目录

[引言 4](#_Toc96701180)

[第一章 绪论 5](#_Toc96701181)

[1.1 研究背景 5](#_Toc96701182)

[1.2 研究内容 5](#_Toc96701183)

[1.3 研究意义 5](#_Toc96701184)

[第二章 基础 6](#_Toc96701185)

[2.1 技术基础 6](#_Toc96701186)

[2.1.1 主要编程语言 6](#_Toc96701187)

[2.1.2 GUI工具 6](#_Toc96701188)

[2.2 分类模型 7](#_Toc96701189)

[2.2.1 随机森林 7](#_Toc96701190)

[2.2.2 K-Means 8](#_Toc96701191)

[第三章 设计 10](#_Toc96701192)

[3.1 设计原则 10](#_Toc96701193)

[3.2 功能设计 10](#_Toc96701194)

[3.3 结构设计 11](#_Toc96701195)

[第四章 实现 12](#_Toc96701196)

[4.1 开发环境 12](#_Toc96701197)

[4.12界面实现 12](#_Toc96701198)

[4.3 逻辑实现 13](#_Toc96701199)

[4.4 功能演示 20](#_Toc96701200)

[第五章 总结 25](#_Toc96701201)

[致谢 26](#_Toc96701202)

[参考 27](#_Toc96701203)

# 摘要

# Abstract

# 引言

# 第一章 绪论

## 1.1 研究背景

遥感图像是地物电磁波谱特征的实时记录。人们可以根据记录在图像上的影像特征——地物的光谱特征、空间特征、时间特征等，来推断地物的电磁波性质[[1]](#endnote-1)。近年来，随着航空航天遥感技术的发展，不同遥感平台和传感器采集数据的能力越来越强，可向全球用户提供越来越高的空间、时间、光谱以及辐射分辨率的遥感数据，这为人类认识、探索和研究地理要素空间分布、时空变化等提供了多源基础数据[[2]](#endnote-2)[[3]](#endnote-3)。面对多源巨量遥感数据，如何高效提取消息成为迫切解决的问题，而遥感影像分类是遥感信息提取的重要手段，是目前遥感技术中的热点研究内容。最初的遥感影像分类是通过目视解译来实现，即主要是根据人的经验和知识通过解译基本要素和具体解译标志来识别地物类型，尽管目视解译技术已经很成熟但是由于人工投入大、结果不确定性高、效率低、精度控制困难、解译经验要求高等缺点的存在使得目视解译技术不适合海量遥感数据处理，计算机自动分类技术成为遥感技术与应用研究的重点4。

## 1.2 研究内容

在过去的几十年里, 国内外的专家学者一直致力于研究分类技术与方法来提高遥感影像分类精度[[4]](#endnote-4)，除了传统的分类方法如ISODATA、K-均值、最小距离、最大似然外，国内外学者提出了许多先进的遥感分类算法，如神经网络、决策树、支持向量机等4。

## 1.3 研究意义

当前具备遥感分类功能的商业软件存在处理方式欠缺灵活、硬件配置要求高、个人使用许可过于昂贵等缺点，另外繁杂的功能，对于部分非专业人士在一定程度上提高了进行遥感分类的的门槛。

Python广泛应用于数据分析领域[[5]](#endnote-5)，亦包括遥感数据的处理，但在GIS软件生态里，更多的是扮演脚本的角色，即发挥Python“胶水”语言的特点，调用其他编程语言的软件接口，批量完成重复性、自动化工作。对于纯Python编写的桌面端遥感分类软件，国内外鲜有尝试，本研究将结合基于Python语言及其开源软件包，结合上述提到的部分分类模型，探究一种纯Python实现的遥感分类软件，简化分类步骤，提高分类效率。

# 第二章 基础

## 2.1 技术基础

### 2.1.1 主要编程语言

Python，由Guido van Rossum 于1989 年底开发，是一门广泛使用的解释型、弱类型、高级和通用的编程语言，支持多种编程范式，包括函数式、指令式、结构化和面向对象编程。它拥有动态类型系统和垃圾回收功能，能够自动管理内存使用，并且其本身拥有一个巨大而广泛的标准5。

Python的鲜明特点：

* 代码简洁易读。与其他编程语言相比，Python继承自ABC语言，而ABC这种语言非常强大，是专门为非程序员设计的，其出生决定了它的语法非常像自然语言，对于非软件专业人士而言，选择Python的成本最低，是目前最易学习的程序设计语言之一[[6]](#endnote-6)。
* 可拓展性，具体体现为模块。Python具有最丰富和强大的类库，这些类库覆盖了文件I/O，GIUI编程、网络编程、数据库访问、文件操作等绝大部分应用场景，此外Python社区也很发达，对于一些小众的应用场景，往往也有其对应的开源模块来提供解决方案。
* 跨平台，Pytho作为一门解释型语言天生具有跨平台性，只要平台提供相应的Pyth解释器就能运行。

### 2.1.2 GUI工具

Qt是一个跨平台的C++应用程序开发框架。它提供给开发者建立图形用户界面所需的功能，广泛用于开发GUI程序，也可用于开发非GUI程序，它是完全面向对象的，很容易扩展，并且允许真正地组件编程。

选择Qt作为GUI程序开发具有如下优势：

* 提供图形工具QtDesigner和命令行工具PyUIC、PyRCC，以方便和加速开发过程，前者负责以可视化的方式设计界面，即用拖拽的方式将Widget（控件）排放在界面上，支持版面配置，后两者均为转换工具，PyUIC负责将QtDesigner设计好的.ui文件转换为.py文件，PyRCC则根据.qrc文件的内容将相关的资源在编译过程中嵌入到Qt应用中。
* Qt不但拥有了完备的C++图形库，而且近年来的版本逐渐集成了数据库、OpenGL库、多媒体库、网路、脚本库、XML库、WebKit库等等，其核心库也加入了进程间通信、多线程等模块，极大地丰富了Qt开发大规模复杂跨平台应用程序的能力

PyQt，作为Python语言的图形用户（GUI）编程解决方案之一，是基于Digia公司强大的图形程式框架Qt5的python接口，由一组python模块构成。PyQt5本身拥有超过620个类和6000函数及方法。在可以运行于多个平台，包括：Unix, Windows以及Mac OS。由于PyQt完成了Qt的所有类的封装，因此理论上说，使用Qt能开发出来的东西，使用PyQt也能实现。[[7]](#endnote-7)

## 2.2 分类模型

### 2.2.1 随机森林

随机森林（Random Forest Classifier），是一种基于无参数回归算法的集成学习策略[[8]](#endnote-8)，其集成思想与基本流程如图 1 所示：

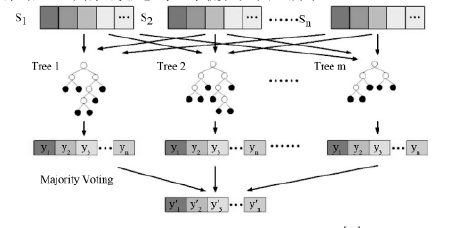


Figure 1[[9]](#endnote-9)

输入原始样本 后，首先从中随机选取一部分构成新的训练集 取代原始训练样本，以降低分类树之间的相关性，提高每棵分类树的精度，进而提高随机森林方法的分类精度。然后从 X 中随机选取一部分样本作  
为 Bootstrap( 进化树) 训练集，构成与该训练集一一对应的回归树 {} ( 其中 是一组独立且具有相同分布的随机向量) ，即分类树。每棵分类树 Tree Predictor( 树预测器) 根据一组与输入样本有的随机向量 进行分裂生长，最终众多分类树构成一个随机森林。最后每棵分类树根据训练集 进行分类，获得各自的分类结果，用多数投票法将所有分类树的分类结果进行综合，就可以得到最终结果。[[10]](#endnote-10)

研究表明，随机森林可以处理高维数据并且适用于大数据量的分类运算，尤其在高维数据分类中，更能体现出其速度快、精度高、稳定性好的优势[[11]](#endnote-11)

### 2.2.2 K-Means

K-均值算法（K-Means）的基本思想是：通过迭代，逐次移动各类的中心，直至得到最好的聚类结果为止。

假设聚类数为已知，其计算步骤为：

1. 任选个初始聚类中心，，括号内的序号为寻找聚类中心的迭代运算的次数序号。为了方便，这里取个模式样本的向量值作为初始聚类中心。
2. 逐个将样本集中各个样本按最小值距离原则分配给个聚类中心的某一个，即：

* 若，则
* 式中， 为运算次数序号，表示地个聚类，其聚类中心为。

1. 计算各向量中心新的向量值。显然，各聚类于中所包含样本的均值向量作为新的聚类中心，即：

* 为第个聚类域中所包含的样本数，以均值向量作为新的聚类中心，可使聚类准则函数
* 最小。这一步要计算个聚类中心的样本均值向量，所谓“k-均值”算法的名称由此得来。

1. 如果，则回到第二步，将模式样本诸葛重新分类，重复迭代计算。

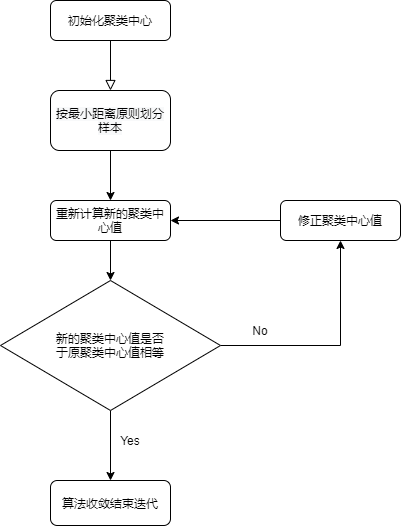
* 若，则算法收敛，计算完毕。[[12]](#endnote-12)
* 

Figure 2 算法流程

# 第三章 设计

## 3.1 设计原则

软件在设计时，应当根据设计设想，满足合理性、适用性、易操作性、可扩性等多个原则。[[13]](#endnote-13)[[14]](#endnote-14)

1. 合理性原则。软件的功能在设计和开发过程中具有可行性，根据遥感分类中的相关方法和用户需求，对软件相关的功能设计时，保证软件功能的可行性，能够完成软件功能的需求设计。
2. 适用性原则。

* 每一项功能设计出来，在实现以后，要具有适用性，能够解决问题，实现用户的需求。

1. 易操作性原则。

* 软件界面应当具有简洁明了、用户友好、操作简便、容易上手。

1. 可拓展性原则。

* 软件设计初版，有很多不足的地方，应在软件结构、功能实现等方面留有余地，以方便后续技术、方法更新，更好地满足用户需求，实现后续的完善和扩充。

## 3.2 功能设计

软件依托国内外提出的相关分类模型，采用Python语言及先开源软件包（package）聚焦于遥感影像分类、分类后可将影像输出为主流图像格式。

1. 影像浏览

* 具备读入本地遥感数据，渲染成像，用户在浏览过程中能实现放大、缩小、平移等基本操作。

1. 影像统计

* 对数据进行基本的单元统计分析，包含计算图像波段的最大值、最小值、亮度范围、峰值的统计图表，供用户决策。

1. 影像分类

* 将影像中每个像元根据其在不同波段的光谱亮度、空间结构特征信息，按照给定规则划分为不同类别，目前仅提供随机森林、K-Means两种分类器。

1. 影像输出

* 将分类后结果输出为主流图像格式，如.tif、.png、.jpg等。

1. 帮助功能

* 提供软件操作指南，帮助目标用户快速熟悉软件操作。

## 3.3 结构设计

软件工程目录如下：

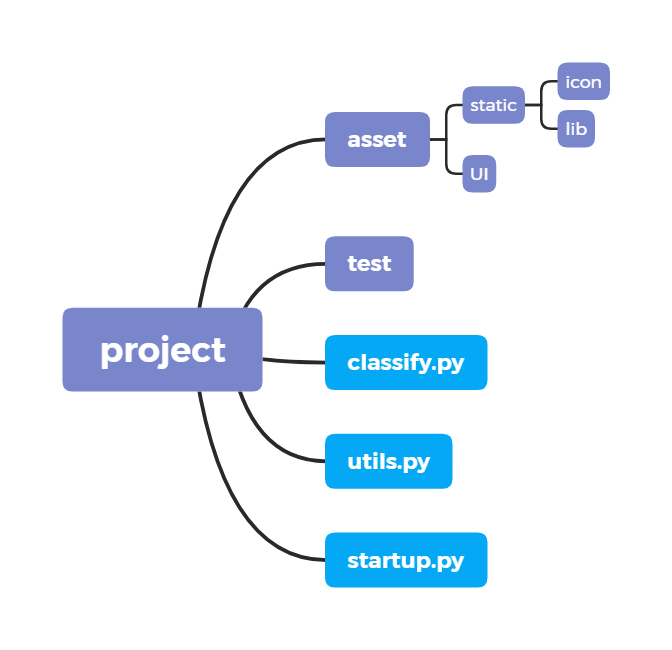


Figure 3结构图

1. asset：资源模块，分别为软件资源目录（static）、ui文件目录、依赖库。
2. test：测试模块，用于软件各部分开发测试。
3. classify.py：分类模块，包含一系列于分类相关的类及函数。
4. utils.py：包含一系列函数，涉及文件I/O、类型转换、拓展线程类等。
5. startup.py：主程序入口文件。

# 第四章 实现

## 4.1 开发环境

PyCharm是一个用于计算机编程的集成开发环境(IDE)，主要用于Python言开发，由捷克公司JetBrains开发，提供代码分析、图形化调试器，集成测试器、集成版本控制系统。

具有如下功能：

* 代码分析与辅助功能，拥有补全代码、高亮语法和错误提示；
* 项目和代码导航：专门的项目视图，文件结构视图和和文件、类、方法和用例的快速跳转；
* 重构：包括重命名，提取方法，引入变量，引入常量、pull，push等；
* 支持网络框架： Django, web2py 和 Flask；
* 集成Python 调试器]；
* 集成单元测试，按行覆盖代码；
* Google App Engine下的Python开发；
* 集成版本控制系统：为Mercurial， Git， Subversion, Perforce 和 CVS提供统一的用户界面，拥有修改以及合并功能。

## 4.12界面实现

根据第三章的设计原则，借助QtDesigner工具，完成整个软件的UI设计出初稿，然后编写代码进行样式修改。

首先创建一个QMainWindow（主窗口）对象，在主窗口上右侧创建菜单栏，放置功能控件按钮，保存为.ui文件，通过PyUIC工具将其转为.py文件，生成一个UI\_MainWindow类。

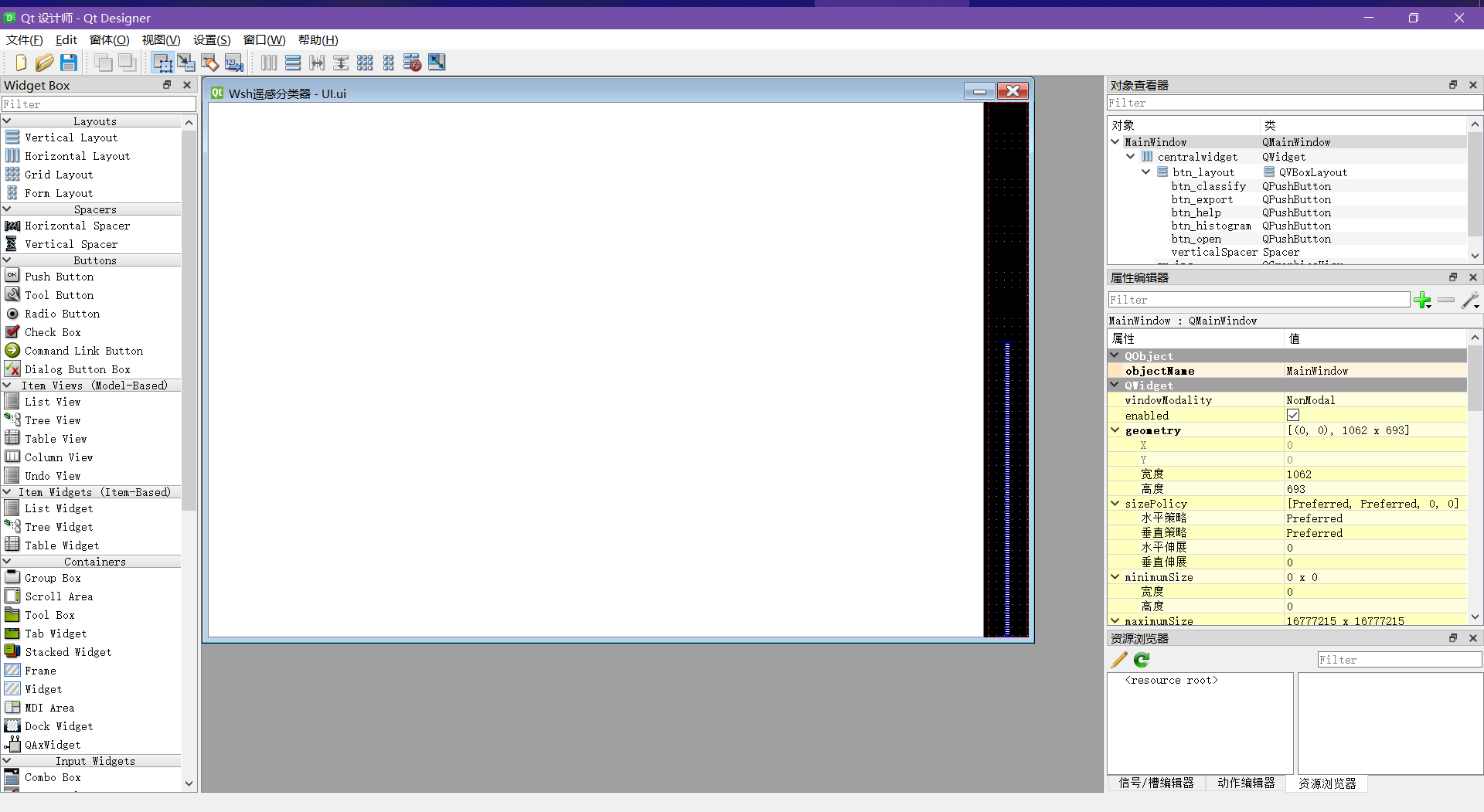


Figure 4 QtDesigner设计界面

## 4.3 逻辑实现

#### 影像浏览与输出

显示GeoTiff文件，且鼠标滚轮放大、缩小、左键平移等操作，需要拓展PyQt模块提供QGraphicsView类，并重写WheelEvent、toggleDragMode、MousePressEvent方法，核心代码如下：

class PhotoViewer(QtWidgets.QGraphicsView):  
 photoClicked = QtCore.pyqtSignal(QtCore.QPoint)  
 def \_\_init\_\_(self, parent):  
 super(PhotoViewer, self).\_\_init\_\_(parent)  
 self.\_zoom = 0  
 self.\_empty = True  
 self.\_scene = QtWidgets.QGraphicsScene(self)  
 self.\_photo = QtWidgets.QGraphicsPixmapItem()  
 self.\_scene.addItem(self.\_photo)  
 self.setScene(self.\_scene)  
 self.setTransformationAnchor(QtWidgets.QGraphicsView.AnchorUnderMouse)  
 self.setResizeAnchor(QtWidgets.QGraphicsView.AnchorUnderMouse)  
 self.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)  
 self.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)  
 self.setBackgroundBrush(QtGui.QBrush(QtGui.QColor(30, 30, 30)))  
 self.setFrameShape(QtWidgets.QFrame.NoFrame)  
  
 def hasPhoto(self):  
 return not self.\_empty  
  
 def fitInView(self, scale=True):  
 rect = QtCore.QRectF(self.\_photo.pixmap().rect())  
 if not rect.isNull():  
 self.setSceneRect(rect)  
 if self.hasPhoto():  
 unity = self.transform().mapRect(QtCore.QRectF(0, 0, 1, 1))  
 self.scale(1 / unity.width(), 1 / unity.height())  
 viewrect = self.viewport().rect()  
 scenerect = self.transform().mapRect(rect)  
 factor = min(viewrect.width() / scenerect.width(),  
 viewrect.height() / scenerect.height())  
 self.scale(factor, factor)  
 self.\_zoom = 0  
  
 def setPhoto(self, pixmap=None):  
 self.\_zoom = 0  
 if pixmap and not pixmap.isNull():  
 self.\_empty = False  
 self.setDragMode(QtWidgets.QGraphicsView.ScrollHandDrag)  
 self.\_photo.setPixmap(pixmap)  
 else:  
 self.\_empty = True  
 self.setDragMode(QtWidgets.QGraphicsView.NoDrag)  
 self.\_photo.setPixmap(QtGui.QPixmap())  
 self.fitInView()  
  
 def wheelEvent(self, event):  
 if self.hasPhoto():  
 if event.angleDelta().y() > 0:  
 factor = 1.25  
 self.\_zoom += 1  
 else:  
 factor = 0.8  
 self.\_zoom -= 1  
 if self.\_zoom > 0:  
 self.scale(factor, factor)  
 elif self.\_zoom == 0:  
 self.fitInView()  
 else:  
 self.\_zoom = 0  
 def toggleDragMode(self):  
 if self.dragMode() == QtWidgets.QGraphicsView.ScrollHandDrag:  
 self.setDragMode(QtWidgets.QGraphicsView.NoDrag)  
 elif not self.\_photo.pixmap().isNull():  
 self.setDragMode(QtWidgets.QGraphicsView.ScrollHandDrag)  
  
 def mousePressEvent(self, event):  
 if self.\_photo.isUnderMouse():  
 self.photoClicked.emit(self.mapToScene(event.pos()).toPoint())  
 super(PhotoViewer, self).mousePressEvent(event)

选择遥感影像，打开浏览后，点击导出按钮，可将影像导出为主流图片格式，示例代码如下：

class Exporter(QThread):  
 def \_\_init\_\_(self,scene, fname):  
 super(Exporter, self).\_\_init\_\_()  
 self.fname = fname  
 self.scene = scene  
 def run(self):  
 try:  
 rect = self.scene.sceneRect()  
 pixmap = QImage(rect.height(), rect.width(), QImage.Format\_ARGB32\_Premultiplied)  
 painter = QPainter(pixmap)  
 rectf = QRectF(0, 0, pixmap.rect().height(), pixmap.rect().width())  
 self.scene.render(painter, rectf, rect)  
 pixmap.save(self.fname)  
 painter.end()  
 except Exception as e:  
 print(e)  
   
# 另存为GeoTiff文件  
 def save(self, X\_cluster):  
 driver = gdal.GetDriverByName("GTiff")  
 out\_img = driver.Create(self.file\_out, self.width, self.height, 1, gdal.GDT\_Byte)  
 out\_img.SetGeoTransform(self.transform)  
 out\_img.SetProjection(self.projection)  
 out\_img.GetRasterBand(1).WriteArray(X\_cluster)  
 out\_img.FlushCache()  
 self.signal.emit("--{:-^20}--".format("保存成功"))

#### 影像统计

在Qt应用展示图表可通过嵌入Matplotlib的Canvas类实现，后者是一个用于绘制静态图表、动画、和交互式可视化图表的Python综合模块。[[15]](#endnote-15)

1. 首先实现一个继承自matplotlib的画布类——MplCanvas，核心代码如下：

* class MplCanvas(FigureCanvasQTAgg):  
   def \_\_init\_\_(self, parent=None, width=16, height=9, dpi=100):  
   fig = Figure(figsize=(width, height), dpi=dpi)  
   super(MplCanvas, self).\_\_init\_\_(fig)

1. 拓展MplCanvas，实现所有嵌入Qt的图表类基类——QtMpl，定义空方法plot，用于绘制图表，核心代码如下：

* class QtMpl(QWidget):  
   def \_\_init\_\_(self):  
   super(QtMpl, self).\_\_init\_\_()  
   self.setGeometry(500, 100,1000, 750)  
   self.set\_ui()  
   def set\_ui(self):  
   self.canvas = MplCanvas()  
   self.tool\_bar = NavigationToolbar(self.canvas, self)  
   self.frame = QVBoxLayout()  
   self.frame.addWidget(self.tool\_bar)  
   self.frame.addWidget(self.canvas)  
   self.setLayout(self.frame)  
    
   def plot(self, \*\*kwargs):  
   pass

1. 拓展图表基类，实现图表类—Chart，相关核心示例代码如下：

* class Chart(QtMpl):  
   def \_\_init\_\_(self, data):  
   super(Chart, self).\_\_init\_\_()  
   self.setGeometry(500, 100,1000, 750)  
   self.setup\_ui()  
   self.data = data  
    
   def setup\_ui(self):  
   # 添加控件，初始化窗体大小、位置等  
   pass  
    
   def plot(self, data=None, flag=None, \*\*kwargs):  
   # 绘制方法  
   self.canvas.figure.clear()  
   ax = self.canvas.figure.add\_subplot(111)  
   ax.spines['top'].set\_visible(False)  
   ax.spines['right'].set\_visible(False)  
   try:  
   if flag == "hist":  
   ax.hist(data, bins=kwargs['bins'], rwidth=kwargs['rwidth'])  
   ax.set\_title("像元分布直方图", fontsize=20, pad=15)  
   ax.set\_xlabel("DN", fontsize=15, labelpad=10)  
   ax.set\_ylabel("频率", fontsize=15, labelpad=10)  
   elif flag == "scatter":  
   ax.plot(data, ".k", markersize=0.1)  
   max\_v = max(data)  
   ax.set\_ylim(0, max\_v + max\_v \* 0.3)  
   elif flag == "line":  
   ax.plot(data, "-.k", label="line")  
   max\_v = max(data)  
   ax.set\_ylim(0, max\_v + max\_v \* 0.3)  
   elif flag == "init":  
   img = Image.open(r'img\_path')  
   ax.imshow(img)  
   ax.set\_yticks([])  
   ax.set\_xticks([])  
   ax.axis('off')  
   self.canvas.draw()  
   except Exception as e:  
   print(e)

#### 影像分类

影像分类属于软件核心功能，点击分类按钮，打开分类面板，可选择合适分类模型，目前内置两种分类模型。

classify.py包含两个分类模型类，拓展自sklearn[[16]](#endnote-16)和Qthread模块，分别为RFC（随机森林）、K\_Means（K近邻），示例代码如下：

class RfC(QThread):  
 signal = pyqtSignal(str)  
 sig\_cls = pyqtSignal(np.ndarray)  
 def \_\_init\_\_(self, file\_input, file\_out, train\_file):  
 super(RfC, self).\_\_init\_\_()  
 self.file\_input = file\_input  
 self.file\_out = file\_out  
 self.train\_file = train\_file  
 self.preparing()  
  
 def preparing(self):  
 img\_ds = gdal.Open(self.file\_input, gdal.GA\_ReadOnly)  
 roi\_ds = gdal.Open(self.train\_file, gdal.GA\_ReadOnly)  
 self.projection = img\_ds.GetProjection()  
 self.transform = img\_ds.GetGeoTransform()  
 self.band\_count = img\_ds.RasterCount  
 self.width = img\_ds.RasterXSize  
 self.height = img\_ds.RasterYSize  
 # 转置影像矩阵  
 img = np.zeros((img\_ds.RasterYSize, img\_ds.RasterXSize, img\_ds.RasterCount),  
 gdal\_array.GDALTypeCodeToNumericTypeCode(img\_ds.GetRasterBand(1).DataType))  
 for b in range(img.shape[2]):  
 img[:, :, b] = img\_ds.GetRasterBand(b + 1).ReadAsArray()  
  
 self.roi = roi\_ds.GetRasterBand(1).ReadAsArray().astype(np.uint8)  
 self.img = img  
 # 关闭  
 img\_ds = None  
 roi\_ds = None  
  
  
  
 def filter\_data(self):  
 img, roi = self.img, self.roi  
 n\_samples = (roi > 0).sum()  
 self.signal.emit('总共有 {n} 条 训练数据'.format(n=n\_samples))  
 self.labels = np.unique(roi[roi > 0])  
 self.signal.emit('训练数据包含 {n} 个地类: {classes}'.format(n=self.labels.size, classes=self.labels))  
  
 X = img[roi > 0, :]  
 y = roi[roi > 0]  
 return X, y  
  
  
 def color\_stretch(self, image, index, minmax=(0, 10000)):  
 colors = image[:, :, index].astype(np.float64)  
 max\_val = minmax[1]  
 min\_val = minmax[0]  
  
 colors[colors[:, :, :] > max\_val] = max\_val  
 colors[colors[:, :, :] < min\_val] = min\_val  
  
 for b in range(colors.shape[2]):  
 colors[:, :, b] = colors[:, :, b] \* 1 / (max\_val - min\_val)  
 return colors  
  
 def create\_rfc(self):  
 X, y = self.filter\_data()  
 img = self.img  
 rf = RandomForestClassifier(n\_estimators=500, oob\_score=True)  
 rf = rf.fit(X, y)  
 # 查看精度  
 self.signal.emit('预测精度为: {oob}%'.format(oob=rf.oob\_score\_ \* 100))  
 new\_shape = (img.shape[0] \* img.shape[1], img.shape[2])  
 img\_as\_array = img[:, :, : 3].reshape(new\_shape)  
 class\_prediction = rf.predict(img\_as\_array)  
 class\_prediction = class\_prediction.reshape(img[:, :, 0].shape)  
 return class\_prediction  
  
 def run(self):  
 cls = self.create\_rfc()  
 self.save(cls)  
 self.sig\_cls.emit(cls)  
 def visualize(self, class\_prediction):  
 plt.imshow(class\_prediction)  
 plt.show()  
   
 def save(self, X\_cluster):  
 driver = gdal.GetDriverByName("GTiff")  
 out\_img = driver.Create(self.file\_out, self.width, self.height, 1, gdal.GDT\_Byte)  
 out\_img.SetGeoTransform(self.transform)  
 out\_img.SetProjection(self.projection)  
 out\_img.GetRasterBand(1).WriteArray(X\_cluster)  
 out\_img.FlushCache()  
 self.signal.emit("--{:-^20}--".format("保存成功"))

class K\_Means(QThread):  
 sig\_cluster = pyqtSignal(np.ndarray)  
 def \_\_init\_\_(self, img\_file, count\_n,iter\_n, out\_file):  
 super(K\_Means, self).\_\_init\_\_()  
 self.n = count\_n  
 self.img\_file = img\_file  
 self.out\_file = out\_file  
 def run(self):  
 cluster = self.classify()  
 self.save(cluster)  
 self.sig\_cluster.emit(cluster)  
  
 def classify(self):  
 img\_ds = gdal.Open(self.img\_file, gdal.GA\_ReadOnly)  
  
 img = np.zeros((img\_ds.RasterYSize, img\_ds.RasterXSize, img\_ds.RasterCount),  
 gdal\_array.GDALTypeCodeToNumericTypeCode(img\_ds.GetRasterBand(1).DataType))  
 [self.cols, self.rows] = img[:, :, 0].shape  
 self.width = img\_ds.RasterXSize  
 self.height = img\_ds.RasterYSize  
 self.trans = img\_ds.GetGeoTransform()  
 self.proj = img\_ds.GetProjection()  
  
 for b in range(img.shape[2]):  
 img[:, :, b] = img\_ds.GetRasterBand(b + 1).ReadAsArray()  
 new\_shape = (img.shape[0] \* img.shape[1], img.shape[2])  
 X = img[:, :, :3].reshape(new\_shape)  
 k\_means = cluster.KMeans(n\_clusters=self.n, max\_iter=500)  
 k\_means.fit(X)  
  
 X\_cluster = k\_means.labels\_  
 self.X\_cluster = X\_cluster.reshape(img[:, :, 0].shape)  
 img\_ds = None  
 return self.X\_cluster  
  
 def save(self, X\_cluster):  
 driver = gdal.GetDriverByName("GTiff")  
 out\_img = driver.Create(self.out\_file, self.width, self.height, 1, gdal.GDT\_Byte)  
 out\_img.SetGeoTransform(self.trans)  
 out\_img.SetProjection(self.proj)  
 out\_img.GetRasterBand(1).WriteArray(X\_cluster)  
 out\_img.FlushCache()  
  
 def visualize(self, data):  
 plt.imshow(self.X\_cluster)  
 plt.show()

#### 使用帮助

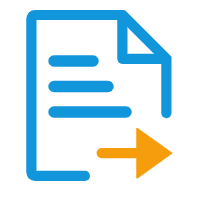
主页面第五个按钮为帮助按钮，点击即可加载本地帮助文档供用户阅读。

PyQt模块内置的QWebEngineView支持渲染html文档，预先编写好软件使用帮助文档，格式为.html。定义show\_help方法，用于在主窗口创建一个子窗口对象和QWebEngineView对象，设定文档来源，并将方法绑定至帮助按钮的点击事件，当该按钮被点击时，即可显示使用文档，示例代码如下：

def show\_help(self):   
 self.help\_win = QMainWindow()   
 self.help\_win.setWindowTitle("帮助文档")   
 self.help\_win.setWindowIcon((QIcon('asset/icon/help.png')))   
 self.help\_win.resize(600, 800)   
 self.wv = QWebEngineView()   
 self.wv.load(   
 QUrl.fromLocalFile(r"E:/Desktop/mypython/PyQt5\RSIC/asset/help.html")   
 )   
 self.help\_win.setCentralWidget(self.wv)   
 self.help\_win.show()

## 4.4 功能演示

1. 影像浏览

选择右侧功能区，单击按钮，进入文件打开对话框，选择目标文件，即可导入影像数据，点击按钮，将当前打开的影像导出为不带地理信息的图片数据。

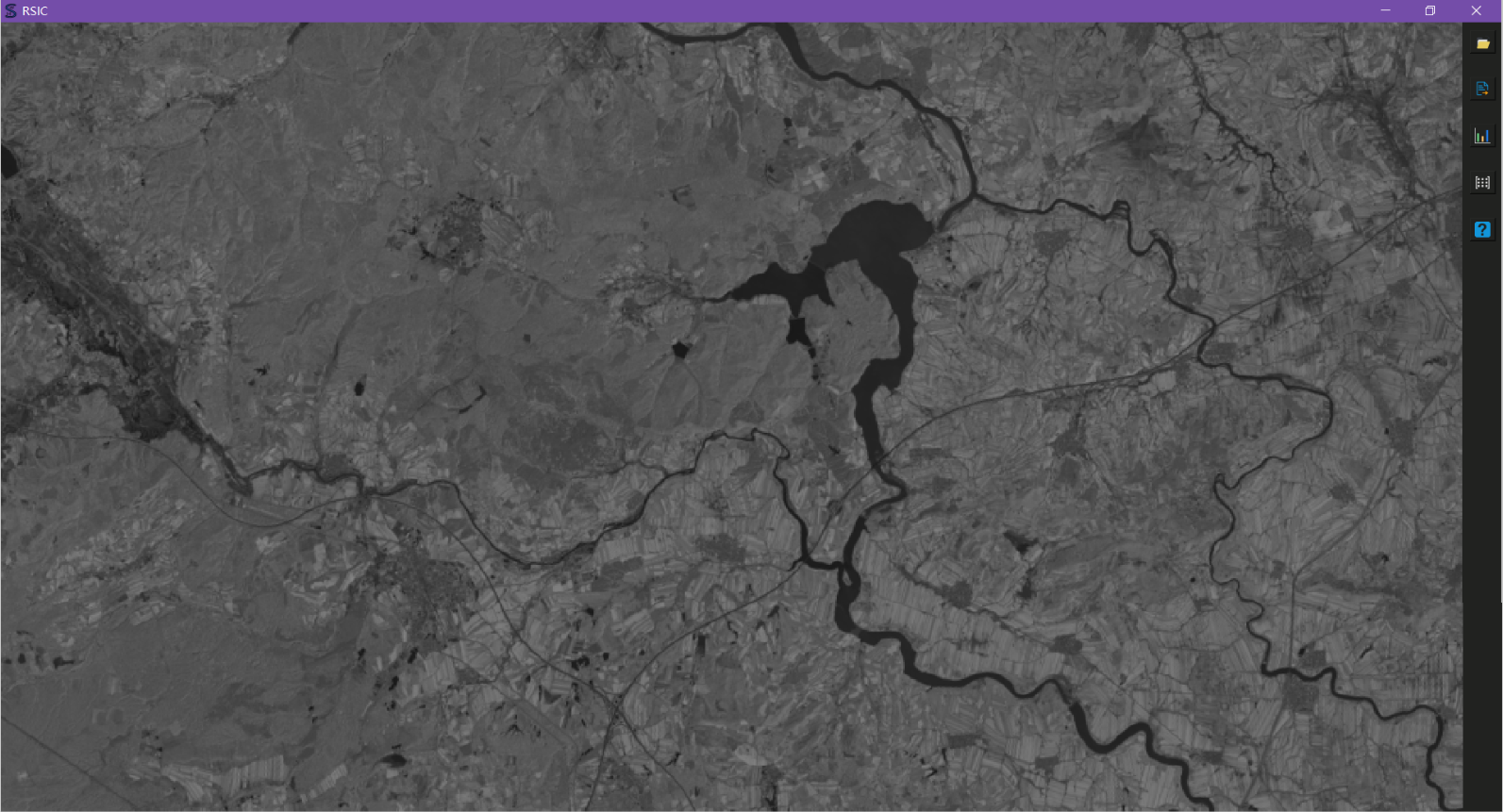
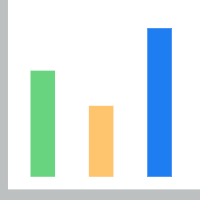


Figure 5浏览影像截图

1. 影像统计

右侧功能区，点击按钮，打开统计面板，选择统计条目，即可显示影像统计信息，面板上方的交互工具可以实现对图表的缩放、平移、导出等操作。

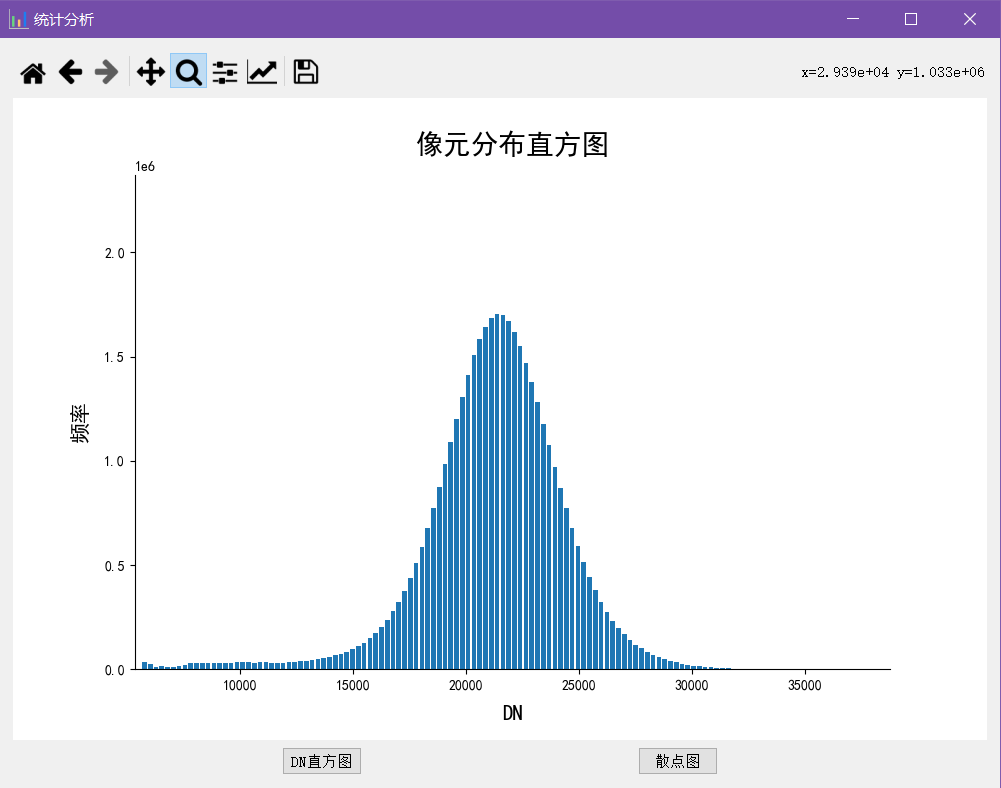
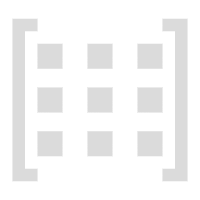


Figure 6像元统计截图

1. 影像分类

右侧功能区，点击按钮，打开分类面板，选择分类模型，输入对应参数。

随机森林模型参数：

1. 输入栅格：待分类遥感影像本地存放路径。
2. 训练数据：训练数据本地存放路径。
3. 输出栅格：分类结果存放路径，默认格式为.tif。

K\_Means模型参数：

1. 输入栅格：待分类遥感影像本地存放路径。
2. 输入类数：指定分类数目。
3. 迭代数：算法迭代次数。
4. 输出栅格：分类结果存放路径，默认格式为.tif

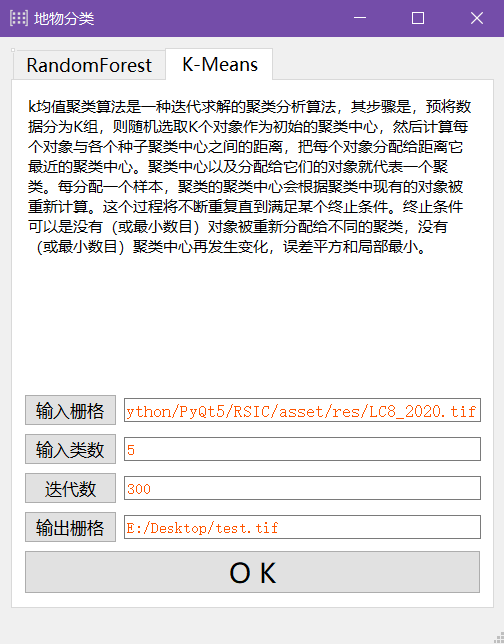


Figure 7分类面板截图

分类完成后，自动预览分类结果，可将结果导出为不带地理信息的图片数据。

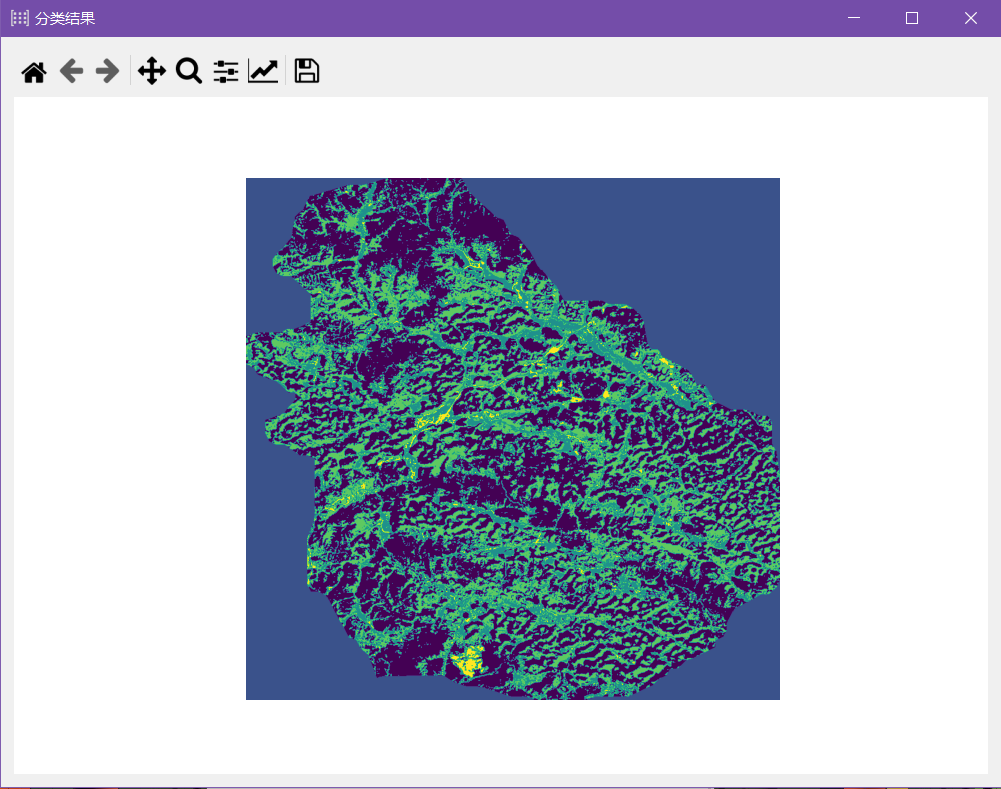


Figure 8分类结构预览截图

1. 使用帮助

右侧功能区，单击，弹出帮助窗口，显示本地帮助文档。



Figure 9帮助页面截图

# 第五章 总结

本文探究了一种将当前流行的编程语言——Python，依托成熟的分类模型，设计并编写纯Python实现的遥感分类软件，从前往后，分别介绍了本次研究的背景、目的、意义、依托的技术基础和理论基础以及最后具体的开发过程。

由于本人能力有限，对于软件的各项功能实现方法也许不是

# 致谢

# 参考

1. 赵英时.遥感应用分析原理与方法（第二版）[M].北京：科学出版社,2013 [↑](#endnote-ref-1)
2. 朱建章,石强,陈凤娥,等.遥感大数据研究现状与发展趋势[J].中国图象图形学报,2016, 21(11): 1425-1439. [↑](#endnote-ref-2)
3. 张兵.智能遥感卫星系统[J].遥感学报,2011,15(3):415-431 [↑](#endnote-ref-3)
4. 贾坤,李强子, 田亦陈,吴炳方.遥感影像分类方法研究进展[J].光谱学与光谱分析,2011,31(10) [↑](#endnote-ref-4)
5. 李刚.疯狂Python讲义[D].北京：电子工业出版社,2019 [↑](#endnote-ref-5)
6. 陶文玲,侯冬青.PyQt5与Qt设计师在GUI开发中的应用[J].湖南邮电职业技术学院学报, 2020(19) [↑](#endnote-ref-6)
7. 王硕,孙,洋.PyQt5快速开发与实战[M].北京：电子工业出版社,2017 [↑](#endnote-ref-7)
8. Breiman L. Random Forests[J].Machine Learning，2001，45 ( 1) : 5-32. [↑](#endnote-ref-8)
9. Yu X W，Hyypp J，Vastaranta M，Holopainen M. Pre-dicting individual tree attributes from airborne laser pointclouds based on the random forests technique[J].Journalof Photogrammetry and Remote Sensing，2011，1( 66) . [↑](#endnote-ref-9)
10. 刘毅,杜培军,郑 辉,夏俊士,柳思聪.基于随机森林的国产小卫星遥感影像分类研究[J].测绘科学,2012,37(4) [↑](#endnote-ref-10)
11. Verikas A，Gelzin A，Bacauskiene M. Mining data with random forests: A survey and results of new tests[J].Pat-tern Recognition，2011，44 ( 2) : 330-349. [↑](#endnote-ref-11)
12. 郭欣.遥感图像的分类[D].北京化工大学硕士研究生学位论文,2004 [↑](#endnote-ref-12)
13. 甘巧燕.基于ArcGIS Engine 的耕地质量等别评价系统[D].2017 [↑](#endnote-ref-13)
14. 陆安琴,秦婵婵,胡圣波,等.基于PyQt5 的数字图像处理实验平台设计[J]. 电子元器件与信息技术,2021,5(2):158-161. [↑](#endnote-ref-14)
15. J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007. [↑](#endnote-ref-15)
16. Scikit-learn: Machine Learning in Python Pedregosa et al., JMLR 12, pp. 2825-2830, 2011. [↑](#endnote-ref-16)