\_line = file.readline().strip()

elif sample\_path.endswith(".csv"):

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

header\_line = file.readline().strip()

# 获取列名

column\_names = header\_line.split(",")

# 重新定义converters字典，跳过第一行（表头）

converters = {i: label\_dict if column\_names[i] == "class" else int for i in range(len(column\_names))}

# 从第二行开始读取数据

data = np.loadtxt(sample\_path, dtype=int, delimiter=",", skiprows=1, converters=converters)

x, y = np.split(data, indices\_or\_sections=(len(column\_names) - 1,), axis=1) # x为数据，y为标签

# msg.emit(x.shape, y.shape)

# ------------------------------- change -------------------------------

# 计算16个波段的相关系数矩阵，上三角矩阵

# PlotHeatmap(np.corrcoef(x.T))

msg.emit("\*" \* 30 + " 获取样本完成 " + "\*" \* 30)

msg.emit("\*" \* 30 + " 分割训练集和验证集 " + "\*" \* 30)

# 分割训练集和验证集

train\_data, test\_data, train\_label, test\_label = Split\_train\_test\_dataset(

x, y, val\_size=test\_size, random\_state=42

)

msg.emit("\*" \* 30 + " 分割完成 " + "\*" \* 30)

msg.emit(f"训练集样本数：{train\_data.shape[0]}")

msg.emit(f"验证集样本数：{test\_data.shape[0]}")

msg.emit(f"样本特征数：{train\_data.shape[1]}")

msg.emit(f"类别数：{len(np.unique(train\_label))}")

msg.emit("\*" \* 30 + " 正在进行超参数优化... " + "\*" \* 30)

# 超参数优化

def HyperOptimize(train\_data, train\_label, test\_data, test\_label, max\_evals=100):

a = []

b = []

def hyperopt\_train\_test(params):

skf = model\_selection.StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

f1\_scores = []

for train\_index, val\_index in skf.split(train\_data, train\_label):

X\_train, X\_val = train\_data[train\_index], train\_data[val\_index]

y\_train, y\_val = train\_label[train\_index], train\_label[val\_index]

clf = RandomForestClassifier(\*\*params, random\_state=42)

clf.fit(X\_train, y\_train.ravel())

pred\_label = clf.predict(X\_val)

f1 = f1\_score(y\_val, pred\_label, average="macro")

f1\_scores.append(f1)

avg\_f1 = sum(f1\_scores) / len(f1\_scores)

a.append(avg\_f1)

b.append(params)

return -avg\_f1 # 注意返回负的 F1 score，因为hyperopt是最小化目标

def f(params):

acc = hyperopt\_train\_test(params)

return {"loss": acc, "status": STATUS\_OK}

trials = Trials()

fmin(f, space, algo=tpe.suggest, max\_evals=max\_evals, trials=trials, show\_progressbar=False)

max\_f1\_dict = b[a.index(max(a))]

msg.emit("最佳超参数：" + str(max\_f1\_dict))

return max\_f1\_dict

# 超参数优化

max\_f1\_dict = HyperOptimize(train\_data, train\_label, test\_data, test\_label, max\_evals=20)

msg.emit("\*" \* 30 + " 超参数优化完成 " + "\*" \* 30)

msg.emit("\*" \* 30 + " 训练中... " + "\*" \* 30)

# 训练模型

classifier = RandomForestClassifier(\*\*max\_f1\_dict, random\_state=42)

classifier.fit(train\_data, train\_label.ravel())

msg.emit("\*" \* 30 + " 超参数优化完成 " + "\*" \* 30)

msg.emit("\*" \* 30 + " 训练结果 " + "\*" \* 30)

msg.emit("训练集精度：" + str(classifier.score(train\_data, train\_label)))

msg.emit("验证集精度：" + str(classifier.score(test\_data, test\_label)))

# 预测输出

predict = classifier.predict(test\_data)

msg.emit("平均绝对误差：" + str(mean\_absolute\_error(test\_label, predict)))

msg.emit("召回率：" + str(recall\_score(test\_label, predict, average="macro")))

msg.emit("准确率：" + str(accuracy\_score(test\_label, predict)))

msg.emit("F1值：" + str(f1\_score(test\_label, predict, average="macro")))

# ------------------------------- change -------------------------------

# 绘制学习曲线

# 交叉验证参数

shuffle = model\_selection.ShuffleSplit(n\_splits=10, test\_size=0.5, random\_state=42)

plot\_learning\_curve(

classifier,

"RandomForest Learning Curve",

x,

y.ravel(),

model\_name="RandomForest",

# ylim=(0.7, 1.01),

cv=shuffle,

)

# ------------------------------- change -------------------------------

# 绘制特征重要性

PlotImportance\_RF(classifier.feature\_importances\_)

msg.emit("\*" \* 5 + " 学习曲线已生成 " + "\*" \* 5)

# 保存模型

SavePickle(classifier, SavePath)

msg.emit("\*" \* 5 + f" 训练完成,模型已保存至 {SavePath} " + "\*" \* 5)

def XGB\_train(sample\_path, SavePath, space, msg, test\_size=0.5):

msg.emit("\*" \* 30 + " 开始训练 " + "\*" \* 30)

msg.emit("模型：XGB\_train")

msg.emit("样本路径：" + sample\_path)

msg.emit("保存路径：" + SavePath)

msg.emit("验证集比例：" + str(test\_size))

msg.emit("\*" \* 30 + " 获取样本中... " + "\*" \* 30)

# 读取第一行以获取列名

if sample\_path.endswith(".txt"):

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

header\_line = file.readline().strip()

elif sample\_path.endswith(".csv"):

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

header\_line = file.readline().strip()

# 获取列名

column\_names = header\_line.split(",")

# 重新定义converters字典，跳过第一行（表头）

converters = {i: label\_dict if column\_names[i] == "class" else int for i in range(len(column\_names))}

# 从第二行开始读取数据

data = np.loadtxt(sample\_path, dtype=int, delimiter=",", skiprows=1, converters=converters)

x, y = np.split(data, indices\_or\_sections=(len(column\_names) - 1,), axis=1) # x为数据，y为标签

# 判断y数据是否符合xgboost的输入要求

y = CheckY(y)

# msg.emit(x.shape, y.shape)

# 计算16个波段的相关系数矩阵，上三角矩阵

# PlotHeatmap(np.corrcoef(x.T))

msg.emit("\*" \* 30 + " 获取样本完成 " + "\*" \* 30)

msg.emit("\*" \* 30 + " 分割训练集和验证集 " + "\*" \* 30)

# 分割训练集和验证集

train\_data, test\_data, train\_label, test\_label = Split\_train\_test\_dataset(x, y, val\_size=test\_size)

msg.emit("\*" \* 30 + " 分割完成 " + "\*" \* 30)

msg.emit(f"训练集样本数：{train\_data.shape[0]}")

msg.emit(f"验证集样本数：{test\_data.shape[0]}")

msg.emit(f"样本特征数：{train\_data.shape[1]}")

msg.emit(f"类别数：{len(np.unique(train\_label))}")

msg.emit("\*" \* 30 + " 正在进行超参数优化... " + "\*" \* 30)

# 超参数优化

def HyperOptimize(train\_data, train\_label, test\_data, test\_label, max\_evals=100):

a = []

b = []

def hyperopt\_train\_test(params):

skf = model\_selection.StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

f1\_scores = []

for train\_index, val\_index in skf.split(train\_data, train\_label):

X\_train, X\_val = train\_data[train\_index], train\_data[val\_index]

y\_train, y\_val = train\_label[train\_index], train\_label[val\_index]

clf = xgb.XGBClassifier(\*\*params, random\_state=42)

clf.fit(X\_train, y\_train.ravel())

pred\_label = clf.predict(X\_val)

f1 = f1\_score(y\_val, pred\_label, average="macro")

f1\_scores.append(f1)

avg\_f1 = sum(f1\_scores) / len(f1\_scores)

a.append(avg\_f1)

b.append(params)

return -avg\_f1 # 注意返回负的 F1 score，因为hyperopt是最小化目标

def f(params):

acc = hyperopt\_train\_test(params)

return {"loss": acc, "status": STATUS\_OK}

if len(np.unique(train\_label)) == 2:

space["objective"] = "binary:logistic"

space["eval\_metric"] = "logloss"

else:

space["objective"] = "multi:softmax"

space["eval\_metric"] = "mlogloss"

space["num\_class"] = len(np.unique(train\_label))

trials = Trials()

fmin(f, space, algo=tpe.suggest, max\_evals=max\_evals, trials=trials, show\_progressbar=False)

max\_f1\_dict = b[a.index(max(a))]

msg.emit(f"最佳超参数：{max\_f1\_dict}")

return max\_f1\_dict

max\_f1\_dict = HyperOptimize(train\_data, train\_label, test\_data, test\_label)

# 训练模型

classifier = xgb.XGBClassifier(\*\*max\_f1\_dict, random\_state=42)

classifier.fit(train\_data, train\_label.ravel())

msg.emit("\*" \* 30 + " 超参数优化完成 " + "\*" \* 30)

msg.emit("\*" \* 30 + " 训练结果 " + "\*" \* 30)

msg.emit("训练集精度：" + str(classifier.score(train\_data, train\_label)))

msg.emit("验证集精度：" + str(classifier.score(test\_data, test\_label)))

# 预测输出

predict = classifier.predict(test\_data)

msg.emit("平均绝对误差：" + str(mean\_absolute\_error(test\_label, predict)))

msg.emit("召回率：" + str(recall\_score(test\_label, predict, average="macro")))

msg.emit("准确率：" + str(accuracy\_score(test\_label, predict)))

msg.emit("F1值：" + str(f1\_score(test\_label, predict, average="macro")))

# -------------------------------------------------change-------------------------------------

# 交叉验证参数

shuffle = model\_selection.ShuffleSplit(n\_splits=10, test\_size=0.5, random\_state=42)

# 绘制学习曲线

plot\_learning\_curve(

classifier,

"XGB Learning Curve",

x,

y.ravel(),

model\_name="XGB",

# ylim=(0.7, 1.01),

cv=shuffle,

)

# 绘制特征重要性

PlotImportance\_XGB(classifier.feature\_importances\_)

msg.emit("\*" \* 5 + " 学习曲线已生成 " + "\*" \* 5)

# -------------------------------------------------------------------------------------------

# 保存模型

SavePickle(classifier, SavePath)

msg.emit("\*" \* 5 + f" 训练完成,模型已保存至 {SavePath} " + "\*" \* 5)

def LightGBM\_train(

sample\_path,

SavePath,

space,

msg,

test\_size=0.5,

):

msg.emit("\*" \* 30 + " 开始训练 " + "\*" \* 30)

msg.emit("模型：LightGBM")

msg.emit("样本路径：" + sample\_path)

msg.emit("保存路径：" + SavePath)

msg.emit("验证集比例：" + str(test\_size))

msg.emit("\*" \* 30 + " 获取样本中... " + "\*" \* 30)

# 读取第一行以获取列名

if sample\_path.endswith(".txt"):

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

header\_line = file.readline().strip()

elif sample\_path.endswith(".csv"):

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

header\_line = file.readline().strip()

# 获取列名

column\_names = header\_line.split(",")

# 重新定义converters字典，跳过第一行（表头）

converters = {i: label\_dict if column\_names[i] == "class" else int for i in range(len(column\_names))}

# 从第二行开始读取数据

data = np.loadtxt(sample\_path, dtype=int, delimiter=",", skiprows=1, converters=converters)

x, y = np.split(data, indices\_or\_sections=(len(column\_names) - 1,), axis=1) # x为数据，y为标签

y = CheckY(y.ravel())

# print(x.shape, y.shape)

msg.emit("\*" \* 30 + " 获取样本完成 " + "\*" \* 30)

msg.emit("\*" \* 30 + " 分割训练集和验证集 " + "\*" \* 30)

# 分割训练集和验证集

train\_data, test\_data, train\_label, test\_label = Split\_train\_test\_dataset(x, y, val\_size=0.5)

msg.emit("\*" \* 30 + " 分割完成 " + "\*" \* 30)

msg.emit(f"训练集样本数：{train\_data.shape[0]}")

msg.emit(f"验证集样本数：{test\_data.shape[0]}")

msg.emit(f"样本特征数：{train\_data.shape[1]}")

msg.emit(f"类别数：{len(np.unique(train\_label))}")

msg.emit("\*" \* 30 + " 正在进行超参数优化... " + "\*" \* 30)

# 定义超参数优化函数

def HyperOptimize(train\_data, train\_label, test\_data, test\_label, max\_evals):

a = []

b = []

def hyperopt\_train\_test(params):

skf = model\_selection.StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

f1\_scores = []

callbacks = [

lgb.early\_stopping(stopping\_rounds=10, verbose=False),

]

for train\_index, val\_index in skf.split(train\_data, train\_label):

X\_train, X\_val = train\_data[train\_index], train\_data[val\_index]

y\_train, y\_val = train\_label[train\_index], train\_label[val\_index]

clf = lgb.LGBMClassifier(\*\*params, random\_state=42, verbose=-1)

clf.fit(

X\_train,

y\_train.ravel(),

callbacks=callbacks,

eval\_set=[(X\_val, y\_val.ravel())],

eval\_metric="multi\_logloss",

)

pred\_label = clf.predict(X\_val)

f1 = f1\_score(y\_val, pred\_label, average="macro")

f1\_scores.append(f1)

avg\_f1 = sum(f1\_scores) / len(f1\_scores)

a.append(avg\_f1)

b.append(params)

return -avg\_f1 # 注意返回负的 F1 score，因为hyperopt是最小化目标

def f(params):

acc = hyperopt\_train\_test(params)

return {"loss": acc, "status": STATUS\_OK}

# 判断任务是否为二分类

if len(np.unique(train\_label)) == 2:

space["objective"] = "binary"

space["metric"] = "binary\_logloss"

space["num\_class"] = 1

else:

space["objective"] = "multiclass"

space["metric"] = "multi\_logloss"

space["num\_class"] = len(np.unique(train\_label))

trials = Trials()

fmin(f, space, algo=tpe.suggest, max\_evals=max\_evals, trials=trials, show\_progressbar=False)

max\_f1\_dict = b[a.index(max(a))]

msg.emit("最佳参数：" + str(max\_f1\_dict))

return max\_f1\_dict

# 超参数优化

max\_f1\_dict = HyperOptimize(train\_data, train\_label, test\_data, test\_label, max\_evals=100)

msg.emit("\*" \* 30 + " 超参数优化完成 " + "\*" \* 30)

msg.emit("\*" \* 30 + " 训练中... " + "\*" \* 30)

# 训练模型

clf = lgb.LGBMClassifier(\*\*max\_f1\_dict, random\_state=42, verbose=-1)

callbacks = [

lgb.early\_stopping(stopping\_rounds=10, verbose=False),

]

clf.fit(

train\_data,

train\_label.ravel(),

callbacks=callbacks,

eval\_set=[(test\_data, test\_label.ravel())],

)

msg.emit("\*" \* 30 + " 超参数优化完成 " + "\*" \* 30)

msg.emit("\*" \* 30 + " 训练结果 " + "\*" \* 30)

msg.emit("训练集精度：" + str(clf.score(train\_data, train\_label)))

msg.emit("验证集精度：" + str(clf.score(test\_data, test\_label)))

# 预测输出

predict = clf.predict(test\_data)

msg.emit("平均绝对误差：" + str(mean\_absolute\_error(test\_label, predict)))

msg.emit("召回率：" + str(recall\_score(test\_label, predict, average="macro")))

msg.emit("准确率：" + str(accuracy\_score(test\_label, predict)))

msg.emit("F1值：" + str(f1\_score(test\_label, predict, average="macro")))

# -------------------------------------------change------------------------------------------

# # 交叉验证参数

shuffle = model\_selection.ShuffleSplit(n\_splits=10, test\_size=0.5, random\_state=42)

# 绘制学习曲线

plot\_learning\_curve(

clf,

"LightGBM Learning Curve",

train\_data,

train\_label.ravel(),

model\_name="LightGBM",

cv=shuffle,

# ylim=(0.7,1.01),

)

# 绘制特征重要性

PlotImportance\_LGBM(clf.feature\_importances\_, column\_names[:-1])

msg.emit("\*" \* 5 + " 学习曲线已生成 " + "\*" \* 5)

# msg.emit("特征重要性：", clf.feature\_importances\_)

# --------------------------------------------------------------------------------------

# msg.emit("\*" \* 30 + " 训练完成 " + "\*" \* 30)

# 保存模型

SavePickle(clf, SavePath)

msg.emit("\*" \* 5 + f" 训练完成,模型已保存至 {SavePath} " + "\*" \* 5)

# -\*- coding: utf-8 -\*-

# File Name：predict.py

# description: 预测

from utils import \*

def RF\_Predict(RF\_model\_path, img\_path, save\_path, class\_path, msg):

def get\_inf(Landset\_Path):

# 读取影像数据

dataset = gdal.Open(Landset\_Path)

img\_width = dataset.RasterXSize

img\_height = dataset.RasterYSize

img\_bands = dataset.RasterCount

img\_geotrans = dataset.GetGeoTransform()

img\_projection = dataset.GetProjection()

\_img\_ = dataset.ReadAsArray(0, 0, img\_width, img\_height)

return img\_width, img\_height, img\_bands, img\_geotrans, \_img\_, img\_projection

start\_time = time.time()

msg.emit("\*" \* 30 + "读取影像数据" + "\*" \* 30)

img\_width, img\_height, img\_bands, img\_geotrans, \_img\_, img\_projection = get\_inf(img\_path)

msg.emit("\*" \* 30 + "读取完毕" + "\*" \* 30)

# 预测

msg.emit("\*" \* 30 + "预测中..." + "\*" \* 30)

pred = Predict\_RF\_func(RF\_model\_path, \_img\_)

msg.emit("\*" \* 30 + "预测完毕" + "\*" \* 30)

# Plot展示

PlotPredictResult(pred, "RandomForest", class\_path)

# 将结果写到tif图像里

msg.emit("\*" \* 30 + "写入" + "\*" \* 30)

writeTiff(pred, img\_geotrans, img\_projection, save\_path)

msg.emit("\*" \* 30 + "写入完毕" + "\*" \* 30)

end\_time = time.time()

msg.emit("\*" \* 25 + " 预测运行时间:" + str(end\_time - start\_time) + " s " + "\*" \* 25)

def XGB\_Predict(XGB\_model\_path, img\_Path, SavePath, classPath, msg):

def get\_inf(Landset\_Path):

# 读取影像数据

dataset = gdal.Open(Landset\_Path)

img\_width = dataset.RasterXSize

img\_height = dataset.RasterYSize

img\_bands = dataset.RasterCount

img\_geotrans = dataset.GetGeoTransform()

img\_projection = dataset.GetProjection()

\_img\_ = dataset.ReadAsArray(0, 0, img\_width, img\_height)

return img\_width, img\_height, img\_bands, img\_geotrans, \_img\_, img\_projection

start\_time = time.time()

msg.emit("\*" \* 30 + "读取影像数据" + "\*" \* 30)

img\_width, img\_height, img\_bands, img\_geotrans, \_img\_, img\_projection = get\_inf(img\_Path)

msg.emit("\*" \* 30 + "读取完毕" + "\*" \* 30)

# 预测

msg.emit("\*" \* 30 + "预测" + "\*" \* 30)

pred = Predict\_XGB\_func(XGB\_model\_path, \_img\_)

msg.emit("\*" \* 30 + "预测完毕" + "\*" \* 30)

# Plot展示

PlotPredictResult(pred, "XGBoost", classPath)

# 将结果写到tif图像里

msg.emit("\*" \* 30 + "写入" + "\*" \* 30)

writeTiff(pred, img\_geotrans, img\_projection, SavePath)

msg.emit("\*" \* 30 + "写入完毕" + "\*" \* 30)

end\_time = time.time()

msg.emit("\*" \* 25 + "程序运行时间：" + str(end\_time - start\_time) + "s" + "\*" \* 25)

def LightGBM\_Predict(LGBM\_model\_path, img\_path, save\_path, classPath, msg):

def get\_inf(Landset\_Path):

# 读取影像数据

dataset = gdal.Open(Landset\_Path)

img\_width = dataset.RasterXSize

img\_height = dataset.RasterYSize

img\_bands = dataset.RasterCount

img\_geotrans = dataset.GetGeoTransform()

img\_projection = dataset.GetProjection()

\_img\_ = dataset.ReadAsArray(0, 0, img\_width, img\_height)

return img\_width, img\_height, img\_bands, img\_geotrans, \_img\_, img\_projection

start\_time = time.time()

msg.emit("\*" \* 30 + "读取影像数据" + "\*" \* 30)

img\_width, img\_height, img\_bands, img\_geotrans, \_img\_, img\_projection = get\_inf(img\_path)

msg.emit("\*" \* 30 + "读取完毕" + "\*" \* 30)

# 预测

msg.emit("\*" \* 30 + "预测中..." + "\*" \* 30)

pred = Predict\_LGBM\_func(LGBM\_model\_path, \_img\_)

# pred结果加1，还原为真实类别

pred = pred + 1

msg.emit("\*" \* 30 + "预测完毕" + "\*" \* 30)

# Plot展示

PlotPredictResult(pred, "lightGBM", classPath)

# 将结果写到tif图像里

msg.emit("\*" \* 30 + "写入" + "\*" \* 30)

writeTiff(pred, img\_geotrans, img\_projection, save\_path)

msg.emit("\*" \* 30 + "写入完毕" + "\*" \* 30)

end\_time = time.time()

msg.emit("\*" \* 25 + "程序运行时间：" + str(end\_time - start\_time) + "s" + "\*" \* 25)